MODC2024 – GRUPO10

Henrique Catarino – 56278

João Osório – 56353

Vasco Maria - 56374

# Phase 2

Este relatório detalha as atividades de descoberta de vulnerabilidades realizadas pelo grupo 10 de segurança em três aplicações: uma aplicação interna (i2) e duas aplicações externas (pt1 e pt2). As ferramentas utilizadas para pentesting foram selecionadas com base na sua eficácia e adequação aos alvos. O relatório inclui a descrição das ferramentas usadas, os comandos executados e os resultados das vulnerabilidades identificadas.

Ferramentas que utilizamos para realizar Pentesting

1. OWASP ZAP

OWASP ZAP é uma das ferramentas mais populares para pentesting, especialmente para análise de segurança de aplicações web. É um scanner de vulnerabilidades que oferece funcionalidades como spidering, fuzzing, e scanner de vulnerabilidades.

OWASP foi usado para detetar vulnerabilidades como xss, com esta ferramenta conseguimos obter informações para penetrar vários níveis do trabalho i2 (Vulnerable-Web-Application (GIT)).

Input que usamos:

1. Introduzimos no nivel 1 - <script>alert(1);</script>
2. Introduzimos no nivel 2,3,4 - <img src=x onerror=prompt()>

2. SQLMap

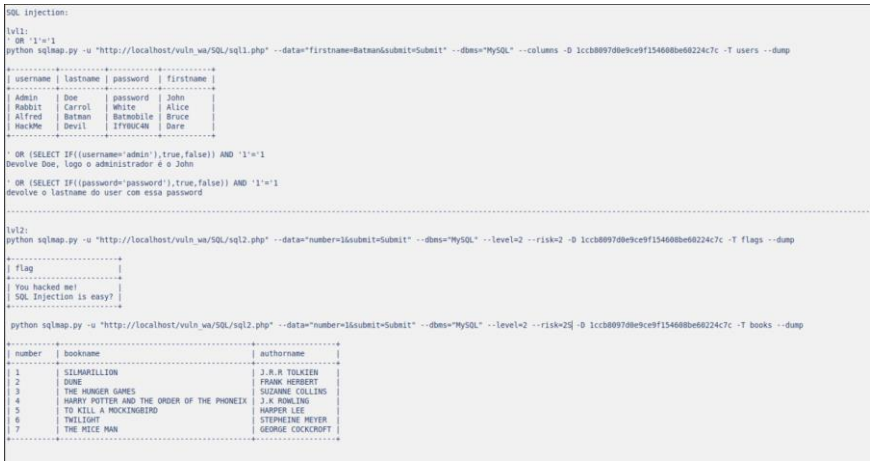
Usamos SQLMap que é uma ferramenta de teste de penetração que automatiza a deteção e exploração de falhas de injeção SQL e também consegue tomar controlo de servidores de banco de dados.
Com o uso desta ferramenta conseguimos obter informações para penetrar vários níveis do trabalho i2 (Vulnerable-Web-Application (GIT)).

Conseguimos penetrar os níveis 1,2,3,4,6. Porém não conseguimos para o nível 5.

Por exemplo para o nível 2 do sql injection utilizamos o certo comando:

a) python sqlmap.py -u "http://localhost/vuln_wa/SQL/sql2.php" --data="number=1&submit=Submit" --dbms="MySQL" --level=2 --risk=2 -D 1ccb8097d0e9ce9f154608be60224c7c -T flags –dump
   i. -u "http://localhost/vuln_wa/SQL/sql2.php": Especifica o URL do alvo que será testado para vulnerabilidades de injeção SQL
   ii. --data="number=1&submit=Submit": Envia dados por o método POST para aoURL especificado. Aqui, os dados enviados são number=1 e submit=Submit para levar os parâmetros do nível 2, ou seja, o Point of attack.
   iii. --dbms="MySQL": Informa ao SQLMap que o sistema de gerenciamento de banco de dados (DBMS) utilizado pelo alvo é MySQL.
   iv. --level=2: Define que usamoso nível 2 de testes que SQLMap deve realizar. O nível 2 é um nível intermediário que realiza testes adicionais além dos padrões.
   v. -T flags: Especifica o nome da tabela dentro do banco de dados que será alvo dos testes de injeção SQL, no caso, a tabela "flags".
   vi. --dump: Indica que SQLMap deve extrair todo o conteúdo da tabela especificada após identificar uma vulnerabilidade de injeção SQL.

