Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra

Análise e solução de vulnerabilidades em ambiente LAMP baseada em experimentação com Kali Linux

Joinville

2016

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra

ANÁLISE E SOLUÇÃO DE VULNERABILIDADES EM AMBIENTE LAMP BASEADA EM EXPERIMENTAÇÃO COM KALI LINUX

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Charles Christian Miers
Orientador

Joinville, Junho de 2016

ANÁLISE E SOLUÇÃO DE VULNERABILIDADES EM AMBIENTE LAMP BASEADA EM EXPERIMENTAÇÃO COM KALI LINUX

Aurelio Grott, Gabriel Dominico, Victor Lucas de M. Mafra

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Charles Christian Miers - Doutor (orientador)

Charles Christian Miers - Doutor

Charles Christian Miers - Doutor

Agradecimentos

"We must know - we will know!"

- David Hilbert

Lista de Figuras

3.1 Como é um ataque de SQL Injection. Fonte: owasp.org 20

3.2 Como é realizado um ataque XSS. Fonte: acunetix.com 21

3.3 Exemplo de funcionamento do XSSer. Fonte: owasp.org 23

4.1 Vulnerabilidades do site: hackthissite.org. Fonte: Próprio autor 26

4.2 Vulnerabilidades do site: testphp.vulnweb.com. Fonte: Próprio autor . . . 26

4.3 Endereço vulnerável em testphp.vulnweb.com. Fonte: Próprio autor 27

4.4 Nome dos BD. Fonte: Próprio autor 27

4.5 Nome das tabelas no BD. Fonte: Próprio autor 28

4.6 Nome das colunas de um BD. Fonte: Próprio autor 28

4.7 *Dump* do campo *uname* da tabela *users* com *sqlmap*. Fonte: Próprio autor 29

4.8 *Dump* do campo *pass* da tabela *users* com *sqlmap*. Fonte: Próprio autor . 29

4.9 *Scripts* inseridos em testes com *XSSer*. Fonte: Próprio autor 30

4.10 Dados estatísticos dos testes com *XSSer*. Fonte: Próprio autor 31

Lista de Tabelas

Lista de Siglas e Abreviaturas

ASP Apache Software Foundation

BD Banco de dados

CERN Conseil Européen pour la Recherche Nucléaire

CGI Common Gateway Interface

CVE Common Vulnerabilities and Exposures

DNS Domain Name System

DoS Denial of Service

GNU Gnu Not Unix

GPL General Public License

HTTP HyperText Transfer Protocol

InP Infrastructure Provider

LAMP Linux Apache MySQL PHP

LAPP Linux Apache PostgreSQL PHP

PHP Hypertext Preprocessor

SGBD Sistema de Gerenciamento de Banco de Dados

SMTP Simple Mail Transfer Protocol

SO Sistema Operacional

SQL Structured Query Language

UDESC Universidade do Estado de Santa Catarina

XSS Cross Site Scripting

Sumário

Lista de Figuras **3**

Lista de Tabelas **4**

Lista de Siglas e Abreviaturas **5**

1 Introdução **8**

2 Conceitos **9**

2.1 LAMP 9

2.1.1 Histórico 9

2.1.2 Aplicabilidade 10

2.2 Funcionamento e componentes básicos 10

2.2.1 Linux 11

2.2.2 Apache 12

2.2.3 MySQL 12

2.2.4 PHP 14

2.3 Fundamentação de ataques e soluções 15

2.4 *Frameworks* e soluções para análise de vulnerabilidades 15

2.5 Normas, recomendações e boas práticas para análise de vulnerabilidades . . 17

3 LAMP & KALI **18**

3.1 LAMP 18

3.1.1 Vulnerabilidades 18

3.1.2 Métodos de exploração 18

3.1.3 Principais ataques 19
3.2 KALI 21
3.2.1 Ferramentas para análise do LAMP 22
3.2.2 TÉCNICAS X MÉTODOS X FERRAMENTAS 24

4 Ataques e Análise **25**
4.1 Ambiente de testes e plano de testes 25
4.2 Sessão de ataques 25
4.2.1 Análise de Vulnerabilidade com *Vega* 25
4.2.2 Ataques de *SQL Injection* 26
4.2.3 Ataques de *Cross Site Scripting* 30
4.3 Análise dos resultados 31
4.4 Soluções propostas 31

5 Conclusão **34**

Referências Bibliográficas **35**

1 Introdução

Dentre os servidores *web* o Linux Apache MySQL PHP (LAMP) ganha cada vez mais espaço no mercado por oferecer soluções open source, ou seja, sem custo e adaptáveis à cada aplicação. Porém, um dos grandes problemas que afetam servidores utilizando LAMP é a facilidade com que atacantes conseguem invadir e muitas vezes realizar extração de informações privilegiadas, o que deixa pessoas um pouco receosas ao utilizá-lo. Entretanto, neste trabalho apresentam-se testes de invasão em servidores LAMP (realizados em ambientes controlados e próprios para isso), mostrando como é feita e comprovando a possibilidade de invasão destes servidores.

No capítulo 1 é explicado o funcionamento do sistema LAMP, a integração dos componentes, o por quê da realização de testes de invasão e quais são as normas e recomendações para a análise de vulnerabilidade. Em 3 decorre-se sobre as vulnerabilidades do ambiente LAMP e quais os métodos de exploração utilizados para identificar e replicar os principais ataques. Detalhes sobre o ambiente de testes bem como dos ataques realizados, sua análise e soluções relacionadas são expostos em 4.

2 Conceitos

Para que se possa ter uma maior compreensão das vulnerabilidades de um sistema, primeiro deve-se conhecer o mesmo. Para isso, nesse capítulo, os conceitos e definições básicas para compreensão de como o ambiente LAMP funciona são apresentados.

2.1 LAMP

LAMP é conjunto de soluções em forma de uma lista dos componentes centrais usados na implementação de uma aplicação web. Seu uso é predominante entre pequenas e médias empresas, já que tais componentes são de código aberto e não têm custo. LAMP pode ser definido pelo seus componentes, sendo eles: **L**inux (sistema operacional), **A**pache (serividor web), **M**ySQL(software de banco de dados) e **P**HP (linguagem de programação), que também pode se referir a Perl ou Python, apesar de não ser muito comum (ROUSE, 2008) (mais detalhes na Seção 2.2).

Um exemplo do funcionamento de cada componente do LAMP, em um servidor *web*, se dá por um visitante entrando em um *site* pelo seu navegador, o qual irá enviar um pedido para o servidor, no qual todos os componentes LAMP estão instalados e sendo executados. Este servidor pode então prover ao usuário, interação com banco de dados através de *scripts* Hypertext Preprocessor (PHP) através de formulários enviados ao navegador pelo protocolo HyperText Transfer Protocol (HTTP) (BROWN, 2005). Essa combinação de softwares de código aberto em todas as etapas do precesso dá ao LAMP a confiabilidade e o baixo custo que fez com que ele se popularizasse. A forma que essa popularização ocorreu pode ser vista na Seção a seguir.

2.1.1 Histórico

O conceito principal em volta do LAMP (um servidor *web* sem custo) foi possível no início de 1995 quando a Conseil Européen pour la Recherche Nucléaire (CERN) introduziu o conceito de Common Gateway Interface (CGI), que tornou possível aos servidores executar códigos para criar páginas dinâmicas (ROBINSON, 2013). Linux, CERN httpd

e linguagens de programação do lado do servidor como Perl estavam disponíveis gratuitamente, mas conseguir um banco de dados gratuito só foi possível mais tarde, com o lançamento de Postgre95 (POSTGRESQL, 2010).

