UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

ESCOLA DE INFORMÁTICA APLICADA

AVALIAÇÃO DE *SCANNERS* DE VULNERABILIDADES

Dez riscos mais críticos em aplicações Web

LUCAS DE ALMEIDA LINS

Orientadora

Prof.ª Dr.ª Leila Cristina Vasconcelos de Andrade

Coorientador

Prof. Dr. Davidson Rodrigo Boccardo

RIO DE JANEIRO, RJ – BRASIL

JULHO DE 2017

Catalogação informatizada pelo(a) autor(a)

Lins, Lucas de Almeida
L759 Avaliação de scanners de vulnerabilidades: dez riscos mais cri?ticos em aplicac?o?es Web / Lucas de Almeida Lins. -- Rio de Janeiro, 2017.
76p

Orientadora: Leila Cristina Vasconcelos de Andrade.
Coorientador: Davidson Rodrigo Boccardo.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2017.

1. Scanner. 2. Vulnerabilidades. 3. Avaliação. 4. Ferramentas. 5. Web. I. Andrade, Leila Cristina Vasconcelos de, orient. II. Boccardo, Davidson Rodrigo, coorient. III. Título.

AVALIAÇÃO DE *SCANNERS* DE VULNERABILIDADES

Dez riscos mais críticos em aplicações Web

LUCAS DE ALMEIDA LINS

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

_____

Prof.ª Dr.ª Leila Cristina V. Andrade (UNIRIO)

_____

Prof. Dr. Davidson Rodrigo Boccardo (Green Hat Labs)

_____

Prof.ª Dr.ª Lucila Maria S. Bento (UFRJ)

_____

Prof. Dr. Luiz Amancio M. Sousa Junior (UNIRIO)

RIO DE JANEIRO, RJ – BRASIL.

JULHO DE 2017

Agradecimentos

Gostaria, primeiramente, de agradecer aos meus pais, Leila e Marcello por nunca terem me deixado faltar apoio, incentivo, ensinamentos e amor. Vocês são a minha inspiração, e tudo para mim.

À minha irmã Leticia, por, mesmo tão pequena, ser grande o suficiente para me mostrar o quanto a vida é mais bela com um sorriso. Você é a pequena que eu amo.

À minha namorada, Thaíssa, por todo companheirismo, paciência, carinho e incentivo em tudo. Com você quero sempre caminhar.

Aos meus amigos, Octávio, Pedro, Raphael, Tiago e William. Vocês são os irmãos que a vida me deu.

À minha orientadora, professora doutora Leila Andrade, pela oportunidade e incentivo como monitor, pela orientação e pelo apoio em toda minha caminhada dentro da UNIRIO.

Ao meu coorientador, professor doutor Davidson Boccardo, sempre disponível para orientar, ensinar e ajudar. Suas contribuições foram muito enriquecedoras para o trabalho e para minha vida profissional.

À professora doutora Lucila Bento e ao professor doutor Luiz Amancio, pela disponibilidade em participarem da banca.

RESUMO

Nos dias atuais, são cada vez mais frequentes ataques virtuais. É importante que se conheça o contexto que possibilita esses ataques, os tipos de vulnerabilidades presentes na Web e as ferramentas de detecção de ameaças. A melhor maneira para minimizar a quantidade de vulnerabilidades em uma aplicação Web é a realização de varreduras por *scanners* de vulnerabilidades. Existem inúmeras ferramentas, livres ou comerciais, disponíveis para esse fim. Portanto, é fundamental que seja estudado se há diferença em seus resultados quanto a detecção de vulnerabilidades. Assim, este trabalho teve como objetivo avaliar a eficácia das ferramentas Arachini, Nessus e Vega na detecção dos dez riscos mais críticos de segurança levantados pelo OWASP. O Nessus destacou-se como um *scanner* eficaz para encontrar as vulnerabilidades que pertencem às categorias do OWASP Top Ten. São sugeridos estudos futuros comparativos entre soluções *open source* e pagas quanto a eficácia na detecção de vulnerabilidades, pesquisas qualitativas sobre *scanners* de vulnerabilidades e um maior investimento em segurança digital.

Palavras-chave: SCANNER, VULNERABILIDADES, AVALIAÇÃO, FERRAMENTAS, WEB.

ABSTRACT

Nowadays virtual attacks are becoming increasingly widespread. It is important that people are aware of different situations that make virtual attacks possible, as well as the types of web vulnerabilities and the tools to detect them. The best way to minimize vulnerabilities into a web application is to perform vulnerability scanning. There are numerous tools available for this purpose, open source or paid for scanners. So, it is important to assess the effectiveness of available tools to detect vulnerabilities. Thus the present study aimed to evaluate the effectiveness of Arachini, Nessus and Vega scanners in detecting the top ten critical vulnerabilities set by OWASP. Among them, Nessus was revealed as the most effective tool in detecting OWASP Top Ten risks (chi square = 8,76, df=2, p-value=0,01). For further studies, it is suggested to compare open source and paid for scanners in terms of the effectiveness in detecting vulnerabilities. It is also proposed the development of qualitative research on scanners. Lastly, investment in digital security is recommended.

Keywords: SCANNER, VULNERABILITIES, ASSESSMENT, TOOLS, WEB.

Índice

Índice de Tabelas

Índice de Figuras

1 **Introdução**

"Mais de 150 países foram afetados por um enorme ataque cibernético"

[The Washington Post maio de 2017].

"Ataques cibernéticos como o do *malware* WannaCry podem continuar se espalhando nos sistemas operacionais que não tenham sido atualizados com os *patches* de segurança" [The New York Times maio de 2017].

Na contemporaneidade, são cada vez mais frequentes ataques virtuais. Para se compreender o contexto que possibilita que estes ataques surjam e continuem causando danos dos mais diversos tipos aos computadores ao redor do mundo, é necessário se ter uma visão clara dos diferentes tipos de vulnerabilidades, do contexto em que se explicitam, a Web e, principalmente, dos seus métodos de detecção.

A Web é um canal recente de veiculação de dados, serviços e aplicações na contemporaneidade. Foi implementada há menos de três décadas por Tim Berners-Lee, físico graduado pela Universidade de Oxford.

Berners-Lee, no final da década de 80, deparou-se com um problema em sua rotina de trabalho. As informações que necessitava para a consecução de seus afazeres encontravam-se espalhadas em diversos bancos de dados, hospedados em máquinas diferentes. Este cenário o impossibilitava de acessá-los simultaneamente. Portanto, quanto mais dados eram produzidos, maior a quantidade de material disperso. Assim, propôs em 12 de março de 1989 o desenvolvimento de um sistema de informação distribuído para atuar no laboratório [Bryant 1999], conforme apresentado a seguir na figura 1.

Em 20 de dezembro de 1990, enquanto Berners-Lee trabalhava nesse projeto que objetivava habilitar um compartilhamento de informações entre equipes dispersadas geograficamente e difundir informações para grupos parceiros, propôs os conceitos básicos da Web, a saber URL, HTTP e HTML.

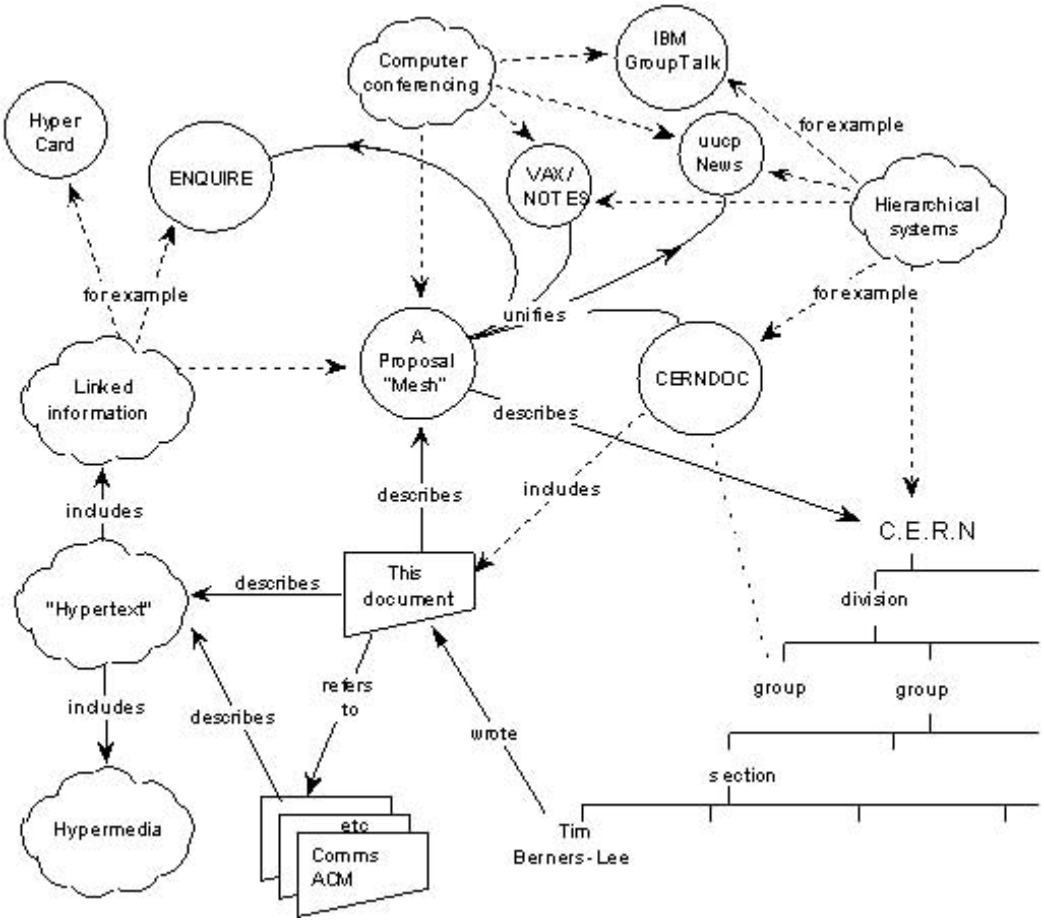


Figura 1. A arquitetura da Wolrd Wide Web sugerida por Berners-Lee

Fonte: https://timeline.Web.cern.ch/timelines/The-birth-of-the-World-Wide-Web

O projeto de Berners-Lee foi logo aprovado e rapidamente começou seu desenvolvimento. No início de 1991, a primeira aplicação Web já estava disponível para uso dos cientistas do CERN (figura 2, a seguir).

Com a implementação finalizada, as informações da aplicação desenvolvida estavam acessíveis a qualquer tipo de computador, em qualquer país e para qualquer pessoa que estivesse autorizada a acessá-la [CERN 2017]. Portanto, isto abriu também um amplo espaço de atuação para empresas.