Como podemos observar pela foto abaixo para o exemplo do nível 1 ele sugere alguns casos possíveis como o do ' OR '1'='1

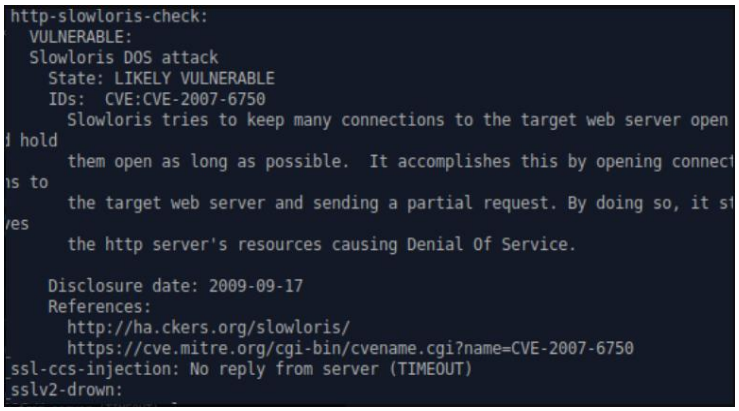


3. NMAP

Utilizamos o Nmap para examinar as portas, identificar sistemas, as suas versões nos trabalhos ssj13 e jvx11.

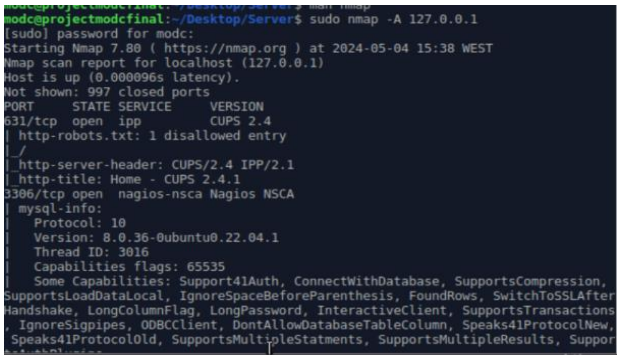
O comando que aplicamos ao tentar examinar as aplicações ssj13 e jvx11 pelo o ip 127.0.0.1 foi o nmap -Pn --script vuln 127.0.0.1.

a. -Pn: Usamos -Pn que vai assumir que o host está ativo
b. --script vuln: O script vuln é um script que executa uma série de testes para várias vulnerabilidades comuns, incluindo aquelas relacionadas a serviços como HTTP, SMB, DNS.




Com o nmap obtemos que os servidores estavam vulneráveis ao slowloris DOS attack. Este tipo de ataque explora a forma como alguns servidores web lidam com conexões HTTP, mantendo múltiplas conexões abertas e inativas, esgotando os recursos do servidor e impedindo-o de

Utilizamos também com a opção -A para ambas as aplicações : sudo nmap -A 127.0.0.1




Assim determinou o sistema operativo em execução no host alvo e identificou as versões dos serviços em execução.

4. Horusec

Para a avaliação de segurança da aplicação SSJ13, utilizamos a ferramenta Horusec para detectar possíveis vulnerabilidades.
Usamos o comando horusec start -D para invocar a ferramenta Horusec e iniciar o processo de análise de segurança, com o modo debug.

Foi detetado sql injection que continha Information Exposure, ao introduzirmos ' OR '1'='1' – no parâmetro da password dava-nos acesso ao login da conta.

Também utilizamos no username ; DROP TABLE GroupChat; -- onde conseguíamos apagar a tabela do chat através de sqlinjection.