Ainda em meados de 1995, o servidor HTTP Apache e PHP foram lançados, compondo então o conjunto de aplicações Linux Apache PostgreSQL PHP (LAPP). Finalmente em 1996, MySQL foi lançado e, com ele, o conjunto LAMP completo fez-se possível. A popularidade do LAMP cresceu rapidamente nos anos 90, já que muitas empresas, por razões monetárias, utilizavam *softwares* de código aberto em suas páginas *web* (GEIPEL, 2010). Essa popularidade inicial contribuiu para que o LAMP fosse amplamente utilizado. Assim, se uma falha é encontrada nele, uma grande quantidade de usuários são afetados.

2.1.2 Aplicabilidade

Sistemas LAMP podem ser encontrados em diversos níveis de aplicações. Quem utiliza um sistema operacional com kernel Linux em seu computador pessoal e deseja desenvolver uma aplicação *web* pode fazê-la através da instalação dos componentes faltantes, ou seja, o Apache como servidor HTTP, MySQL como banco de dados local e PHP como linguagem de *script* de servidor local. Após essas configurações, o sistema está pronto para exibir *web sites* criados localmente, os quais poderão conter formulários a serem processados e enviados a outro servidor ou armazenados na própria máquina. Com o ajuste de alguns detalhes (configurações de *firewall* e permissões), este *site* desenvolvido localmente pode se tornar acessível à usuários externos, fazendo com que o sistema se assemelhe à um servidor contratado.

Não se limitando a projetos de desenvolvimento local, o LAMP se encontra frequentemente em servidores providos por Infrastructure Provider (InP)s, nos quais a principal diferença de um sistema local é a falta de interface gráfica, para alguns usuários experientes isso não é um problema e então a instalação dos componentes se dá de forma muito parecida, através de linhas de comando.

2.2 Funcionamento e componentes básicos

Para garantir o bom funcionamento do sistema, é necessário a boa comunicação e integração entre todos os componentes do LAMP. Apesar de serem serviços independentes,

juntos eles constituem um servidor *web* capaz de executar *scripts* (do lado do servidor) e armazenar dados. Detalhes dos componentes do sistema são descritos nas seções 2.2.1 à 2.2.4.

2.2.1 Linux

Linux é o *kernel* atuante como base de diversos Sistema Operacional (SO)s de software livre, tais como Arch, Fedora, Debian e outros. Como *kernel* de diversas distribuições, o Linux é o núcleo do SO atuando em baixo nível, permitindo o bom gerenciamento do SO sobre o *hardware* (LIP, 2004).

Desde sua origem em 1991, sistemas Linux vem crescendo e ganhando uma grande força na computação, atualmente presente em lugares desde a bolsa de valores de Nova York e supercomputadores à telefones celulares e computadores pessoais, o Linux é um software livre desenvolvido de maneira colaborativa (PROFFITT, 2009). Mais de 1.000 desenvolvedores de pelo menos 100 diferentes companhias, contribuíram para cada versão do *kernel* sob a licensa General Public License (GPL) que é baseada em quatro liberdades (FSF, 2016):

- A liberdade de executar o programa como quiser, para qualquer propósito;
- A liberdade para estudar como o programa funciona, e alterá-lo para que ele execute como você queira. Ter acesso ao código fonte é necessário para tal;
- A liberdade para redistribuir cópias para ajudar o próximo; e
- A liberdade para distribuir cópias de suas versões modificadas para outros. Fazendo isso você concede à comunidade a chance de se beneficiarem de suas alterações. Ter acesso ao código fonte é necessário para tal.

Por esses motivos o Linux tem sido bem-sucedido, particularmente como plataforma de servidor: até mesmo em organizações que confiam veemente em sistemas operacionais comerciais como Microsoft Windows, o Linux aparece frequentemente em papeis infraestruturais, como em *gateways* de Simple Mail Transfer Protocol (SMTP) e servidores Domain Name System (DNS) devido a sua confiança, segurança, baixo custo e a qualidade execepcional das aplicações do servidor (BAUER, 2005).

No acrônimo LAMP o componente Linux é o primeiro a aparecer. Sendo a base do sistema, todos os outros componentes (Apache, MySQL e PHP) são executados sob *kernel* Linux, constituindo assim um serviço gratuito disponibilizado através da integração de um conjunto de *softwares* livres.

2.2.2 Apache

Atuando como servidor HTTP no sistema LAMP, o Apache é o servidor *web* mais popular na Internet desde Abril de 1996 (ASF,). O Apache é um projeto de código livre (sob a licensa GPL) da Apache Software Foundation (ASP), o qual tem como objetivo manter um seguro, eficiente e extensível servidor que provê serviços HTTP de acordo com os padrões HTTP atuais.

Como segundo componente do acrônimo LAMP, o servidor Apache é o provedor de serviços HTTP. Sendo capaz de aceitar requisições HTTP, é este componente o responsável por responder ao *browser* uma requisição de página *web*, por exemplo. Solicitações também podem ser efetuadas através de execuções de *scripts* PHP.

Páginas mais complexas, consequentemente, são resultados de sistemas mais complexos. A busca e amostragem de dados em uma página *web* não se dá pelo retorno imediato do servidor à aplicação solicitante, dependendo também da execução de um *script* solicitando um certo conjunto de dados ao banco de dados local através de consultas em Structured Query Language (SQL) executadas pelo Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL, terceiro componente do acrônimo.

2.2.3 MySQL

O Banco de dados (BD) MySQL, foi projetado com base no mSQL, o qual tinha muitos problemas, como não ser rápido e flexível o suficiente para o uso dos usuários (MYSQL, 2016), com isso a necessidade de um novo BD foi aumentando e com base nesse conceito foi desenvolvido o que hoje é conhecido como MySQL, um SGBD que viabiliza programação em SQL presente no servidor LAMP, o qual se comunica com perfeição com a linguagem de programação presente no LAMP (PHP), com isso veio a ser adotado como principal BD nos servidores.

Um BD pode ser definido como uma coleção de dados, para conseguir acessar os

dados armazenados nesse sistema, teve-se a necessidade de criar algum tipo de gerenciador, sendo o MySQL um dos mais usados. Algumas características desse sistema (MYSQL, 2013a):

- **Banco de dados relacional:** a principal diferença desse tipo de BD para os outros é que os dados são guardados em pequenas tabelas de uma forma que seu acesso seja da forma mais eficiente o possível, obtendo um tempo menor de resposta com o servidor e com isso passou a ganhar espaço no mercado.
- ***Open Source:*** esse termo corresponde que qualquer pessoa pode modificar o *software* do jeito que preferir, podendo ajustá-lo conforme a sua necessidade, sendo perfeito para programadores experientes que precisam adaptá-los a seus critérios.
- **Rápido, confiável, escalável e fácil de usar:** como foi criado para atender a grandes quantidades de dados de uma forma mais rápida que seus concorrentes, foi apenas lógico que se tornasse um dos mais rápidos BD. Portanto começou a ser utilizado em grande escala, consequentemente a segurança foi aumentando juntamente com sua escalabilidade para atender a demanda de usuários.

Mesmo com a segurança presente, precisamos ainda tomar algumas atitudes para dificultar que o BD seja acessado por pessoas não autorizadas. Alguns métodos básicos que ajudam a proteger seu BD (MYSQL, 2013b):

- Não prover acesso a ninguém para a tabela usuário do BD MySQL.
- Não guardar senhas sem algum tipo de função *hash* (algoritmo usado para transformar sua senha para uma *string* ilegível.
- Crie senhas aleátorias, porém de fácil memorização.
- Invista em um *firewall*, protegem pelo menos 50% dos ataques feitos contra seu *software*.
- Sempre criptografe os dados que precisam ser enviados pela internet.

Portanto, seguindo algumas boas práticas de segurança relativamente simples, têm-se um BD protegido e qualquer pessoa não autorizada não poderá acessá-lo.