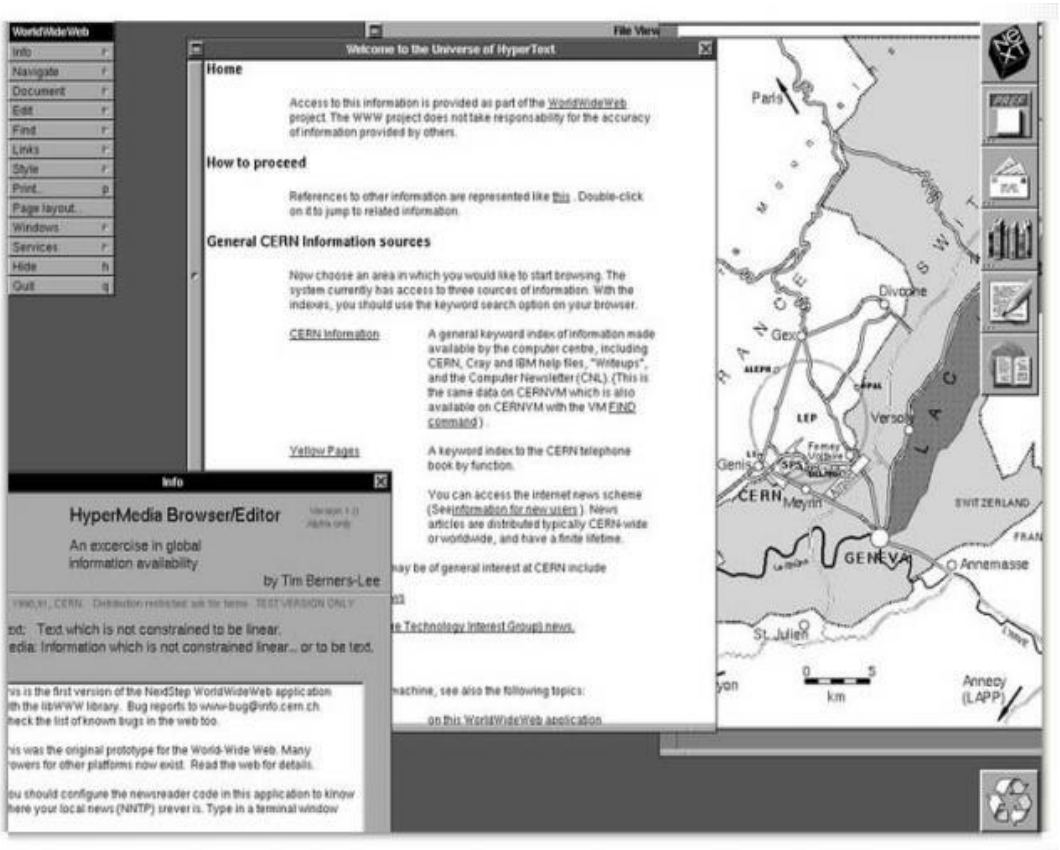


Figura 2. O primeiro servidor e site em ambiente de produção na CERN

Fonte: https://timeline.Web.cern.ch/timelines/The-birth-of-the-World-Wide-Web/overlay#1990-12-20 00:00:00

Atualmente, ainda é crescente o número de organizações que utilizam a Web e suas facilidades. Entretanto, neste cenário de exposição de marcas na Web, as empresas não podem correr o risco de por a reputação de seus nomes em jogo com um incidente público de segurança.

1.1 Motivação

Ataques como o supracitado WannaCry exemplificam a problemática tratada no artigo *Web Application Security Assessment Tools* [Curphey e Araujo 2006]. Uma das principais causas de ataques bem-sucedidos referenciadas por ambos é o alto gasto dos recursos de segurança na "porta de entrada" das aplicações Web, em sua grande maioria, os *sites*. Esta questão, porém, reflete um investimento reduzido e, muitas vezes, escasso em outras áreas críticas da aplicação tornando-as bastante expostas.

Partindo de um ponto de vista técnico, a repercussão alcançada pelo *malware* WannaCry deve-se, em uma análise imediata, a dois fatores. O primeiro é a "inteligência" por trás de seu código, onde seu comportamento auto replicável na rede, permitiu que se espalhasse sem necessidade de interação humana. O segundo fator é justamente o ponto ressaltado no artigo acima. As empresas afetadas se preocuparam em alocação dos recursos de segurança nas áreas que existem contato direto com o usuário, negligenciando a segurança em partes da aplicação Web que passam despercebidas à maioria dos utilizadores da aplicação, como é o caso do protocolo SMB v1 que é vulnerável.

No artigo *Web Application Scanners: Definitions and Functions* [Fong e Okun 2007] é apontado que a melhor maneira para minimizar a quantidade de vulnerabilidades em uma aplicação Web é a realização de varreduras por *scanners* de vulnerabilidades. Estas ferramentas, em um primeiro momento, realizam um *crawling* em toda a aplicação, para mapear o escopo do teste e, em seguida, procuram por vulnerabilidades simulando ataques a todo o escopo mapeado.

Porém, existem inúmeras ferramentas, tanto livres quanto comerciais, para a realização de *scan* de vulnerabilidades em aplicações Web, que garantem eficácia na detecção e prevenção de vulnerabilidades. Em função da grande quantidade de *scanners* disponíveis, torna-se fundamental que seja estudado se há diferenças em seus resultados quanto à detecção de vulnerabilidades, tomando-se como referência as dez vulnerabilidades mais recorrentes atualmente na Web.

1.2 Objetivos

Este trabalho teve como objetivo avaliar a eficácia de ferramentas públicas e de mercado na detecção de vulnerabilidades conhecidas na Web. Especificamente, avaliará a eficácia das ferramentas Arachini, Nessus e Vega na detecção dos dez riscos mais críticos de segurança levantados pelo OWASP.

1.3 Organização do texto

O presente trabalho está estruturado em cinco capítulos, como descrito a seguir:

- Capítulo I: Introdução – contextualiza a Web com suas vulnerabilidades e apresenta o objetivo do estudo.

- Capítulo II: Avaliação de vulnerabilidades na Web – apresenta o crescimento da Web e o surgimento de vulnerabilidades, a compreensão da tríade da Segurança da Informação e a emergência de bases de dados unificadas para a padronização das vulnerabilidades.
- Capítulo III: Projetos OWASP e bWAPP – retrata o projeto OWASP e alguns de seus projetos derivados que objetivam minimizar a frequência de vulnerabilidades em aplicações. A bWAPP como máquina planejadamente vulnerável para testes de segurança.
- Capítulo IV: *Scanners* de vulnerabilidade – conceitua um *scanner* de vulnerabilidade, descreve as três ferramentas utilizadas e apresenta os achados do estudo.
- Capítulo V: Conclusões – trata das considerações finais, sintetizando as contribuições da pesquisa e indicando caminhos para futuros estudos.

2 Avaliação de vulnerabilidades na Web

Tendo em vista uma melhor apresentação sobre avaliação de vulnerabilidades na Web, apresento em diferentes seções, a seguir, o funcionamento da Web, suas vulnerabilidades, a importância do processo de identificação e quantificação das vulnerabilidades, os componentes da segurança que guiam as políticas de segurança da informação e a caracterização de um banco público para consulta de vulnerabilidades.

2.1 Compreendendo a Web

A Web, a World Wide Web de Berners-Lee, é um repositório de informações conectados por pontos em todo o mundo. Ela tem uma combinação única de flexibilidade, portabilidade e características que a torna de fácil utilização, distinguindo-a de outros serviços fornecidos pela Internet [Forouzan 2007].

Para se compreender o funcionamento da Web é necessário entender que ela e a Internet são tecnologias diferentes. Porém, desempenham um trabalho conjunto que é essencial às suas funcionalidades nos dias atuais. Atualmente, a Web é o sistema de informação implementado mais avançado na Internet. É uma espécie de *software* com um conjunto de protocolos e convenções. Devido à utilização de hipertextos, a Web possui como uma de suas características principais a navegabilidade. Já a Internet pode ser destacada como "a mãe das redes" e seu principal objetivo é conectá-las utilizando linhas de telefone, cabos de fibra ótica, ou *links* em satélites.

Tendo em vista uma melhor apreensão desta questão, cabe uma comparação. Imagine-se um cenário onde existam ruas e prédios. Cada prédio possui informações que deseja receber ou enviar (Web) e cada rua é o caminho que possibilita o acesso a esta informação desejada (Internet). Se é conhecido o endereço do prédio que contém a informação e o apartamento no qual o dado desejado se encontra, é possível acessá-lo, caso se tenha autorização para entrar no prédio e apartamento em questão. É exatamente dessa maneira que funcionam a Web e a Internet.

É curioso um pequeno detalhe sobre o nome, World Wide Web, escolhido por Tim Berners-Lee. As duas primeiras partes do nome fazem referência ao fato do projeto ter sido elaborado para físicos de diversas partes do mundo que precisavam compartilhar informações entre si, e não ao fenômeno de vir a ser utilizada por todo o mundo.

Quatro componentes fundamentais compõem a tecnologia Web: navegador, servidor, protocolo e linguagem. O navegador é o *software* que gera uma série de requisições de dados da Web e os retorna na forma gráfica para o usuário. O servidor é o *hardware* que hospeda e executa aplicações e *softwares*. Ele é responsável por receber as requisições emitidas pelos navegadores dos usuários, interpretá-las e enviar o resultado gerado de volta para quem o requisitou. O protocolo – HTTP - é o método responsável pela comunicação entre o cliente, na forma de navegador, e o servidor. A linguagem – HTML - é um código de marcação baseado em hipertextos utilizado para desenvolver as páginas Web que ficam hospedadas nos servidores [Kurose e Ross 2013].

Com o intuito de tornar a Web *open source* o CERN, em abril de 1993, tornou públicos o protocolo e o código por ela utilizados [CERN 2017]. Desta maneira, começou a ser possível o desenvolvimento de páginas pessoais e negócios na Web, aumentando o número de usuários da tecnologia. Todavia, nem todos eram capazes de possuir suas próprias páginas, devido a dois grandes fatores limitadores. O computador ainda não era um equipamento maciçamente encontrado nas residências. As páginas só podiam ser feitas através da codificação manual de HTML.

O aumento significativo do uso da Internet se deu quando o computador se tornou um item domiciliar, tornando o acesso à Web possível de dentro das casas dos usuários. O fator de maior atração para os usuários foi a característica da Web operar *on demand*: eles recebiam o que queriam, quando queriam. Além disso, com o desenvolvimento da Web, o aumento de servidores também acompanhou esse crescimento, tornando possível que as pessoas começassem a se comunicar e compartilhar mais.

A segunda metade dos anos 90 caracterizou-se como um período de tremendo crescimento e inovação da Web. Grandes corporações criaram produtos e hospedaram serviços, tais como *email*, páginas Web, bancos de dados, entre outros. Este crescimento exponencial da Web a posicionou como fundamental no modelo de negócio de qualquer empresa porque, na Web, o consumidor é apresentado a negócios e serviços oferecidos.

São diversas as áreas de negócios difundidos pela Web: publicidade e *marketing*; comércio *online*; desenvolvimento e pesquisas; e comunicação com o consumidor, para mencionar alguns. A Web é tida como a ferramenta de negócio do futuro. Considera-se que os negócios que não se expandirem através desta tecnologia deverão sofrer uma desvantagem comercial.

Assim, a Web caracterizou-se como um novo mercado para ser explorado pelas empresas. Potenciais clientes são atraídos pela facilidade de utilização e pela abordagem multimídia nas quais as informações são apresentadas. Além disso, os *links* em hipertextos tornam fácil a navegação sem que seja necessária a aprendizagem de manuseio de comandos complexos. [Cockburn e Wilson 1996].