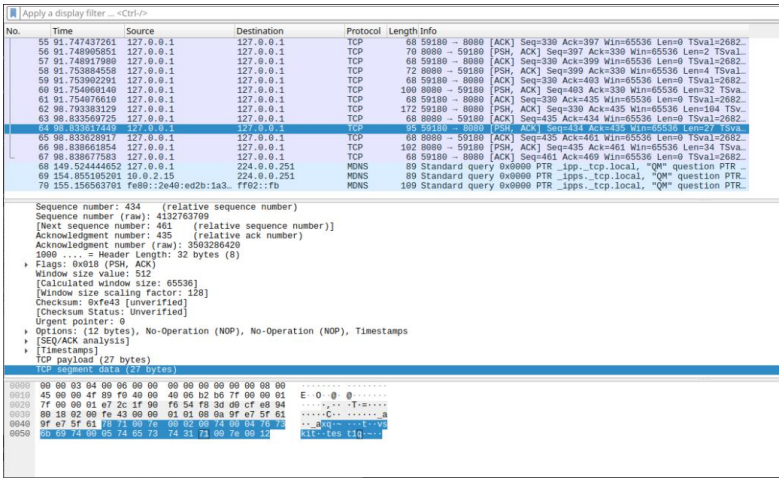
5. Wireshark

Nas aplicações SSJ13 e JVX11, foi utilizado o wireshark para detectar vulnerabilidades relacionadas a sockets TCP, monitorar e analisar o tráfego de rede em tempo real, permitindo identificar e diagnosticar problemas de segurança.

Durante a captura de tráfego das aplicações SSJ13 e JVX11, foi possível observar:

1. Transmissão de Credenciais:
   As credenciais de login, incluindo nome de usuário e senha, foram transmitidas em texto claro onde qualquer pessoa com acesso ao tráfego de rede poderia intercetar e ler essas informações.

2. Troca de Mensagens no Chat:
   As mensagens trocadas entre usuários no chat da aplicação também foram visíveis em texto, sem qualquer forma de criptografia.




Nestes prints podemos observar que é possível observar em texto as credenciais introduzidas por um cliente.

# Phase 3 – Step 3

## Introdução

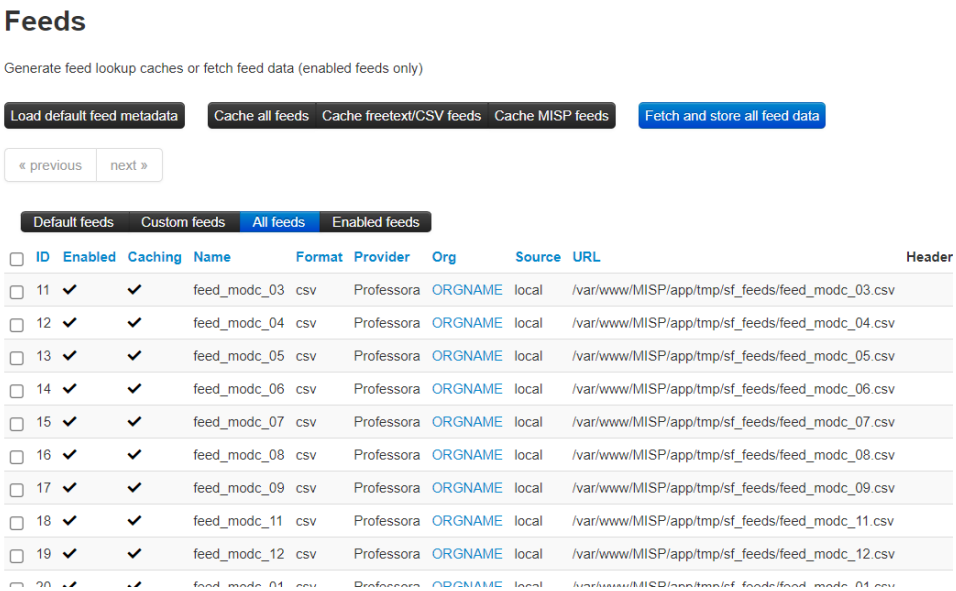
O relatório tem como objetivo falar sobre a utilização do CTI fornecido por feeds para auxiliar no monitoramento e resposta a incidentes. Este processo envolve a adição de feeds ao MISP, configuração de delimitadores de CSV, análise de eventos e correlações, e definição de medidas de remediação. Os pontos abordados são:

1. Verificar se os outros grupos foram capazes de descobrir as vulnerabilidades inseridas na aplicação i1.
2. Para a aplicação i2, verificar se os outros grupos descobriram outras vulnerabilidades além das identificadas pelo grupo.
3. Para a aplicação e1, identificar quais vulnerabilidades estão presentes e definir medidas de remediação.

## Adição e Configuração de Feeds:

Adição dos Feeds: Os feeds dos grupos foram adicionados ao MISP, garantindo que todas as fontes de dados relevantes fossem integradas.

Configuração de Delimitadores CSV: Cada feed foi configurado adequadamente no MISP, ajustando os delimitadores de CSV conforme necessário para assegurar a correta importação dos dados.