2.2.4 PHP

O PHP foi criado em 1994 por Rasmus Lerdof, o projeto inicial era um simples conjunto de CGIs[1] binários escritos na linguagem de programação C, usados para rastrear as visitas ao seu *site*(GROUP, 2016). Com o tempo, otimizações foram sendo feitas e funcionalidades adicionadas. Sendo lançado em 1998, o PHP 3.0 foi a primeira versão que contém traços do PHP de hoje em dia, incluindo o suporte a programação orientada a objeto. Porém essa versão tinha muita dificuldade em processar aplicações complexas, foi com base nessa premissa que foram lançadas as versões 4.0 e 5.0 (Julho de 2004), principalmente para melhorar seu antecessor e acrescentar dezenas de novos recursos.

Usado principalmente para desenvolvimento *web*, é um *script open source* de uso geral. As principais áreas que *scripts* PHP são mais utilizados (PHP, 2016):

- **Scripts no lado do servidor.** Podendo acessar os resultados do seu programa com um navegador web.
- **Scripts de linha de comando.** Executar os scripts sem um servidor ou navegador, apenas necessita de um interpretador PHP.
- **Escrever aplicações desktop.** Não é a melhor linguagem para se desenvolver aplicações desktop, porém para um programador experiente o PHP tem alguns recursos avançados que permitem escrever esse sistema.

Com as características destacadas acima, pode-se perceber o quão viável o PHP é como linguagem de programação dos servidores LAMP, sendo principalmente por seus *scripts* no lado do servidor, os quais retornam informações de uma forma rápida, eficiente e de fácil acesso.

Uma característica é a escalabilidade que o PHP possui, podendo ser utilizado na maioria dos sistemas operacionais e servidores *web* (**??**). Portanto, pelas características descrita, ele vem sendo aplicado cada vez mais em servidores LAMP, por suas várias extensões que facilitam a conectividade com diversos banco de dados e sua fácil integração com servidores *web*. Logo, observa-se o porquê do PHP ser aplicado em tantas aplicações, seja por seu fácil entendimento, sua maneira simples de programar ou até mesmo sua eficiente integração com os demais componentes do LAMP.

[1]Uma maneira padrão para um servidor *web* passar a solitação do usuário para uma aplicação e receber dados para enviar ao usuário.

2.3 Fundamentação de ataques e soluções

Testes de penetração de servidores *web* não devem ser confundidos com ataques maliciosos. Apesar de possuírem rotinas parecidas, possuem objetivos distintos. Ataques maliciosos são realizados para roubar informações, causar indisponibilidade de serviços (Denial of Service (DoS)) ou qualquer outro evento indesejado ao responsável pelo servidor. Já um teste de penetração consiste em um processo autorizado, programado e sistemático onde se faz uso de vulnerabilidades conhecidas para realizar tentativas de invasão a um servidor, rede ou conteúdos de aplicações.

Testes de penetração podem ser executados de mais de uma maneira para propor mais de um ponto de vista sob a mesma organização. Para tal, existem dois tipos de testes (não exclusivos) que podem ser conduzidos (SANS, 2002):

Teste interno de penetração Realizado com o objetivo de identificar vulnerabilidades com acesso físico ou exposição a engenharia social. Servem para determinar quais vulnerabilidades existem no sistema interno, acessível somente à pessoas autorizadas com acesso a rede interna da organização.

Teste externo de penetração Realizado com o objetivo de identificar vulnerabilidades presentes através de conexões que foram estabelecidas através da conexão entre a organização e a Internet (através do *firewall* ou *gateway*).

Testes de penetração são um método de segurança ofensiva, na qual ocorre a reprodução de uma tentativa real de invasão para que assim sejam identificadas vulnerabilidades na segurança do sistema. Para a execução dos testes existem ferramentas (ou *frameworks*) com configurações predefinidas para auxiliar no decorrer do processo.

2.4 *Frameworks* e soluções para análise de vulnerabilidades

Frameworks para análise de vulnerabilidades são ferramentas para facilitar a detecção de vulnerabilidades em determinado sistema/página *web*, também conseguindo prever a efetividade das medidas tomadas para combater as vulnerabilidades encontradas pelos

frameworks. Porém, para se obter a máxima eficiência destas ferramentas, precisa-se seguir alguns passos para a análise de vulnerabilidades (SANS, 2001):

- **Conduzir avaliação:** consiste em dois objetivos principais, o de planejamento (coletar informações, definir escopo) e o método de aplicar a análise de vulnerabilidades (entrevistar administradores e escanear a segurança.)
- **Identificar exposições:** revisar os dados coletados na fase anterior e classificá-los de acordo com seu nível de perigo, e assim propor soluções.
- **Endereçar exposições:** tentar resolver as exposições identificadas na fase anterior.

Uma dessas ferramentas é o *Nikto*, o qual é uma ferramenta de segurança específica para *websites*, podendo verificar: mais de 6.000 ameaças em potencial localizadas em arquivos ou programas, mais de 1.200 versões desatualizadas de servidores e mais de 270 problemas em servidores específicos (SULLO; LODGE,). Alguns dos principais problemas frequentemente detectados são: arquivos perigosos, serviços mal configurados, *scripts* vulneráveis, entre outros.

Outra ferramenta que pode ser citada é o *Vega*, com objetivos semelhantes ao *Nikto*, porém com dois modos de operação (SUBGRAPH, 2014a):

- ***Scanner* automático:** rastreia automaticamente por *websites*, extraindo *links*, e executa módulos em possíveis pontos de vulnerabilidades.
- ***Proxy* de interceptação:** permite análises detalhadas sobre a interação navegador-aplicação.

Pode-se citar ainda o OpenVAS, uma ferramenta para varredura e gerenciamento de vulnerabilidades. Podendo ser subdividido em duas arquiteturas (OPENVAS, 2016):

Rastreados OpenVAS executa os testes de vulnerabilidades.

Gerenciador OpenVAS consolida a varredura de vulnerabilidade em uma solução completa de gerenciamento de vulnerabilidade.

2.5 Normas, recomendações e boas práticas para análise de vulnerabilidades

Vulnerabilidade pode ser definida como um erro no *software* que permite que uma pessoa não autorizada ganhe acesso ao sistema ou a rede interna (CVE, 2016). Primeiro monitora-se com qual frequência determinada vulnerabilidade pode ocorrer, e o quão prejudicial é para o programa. Quando é descoberto alguma vulnerabilidade com os testes realizados, ocorre a verificação em um banco de dados de vulnerabilidades, sendo o Common Vulnerabilities and Exposures (CVE) uma boa fonte para conhecer um pouco mais sobre o problema encontrado.

Existem também os escaneadores de vulnerabilidade, os quais são *softwares* que ajudam na identificação de possíveis problemas que podem facilitar a corrupção do sistema, identificando, como por exemplo: versões de *softwares* desatualizadas, configurações falhas. Segundo (STANDARDS; TECHNOLOGY, 2008), esses escaneadores podem:

- **Verificar políticas de segurança.**
- **Prover informações sobre alvos para testes de penetração.**
- **Fornecer informações sobre como diminuir as vulnerabilidades descobertas.**

Contudo, esses escaneadores atuam de forma local. Ao identificar de maneira isolada cada falha e vulnerabilidade muitas vezes não passam a proporção real do problema, que se daria em um único contexto com todas as falhas identificadas. Ou seja, várias pequenas vulnerabilidades podem, juntas, proporcionar um grande risco ao servidor.

Outro meio de se prevenir contra ataques pode ser feito com uma simples revisão dos códigos fontes quando estes estão sendo implementados, não apenas nos seus estados finais. Também é viável realizar testes unitários para verificar sua consistência.