A facilidade de exposição de uma marca, os produtos oferecidos e as tecnologias utilizadas convencem consumidores de que o produto vale a pena. Isto se reflete no dado que 100% das maiores empresas do mundo em 2016 estão na Web [Yeomans 2016]. Elas possuem e dependem de vários serviços em seus servidores para o seu correto funcionamento: de troca de e-*mails*, páginas Web, de conexão remota, de transferência de arquivos, de resolução de nomes, de banco de dados, de páginas Web com criptografia, entre outros. Cada serviço possui, portanto, uma funcionalidade.

2.2 Vulnerabilidades na Web

Os serviços, por mais que possuam uma ampla e qualificada equipe envolvida em seu planejamento e construção, possuem lacunas que podem ser descobertas por usuários, após algum tempo de uso no mercado. Essas lacunas podem ser desde erros durante a fase de projeto até falta de padrões de codificação seguros. Usuários maliciosos se aproveitam desses erros e realizam diversas tentativas de exploração dessas brechas, até que a exploração seja bem-sucedida. A partir do momento em que um usuário malicioso encontra uma forma bem-sucedida de explorar, para diversos fins, uma dessas brechas, caracteriza-se a vulnerabilidade, a fraqueza, do sistema. De acordo com o Instituto SANS (*SysAdmin, Audit, Network and Security*), vulnerabilidades são portas através das quais as ameaças se manifestam [Cima 2001].

Apesar de todos os benefícios trazidos aos negócios, as aplicações Web são afetadas por uma série de problemas decorrentes de erros de projeto e de codificação. Entre os erros de codificação, pode-se destacar o acesso direto, para se obter os dados sensíveis residentes, tanto a banco de dados (informações sobre dados pessoais e financeiros), quanto a servidores, além de invasões às páginas Web de empresas.

A questão agravante aí colocada é a dependência existente entre as páginas Web e os bancos de dados, já que as primeiras necessitam dos bancos para entregarem as informações demandadas pelos usuários. Deste modo, se uma página Web está suscetível

a um dos diversos tipos de ataques existentes, os servidores de banco de dados que a ela está conectado também se encontrará comprometido [Acunetix 2017].

Tratando-se de grandes companhias com serviços de vital importância para seu funcionamento, atacantes encontram formas de explorar esses serviços buscando o comprometimento de algum dos pontos da tríade que compõe a segurança da informação: disponibilidade, integridade e confidencialidade.

Tem-se a seguir alguns exemplos de vulnerabilidades existentes nos principais serviços Web presentes na maioria das aplicações Web das empresas.

No banco de dados Oracle 11.1.0.7, é possível ver que existem: quatro vulnerabilidades que permitem o atacante fazer execução de códigos neste banco, uma vulnerabilidade de injeção de comando SQL, duas vulnerabilidades que dão ao atacante ganho de informação e uma vulnerabilidade que permite ao atacante realizar um *overflow*. Como apresentado na figura 3.

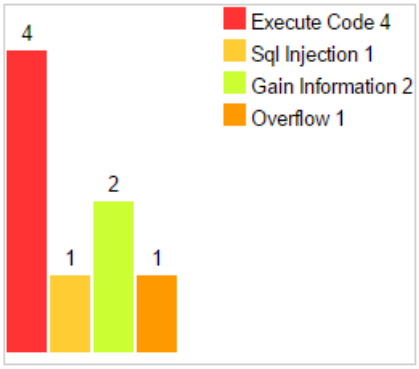


Figura 3. Vulnerabilidades do Oracle Database Server 11.1.0.7

Fonte: https://www.cvedetails.com/version/79103/Oracle-Database-Server-11.1.0.7.html

No servidor Web Microsoft IIS 6.0, é possível que um usuário malicioso o explore de diversas formas. Pode-se fazer uma execução de código de quatro formas distintas. Pode realizar um *overflow* de quatro maneiras diferentes. Pode disparar um ataque de negação de serviço – DoS – de duas maneiras desiguais. Pode corromper a memória duas de dois jeitos distintos. Pode atravessar algum tipo de bloqueio imposto pelo servidor de duas formas diferentes. E, por fim, pode permitir a execução de um ataque do tipo XSS de uma única forma. Como representado na figura a seguir.

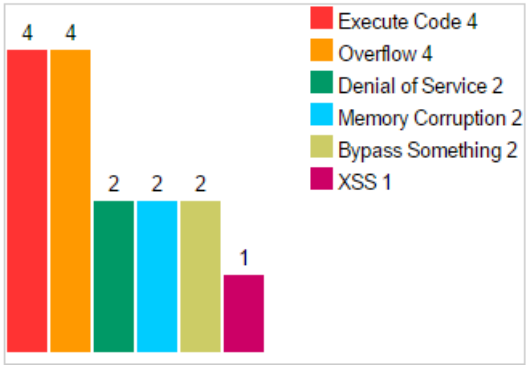


Figura 4. Vulnerabilidades do Microsoft IIS 6.0

http://www.cvedetails.com/version/13492/Microsoft-IIS-6.0.html

Já, no OpenSSH 7.3, que é utilizado para permitir que usuários acessem, de forma criptografada, o servidor na qual o serviço está executando, é possível: de duas maneiras, conseguir um ganho de privilégio; de uma maneira, criar uma negação de serviço do servidor; de uma maneira, criar um *overflow*; e, por fim, de uma maneira ganhar informações confidenciais.

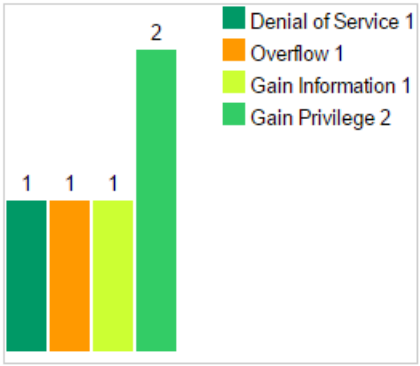


Figura 5. Vulnerabilidades no OpenBSD OpenSSH 7.3

http://www.cvedetails.com/version/205277/Openbsd-Openssh-7.3.html

Sabendo das potenciais vulnerabilidades e conhecendo técnicas cada vez mais elaboradas, os atacantes conseguem invadir sistemas e comprometer alguns serviços. O ataque ganha repercussão e prejudica a imagem do negócio da empresa. Dentre os ataques de maior impacto que já existiram na Web, dois chamaram muita atenção: o "Heartbleed", no período de 2012-2014, e o "PlayStation Network" em 2011 [Newton 2016].

O primeiro foi um *bug* que erroneamente foi escrito no OpenSSL. Ele permitiu que atacantes criassem um *gateway* em banco de dados. Foi considerado um dos maiores ataques da história, onde alguns laudos indicaram que afetou 17% de todas as páginas da Web. Ele permitiu que os atacantes ganhassem acesso, através do *gateway* estabelecido no sistema, a conversas privadas dos usuários, a qualquer momento, sem que eles soubessem.

O segundo ocorreu em abril de 2011 quando a Sony reconheceu que algumas funções da PSN estavam foram fora do ar. O ataque, que durou dois dias, viu o serviço *online* do PlayStation afetado por quase um mês. Um total de 77 milhões de contas foram comprometidas no período de 23 dias. Paralelamente a este dado, mais de 12 mil cartões de crédito válidos foram roubados, resultando na maior invasão já ocorrida na Web em toda sua história. A Sony foi obrigada a responder por processos devido às suas precárias medidas de segurança, recebendo uma multa de 250 mil Libras. Posteriormente, ela confirmou que a interrupção de 23 dias lhe custou 140 milhões de Libras.

2.3 Importância da avaliação de vulnerabilidades

Vulnerabilidades são portas pelas quais ameaças são manifestadas em um dado sistema, denominando-se avaliação de vulnerabilidade o processo de identificar e quantificar as vulnerabilidades encontradas. Estas avaliações podem ser usadas em diferentes tipos de sistemas, desde os bancários até os de proteção de usinas nucleares [Cima 2001].

Uma avaliação de vulnerabilidade tem ainda como objetivo apresentar formas de reduzir ameaças, visando a melhoria da capacidade do sistema para lidar com possíveis incidentes futuros.

Dois pontos devem ser considerados para que se compreenda a origem de tais fraquezas: brechas de segurança presentes em muitos sistemas que podem, ou não, ser fruto de configurações padrões de suas aplicações e más configurações dos sistemas realizadas por administradores de rede.

As soluções para os problemas de fraquezas são variadas, como criação e adoção de *baselines* de segurança, instalação de patches dos fabricantes, implementação de ativos para defesa de perímetro (*firewalls*, IPS e IDS) e avaliações de vulnerabilidades através

de *scans*. Cabe ainda acrescentar que, quando uma fraqueza é encontrada, há o risco de ter ocorrido comprometimento do sistema.

O conceito de avaliação de vulnerabilidade é um processo crucial que deve ser feito em todas as organizações, como forma de identificar, avaliar e prevenir as novas vulnerabilidades antes que elas se tornem ameaça para o nome da organização. Com o atual cenário da exposição das marcas na Web, as empresas não podem correr o risco de por a reputação de seu nome em jogo com um incidente público de segurança. Estes podem vir a causar severas perdas financeiras, caso sua marca seja ligada a um determinado incidente. O medo gerado por possíveis ataques as motiva a tomarem medidas proativas que minimizem o comprometimento de sua infraestrutura ou aplicações Web [Houghton 2003].

2.4 A tríade da segurança de informação

Disponibilidade, Integridade e Confidencialidade, do inglês *CIA triad*, é um modelo projetado para guiar as políticas de segurança da informação. Estes três componentes são considerados os componentes mais cruciais da segurança. No contexto, disponibilidade é a garantia de acesso à determinada informação por uma pessoa autorizada, integridade é a afirmação que uma informação é legítima e precisa, e confidencialidade é um conjunto de regras que limita o acesso para determinada informação.

A disponibilidade é melhor garantida quando existe uma boa manutenção de todos *hardwares*, reparando-os imediatamente quando necessário e livrando-os de conflitos com *software*, tornando possível seu correto funcionamento. É igualmente importante manter o sistema atualizado com suas últimas versões, e possuir uma largura de banda para comunicação adequada, que previna a ocorrência de gargalos na rede. Como forma de mitigar possíveis problemas de *hardware*, deve-se implementar redundâncias, *failovers* e ter um plano de implementação rápida, *Rapid Disaster Plan*, para situações que envolvam cenários extremos.

A proteção contra a indisponibilidade, inclui estar protegido contra perda de dados ou interrupções de conexão. Para o primeiro, deve-se ter um *backup* guardado em um local geograficamente isolado, o cenário ideal é um cofre a prova de chamas e água. Já, para a proteção contra interrupção de conexão, deve-se utilizar equipamentos ou

softwares como *firewalls* ou servidores de *proxy* que forneçam a proteção contra dados que porventura fiquem indisponíveis ou inalcançáveis, geralmente frutos de um ataque de negação de serviço (DoS).