**Feeds**

Generate feed lookup caches or fetch feed data (enabled feeds only)

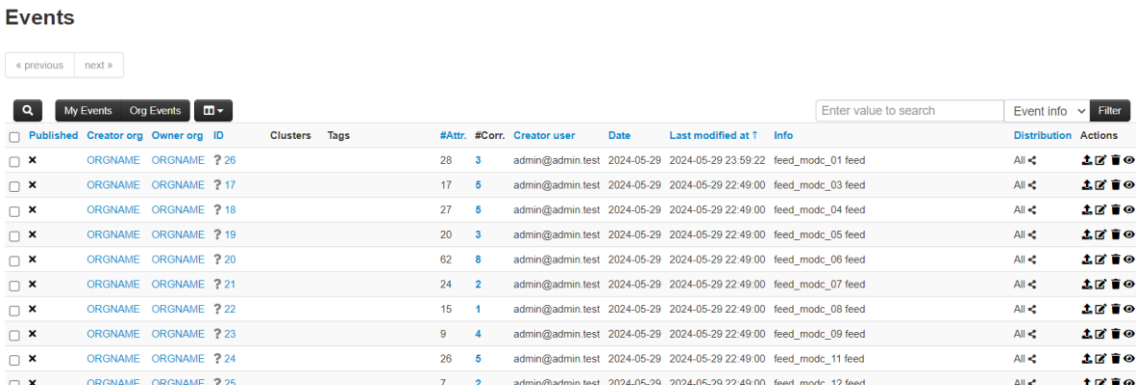
Load default feed metadata | Cache all feeds | Cache freetext/CSV feeds | Cache MISP feeds | Fetch and store all feed data

« previous | next »

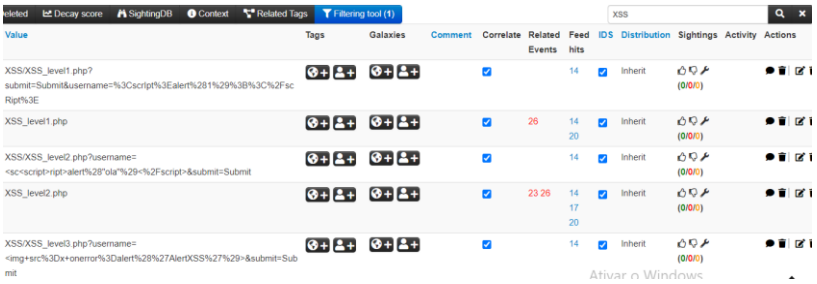
Default feeds | Custom feeds | All feeds | Enabled feeds

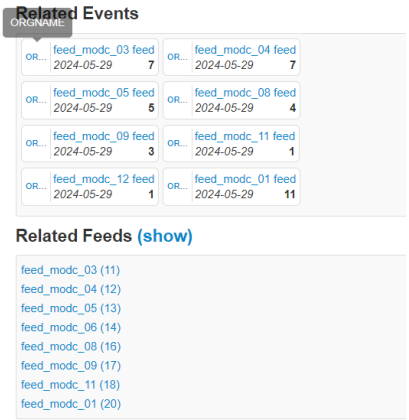
| ID | Enabled | Caching | Name | Format | Provider | Org | Source | URL | Header |
|---|---|---|---|---|---|---|---|---|---|
| 11 | ✓ | ✓ | feed_modc_03 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_03.csv | |
| 12 | ✓ | ✓ | feed_modc_04 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_04.csv | |
| 13 | ✓ | ✓ | feed_modc_05 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_05.csv | |
| 14 | ✓ | ✓ | feed_modc_06 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_06.csv | |
| 15 | ✓ | ✓ | feed_modc_07 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_07.csv | |
| 16 | ✓ | ✓ | feed_modc_08 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_08.csv | |
| 17 | ✓ | ✓ | feed_modc_09 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_09.csv | |
| 18 | ✓ | ✓ | feed_modc_11 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_11.csv | |
| 19 | ✓ | ✓ | feed_modc_12 | csv | Professora | ORGNAME | local | /var/www/MISP/app/tmp/sf_feeds/feed_modc_12.csv | |

# Análise de Eventos e Atributos




- Visualização de Correlações: Utilizamos a funcionalidade do MISP para visualizar correlações entre os feeds, o que permitiu identificar interseções e redundâncias nos dados de vulnerabilidades relatadas.
- Filtros nos Atributos: Aplicamos filtros nos atributos de cada evento relacionado a um feed para isolar informações específicas e relevantes.