3 LAMP & KALI

Neste capítulo será apresentado as principais vulnerabilidades em sistemas LAMP, bem como os principais ataques realizados. Estes ataques/vulnerabilidades serão analisadas com a ajuda do sistema operacional KALI, baseada em distribuição Linux, o qual é especializado em realizar testes de intrusão. Portanto, algumas ferramentas serão descritas e futuramente usadas para a realização de testes.

3.1 LAMP

Para facilitar a análise de um sistema, ou um conjunto de soluções, é interessante separá-lo em componentes menores. Por exemplo, no LAMP cada elemento tem probabilidade de ter uma vulnerabilidade e além dessa vulnerabilidade, como esses elementos interagem entre si, há também essas interações a serem consideradas como possíveis fatores de risco, tendo em vista que elas podem ser feitas de maneira errônea.

3.1.1 Vulnerabilidades

Como os elementos do LAMP oferecem riscos por si só, uma boa prática para minimizar esses riscos é manter todos os componentes devidamente atualizados, pois assim ataques já conhecidos serão prevenidos (OWASP, 2013a). Outra grande parcela, se dá por configuração incorreta do LAMP (OWASP, 2013b). Por isso, sempre recomenda-se instalação por usuários não leigos.

3.1.2 Métodos de exploração

Vários métodos diferentes podem ser utilizados para encontrar uma vulnerabilidade. Se as vulnerabilidades mais elementares não forem prevenidas, até mesmo serviços mais básicos encontrados no Kali como o *whois* (que pode ser também encontrado *online*) podem ser utilizados para encontrar tais vulnerabilidades. Métodos mais avançados para analisar servidores LAMP são discutidos na Seção 3.2.1.

3.1.3 Principais ataques

Ataques direcionados a servidores baseados em LAMP tem como objetivo explorar as vulnerabilidades das aplicações. Geralmente ataques são confundidos com vulnerabilidades, porém os mesmos não são sinônimos, ataque é algo que o atacante realiza e não uma falha do sistema. Alguns dos principais ataques serão apresentados, bem como sua descrição e os problemas que são causados caso o sistema seja afetado por esses ataques.

SQL Injection

Um dos principais ataques é conhecido como SQL *injection*, o qual consiste em inserir uma consulta SQL do cliente para a aplicação. Se o ataque for bem-sucedido o atacante conseguirá acessar e modificar informações do BD, o qual guarda todos os dados referentes ao sistema, com isso pode-se arruinar todo a aplicação.

Portanto, as consequências são graves. Pode-se citar algumas delas quando o sistema for comprometido (OWASP, 2016c), como:

- **Confidenciabilidade:** como os BD guardam informações sobre tudo, perca de confidenciabilidade é um problema frequente.
- **Autenticação:** se os comandos SQL que são usados para verificar senhas são mal escritos, há a possibilidade do atacante se conectar ao sistema como outro usuário.
- **Autorização:** se a informação de autorização está no BD, então é possível que o atacante mude essa informação a ser favor.
- **Integridade:** se é possível a leitura de informações restristas, então o atacante também conseguirá mudar ou até mesmo deletar informações com o ataque de SQL *injection*.

Com isso, percebe-se o grave prejuízo do sistema vulnerável ao ataque de SQL *injection*, com perdas de informações importantes e até mesmo o comprometimento total do sistema uma vez que o atacante tem acesso ao BD da aplicação.

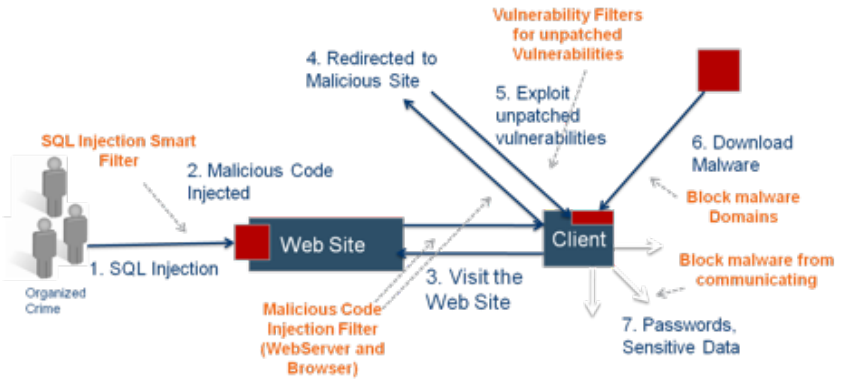

Figura 3.1: Como é um ataque de SQL Injection. Fonte: owasp.org

Na fig. 3.1 têm-se como os atacantes conseguem se infiltrar no sistema e assim roubar dados dos usuários.

Cross-site Scripting (XSS)

Outra vulnerabilidade utilizada pelos atacantes para conseguir infiltrar em seu sistema é o Cross-site Scripting. Sendo que o atacante pode injetar *scripts* maliciosos dentro de seu sistema e conseguir com que o *browser* execute esse script por parte do cliente (*client-side script*). Um exemplo deste tipo de ataque pode ser vista em uma página qualquer que possui campo de login e senha, sendo estes dados, quando inseridos pelos usuários, enviados a um BD e assim o atacando consegue visualizar o login e senha do usuário.

Um ataque pode ser caracterizado como Cross Site Scripting (XSS) segundo (OWASP, 2016a) quando:

- Dados entram na aplicação *Web* por uma fonte de dado não confiável, geralmente por uma requisição *web*.
- Os dados que são incluídos em contexto dinâmico e que foram enviados por um usuário sem que tenha sido validado para conteúdo malicioso.

Muitas vezes atacantes utilizam de códigos JavaScript para conseguirem se infiltrar em sistemas com vulnerabilidade e assim roubar dados de usuários e até mesmo dos própios proprietários do sistema.

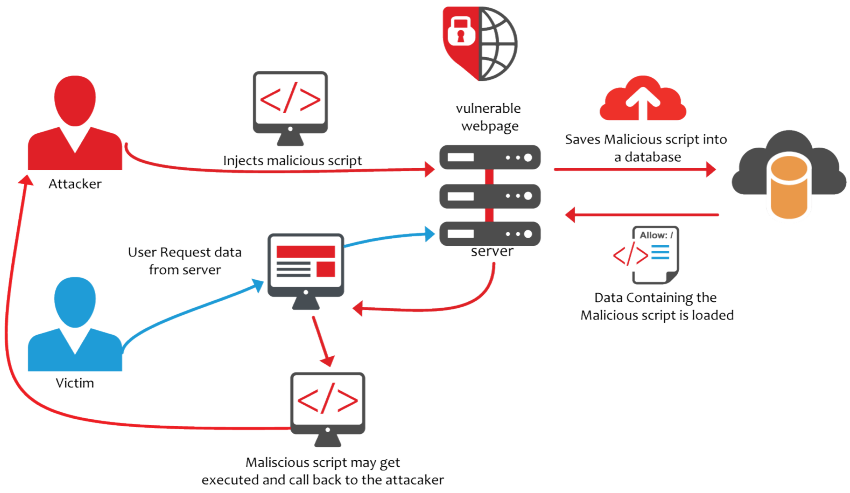

Figura 3.2: Como é realizado um ataque XSS. Fonte: acunetix.com

Na fig. 3.2 têm-se um exemplo da arquitetura de um ataque XSS.

3.2 KALI

O sistema operacional KALI Linux é uma distribuição Linux especializada em testes de intrusão. Com isso, o sistema possui muitas funcionalidades para que possa ser um dos melhores sistemas para a realização de testes de penetração (KALI, 2016b), como:

- **Mais de 600 ferramentas para testes de penetrações.**
- ***Open source* e gratuito.**
- **Desenvolvido por um pequeno grupo, utilizando protocolos seguros.**
- **Suporte a múltiplos idiomas.**
- **Completamente customizável.**

Com todas essas características percebe-se o quão poderoso o Kali Linux pode ser quando utilizado por pessoas que necessitam realizar testes de penetração. Porém, como o sistema foi desenvolvido para necessidades específicas, usuários que não conhecem/utilizam distribuições Linux podem ter dificuldades com relação as ferramentas, pois

mesmo ele sendo *open source* não são todos os usuários que podem contribuir para o seu desenvolvimento (questões de segurança).