A integridade envolve a manutenção da consistência, precisão e invariabilidade dos dados em todo o seu ciclo. Eles não podem ser alterados enquanto estiverem em trânsito e medidas devem ser tomadas para garantir que ele não será alterado por alguém não autorizado. As medidas de proteção envolvem permissão de arquivos, controle de acesso de usuários, verificação através de *checksums*, entre outras. Versões de controle também são uma boa maneira para prevenir mudanças equivocadas ou deleções acidentais.

A confidencialidade, por sua vez, abrange o princípio do "menor privilégio". Este princípio diz que o acesso à informação (ou ativos) é dado somente àqueles que realmente dela necessitam (e de forma controlada), e não para todos [Rouse 2008].

Medidas são planejadas para garantir a confidencialidade e prevenir que informações sensíveis cheguem até pessoas erradas, restringindo-se o acesso aos dados a usuários autorizados.

É comum que um dado seja categorizado de acordo com o tipo e tamanho do dano que pode causar caso caia nas mãos erradas. Devido a este risco, a proteção da confidencialidade dos dados deve envolver uma série de procedimentos para torná-los seguros, tais como boas práticas na cultura de senhas, conscientização dos portadores dos dados sobre ataques de engenharia social, criptografia de dados sensíveis, *tokens* de segurança (2FA), entre outros [Infosec Institute 2017].

2.5 Bancos públicos para consultas

Em um processo de avaliação de vulnerabilidade de uma aplicação ou de um sistema é necessário se ter uma convenção das métricas e pontuação dos riscos. Assim o auditor, ao utilizar diversas ferramentas de avaliação de vulnerabilidade durante a auditoria de segurança de aplicações, consegue realizar um relatório consistente e que retrate para o cliente o real cenário de vulnerabilidade que ele se encontra. Não é difícil encontrar uma situação em que duas ferramentas diferentes retratem uma mesma vulnerabilidade com nomes e pontuações de impacto diferentes. Como maneira de padronizar os relatórios, e até mesmo a pontuação de criticidade de cada vulnerabilidade,

o governo dos Estados Unidos criou serviços especializados no armazenamento e avaliação das vulnerabilidades de segurança virtuais.

Segundo o *Center for Homeland Defense and Security* (CHDS - Estados Unidos):

> "O *National Vulnerability Database* (NVD - Estados Unidos) e seu agregado, o repositório do *National Checklist Program* (NCP - Estados Unidos), oferecem um conjunto de serviços flexíveis e valiosos para usuários ao redor do mundo. O NVD foi estabelecido em 2005 para fornecer ao governo dos Estados Unidos um repositório de dados sobre todas as vulnerabilidades e configurações de *softwares*, à medida que alavanca padrões para fornecerem informações confiáveis e de fácil execução sobre severidade dos impactos, métodos de avaliação e referências para remediação". [Harold, Greg e Doug 2013].

O NVD é um produto da *Computer Security Division* que pertence ao *National Institute of Standards and Technology* (NIST) e que recebe patrocínio da *National Cyber Security Division* do *Department of Homeland Security* (DHS). Trata-se de um banco de dados, disponível publicamente [CVE 2017], do governo dos Estados Unidos contendo as vulnerabilidades que envolvem segurança cibernética e fornecendo referências de tais vulnerabilidades para a indústria. As informações contidas no NVD são acessadas através da Web. Seu mecanismo de estatísticas possui uma funcionalidade de reporte, possibilitando ver as tendências das vulnerabilidades através do tempo. Isto permite ao usuário avaliar as mudanças ocorridas em relação às taxas de detecção de vulnerabilidades em um produto específico ou em um tipo específico de produto.

O NVD é baseado no *Common Vulnerabilities and Exposures* (CVE), que é um dicionário de nomes comuns dados para as vulnerabilidades de segurança cibernética conhecidas publicamente, e as pontua de acordo com *Common Vulnerability Scoring System* (CVSS). Vale salientar que o NVD é o dicionário CVE com recursos adicionais, como análise, banco de dados e mecanismo de busca refinado. Desta maneira, qualquer atualização no CVE, imediatamente acontece no NVD.

Seguem algumas considerações sobre a CVE e o CVSS.

Por conta das diferentes nomenclaturas dadas à mesma vulnerabilidade, tanto por profissionais de segurança quanto por fabricantes, o trabalho dos auditores de segurança tornava-se confuso. Dessa maneira, as organizações se mobilizaram e desenvolveram

uma linguagem única, a CVE - *Common Vulnerabilities and Exposures List*, patrocinada pela Mitre Corporation.

Já o CVSS é um sistema padrão de medidas adequado para a indústria, organizações e governos que precisam das pontuações dos impactos de uma forma precisa e consistente. O Common Vulnerability Scoring System fornece um framework que permite a comunicação das características e impactos das vulnerabilidades de tecnologia da informação. O seu modelo permite uma medição precisa, enquanto permite aos usuários ver as características de determinada vulnerabilidade que foram utilizadas para medir a pontuação.

Os dois usos mais comuns do CVSS é a priorização das atividades para remediação das vulnerabilidades e o cálculo da severidade das vulnerabilidades encontradas em determinado sistema. Quanto a este último, o NVD fornece pontuação CVSS para praticamente todas as vulnerabilidades conhecidas.

Acrescenta-se a isso, que o NVD suporta a versão dois do CVSS para todas as vulnerabilidades CVE. Ele fornece pontuações bases do CVSS que representam as características inatas de cada vulnerabilidade. Não é fornecida a pontuação temporal da vulnerabilidade, que é a pontuação que muda de acordo com o tempo devido a eventos externos à vulnerabilidade. Porém, para suprir este ponto, o NVD fornece uma calculadora dos pontos CVSS que permite ao usuário adicionar dados temporais para calcular o impacto da vulnerabilidade na própria organização.

Em junho de 2015, após três anos de desenvolvimento, o novo sistema do *Common Vulnerability Scoring System* ficou disponível na sua nova versão, a terceira. O CVSSv3 introduz mudanças no sistema de pontuação, que reflete com mais precisão as vulnerabilidades das aplicações Web. Enquanto os três grupos de métricas (*Base*, *Temporal* e *Environmental Score*) permaneceram os mesmos, novos grupos foram adicionados, como *Scope* e *User Interaction*, além de antigas métricas como *Authentication* serem modificadas para novas, como *Privileges Required*. Em relação à métrica *Environmental*, esta teve um novo acréscimo com a *Modified Base*, permitindo aos analistas customizarem pontuações CVSS de acordo com o ativo que foi afetado dentro da organização [Brigjaj 2016].

Figura 6. Detalhes técnicos de uma vulnerabilidade - EternalBlue

Fonte: https://nvd.nist.gov/vuln/detail/CVE-2016-7053

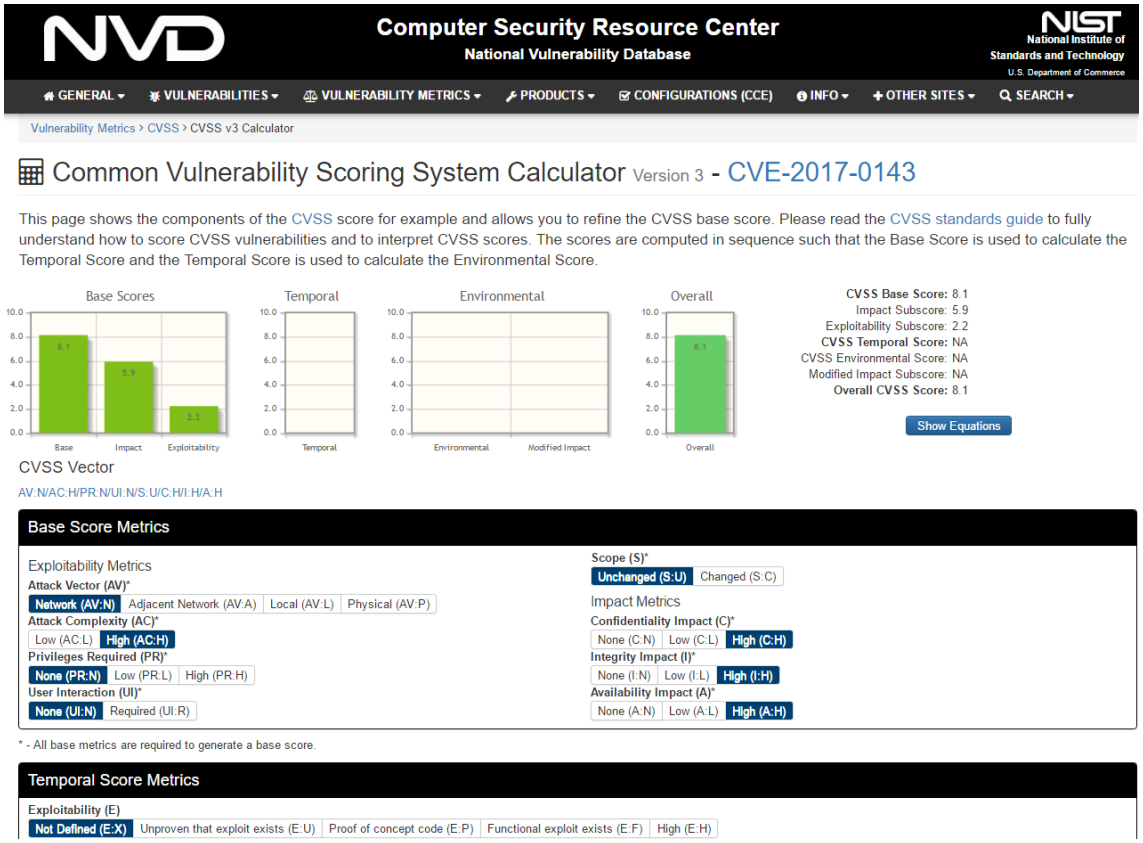


Figura 7. Calculadora CVSS para uma vulnerabilidade – EternalBlue

Fonte:https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator?name=CVE-2016-7053&vector=AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Em função dos fatos aqui expostos, especialmente a falta de privacidade e segurança na internet, há céticos sobre o futuro do "*cyber-business*". Entretanto, a maioria dos problemas são alvo de pesquisas e soluções vêm sendo desenvolvidas. Para evitar o comprometimento de sistemas, diversas ferramentas têm sido construídas. Atualmente, já existe uma série de sistemas mais seguros disponíveis na Web. Aliado ao avanço na parte da tecnologia, vários projetos e comunidades sem fim lucrativos vêm se formando e se especializando no combate do ciber-crime.

3 Projetos OWASP e bWAPP

A falta de segurança na Internet conduziu à troca e disponibilização de informações e soluções sobre práticas e recursos para a proteção contra vulnerabilidades. Neste sentido, destacam-se o projeto OWASP e bWAPP.

3.1 O Open Web Application Security Project: OWASP

Qualquer aplicação Web necessita de uma fonte de informação imparcial que disponibilize as melhores práticas de segurança para tornar seu ambiente seguro. No cenário da segurança de aplicações, o Open Web Application Security Project (OWASP) funciona sem fins lucrativos e sem afiliação a nenhuma empresa de tecnologia. Isto se traduz em possuir uma posição única para fornecer informação imparcial sobre segurança de aplicações para pessoas, corporações, universidades, agências governamentais e outras organizações mundo a fora. Funciona como uma comunidade de profissionais com elevado conhecimento na área de segurança e que possuem o mesmo propósito - diminuir a insegurança na Web. O OWASP produz ferramentas e documentos voltados à segurança das aplicações. Todos seus artigos, metodologias, tecnologias e projetos são disponibilizados à grande rede de forma gratuita.