1. Verificação de Vulnerabilidades na Aplicação i1 (app desenvolvida por nós)

Foi realizado um cruzamento de dados entre os feeds para verificar se os outros grupos identificaram as vulnerabilidades inseridas na aplicação i1 que a nossa é a abc10. A análise revelou as vulnerabilidades:

1. Cleartext Transmission of Sensitive Information CWE-319
    a. Com a ferramenta Snyk
2. SQL Injection
    a. ' OR '1' = '1 Com a ferramenta Snyk
3. TCP Server Socket
    a. Com a ferramenta Horusec
4. Password Store in Java String Object
    a. Com a ferramenta HCL Appscan CodeSweep
5. Resource Injection in the Socket
    a. Com a ferramenta HCL Appscan CodeSweep
6. Deserialization of Untrusted Data
7. Server-Side Request Forgery

2. Para a aplicação i2, verificamos que descobriram vulnerabilidades adicionais como :

- OS Command Injection:
    1. Vulnerabilidade que permite a execução de comandos do sistema operacional no servidor.
- Remote File Inclusion (RFI):
    1. Vulnerabilidade que permite a inclusão de arquivos remotos no servidor.
- Path Traversal:
    1. Vulnerabilidade que permite o acesso a diretórios e arquivos fora da raiz da web.
- File Upload:
    1. Descrição: Após o envio do arquivo malicious.php, foi possível explorar a vulnerabilidade de upload de arquivos.
- PHP Backdoor:
    1. Descrição: Upload de um backdoor PHP com o Weevely, permitindo controle remoto.

3. Para evitar duplicações nós optamos, que a resposta para a questão três esteja incluída no relatório da fase 4.

# Phase 4

## 1. Introdução

Neste relatório, apresentamos uma análise de risco das vulnerabilidades identificadas na aplicação e1. Inicialmente, foram realizadas diversas avaliações de segurança para identificar possíveis falhas no sistema. Posteriormente, cada vulnerabilidade foi analisada detalhadamente, considerando-se a probabilidade de exploração e o impacto potencial sobre a empresa. Além disso, propomos estratégias de remediação específicas para mitigar os riscos associados, garantindo a integridade, confidencialidade e disponibilidade dos dados. O objetivo deste relatório é fornecer uma visão abrangente dos riscos de segurança e das medidas necessárias para proteger a aplicação e1 de possíveis ameaças e garantir a continuidade das operações da empresa.

## 2. Listagem das Vulnerabilidades

Foram identificadas as seguintes vulnerabilidades:

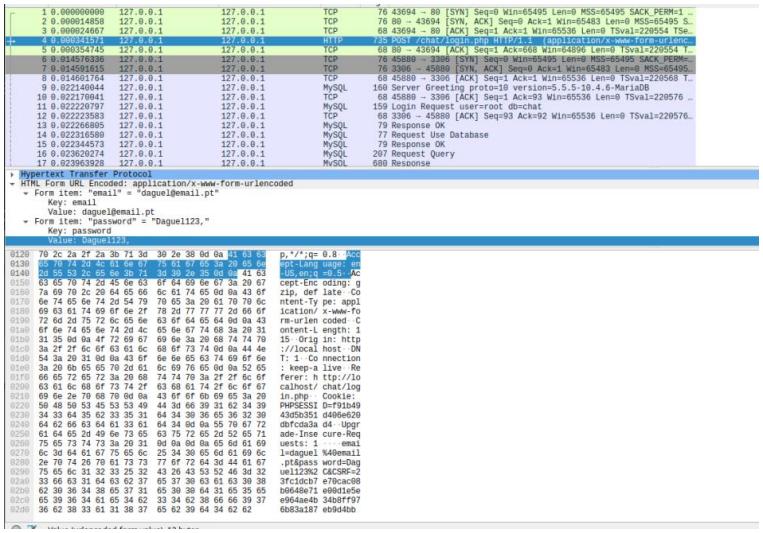
**Injeção de SQL**: Foi descoberto que a aplicação é vulnerável a ataques de injeção SQL, conforme confirmado pelo uso do SQLMap. Esta vulnerabilidade potencialmente permite que atacantes executem comandos SQL maliciosos, comprometendo a integridade e a confidencialidade dos dados.