3.2.1 Ferramentas para análise do LAMP

Ferramentas para análise do LAMP são acessórios que facilitam ao se procurar por vulnerabilidades. Nesta seção apresentar-se-á algumas dessas ferramentas, descrevendo-as e informando algumas características específicas, todas as ferramentas aqui apresentadas estão disponibilizadas sem custo e nativamente quando instalado o sistema operacional Kali Linux.

Vega

Uma ferramenta que será utilizada para a análise do LAMP será o *Vega*, o qual é gratuito e *open source*, sua especialidade consiste em análise de vulnerabilidade em servidores *web*. Neste acessório têm-se incluido um escaneador automático, o qual possui a função de realizar testes rápidos, as vulnerabilidades que o *Vega* pode encontrar: XSS (*cross-site scripting*, SQL injection, dentre outras (KALI, 2016a; SUBGRAPH, 2014b).

Algumas características que esta ferramenta apresenta:

- **Escaneador automático.**
- **Interceptador de *proxy*.**
- **Escaneador de *proxy*.**

Portanto, percebe-se que caso o objetivo seja procurar por vulnerabilidades em relação a *websites* utilizando LAMP, o *Vega* é uma ferramenta simples, fácil de usar e que cumpre seu objetivo com perfeição.

sqlmap

Outra ferramenta que será utilizada para testes e posteriormente apresentado resultados na seção será o *sqlmap*, o qual é uma ferramenta *open source* especializada em SQL injection e que automatiza o processo de detecção e exploração. Segundo (SQLMAP, 2016), esta ferramenta possui algumas funcionalidades/características:

- Suporte para diferentes Banco de Dados.
- Suporte para diferentes técnicas de SQL *injection*.
- Suporte para enumerar usuários, *password hashes*, tabelas e colunas do BD.
- Suporte para derrubar tabelas do BD.

Entretanto, mesmo com esta ferramenta proporcionando várias funcionalidades, necessita-se de um bom conhecimento para a realização de um ataque.

XSSer

Uma ferramenta que pode facilitar na hora de realizar ataques de XSS é a ferramenta XSSer, a qual é automizada para detectar, explorar e reportar vulnerabilidades de XSS em aplicações *web*. Para utilizar desta ferramenta é necessário ter a linguagem Python instalada e algumas bibliotecas específicas.

Na figura 3.3 têm-se um exemplo do funcionamento da ferramenta XSSer.

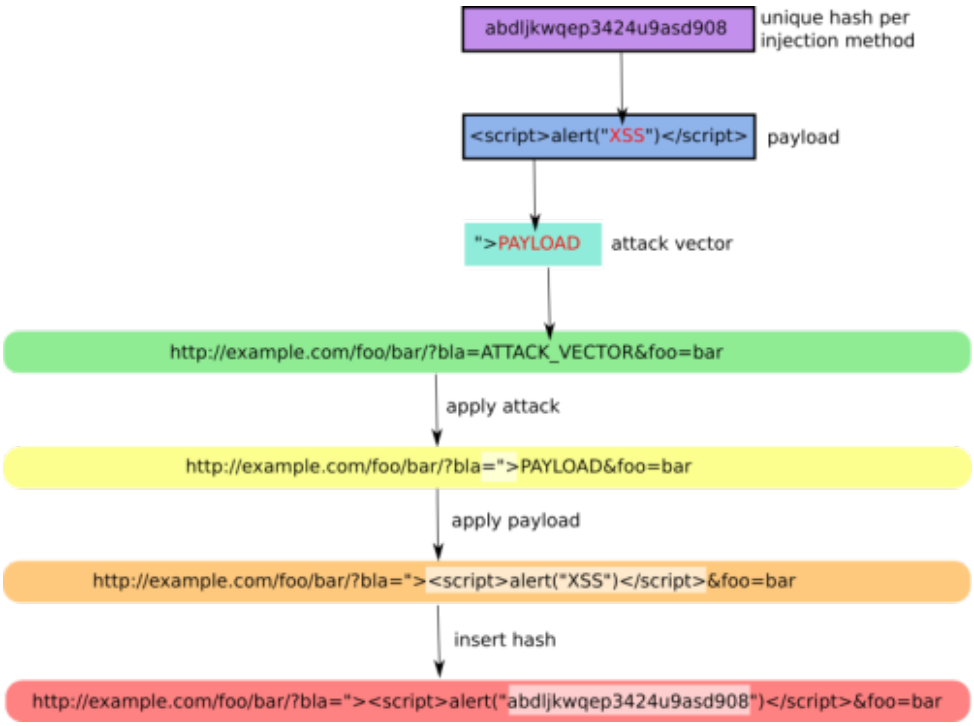

Figura 3.3: Exemplo de funcionamento do XSSer. Fonte: owasp.org

3.2.2 TÉCNICAS X MÉTODOS X FERRAMENTAS

Técnicas, métodos e ferramentas se trabalhados em conjunto conseguem realizar até os mais minuciosos testes de vulnerabilidades e consequentemente verificar e possivelmente corrigir os problemas apresentados. Porém, integrar estes três pode ser uma tarefa bastante complexa, sendo necessário a experiência de quem ficará responsável pelo monitoramente e a realização de testes de penetração das aplicações LAMP.

Se vistos de forma separados, técnica pode ser dita como a experiência do atacante, representando como o ataque irá ser realizado, já método pode ser visto como qual será o ataque a ser realizado, e ferramenta como um facilitador para que o atacante consiga acessar sua aplicação de forma ilegal. Portanto, percebe-se que não pode faltar nenhum desses componentes para que haja um aproveitamento máximo dos testes de penetração.

4 Ataques e Análise

Neste capítulo será apresentado os ataques realizados e algumas análises dos mesmos, bem como possíveis soluções para os problemas encontrados. Para a realização dos ataques será utilizado algumas ferramentas auxiliares, como: *Vega*, *sqlmap* e *XSSer* (ver seção 3 para mais detalhes).

4.1 Ambiente de testes e plano de testes

Todos os testes aqui apresentados foram executados em *sites* próprios para testes de penetração, ou seja, são *sites* próprios para que sejam feitos os devidos testes e com isso aprimorar habilidades sem nenhum tipo de dano a terceiros. Estes *sites* são disponibilizados pela (OWASP, 2016b), são eles: `www.testphp.vulnweb.com` e `www.hackthissite.org`.

Também foram realizados testes com auxílio de ferramentas específicas para cada tipo de ataque (comentado em 3.1.3).

4.2 Sessão de ataques

Viabilizada por ferramentas *open-source* já citadas em 3.2.1, a sessão de ataques seguiu a seguinte ordem lógica:

1. Análise das vulnerabilidades com o *Vega*;
2. Utilização do *sqlmap* para execução de *SQL Injection*; e
3. Utilização do *XSSer* para execução de *Cross Site Scripting*.

4.2.1 Análise de Vulnerabilidade com *Vega*

As figuras 4.1 e 4.2 representam as vulnerabilidades encontradas quando utilizado do *Vega Scan* (ver 3.2.1 para mais detalhes).