O OWASP teve seu início em 2001, porém somente em 2004 começou efetivamente a operar, devido ao apoio recebido da sua fundação, o OWASP Foundation, que ainda contribui mantendo a comunidade e seus projetos [OWASP b 2017].

A liderança do OWASP é completamente voluntária e toma as decisões acerca de diretivas técnicas, prioridades dos projetos, planejamento de tarefas e calendários, e lançamentos. O OWASP possui apenas três funcionários, tornando viável seu funcionamento com baixíssimo custo. Para gerar subsídios e possibilitar pesquisas e projetos de ponta para a área de segurança de aplicações, o OWASP cobra taxas de associação de empresas e de indivíduos e taxas de conferências. Sua principal missão é fazer com que a questão da segurança seja visível por todos, para que tanto as organizações, quanto as pessoas que buscam a expansão de seus negócios no mundo virtual, sejam capazes de tomar decisões sensatas de acordo com as informações fornecidas por seus projetos.

3.2 O OWASP como um ponto de referência à segurança

Os materiais criados pelo OWASP, fontes de projetos, estudos e vivência no mundo da segurança são de tamanha confiança, abrangência e credibilidade que diversas organizações governamentais, empresas multinacionais de impacto na economia e institutos incorporam seus projetos em seu dia a dia. Tomando-se os países integrantes do G8, apenas a Itália e a Rússia não possuem alguma organização governamental que documenta a necessidade de tomar o OWASP como referência para problemas relacionados à segurança, tal como ilustrado a seguir [OWASP 2016].

Na França, a *L'Agence Nationale de Sécurité des Systèmes d'Information* possui um guia de padrões e documentos técnicos referentes à auditoria de segurança dos sistemas de informação - *Prestataires D'Audit de la Securite des Systemes D'Information* – que estabelece a adoção da documentação e orientações do OWASP.

A mesma orientação acontece na Alemanha, no Escritório Federal de Segurança da Informação - *Bundesamt für Sicherheit in der Informationstechnik* (BSI). Há um conjunto de boas práticas para segurança de aplicações Web, a *Sicherheit von Webanwendungen: Maßnahmenkatalog und Best Practices*, que define o OWASP como uma fonte confiável de recursos que permitem seguir as ações estabelecidas.

No Canadá, o Centro de Respostas a Incidentes Virtuais do Canadá - *Canadian Cyber Incident Response Centre* - possui um guia de mitigação para ameaças de injeção de SQL, o TR08-001, que cita o projeto OWASP Testing Guide entre as boas práticas para a segurança de aplicações Web.

No Japão, a Agencia de Promoção da Tecnologia e Informação - *Information-Technology Promotion Agency* (IPA) - possui o estudo do controle de acesso obrigatório em servidores Web, o *Study of Web Server Mandatory Access Control*, que se baseia nos guias do OWASP como contramedida para possíveis fraquezas que as aplicações Web possam apresentar.

No Reino Unido, o Centro de Proteção Nacional da Infraestrutura - *Centre for the Protection of National Infrastructure* (CPNI) - faz menção à utilização do projeto do OWASP, o Top 10, para avaliação dos riscos na segurança das aplicações Web.

A mesma abordagem é encontrada na Agência de Segurança Nacional dos Estados Unidos - *National Security Agency* (NSA) - em seu projeto *Web Application Security*

Overview and Web Application Security Vulnerabilities, que cita a utilização do projeto OWASP Top 10 para a busca de falhas e para remediar aplicações Web.

Além de organizações governamentais, a utilização de projetos do OWASP também é encontrada em Conselhos mundialmente famosos, como é o caso do PCI (*Payment Card Industry*). Trata-se de uma organização global que mantém, desenvolve e promove padrões na indústria de pagamentos com cartões e que protege os dados do dono do cartão em todo o mundo [PCI 2017]. O PCI *Council*, utiliza o projeto OWASP Top 10 como parte do PCI DSS - *Payment Card Industry Data Security Standard*.

Já na esfera das empresas de grande impacto no mercado, também existe a adoção de projetos do OWASP como forma de mitigar ou prevenir possíveis riscos ou vulnerabilidades na Web. A Microsoft, por exemplo, faz a utilização do projeto OWASP Top 10 como forma de encontrar possíveis problemas de segurança no ciclo de vida de desenvolvimento de seus softwares [Sullivan 2008].

3.3 Projetos em destaque

A amplitude e atualidade dos conteúdos presentes em cada um dos seus 357 projetos [OWASP 2017b] fez o OWASP a referência mundial no quesito projetos abertos voltados para a segurança. Seus projetos englobam áreas fundamentais para as aplicações, como desenvolvimento de maneira segura (para evitar riscos já conhecidos), a revisão de códigos criados por desenvolvedores, teste de aplicações Web para mitigar possíveis ataques, uma lista atualizada das dez maiores vulnerabilidades encontradas na Web, entre outros.

O primeiro projeto do OWASP, e um dos mais acessados, é o *Developer Guide*, que foi publicado pela primeira vez em 2002. Porém, em suas versões iniciais, o guia não se engajou com seu público-alvo desejado: os desenvolvedores. Os guias iniciais se referiam mais a como realizar um teste de invasão em aplicações Web do que propriamente um conjunto de boas práticas para o desenvolvimento de uma aplicação.

Em sua versão atual, *Developer Guide* 2014, assume um papel de livro dos princípios básicos, não sendo específico para nenhuma linguagem ou *framework*. Possui questões altamente específicas em várias linguagens, como configurações em PHP ou problemas com o *Spring* MVC. Os temas principais abordados no livro passam desde arquitetura e *design* de sistemas até sua configuração e operação.

O material do guia original está sendo reconstruído, sendo adaptado ao cenário atual e se direcionando às aplicações Web contemporâneas, como as que usam AJAX e aplicações móveis. Os materiais posteriores ao guia original, como a revisão completa de um código previamente desenvolvido e um guia para testes de vulnerabilidade das aplicações Web, foram movidos para projetos individuais, devido à sua importância e criticidade na segurança das aplicações [Github, 2017].

Devido à necessidade de um código sempre se manter conciso e livre de falhas, o projeto *Code Review Guide* ocupa uma das primeiras posições nos *downloads* do OWASP. É um livro técnico escrito para todos que são responsáveis pela revisão de códigos, como gestores, desenvolvedores, profissionais de segurança, entre outros. O foco principal do livro é dividido em duas seções principais. A primeira é o porquê e como se faz uma revisão de código. Já a segunda é voltada para quais vulnerabilidades devem ser procuradas em uma revisão manual de código. Mesmo com o melhoramento constante dos *scanners* de vulnerabilidade, a revisão manual de segurança no código ainda precisa ter um lugar proeminente no ciclo de vida do desenvolvimento seguro, requerendo um código seguro em produção [OWASP 2017a].

Com o código desenvolvido de maneira segura e revisado de forma correta para concluir a eficiência de uma aplicação, no que tange à segurança, ainda torna-se necessária a realização de exaustivos testes atrás de possíveis falhas. Com o intuito de preencher esta lacuna pode-se utilizar o OWASP *Testing Guide*.

O OWASP *Testing Guide* é o documento projetado para ajudar organizações a entenderem o que compreende um programa de teste e a identificarem os passos que precisam ser dados para construir e operar o programa em suas aplicações Web. O guia oferece uma ampla visão dos elementos necessários para fazer um programa abrangente de segurança em aplicações Web. Devido a seu conteúdo, pode ser utilizado para entender a magnitude dos recursos necessários para testar e manter o *software* ou para preparar as aplicações para uma auditoria de segurança. Além de fornecer uma estrutura organizacional de segurança, o guia entra em detalhes técnicos de como testar uma aplicação, cobrindo partes de um teste de invasão. Desta forma, ele pode ser usado tanto como referência, quanto como metodologia para ajudar a determinar o espaço existente entre as práticas existentes de segurança e as boas práticas da indústria

[OWASP 2017c].

3.4 O OWASP Top 10

Projetos como os recém citados são amplos e contemplam áreas críticas de qualquer aplicação, inclusive o código fonte, que é a origem de todo seu funcionamento. É provável que, se o desenvolvimento de uma aplicação foi feito seguindo o *Developer Guide*, sua revisão de código foi guiada pelo *Code Review Guide*. Com a aplicação em ambiente de produção, foram testadas diversas vulnerabilidades presentes no *Testing Guide*, logo ela provavelmente estará livre de vulnerabilidades, ou bem perto disso. Porém, para implementar esses três testes principais, além de diversos outros mais específicos que são disponibilizados pelo OWASP, há um gasto alto com recursos financeiros e tempo.

Por conta deste cenário, muitos *softwares* são desenvolvidos e não é dada a devida atenção à sua segurança, gerando aplicações inseguras que põem em risco infraestruturas críticas como a financeira, a de defesa, entre outras. No panorama atual, com aplicações interligadas e cada vez mais complexas, a dificuldade de se garantir a segurança aumenta de forma exponencial. Como resposta a isto, o OWASP desenvolveu o OWASP Top 10, um guia que identifica as vulnerabilidades mais críticas presentes nas organizações, protege-lhes contra as dez ameaças mais críticas e recorrentes no mundo digital e sensibiliza as organizações quanto a segurança.

Para cada risco identificado, são especificados a probabilidade de ocorrência e o impacto técnico que pode ser gerado. O impacto do risco pode variar de organização para organização. Em algumas, certo tipo de ataque pode ser prejudicial e, em outras, o impacto deste ataque pode ser irrelevante. O nome de cada um dos riscos apresentados no guia vem do tipo de ataque, do tipo de vulnerabilidade ou do tipo de impacto causado.

A longo prazo, a conscientização proposta pelo OWASP resulta na criação de um programa de segurança de aplicações compatível com a cultura e tecnologia utilizadas pela organização.

A lista apresentada no OWASP Top 10 não é o milagre para os problemas de segurança relacionados às aplicações Web, mas serve como um guia que as organizações devem seguir para garantir que suas aplicações Web não estejam expostas às vulnerabilidades que mais são exploradas e aos problemas de segurança mais recorrentes na atualidade. Assim, obtendo a certeza de que todos os tópicos do OWASP Top 10 foram

seguidos em seus mínimos detalhes, a organização poderá manter seu negócio em um patamar de segurança acima da grande maioria que existe no mercado [OWASP 2017 e].

Como resposta a isto, o OWASP desenvolveu o OWASP Top 10, um guia que identifica as vulnerabilidades mais críticas presentes nas organizações, protege-lhes contra as dez ameaças mais críticas e recorrentes no mundo digital e sensibiliza as organizações quanto a segurança.

Para cada risco identificado, são especificados a probabilidade de ocorrência e o impacto técnico que pode ser gerado. O impacto do risco pode variar de organização para organização. Em algumas, certo tipo de ataque pode ser prejudicial e, em outras, o impacto deste ataque pode ser irrelevante. O nome de cada um dos riscos apresentados no guia vem do tipo de ataque, do tipo de vulnerabilidade ou do tipo de impacto causado.