Por exemplo com o input:
python sqlmap.py -u http://localhost/chat/gestor.php --data="service=getMensagem&sala=cafe" -D chat -T chat_mensagem –dump
É possivel ter acesso e observar o chat.

| id | ip | nome | sala | dataHora | mensagem |
|---|---|---|---|---|---|
| 1 | 127.0.0.1 | c4ca4238a0b923820dcc509a6f75849b (1) | cafe | 2024-05-03 14:25:40 | boas pessoal daqui Ã© o tiagovski e sbem vindos a maz um video de slots |
| 2 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:25:58 | omgggggggggggg tiagovski!!!!!!!!!! |
| 3 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:06 | abre um buraco |
| 4 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:14 | amor |
| 5 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:19 | <blank> |
| 6 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:28 | santa clara |
| 7 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:35 | a |
| 8 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:42 | a |
| 9 | 127.0.0.1 | c81e728d9d4c2f636f067f89cc14862c (2) | cafe | 2024-05-03 14:26:56 | porto |
| 10 | 127.0.0.1 | c4ca4238a0b923820dcc509a6f75849b (1) | OR | 2024-05-03 14:27:56 | bb |

**Transmissão de Mensagens Não Criptografadas**: Observamos que as mensagens transmitidas pela aplicação não estão adequadamente encriptadas, facilitando a interceptação não autorizada de informações confidenciais. Isso foi evidenciado pelo acesso às mensagens através do Wireshark.

Utilizamos também a ferramenta Horusec que nos permitiu compreender os seguintes outputs:



Como vemos pela a print no Wireskark é possível visualizar informações que não deveríamos, como por exemplo saber que a password é igual a Daguel123

**Hard-coded password**: Detecção de credenciais hard-coded, como passwords ou chaves criptográficas, utilizadas para autenticação interna ou comunicação externa, aumentando o risco de exposição de dados sensíveis.
**Password in hardcoded URL**: Password encontrada num URL hard-coded, o que pode levar ao vazamento de passwords e possibilitar ataques CSRF e SSRF mais sofisticados. **Alert statements in JavaScript**: Uso de alert(...) e confirm(...) que podem expor informações sensíveis a atacantes se utilizados em ambiente de produção.
**Repetição de alert statements:** Repetição do problema com alert(...) e confirm(...) em diferentes partes do código, representando o mesmo risco de exposição de informações sensíveis.
**Cross-origin communication verification:** Falta de verificação de origens durante comunicações entre diferentes origens, potencialmente permitindo ataques de falsificação de identidade entre janelas.

## 3. Análise de Risco

A análise de risco é um processo fundamental em segurança da informação, que visa identificar, avaliar e priorizar as vulnerabilidades de uma aplicação ou sistema. As etapas usuais incluem a identificação das ameaças, a avaliação da probabilidade e do impacto de cada ameaça, e a definição de medidas mitigadoras para reduzir os riscos a níveis aceitáveis. A importância desta análise reside na capacidade de proteger os ativos da organização, garantir a continuidade dos negócios, e evitar potenciais perdas financeiras e danos reputacionais. Além disso, uma análise de risco bem conduzida ajuda na conformidade com regulamentos e normas de segurança.

***Riscos Potenciais:***

- Perda de dados devido a SQL Injections, permitindo acesso não autorizado, manipulação de dados e exposição de informações confidenciais.

  Probabilidade: Muito provável (50%+)

  Impacto: Crítico

  Justificação: As vulnerabilidades de SQL Injection são relativamente fáceis de explorar com ferramentas automatizadas, como o SQLMap, e podem causar danos significativos à integridade, confidencialidade e disponibilidade dos dados. Dada a criticidade dos dados envolvidos e a facilidade de exploração, o risco é considerado alto.

  Proprietário do Risco: Gestores de Segurança da Informação

- Perda de dados sensíveis devido ao armazenamento e transmissão insegura de credenciais, incluindo passwords hard-coded no código e URLs, bem como a transmissão de dados não criptografados.

  Probabilidade: Muito provável (50%+)

  Impacto: Crítico

  Justificação**:** A presença de credenciais hard-coded e a transmissão de dados sem criptografia aumentam significativamente o risco de exposição de informações sensíveis. Tais práticas facilitam a exploração por atacantes, resultando em acesso não autorizado e comprometimento da segurança dos dados. Ataques como interceptação de dados são comuns e podem ter graves consequências neste caso, podendo ter consequências catastróficas.

  Proprietário do Risco: Gestores de Segurança da Informação

- Exposição de informações sensíveis devido ao uso de alert statements em JavaScript, incluindo a repetição desse problema em diferentes partes do código.