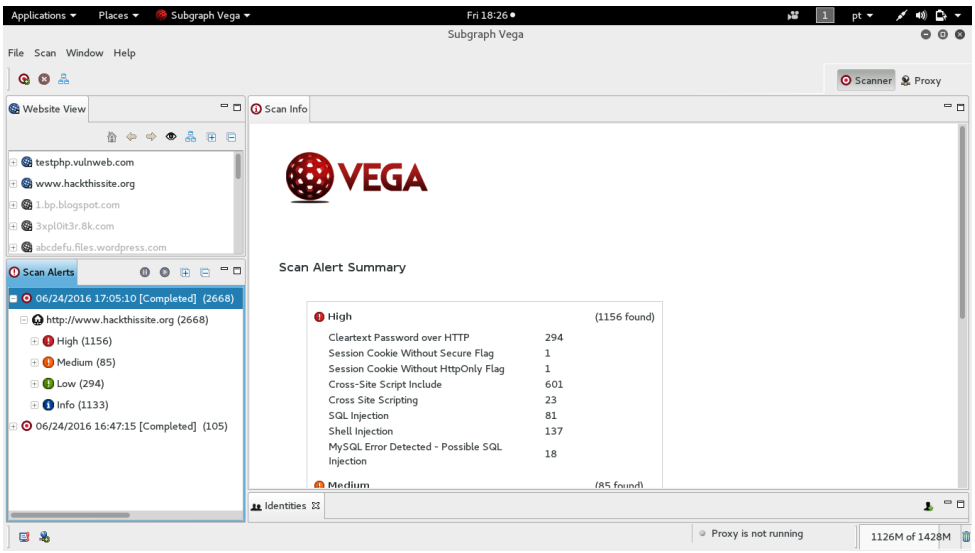

Figura 4.1: Vulnerabilidades do site: hackthissite.org. Fonte: Próprio autor

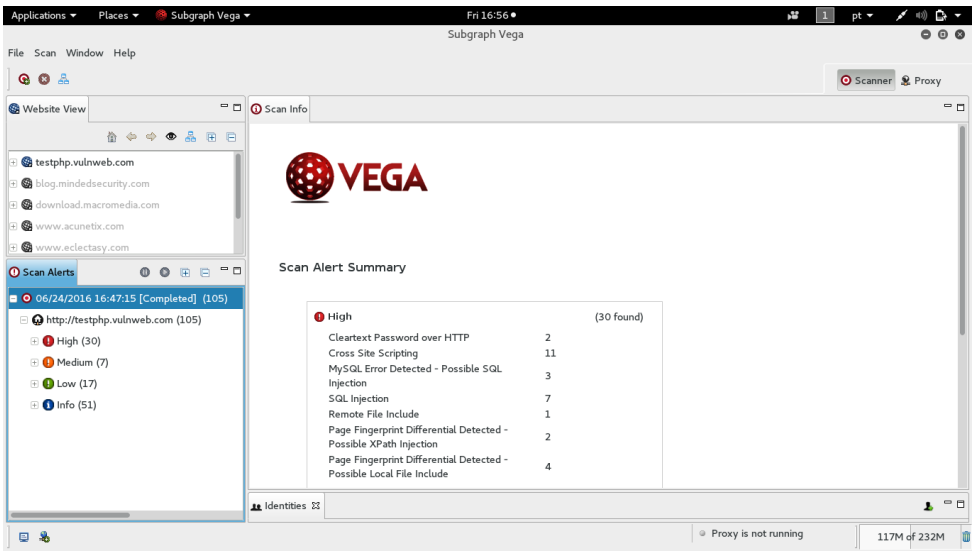

Figura 4.2: Vulnerabilidades do site: testphp.vulnweb.com. Fonte: Próprio autor

O *Vega* realiza então a classificação das ameaças como sendo de risco alto, médio ou baixo. Tendo então detectado um total de 1535 ameaças em `www.hackthissite.org` e 54 ameaças em `www.testphp.vulnweb.com`.

4.2.2 Ataques de *SQL Injection*

De acordo com a análise do *Vega* dentre as 54 ameaças encontradas em `www.testphp.vulnweb.com` (figura 4.2) existem 7 possíveis endereços vulneráveis à *SQL Injection* e um dos possíveis endereços pode ser visualizado na figura 4.3.

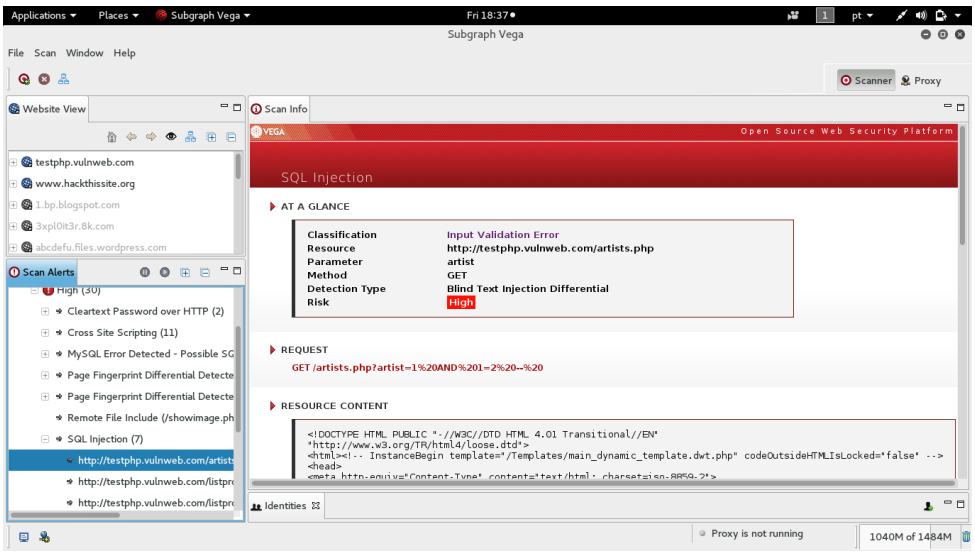

Figura 4.3: Endereço vulnerável em testphp.vulnweb.com. Fonte: Próprio autor

Após obter através do *Vega* o endereço onde será realizado o ataque, o próximo passo fora a utilização do *sqlmap* para obter inicialmente o nome dos bancos de dados disponíveis.

Listing 4.1: Comando para retornar a lista de bancos disponíveis

```
1 $ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1%20AND%201=2%20--%20
2 --dbs --threads 3
```

Com o comando 4.1 é possível obter a lista dos bancos de dados, como visto na figura 4.4.

Figura 4.4: Nome dos BD. Fonte: Próprio autor

Dentre os nomes identificados tem-se o banco *information_schema*, o qual costuma ser um banco de dados padrão do sistema. Logo, o foco dos ataques será o banco *acuart*. Para isso executa-se então o comando 4.2.

Listing 4.2: Comando para retornar a lista de tabelas

```
$ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1%20AND%201=2%20--%20
--tables --threads 3
```

A figura 4.5 exemplifica o retorno da operação realizada.

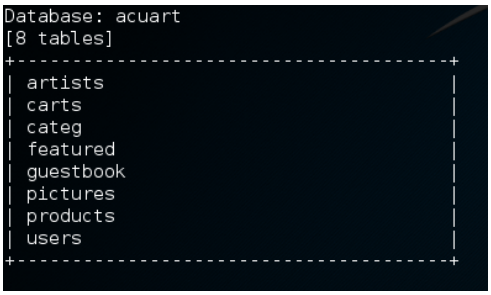

Figura 4.5: Nome das tabelas no BD. Fonte: Próprio autor

Tendo então o nome das tabelas, é possível adicionar mais parâmetros ao comando *sqlmap* para obter as colunas de uma determinada tabela, por exemplo. Simulando um sistema onde o banco de dados possui usuários cadastrados, é possível visualizar o nome *users*, uma tabela promissora para conter o nome de usuário e senha para um possível *login*.

Listing 4.3: Comando para retornar as colunas da tabela users

```
$ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1%20AND%201=2%20--%20
-D acuart -T users --columns
```

Verificar os campos (colunas) dessa tabela com *sqlmap* é fácil com o comando 4.3, sendo o resultado apresentado na figura 4.6.