A longo prazo, a conscientização proposta pelo OWASP resulta na criação de um programa de segurança de aplicações compatível com a cultura e tecnologia utilizadas pela organização.

A lista apresentada no OWASP Top 10 não é o milagre para os problemas de segurança relacionados às aplicações Web, mas serve como um guia que as organizações devem seguir para garantir que suas aplicações Web não estejam expostas às vulnerabilidades que mais são exploradas e aos problemas de segurança mais recorrentes na atualidade. Assim, obtendo a certeza de que todos os tópicos do OWASP Top 10 foram seguidos em seus mínimos detalhes, a organização poderá manter seu negócio em um patamar de segurança acima da grande maioria que existe no mercado [OWASP 2017 e].

3.5 Dez riscos mais críticos em aplicações Web – A lista do OWASP Top Ten

3.5.1 A1: Injeção de código

A primeira vulnerabilidade descrita no guia é a referente à injeção de código. Referenciada como A1, as falhas de injeção, seja ela de SQL, de SO ou de LDAP, acontecem quando dados não confiáveis são enviados para um interpretador, que não possui a devida sanitização destes dados, como parte de uma consulta ou de um comando. Estes dados são manipulados pelo atacante de forma que o interpretador não os distinga como maliciosos e execute comandos indesejados ou forneça acesso a dados não autorizados.

A vulnerabilidade se evidencia quando um usuário interno ou externo, ou ainda um administrador, possui a opção de enviar dados não confiáveis para o sistema. A partir deste ponto, o atacante envia ataques simples que se baseiam em *strings* que exploram a sintaxe do interpretador alvo. O sucesso do ataque se dá quando a aplicação envia os dados não confiáveis para o interpretador e este não faz o devido filtro da *string* que lhe foi passada. Esta falha de injeção pode ser encontrada em consultas SQL, LDAP, Xpath ou NoSQL, comandos de Sistemas Operacionais, cabeçalhos SMTP, entre outros. Detectar que um sistema está vulnerável a esta falha é fácil, se existir uma análise do código utilizado, porém existe uma dificuldade maior de detectá-la através de testes. O ataque bem sucedido e a exploração bem feita de uma injeção podem resultar em vazamento ou corrupção de dados. A injeção pode chegar a tal ponto de levar ao comprometimento completo do servidor.

Para detectar a exposição de uma aplicação à injeção, é necessário verificar se todos os usos dos interpretadores separam os dados não confiáveis do comando ou da consulta. No caso específico da injeção de comandos SQL, isto significa utilizar variáveis de ligação em todas as instruções preparadas e procedimentos armazenados, e evitar consultas dinâmicas. Uma outra maneira de detecção, rápida e precisa, é realizar a verificação do código fonte. A utilização de ferramentas de análise de código juntamente com testes de invasão ajuda o analista de segurança a traçar fluxo de dados através da aplicação e a validar a vulnerabilidade do interpretador através da utilização de *exploits*. A detecção de forma automatizada pode ser concluída realizando varreduras dinâmicas que exercitem a aplicação através de ferramentas de *scan* de vulnerabilidade.

A prevenção dessa vulnerabilidade pode ser alcançada em três vias distintas. A primeira, e mais eficiente, é a utilização de uma API segura que evite o uso total do interpretador ou que forneça uma interface parametrizada. O segundo modo é a situação na qual a API parametrizada não está disponível, onde deve-se filtrar os caracteres especiais utilizando a sintaxe específica do interpretador. A última abordagem, não completamente segura, caracteriza-se pela criação de uma *white list*. Porém, não resulta em uma proteção completa, pois muitas aplicações necessitam de caracteres especiais em suas entradas [OWASP 2017d].

3.5.2 A2: Quebra de autenticação e gerenciamento de sessão

O segundo item do guia, A2, é a quebra de autenticação e gerenciamento de sessão. Esta vulnerabilidade é caracterizada pela implementação de maneira incorreta das funções das aplicações relacionadas à autenticação e gerenciamento de sessão. Permite que os atacantes estejam aptos a comprometer senhas, identificadores de sessão, ou ainda consigam explorar outra falha na implementação da aplicação tornando possível que se assuma a identidade de outro usuário.

Esse tipo de vulnerabilidade pode ser gerado por atacantes externos, não autorizados na aplicação, ou usuários internos à aplicação, autenticados, que têm como objetivo roubar contas de outros usuários da mesma aplicação ou disfarçar suas ações, dando a entender que foi outro usuário que a fez. A exploração desta vulnerabilidade é feita quando o atacante utiliza informações vazadas ou falhas nas funções de autenticação e gerenciamento de sessão (como credenciais expostas, identificadores de sessão) para tomar para si a identidade de outro usuário. O ponto chave deste ataque está na implementação das funções de autenticação e gerenciamento de sessão pelos desenvolvedores, que em sua grande maioria são realizadas de forma personalizada. Sua correta implementação é difícil. Diversos são os casos em que os esquemas personalizados possuem falhas na função de *logout*, gestão de senhas, tempo de expiração do *token* da funcionalidade "lembrar senha", pergunta secreta, entre outras. Devido à tendência de unicidade de cada implementação, a detecção desta vulnerabilidade não é trivial. Mas, por contrapartida, uma vez detectada e explorada, essas falhas trazem um impacto severo, permitindo que uma conta seja comprometida e atacada. Dentro da conta da vítima, o atacante tem o poder de fazer qualquer ação que a vítima faria, tendo como maiores alvos, contas de alto privilégio.

A detecção dessa vulnerabilidade na aplicação pode ser constatada ao se procurar por falhas nos ativos de gerenciamento de sessão, tais como as destacadas a seguir. Há casos de credenciais de autenticação de usuário não protegidas através de *hash* ou criptografia enquanto armazenadas. Encontram-se também funções frágeis de gerenciamento de conta (como alteração e recuperação de senhas e identificadores de sessão fracos), ou problemas relacionados a identificadores de sessão (expostos em URL, ou que não expiram, ou ainda que não são rotacionados após um *login* bem sucedido). Por fim, pode-se citar também problemas com tráfego de senhas, identificadores de sessão e outras credenciais enviadas através de conexões não criptografadas.

Como forma preventiva dos problemas citados acima, deve-se disponibilizar aos desenvolvedores um conjunto único de controles fortes de autenticação e gerenciamento de sessão, tal como o cumprimento de requisitos de autenticação e gerenciamento de sessão pré-definidos, e a concentração de esforços para evitar falhas de XSS, que podem ser utilizadas para roubar os identificadores de sessão [OWASP 2017d].

3.5.3 A3: *Cross-Site Scripting*

A terceira vulnerabilidade mais comum nas aplicações Web é o *Cross-Site Scripting* (XSS). Este tipo de falha ocorre sempre que a aplicação receber dados não confiáveis e não realizar nenhum tipo de filtro ou validação destes, enviando-os diretamente para o navegador. O XSS dá ao atacante a capacidade de executar *scripts* no navegador da vítima que podem fazer sequestro de sessão de usuário, desfigurar *sites* ou redirecionar a vítima para *sites* não confiáveis.

O ataque é originado por qualquer um que tenha o poder de enviar dados não-confiáveis para o sistema. O funcionamento do ataque consiste no envio de *scripts* maliciosos que exploram o interpretador do navegador por parte do atacante.

O *Cross-Site Scripting* é a falha de segurança mais predominante nas aplicações Web. Ocorre quando a aplicação, sem nenhum tipo de filtro, trata os dados enviados ao navegador pelo usuário malicioso. A exploração deste ataque pode ser em três tipos: de maneira persistente, refletida ou baseada em DOM. O primeiro tipo de exploração ocorre quando o código malicioso fica armazenado na aplicação e é executado pelo navegador de qualquer vítima que acesse a página com tal código. É frequentemente encontrado em ambientes de fóruns, por exemplo. O segundo é feito quando o atacante envia para a vítima um *link* que, ao ser acessado, automaticamente executa o código no navegador do alvo. Já o terceiro utiliza o *Document Object Model* (DOM) que é o padrão usado para interpretar o código HTML em objetos a serem executados pelos navegadores Web. Ele permite a modificação de propriedades destes objetos diretamente no navegador da vítima, não dependendo de nenhuma interação por parte do servidor que hospeda o serviço.

O impacto de um ataque de XSS bem sucedido pode gerar sequestro de sessão da vítima, inserção de conteúdos hostis em uma página Web, redirecionamento de usuários para destinos inseguros, entre outros.

Para assegurar que a aplicação não está vulnerável a esta falha, é necessário garantir que todas as entradas que interajam com o usuário tenham seus conteúdos apropriadamente sanitizados. Caso o filtro de controle de entrada não esteja adequado, a *string* enviada pelo usuário será tratada como conteúdo ativo no navegador.

Uma aliada na detecção de XSS são as ferramentas de varredura de vulnerabilidades. Elas são capazes de encontrar automaticamente falhas que levem ao XSS. Porém, a grande quantidade de interpretadores existentes e a possibilidade de utilizar cada um deles criam dificuldades para a detecção automática através dessas ferramentas. Consequentemente, para realizar uma varredura completa desta vulnerabilidade, é necessário combinar, além das ferramentas automatizadas, uma análise de código e testes de invasão na aplicação.

Como forma de evitar XSS, é necessário realizar a separação do dado não-confiável do conteúdo ativo no navegador. Isto pode se dar de diversas maneiras. A primeira abordagem, e mais apropriada, é realizar um filtro de todos os dados não confiáveis, tomando como base o contexto HTML que os dados serão colocados. Uma outra maneira é a criação de *white list* mas, assim como na vulnerabilidade de injeção, esta lista não constitui uma defesa completa, pois muitas aplicações utilizam caracteres especiais em suas entradas. A última solução é considerar, na aplicação, a utilização de bibliotecas de auto sanitização [OWASP 2017d].

3.5.4 A4: Referência insegura e direta a objetos

Imediatamente após ao *Cross-Site Scripting*, encontra-se a vulnerabilidade de referência insegura e direta a objetos. Esta vulnerabilidade tem sua ocorrência quando um programador expõe uma referência à implementação interna a um objeto, podendo este ser um arquivo, diretório ou registo do banco de dados. Quando não há uma verificação do controle de acesso, ou outra proteção, a esses objetos, os atacantes podem manipular as referências para acessarem dados não-autorizados.

O ataque é realizado por um usuário autorizado que possua acesso a um determinado objeto. Ao modificar o valor de um parâmetro que se refere diretamente ao objeto do sistema, o atacante torna possível que outros usuários do sistema, que até então não possuíam autorização de acesso ao objeto, tenham seu acesso a ele concedido. A exploração deste ataque é possível pois as aplicações frequentemente usam o nome real, ou a chave de um objeto, ao gerar páginas Web. Essas aplicações nem sempre verificam

se o usuário que está tentando acesso direto ao objeto alvo está autorizado para vê-lo. A detecção da falha é facilmente realizada por testadores, que podem analisar o código para verificar se a autorização é solicitada mediante acesso ao objeto.

O resultado da referência direta e insegura a objetos é o comprometimento de qualquer dado que pode ser referenciado diretamente pelos parâmetros da aplicação. A menos que as referências para os objetos sejam imprevisíveis, torna-se fácil para um atacante acessar os dados disponíveis desse tipo.