  Probabilidade: Muito provável (50%+)

  Impacto: Crítico

  Justificação: Alertas em JavaScript, quando utilizados em produção, podem revelar informações internas aos usuários, incluindo dados sensíveis. Isso pode ser explorado por atacantes para obter informações

adicionais sobre a aplicação e seus dados. A repetição do uso de alertas em múltiplas partes do código aumenta a superfície de ataque e proporciona múltiplos vetores de exploração. A remoção e substituição desses alertas por métodos seguros são cruciais para a segurança.

Proprietário do Risco: Gestores de Segurança da Informação

- Exposição de dados sensíveis devido à falta de verificação de origem durante comunicações entre diferentes origens, permitindo ataques de falsificação de identidade entre janelas.

  Probabilidade: Muito provável (50%+)

  Impacto: Crítico

  Justificação: A ausência de verificação de origem nas comunicações entre janelas aumenta significativamente o risco de ataques de falsificação, onde um atacante pode enviar ou interceptar mensagens maliciosas. Isso pode resultar na execução de ações indesejadas ou na obtenção de dados sensíveis, comprometendo a segurança da aplicação e dos usuários. É essencial implementar verificações adequadas de origem para mitigar esses riscos.

  Proprietário do Risco: Gestores de Segurança da Informação

Utilizando uma Matriz de Risco Padrão, como por exemplo na figura abaixo:

| Probabilidade | Muito Baixo | Baixo | Moderado | Alto | Muito Alto |
|---|---|---|---|---|---|
| 90% | Média | Média | Alta | Alta | Alta |
| 70% | Baixa | Média | Média | Alta | Alta |
| 50% | Baixa | Baixa | Média | Alta | Alta |
| 30% | Baixa | Baixa | Média | Média | Alta |
| 10% | Baixa | Baixa | Baixa | Baixa | Média |
| | Impacto | | | | |

Podemos correlacionar a probabilidade com o impacto para obtermos o nível de risco, e podemos verificar que para este caso todos os riscos aqui presentes apresentam nível alto, sendo que necessitam por isso de ser tratados, através de medidas de mitigação, para reduzir o seu nível de risco até um valor que a empresa consiga suportar.

## 4. Estratégias de Mitigação

Para remediar as vulnerabilidades identificadas, recomendamos as seguintes ações:

- **<u>Perda de dados devido a SQL Injections, permitindo acesso não autorizado, manipulação de dados e exposição de informações confidenciais.</u>**

  Propomos a adoção de consultas preparadas (prepared statements) em vez de concatenar diretamente os valores de entrada nas consultas SQL. Isso pode ser alcançado facilmente no PHP e ajuda a prevenir ataques de injeção de SQL, como demonstrado no patch abaixo:
  *Patch*: O patch proposto envolve a utilização de prepared statements, que ajudam a prevenir ataques de injeção de SQL. Portanto, todas as funções devem ser revistas e atualizadas para adotar essa abordagem segura. Por exemplo, na função `obtemMensagens`, a consulta SQL é construída concatenando diretamente o valor da variável `$sala` à consulta, o que a torna vulnerável a ataques de injeção de SQL. Atualizando essa função para utilizar prepared statements, garantimos que os valores de entrada sejam tratados de forma segura, eliminando a possibilidade de injeção de código malicioso.

```
function obtemMensagens($sala){
    $q = "SELECT q.* FROM (SELECT id, nome, dataHora, mensagem from chat_mensagem where sala = '".$sala."' order by id desc limit 20) q ORDER BY q.id ASC";
    $consulta=$this->dados->Execute($q);
    if ($consulta && $consulta->RecordCount()>0) {
        return $consulta;
    }else{
        return false;
    }
}
```

  passaria a:

```
function obtemMensagens($sala) {
    // Prepara a consulta com um espaço reservado (?)
    $q = "SELECT q.* FROM (SELECT id, nome, dataHora, mensagem FROM chat_mensagem WHERE sala = ? ORDER BY id DESC LIMIT 20) q ORDER BY q.id ASC";
    // Prepara a consulta com o espaço reservado substituído pelo valor de $sala
    $consulta = $this->dados->Execute($q, array($sala));
    if ($consulta && $consulta->RecordCount() > 0) {
        return $consulta;
    } else {
        return false;
    }
}
```

  *Testes Pós-Correção:* Após a implementação das correções, recomendamos realizar testes de penetração para confirmar a eliminação das vulnerabilidades de SQL Injection, utilizando ferramentas como SQLMap. Além disso, deve-se conduzir uma revisão de código para garantir que todas as funções foram atualizadas para utilizar prepared statements e

realizar testes de unidade para verificar a funcionalidade correta das funções. Testes de integração são essenciais para assegurar que a comunicação entre os componentes da aplicação continua a funcionar corretamente. Por fim, implementar monitorização contínua para detectar possíveis tentativas de exploração de vulnerabilidades e configurar alertas para atividades suspeitas.