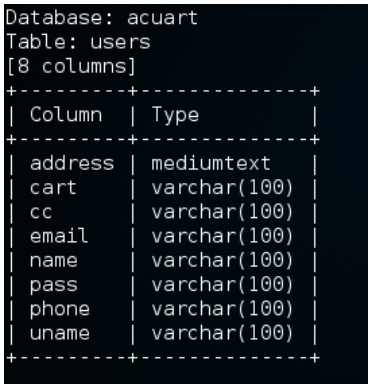

Figura 4.6: Nome das colunas de um BD. Fonte: Próprio autor

Neste retorno observa-se os campos *uname* e *pass*, os quais provavelmente

contém nome de usuário e senha para acesso ao perfil do usuário.

Para verificar estes valores tem-se os comandos 4.4 e 4.5. O retorno desses comandos estão expostos nas figuras 4.7 e 4.8 respectivamente.

Listing 4.4: Comando para retornar os valores da coluna *uname*

```
1 $ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1%20AND%201=2%20--%20
2 --threads 3 -D acuart -T users -C uname --dump
```

Listing 4.5: Comando para retornar os valores da coluna *pass*

```
1 $ sqlmap -u http://testphp.vulnweb.com/artists.php?artist=1%20AND%201=2%20--%20
2 --threads 3 -D acuart -T users -C pass --dump
```

Figura 4.7: *Dump* do campo *uname* da tabela *users* com *sqlmap*. Fonte: Próprio autor

Figura 4.8: *Dump* do campo *pass* da tabela *users* com *sqlmap*. Fonte: Próprio autor

A partir da integração entre análise de vulnerabilidade e ataques de teste foi possível obter os nomes de usuário e suas respectivas senhas de acesso ao endereço `www.testphp.vulnweb.com`.

4.2.3 Ataques de *Cross Site Scripting*

De acordo com a análise do *Vega* dentre as 1535 ameaças encontradas em `www.hackthissite.org` (figura 4.1) existem 23 possíveis endereços vulneráveis à *Cross Site Scripting*. Os próximos passos se dão com o uso do *XSSer*.

Através do endereço vulnerável disponibilizado pelo *Vega* em sua análise de vulnerabilidade, inicia-se o *XSSer* para iniciar os testes de inserção de *scripts*. Inúmeros casos de teste são executados, como é possível observar na figura 4.9.

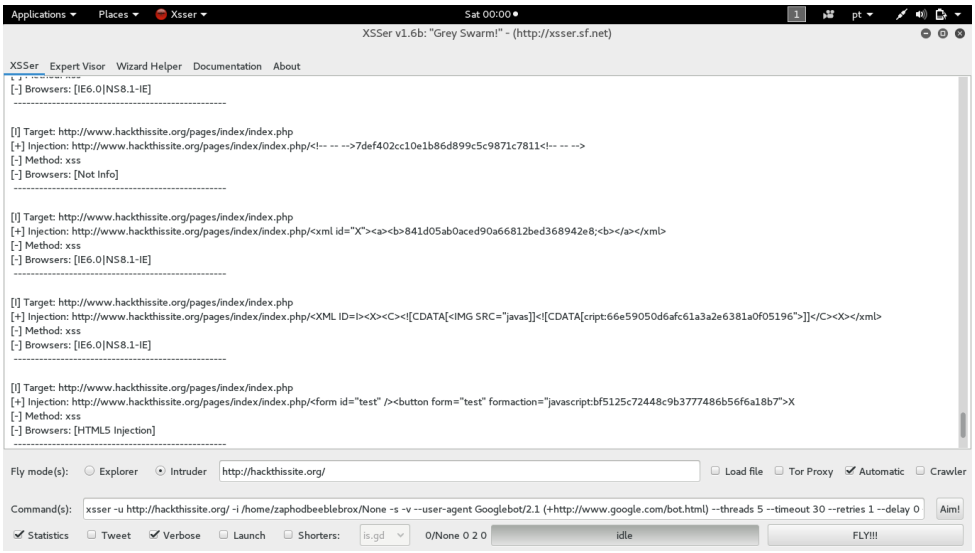

Figura 4.9: *Scripts* inseridos em testes com *XSSer*. Fonte: Próprio autor

Ao final da execução, o *XSSer* fornece dados estatísticos com relação aos testes realizados (figura 4.10).

Figura 4.10: Dados estatísticos dos testes com *XSSer*. Fonte: Próprio autor

Diferente da coleta de dados restritos observada em 4.2.2, os testes de inserção com *XSSer* modificam a página com *scripts* de outra origem que não o servidor de `www.hackthissite.org`.

4.3 Análise dos resultados

Os ataques realizados em 4.2 foram bem sucedidos, a manipulação de dados (vide 4.2.2) se mostrou 100% efetiva e a inserção de *scripts* de outra origem (XSS) (vide 4.2.3) obteve 82% de sucesso em seus casos de teste (figura 4.10).

A detecção das vulnerabilidades pelo *Vega* garantem a possível manipulação do banco de dados (vide 4.2.2) e inserção de *scripts* maliciosos de origem desconhecida (vide 4.2.3) por atacantes que podem se prover de acesso de escrita e leitura para dados que não deveriam ter permissão (*SQL Injection*) ou então manipular o conteúdo do site, suas funções e aparência, manipular a aplicação sem conhecimento do usuário, retransmitir *cookies* e outras funcionalidades.

4.4 Soluções propostas

Algumas boas práticas podem ajudar a solucionar problemas com vulnerabilidades. Para *SQL Injection* tem-se alguns tópicos chave (OWASP, 2016d):

- O desenvolvedor deve avaliar o pedido e resposta contra o código para verificar manualmente se há ou não uma vulnerabilidade presente.
- A melhor defesa contra vulnerabilidades de *SQL Injection* é utilizar instruções parametrizadas.
- Filtrar a entrada pode impedir essas vulnerabilidades. Variáveis de tipos *string* devem ser filtradas para caracteres de escape, e tipos numéricos devem ser verificados para garantir que eles são válidos.
- O uso de procedimentos armazenados pode simplificar consultas complexas e permitir configurações de controle de acesso mais restritivas.
- Configurar controles de acesso de banco de dados pode limitar o impacto das vulnerabilidades exploradas. Esta é uma estratégia atenuante que pode ser utilizada em ambientes em que o código não é modificável.
- Um mapeamento objeto-relacional elimina a necessidade de SQL.

O mesmo é encontrado para prevenção de problemas relacionados à vulnerabilidades em XSS (OWASP, 2016e):

- O desenvolvedor deve identificar como os dados não confiáveis estão sendo emitidos para o cliente sem filtragem adequada.
- Existem várias técnicas específicas de linguagem / plataforma para a filtragem de dados não confiáveis.

Algumas regras para prevenção de XSS:

Regra #0: Nunca insira dados não confiáveis, com exceção de locais permitidos. A primeira regra é negar tudo - não coloque dados não confiáveis em seu documento HTML. Isso inclui "contextos aninhados"como um URL dentro de um javascript - as regras de codificação para esses locais são complicadas e perigosas. Mais importante, nunca aceitar qualquer código *JavaScript* real a partir de uma fonte não confiável e, em seguida, executá-lo.

Regra #1: Quando você quiser colocar dados não confiáveis diretamente no corpo HTML em algum lugar (isso inclui dentro de tags normais como div, p, b, td, etc.), saiba que a maioria dos frameworks web tem um método de escape para HTML em certos contextos. No entanto, isto não é absolutamente suficiente para outros contextos HTML. É necessária a implementação das demais regras aqui também.