A melhor maneira de testar a vulnerabilidade de uma aplicação para esse ataque é verificar se todos os objetos que são referenciados possuem um controle de acesso apropriado. Em referências diretas a recursos restritos, é necessário que a aplicação verifique se o usuário que está tentando acessar determinado recurso está autorizado para essa requisição. Testes e revisões de códigos são duas possíveis soluções para identificar referências diretas a objetos e se elas são seguras.

Para que uma aplicação fique devidamente prevenida à referência insegura e direta a objetos, é necessária a seleção de uma abordagem para controle de acesso de cada objeto que possa ser acessível ao usuário. Uma primeira abordagem possível é a utilização de referências indiretas a objetos por usuário ou sessão. Isso impede que o atacante chegue diretamente aos recursos que ele não está autorizado. Uma segunda abordagem é a verificação de acesso. Cada referência direta a objeto de uma origem não confiável deve incluir uma verificação de controle de acesso. Desta forma, garante que o usuário que está solicitando acesso a determinado objeto está autorizado para acessá-lo [OWASP 2017d].

3.5.5 A5: Configuração incorreta de segurança

A quinta vulnerabilidade destacada no OWASP Top 10, A5, é a configuração incorreta de segurança. Para garantir que uma aplicação possua uma boa segurança, é necessário que a configuração esteja bem definida, segura e implementada na aplicação, nos frameworks, no servidor de aplicação, no servidor Web, no banco de dados e na plataforma. Todas essas configurações devem ser definidas, implementadas e mantidas, já que normalmente a configuração padrão é insegura. Paralelamente, é preciso que o *software* sempre esteja atualizado.

Atacantes em potencial podem ser usuários externos à aplicação ou usuários internos autenticados que têm como objetivo comprometer o sistema. A má configuração de segurança permite a esses usuários maliciosos acessarem contas padrão, páginas não

autorizadas, falhas não corrigidas e outros caminhos possíveis, atrás da obtenção de acesso não autorizado ou reconhecimento do sistema. A exploração do ataque pode se dar em qualquer nível da aplicação, como na plataforma, servidor Web, banco de dados, códigos e outros. Desta maneira, desenvolvedores e administradores precisam trabalhar em conjunto para garantir que o sistema esteja configurado corretamente. O impacto da exploração pode ser desde o acesso não-autorizado do atacante a dados ou funcionalidades do sistema, até um comprometimento completo do mesmo.

A detecção dessa vulnerabilidade pode ser feita através da verificação da segurança em diversas partes da aplicação. Nos *softwares*, pode-se verificar se todos estão atualizados com as últimas versões disponibilizadas pelo fabricante, como o SO, servidor Web, SGBD, entre outros. É recomendável analisar os recursos e contas, checar se todos os recursos desnecessários estão desativados - o mesmo aplica-se para as contas padrão. As mensagens de erro devem ser observadas para que não revelem informações sensíveis da aplicação, como caminhos internos, por exemplo.

Para evitar que a aplicação esteja comprometida com configuração incorreta de segurança, existem algumas recomendações que devem ser seguidas. Uma delas é a realização de um processo de endurecimento (*hardening*) recorrente para implantar de maneira rápida e fácil outro ambiente. Outra recomendação a ser mencionada é a utilização de uma arquitetura de aplicação forte que forneça a separação segura e eficaz entre os componentes [OWASP 2017d].

3.5.6 A6: Exposição de dados sensíveis

A sexta vulnerabilidade do guia diz respeito à exposição de dados sensíveis, como cartões de crédito, dados fiscais e credenciais de autenticação em uma aplicação. Com a alta do *e-commerce*, muitos destes dados começaram a ser armazenados e transitar entre as aplicações, e grande parte delas não tem a proteção devida. Atacantes têm o poder de modificar ou roubar informações sensíveis não protegidas com a finalidade de fraudar cartões de crédito, adquirir novas identidades, ou qualquer outro tipo de crime. Portanto, dados sensíveis necessitam da utilização de criptografia para serem armazenados na aplicação, ou para serem transportados pela Web.

Um ataque pode partir de qualquer um que tenha acesso a dados sensíveis e seus backups. Os dados que podem ser ameaçados, nessa situação, englobam os dados em repouso, em tráfego e os que estão presentes nos navegadores dos clientes. Desta maneira,

ataques externos também são possíveis. Contudo, um ataque em si não é tão fácil de ser executado. Por isto, normalmente, os atacantes não buscam a quebra direta da criptografia. Buscam maneiras menos complicadas e mais rápidas de se obter o resultado, como roubar chaves, aplicar uma técnica de ataque chamada *Man-In-The-Middle*, ou roubar dados que estejam transitando em texto puro quando estão fora do servidor. A falha mais comum que leva a esse tipo de vulnerabilidade é não criptografar os dados sensíveis, ou quando criptografado, utilizar técnicas de *hashing* ou chaves fracas, sendo este último resultado de uma má geração ou gerenciamento das chaves.

O impacto de uma exploração bem sucedida de dados sensíveis é muito alto. Por tratar-se de dados que incluem informações confidenciais, a falha frequentemente compromete todos estes dados que deveriam estar protegidos [OWASP 2017d].

3.5.7 A7: Falta de função para controle do nível de acesso

A sétima vulnerabilidade da lista diz respeito à falta de função para controle do nível de acesso. Diversas aplicações Web realizam uma verificação nas permissões de acesso de determinado usuário nas funções da aplicação, antes de expor essa funcionalidade na interface do usuário. Porém, é necessário que ocorra essa mesma verificação de controle de acesso no servidor, no momento em que cada função é requisitada. Caso as requisições não forem verificadas, ignorando-se o controle do nível de acesso, atacantes serão capazes de manipulá-las para ter acesso a determinadas funcionalidades sem autorização adequada.

A ameaça pode partir tanto de usuários anônimos que possuam acesso à rede na qual a aplicação está hospedada, enviando requisições manipuladas para explorar a falha, quanto de usuários registrados na aplicação que possuem acesso até determinado nível do sistema. A execução do ataque é realizada quando o atacante, que pode estar cadastrado na aplicação ou não, faz alteração de uma URL ou de parâmetros de uma função que requisitem privilégios maiores que o possuído. No cenário de uma aplicação vulnerável a esta falha, o acesso à função é concedido. A chance de sucesso deste ataque se dá pelo fato das aplicações nem sempre possuírem proteção adequada das funções internas. Ocorrem casos em que a proteção no nível da função é gerenciada por configuração, mas o sistema está mal configurado, resultando num cenário vulnerável. Uma maneira de melhorar a segurança neste ponto é a inclusão de verificações de código adequadas por parte dos desenvolvedores. A detecção das falhas relacionadas à vulnerabilidade da falta

de função para controle do nível de acesso das aplicações é fácil de ser percebida. A real dificuldade se encontra na identificação das páginas (URLs) e das funções vulneráveis ao ataque. Ao realizar um ataque bem sucedido, o atacante se torna capaz de acessar funcionalidades não autorizadas em uma aplicação, especialmente funções administrativas, os principais alvos. Por conta deste cenário, um ataque bem sucedido possui impacto moderado.

Para verificar se a aplicação falha na restrição do acesso nas funções é necessário percorrer todas as funções desta aplicação, como a checagem da autenticação ou autorização no lado do servidor, ou então avaliar se as verificações no lado do servidor dependem apenas de informações fornecidas pelo usuário. O teste na aplicação pode ser feito enquanto se navega através dela com um *proxy* ativo. Em um primeiro momento, deve-se fazer esta navegação utilizando um usuário com privilégios maiores e, em seguida, tentar refazer os mesmos caminhos através de um usuário com privilégios inferiores. Ao final, se o usuário com menor privilégio tiver conseguido replicar todos os passos sem ter um erro gerado pela aplicação, significa que ela se encontra vulnerável. Uma outra abordagem para constatar falha na restrição do acesso nas funções é verificar a implementação do controle de acesso no código fonte da aplicação. A ideia é observar uma requisição privilegiada através do código e verificar o padrão de autorização cabendo, posteriormente, uma pesquisa no código fonte para encontrar onde o padrão não é seguido. Devido à necessidade de verificação de código e observação de respostas do *proxy*, a detecção desse problema através de ferramentas automatizadas é mais improvável.

Normalmente, para garantir a segurança da aplicação, são utilizados componentes externos ao código que fornecem essa proteção. Neste ponto, o ideal é ter um módulo de autorização consistente e de fácil análise utilizado por todas as funções de negócio. Um bom ponto de partida para garantir uma configuração que evite falhas, é executar mecanismos que negue todo acesso por padrão, exigindo direitos explícitos para privilégios específicos no acesso a todas as funções [OWASP 2017d].

3.5.8 A8: *Cross-Site Request Forgery*

O ponto seguinte na lista é o ataque CSRF - *Cross-Site Request Forgery*. Este ataque força que a vítima, autenticada na aplicação, submeta uma requisição HTTP

maliciosa forjada pelo atacante, contendo o *cookie* de sessão da vítima e outras informações de autenticação da sessão, para uma aplicação Web vulnerável. Esta falha permite ao atacante forçar o navegador da vítima a criar requisições que a aplicação vulnerável aceite como requisições legítimas feitas pela vítima.

Esta ameaça pode estar presente em qualquer *site* ou outro serviço HTML que usuários possam acessar. Desta forma, qualquer pessoa que possa carregar conteúdo nos navegadores dos usuários e forçá-los a fazer uma requisição para um *site* específico, é considerada um atacante. O ataque consiste quando um usuário malicioso forja requisições HTTP e as submete à vítima através *tags* de imagem, XSS ou outras técnicas. A eficiência da exploração do ataque depende diretamente do fato da vítima estar autenticada configurando, deste modo, um ataque bem sucedido.

O CSRF se baseia no fato da maioria das aplicações Web permitirem que todos os detalhes de uma ação particular da aplicação possam ser previstos por qualquer usuário. Como os navegadores automaticamente trafegam credenciais, como *cookies* de sessão, os atacantes podem criar páginas Web que possuem referências à aplicação alvo. Assim, o usuário que estiver autenticado na aplicação e que clique no *link*, automaticamente cria requisições maliciosas forjadas que são indistinguíveis das legítimas pela aplicação alvo. A detecção de falhas de CSRF é facilmente descoberta por testes de invasão ou análise de código. O impacto do ataque é notado quando o atacante engana a vítima fazendo com que ela realize qualquer mudança de estado na aplicação na qual ela está autorizada. Se a vítima for um usuário com privilégios normais, operações como alterações do cadastro, realização de compras, *login* e *logout* podem ser executadas. Porém, se o alvo do ataque for um administrador do sistema, toda a aplicação estará comprometida, refletindo em um impacto severo.

Para testar a vulnerabilidade da aplicação à CSRF, é necessário verificar se qualquer *link* e formulário possuem um *token* de CSRF. Caso o *token* não esteja presente, atacantes podem forjar requisições. Uma alternativa para defesa é solicitar a interação do usuário com a aplicação ao submeter uma requisição. Essa interação pode ser através de uma autenticação, CAPTCHA, ou qualquer outra prova de que se trata de um usuário real. O principal foco, neste ponto, é verificar links e formulários que chamam funções de mudança de estado, por serem os alvos mais importantes de um CSRF.