- **Perda de dados sensíveis devido ao armazenamento e transmissão insegura de credenciais, incluindo passwords hard-coded no código e URLs, bem como a transmissão de dados não criptografados.**

  Recomendamos a implementação do protocolo HTTPS em toda a comunicação entre o cliente e o servidor para garantir que os dados sejam transmitidos de forma segura e cifrada. Isto é fundamental para proteger a confidencialidade e integridade das informações trocadas, prevenindo a interceptação e manipulação de dados por terceiros. Além disso, todas as credenciais hard-coded devem ser removidas do código fonte.
  *Patch*: O patch proposto envolve a configuração do servidor web para suportar HTTPS e a obtenção de um certificado SSL/TLS válido. Todos os URLs na aplicação devem ser atualizados para utilizar "https://" em vez de "http://". Além disso, é necessário redirecionar automaticamente todas as solicitações HTTP para HTTPS para garantir que nenhuma comunicação seja transmitida sem criptografia. O patch inclui também substituir todas as credenciais hard-coded e rever o código para garantir que nenhuma password ou chave criptográfica esteja embutida

  *Testes pós correção:* Após a implementação do HTTPS, realizar testes de navegador para garantir a carga correta via HTTPS, testes de penetração com ferramentas como SSL Labs, revisão de código para atualizar URLs, testes de unidade para verificar a funcionalidade das comunicações, e monitoramento contínuo para detectar problemas de segurança e expiração de certificados. Adicionalmente, conduzir uma revisão de código para assegurar que todas as credenciais hard-coded foram removidas e substituídas por referências seguras

- **Exposição de informações sensíveis devido ao uso de alert statements em JavaScript, incluindo a repetição desse problema em diferentes partes do código.**

  Recomendamos a remoção de todos os alert statements (`alert(...)`) do código JavaScript em produção, substituindo-os por métodos seguros e adequados para comunicação e notificação de erros, como logs de servidor ou notificações de interface de utilizador controladas.

*Patch:* O patch proposto envolve rever e atualizar todas as funções que utilizam alert statements. Por exemplo, na função que utiliza `alert(...)` em `/mnt/data/erro.php` e `/mnt/data/index.php`, deve ser alterada para utilizar métodos seguros, como `console.log(...)`.
*Testes pós-correção:* Após a implementação das correções, realizar testes de revisão de código para garantir que todos os alert statements foram removidos ou substituídos, implementar testes de unidade para verificar que as novas abordagens de comunicação e notificação funcionam corretamente. Conduzir testes de penetração para garantir que as mudanças não introduziram novas vulnerabilidades, monitorizar a aplicação em produção para detectar possíveis problemas de comunicação e assegurar que as informações sensíveis não estão a ser expostas.

- **Exposição de dados sensíveis devido à falta de verificação de origem durante comunicações entre diferentes origens, permitindo ataques de falsificação de identidade entre janelas.**

  Para mitigar a exposição de dados sensíveis devido à falta de verificação de origem durante comunicações entre diferentes origens, recomendamos implementar verificações adequadas de origem (origin) nas comunicações entre diferentes janelas (cross-origin). Certificar de que todas as mensagens enviadas especificam explicitamente o destinatário correto, e que todas as mensagens recebidas verificam a origem do remetente antes de processar os dados.
  *Patch:* O patch proposto envolve rever todos os envios de mensagem e eventos de `message` para incluir verificações de origem. Alterar o código para verificar a origem das mensagens recebidas, processando apenas se a origem for confiável.
  *Testes pós correção:* realizar uma revisão de código para garantir que todas as comunicações entre diferentes origens incluem verificações de origem, implementar testes de unidade para verificar que as mensagens são processadas corretamente apenas de origens confiáveis, conduzir testes de penetração para garantir que as mudanças protegem contra ataques de falsificação de identidade, e monitorizar continuamente as comunicações entre janelas para detectar tentativas de exploração de vulnerabilidades.