
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

Agradecimientos

“We must know - we will know!”

- David Hilbert

Resumo

[illegible]

Palavras-chaves:

Abstract

[illegible]

Keywords:

Lista de Figuras

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

Sumário

Lista de Figuras	5
Lista de Tabelas	6
Lista de Siglas e Abreviaturas	7
1 Introdução	9
2 Conceitos	10
2.1 LAMP	10
2.1.1 HISTÓRICO	10
2.1.2 APLICABILIDADE	11
2.2 FUNCIONAMENTO E COMPONENTES BÁSICOS	12
2.2.1 Linux	12
2.2.2 Apache	13
2.2.3 MySQL	13
2.2.4 PHP	15
2.3 FUNDAMENTAÇÃO DE ATAQUES E SOLUÇÕES	16
2.4 FRAMEWORKS E SOLUÇÕES PARA ANÁLISE DE VULNERABILIDADES	17
2.5 NORMAS, RECOMENDAÇÕES E BOAS PRÁTICAS PARA ANÁLISE DE VULNERABILIDADES	18
3 Conclusão	20
Referências Bibliográficas	21

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 Linux Apache MySQL PHP (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 (servidor 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 processo 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 licença 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 papéis 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 excepcional 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 licença 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, conseqüentemente, 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 aleatórias, 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 Lerdorf, o projeto inicial era um simples conjunto de CGIs¹ 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

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

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.

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, 2014):

- **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.

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).

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).

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).

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).

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