Regra #2: Nunca coloque dados não confiáveis em valores de atributos típicos como largura, nome, valor, etc. Isso não deve ser usado para os atributos complexos como *href, src, style,* ou qualquer um dos manipuladores de eventos como *onmouseover*. É extremamente importante que os atributos manipuladores de eventos sigam a Regra #3. Com exceção de caracteres alfanuméricos, escapar todos os caracteres com valores ASCII inferior a 256 para evitar a mudança fora do atributo.

Regra #3: Com relação a códigos *JavaScript* - ambos os blocos de *script* e atributos de manipulador de eventos. O único lugar seguro para colocar dados não confiáveis para este código está dentro de um "valor de dado"entre aspas. Incluir dados não confiáveis dentro de qualquer outro contexto é bastante perigoso, pois é extremamente fácil de mudar para um contexto de execução com caracteres incluindo (mas não limitado a) ponto e vírgula, iguais, espaço, mais, e muitos outros, assim use com cuidado.

Regra #4: Esta regra é para quando você quiser colocar dados não confiáveis em um arquivo de estilo ou uma *tag* de estilo. CSS é surpreendentemente poderoso, e pode ser usado por vários ataques. Portanto, é importante que você use somente dados não confiáveis em um valor de propriedade e não em outros lugares dentro de dados de estilo. Você deve ficar longe de colocar dados não confiáveis em propriedades complexas como *url*, comportamento e personalizado.

5 Conclusão

Ao longo deste trabalho foi possível compreender a importância da prática de programação segura em um ambiente LAMP. Sendo um ambiente completo, gratuito e bem disseminado, sua robustez se torna parte essencial do processo.

Embora seja essencial, não é algo garantido. Esta segurança deve ser provida no decorrer do desenvolvimento da aplicação, caso contrário testes realizados neste trabalho concluem que é possível, com o uso de poucas ferramentas, a extração de dados sigilosos e modificação da aplicação por usuários mal intencionados.

Detalhes sobre programação segura contra os ataques abordados neste trabalho foram explanados em 4.4, espera-se então a conscientização sobre a importância da programação segura de um ambiente com interação de usuários, testes foram feitos para alertar sobre a existência das vulnerabilidades e existem soluções para cada uma delas.

Referências Bibliográficas

ASF, A. S. F. *The Apache HTTP Server Project*. https://httpd.apache.org/. (Accessed on 04/08/2016).

BAUER, M. *Linux server security*. Sebastapol, CA Cambridge: O'Reilly, 2005. ISBN 978-0-596-00670-9.

BROWN, M. *Understanding LAMP*. 2005. http://www.serverwatch.com/tutorials/article.php/3567741/Understanding-LAMP.htm. (Accessed on 04/21/2016).

CVE. *CVE - Frequently Asked Questions*. 2016. http://cve.mitre.org/about/faqs.html. (Accessed on 04/08/2016).

FSF. *What is free software? - GNU Project - Free Software Foundation*. January 2016. http://www.gnu.org/philosophy/free-sw.en.html. (Accessed on 04/08/2016).

GEIPEL, M. M. *Dynamics of Communities and Code in Open Source Software*. 2010. http://searchenterpriselinux.techtarget.com/definition/LAMP. (Accessed on 04/21/2016).

GROUP, T. P. *PHP: Historia do PHP - Manual*. 2016. http://php.net/manual/pt_BR/history.php.php. (Accessed on 04/20/2016).

KALI. *Vega — Penetration Testing Tools*. 2016. http://tools.kali.org/web-applications/vega. (Accessed on 05/08/2016).

KALI. *What is Kali Linux ? — Kali Linux*. 2016. http://docs.kali.org/introduction/what-is-kali-linux. (Accessed on 05/08/2016).

LIP, T. L. I. P. *Kernel Definition*. May 2004. http://www.linfo.org/kernel.html. (Accessed on 04/21/2016).

MYSQL. *MySQL :: MySQL 5.7 Reference Manual :: 1.3.1 What is MySQL?* April 2013. https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html. (Accessed on 04/09/2016).

MYSQL. *MySQL :: MySQL 5.7 Reference Manual :: 6.1.1 Security Guidelines*. April 2013. http://dev.mysql.com/doc/refman/5.7/en/security-guidelines.html. (Accessed on 04/09/2016).

MYSQL. *MySQL :: MySQL 5.7 Reference Manual :: 1.3.3 History of MySQL*. 2016. http://dev.mysql.com/doc/refman/5.7/en/history.html. (Accessed on 04/20/2016).

OPENVAS. *OpenVAS - About OpenVAS Software*. 2016. http://www.openvas.org/software.html. (Accessed on 04/09/2016).

OWASP. *Top 10 2013-A9-Using Components with Known Vulnerabilities - OWASP*. 2013. https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities. (Accessed on 06/26/2016).

OWASP. *Top 10 2013-Top 10 - OWASP*. 2013. https://www.owasp.org/index.php/Top_10_2013-Top_10. (Accessed on 06/26/2016).

OWASP. *Cross-site Scripting (XSS) - OWASP*. 4 2016. https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29. (Accessed on 06/24/2016).

OWASP. *OWASP Vulnerable Web Applications Directory Project - OWASP*. 1 2016. https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project#tab=On-Line_apps.

OWASP. *SQL Injection - OWASP*. 2016. https://www.owasp.org/index.php/SQL_Injection. (Accessed on 05/08/2016).

OWASP. *SQL Injection Prevention Cheat Sheet - OWASP*. 5 2016. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. (Accessed on 06/25/2016).

OWASP. *XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP*. 3 2016. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. (Accessed on 06/25/2016).

PHP. *PHP: O que o PHP pode fazer? - Manual*. March 2016. http://php.net/manual/pt_BR/intro-whatcando.php. (Accessed on 04/09/2016).

POSTGRESQL. *PostgreSQL 8.1.23 Documentation*. 2010. http://www.postgresql.org/docs/8.1/interactive/history.html. (Accessed on 04/21/2016).

PROFFITT, B. *What Is Linux: An Overview of the Linux Operating System — Linux.com — The source for Linux information*. April 2009. https://www.linux.com/learn/what-linux-overview-linux-operating-system. (Accessed on 04/09/2016).

ROBINSON, D. *The Common Gateway Interface (CGI) Version 1.1*. RFC Editor, March 2013. RFC 3875. (Request for Comments, 3875). Disponível em: <https://rfc-editor.org/rfc/rfc3875.txt>.

ROUSE, M. *LAMP (Linux, Apache, MySQL, PHP)*. December 2008. http://searchenterpriselinux.techtarget.com/definition/LAMP. (Accessed on 04/21/2016).

SANS. *Vulnerability Assessments: The Pro-active Steps to Secure Your Organization*. 2001. https://www.sans.org/reading-room/whitepapers/threats/vulnerability-assessments-pro-active-steps-secure-organization-453. (Accessed on 04/22/2016).

SANS. *Penetration Testing - Is it right for you?* 2002. https://www.sans.org/reading-room/whitepapers/testing/penetration-testing-you-265. (Accessed on 04/09/2016).

SQLMAP. *sqlmap: automatic SQL injection and database takeover tool*. 2016. http://sqlmap.org/. (Accessed on 06/24/2016).

STANDARDS, N. I. of; TECHNOLOGY. *Technical Guide to Information Security Testing and Assessment*. september 2008. http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf. (Accessed on 04/08/2016).

SUBGRAPH. *About Vega*. 2014. https://subgraph.com/vega/documentation/about-vega/index.en.html. (Accessed on 04/09/2016).

SUBGRAPH. *Vega Vulnerability Scanner*. 2014. https://subgraph.com/vega/. (Accessed on 05/08/2016).

SULLO, C.; LODGE, D. *Nikto2 — CIRT.net*. https://www.cirt.net/Nikto2. (Accessed on 04/11/2016).