A melhor forma de se prevenir de um CSRF é incluir um *token* imprevisível e único, em cada requisição HTTP. O procedimento mais recomendável é sua inclusão

através de um campo oculto, forçando que ele seja enviado no corpo da requisição e evitando sua inserção na URL. Em situações que o *token* é incluído na URL, ou em parâmetros dela, o risco de interceptação deste *token* é maior, podendo compromete-lo [OWASP 2017d].

3.5.9 A9: Utilização de componentes vulneráveis conhecidos

A penúltima vulnerabilidade é a utilização de componentes vulneráveis conhecidos, que englobam desde bibliotecas, frameworks, até outros módulos de software. Para executá-los, quase sempre são necessários privilégios elevados. Uma vez que um desses componentes seja explorado, o atacante tem o poder de causar perda de dados ou um comprometimento do servidor. Aplicações que utilizam componentes vulneráveis acabam por comprometer a defesa e permitir diversos tipos de ataques.

A ameaça pode ser causada através da identificação e exploração de alguns componentes que possuem vulnerabilidades, utilizando-se ferramentas automatizadas que criam uma expansão do leque dos causadores dessa fraqueza. A execução do ataque ocorre a partir do momento que o atacante identifica um componente vulnerável na aplicação, seja através de varredura, ou através de análise manual. A partir deste ponto, ele personaliza o *exploit* de acordo com sua necessidade e executa o ataque. A ocorrência desta vulnerabilidade é muito alta. É comum que uma grande parte das aplicações possuam este problema, pois a maioria dos times de desenvolvimento não se preocupam em garantir que os componentes utilizados, bibliotecas e outros artefatos, estejam atualizados. Em um grande número de casos, os desenvolvedores sequer conhecem todos os componentes que estão sendo utilizados e suas respectivas versões, agravando ainda mais a situação quando os componentes possuem dependência. A fraqueza relacionada aos componentes vulneráveis tornam possíveis uma grande quantidade de ataques, como injeções, falhas no controle de acesso à aplicação, XSS, entre outros. Isto possui como consequência um impacto que pode variar de pequeno, comprometendo alguns dados, até extremo, onde um *host* completo é afetado.

Determinar se certa aplicação está utilizando componentes vulneráveis deveria ser algo fácil, porém relatórios de vulnerabilidades de *software* nem sempre especificam exatamente quais versões possuem vulnerabilidades. Adicionalmente, diversas vulnerabilidades não são reportadas para um local de centralização de vulnerabilidades, como a CVE, por exemplo. Para melhorar a chance de detecção de certo componente ser

vulnerável, são cabíveis pesquisas nestes bancos de dados especializados - CVE, NVD. Existindo a situação de algum componente ter uma vulnerabilidade, deve-se avaliar cuidadosamente se o sistema realmente está vulnerável. Cabe aqui analisar se o código utiliza partes do componente com a vulnerabilidade e se a falha pode resultar em um impacto que preocupe o negócio.

Para evitar a exposição a esse tipo de vulnerabilidade, a principal solução é manter os componentes sempre atualizados. Muitos projetos de componentes não criam correções de vulnerabilidades para versões antigas, e sim corrigem os problemas através de atualizações. Outra forma de prevenção é incluir rotinas no processo do negócio, como identificar todos os componentes e as versões que estão sendo utilizadas, monitorar a segurança deles em bancos de dados públicos e em listas de email de projetos de segurança e estabelecer políticas de segurança que definam de forma detalhada o uso do componente internamente no sistema [OWASP 2017d].

3.5.10 A10: Redirecionamentos e encaminhamentos inválidos

Para finalizar, a última vulnerabilidade do OWASP Top 10 são os redirecionamentos e encaminhamentos inválidos. As aplicações Web frequentemente redirecionam e encaminham usuários para outras páginas e *sites*, e utilizam dados não confiáveis para determinar a página de destino. Não existindo uma validação adequada, os atacantes estão aptos a redirecionar as vítimas para os *sites* que desejarem, podendo ser *sites* de *phishing* ou *malware*, ou então usar encaminhamentos para acessar páginas não autorizadas.

O atacante pode ser qualquer pessoa que engane os usuários para que enviem uma requisição para um determinado *site*. A execução do ataque ocorre quando o usuário malicioso cria um redirecionamento inválido e leva as vítimas a clicarem nele. Elas são propensas a clicar, pois o *link* aponta para um *site* real e válido. O atacante, então, cria um encaminhamento inseguro para evitar verificações de segurança.

Encontrar redirecionamentos inválidos não é muito comum. As aplicações frequentemente redirecionam usuários para outras páginas, ou utilizam encaminhamentos internos à própria aplicação. Em poucas vezes, a página alvo é especificada através de um parâmetro inválido que permitem aos atacantes escolherem a página de destino. Detectar redirecionamentos inseguros é fácil, basta procurar por aqueles que é possível

inserir uma URL completa. Em contrapartida, detectar encaminhamentos inseguros é mais difícil, pelo fato de apontarem para páginas internas.

O impacto do redirecionamento pode ser desde uma tentativa de instalação de *malware*, ou até enganar a vítima para descobrir senhas ou outras informações sensíveis. Quanto aos encaminhamentos, o impacto pode ser de atravessar controles de acesso de uma aplicação.

Existem três principais maneiras de descobrir se uma aplicação é vulnerável a redirecionamentos ou encaminhamentos inválidos. A primeira é realizar uma revisão do código em todas as áreas que tem utilização de redirecionamento ou encaminhamento. Para cada uso, é necessário identificar se a URL que está como destino está incluída em algum parâmetro. Caso ela esteja e não seja validada por uma *whitelist*, significa que a aplicação se encontra vulnerável. A segunda maneira é fazer uma varredura no *site* para ver se ele gera algum redirecionamento (respostas 3XX do HTTP). Em caso positivo, é necessário analisar os parâmetros anteriores a ele para verificar se parecem ser uma URL de destino ou apenas parte dela. Caso seja uma URL de destino, é necessário alterá-la para observar se o *site* irá realizar o redirecionamento. A última maneira é a situação na qual o código não está disponível. Nesta, é necessário verificar todos os parâmetros para avaliar se parecem ser parte de um redirecionamento ou encaminhamento para, então, testar todos.

Nas maneiras de evitar redirecionamentos e encaminhamentos inseguros, duas abordagens são possíveis. A primeira é simplesmente não utilizá-los. A segunda é utilizar os redirecionamentos, porém sem envolver parâmetros do usuário no cálculo do destino [OWASP 2017d].

3.6 Treinamento para a detecção de vulnerabilidades

É perceptível a necessidade da Web e suas aplicações para o sucesso e nas rotinas diárias de organizações, empresas e qualquer tipo de negócio. Sua ampla variedade de aplicações e seus respectivos serviços resultam em um cenário onde, independentemente do tipo de negócio aplicado, diversas serão as aplicações utilizadas e modificações empreendidas, resultando em adaptações para a marca e objetivos em questão. Por maior que seja o valor investido em segurança, as aplicações só atingirão uma boa margem de

segurança quando houver a certeza de que todas foram submetidas a testes e que a dificuldade de exploração é muito alta.

Por diversos motivos, principalmente custos de tempo, nem todos testes podem ser realizados da maneira desejada e nem todas as aplicações possuem suas funcionalidades submetidas aos testes. Portanto, torna-se mais viável a realização de testes por vulnerabilidades, recomendando-se que nunca se deixe de testar todas as vulnerabilidades expostas pelo OWASP Top 10.

A utilização do OWASP Top 10 requer um ambiente adequado à realização de testes por vulnerabilidades. Ambiente, portanto, vulnerável, mas controlado. O ambiente deve possuir diversas máquinas intencionalmente projetadas com falhas, desde as mais graves até as simples. Uma delas, porém, deve possuir claramente um objetivo muito valioso para qualquer negócio: preparar a equipe para lidar diretamente com as falhas que possam vir a ser encontradas na aplicação, inclusive as dez vulnerabilidades mais críticas apontadas pelo OWASP Top 10.

3.7 Combate às vulnerabilidades: a bWAPP

O contexto atual das organizações e empresas envidando esforços para tornar seu negócio difundido e acessível por todos via Web, traz como consequência a necessidade de preocupação extrema com a segurança. Porém, o cenário que se encontra não é o mais esperado. A segurança de aplicações Web é, atualmente, o aspecto mais negligenciado de segurança de uma empresa. Isso caracteriza um cenário onde hackers maliciosos veem neste contexto uma oportunidade para realizar ataques direcionados, concentrando seus esforços nas aplicações Web e sites das mais variadas empresas.

Um conjunto de fatores contribui para tornar as aplicações um alvo tão constante, e em muitas vezes bem sucedido, de ataques. Sua disponibilidade 24/7 não garante que durante todo este período haverá um time de segurança monitorando as atividades e pronto para responder a incidentes. Devido a isso, um usuário mal intencionado poderá realizar um ataque que gere bastante ruído interno e não terá ninguém para respondê-lo a tempo.

Outro fator que motiva os hackers maliciosos a mirarem nas aplicações Web é a quantidade de dados sensíveis que estão armazenados em aplicações críticas da empresa. Determinadas aplicações, como servidores de bancos de dados ou servidores de arquivos,

possuem dados de extrema sensibilidade interna da organização. Dados estes que, muitas vezes, somente alguns usuários internos (que possuam alto privilégio na aplicação), são capazes de acessá-los. Um ataque bem sucedido, que resulte no contato direto do invasor com os dados, causará um impacto com valor incalculável para a empresa.

Também pode ser citada a confiança que certas empresas depositam em seus equipamentos ou medidas adotadas de segurança, nem sempre tão eficazes. A utilização de *hardwares* de segurança, como *firewalls* tradicionais, IPS, IDS e outros, não garantem uma proteção total do sistema. Atacantes podem encontrar métodos de atravessá-los, ou então, vulnerabilidades de outros componentes que lhes permitam acessar áreas restritas da aplicação, capacitando-lhes acessar o sistema em diversos níveis.

O último fator a ser destacado é a customização que está presente na maioria das aplicações de uma organização. É normal que as aplicações puras, lançadas diretamente pelo fabricante, tenham vulnerabilidades descobertas depois de um determinado tempo. Porém a vulnerabilidade se torna ainda maior quando se resolve customizar essas aplicações para atenderem aos requisitos ou identidade da empresa. Durante este processo, e aliado com o tempo curto para entrega dessa aplicação customizada, diversos testes não são realizados, resultando em uma alta propensão a presença de vulnerabilidades. Conhecendo este cenário, os atacantes, ao encontrarem aplicações que se enquadram neste tipo, gastam esforços para encontrar brechas que os tornem possíveis de entrar no sistema.

Conhecendo este alarmante contexto atual das aplicações Web e sabendo da extrema necessidade de defesa que se torna necessária para proteger negócios dos hackers, a MME BVBA criou um projeto chamado de bWAPP.

A MME BVBA, é uma empresa independente de segurança de tecnologia de informação especializada em auditorias de segurança, testes de invasão, hacktivismo ético e treinamentos de segurança. Possuí uma mentalidade livre de preconceitos a respeito dos incidentes de segurança e proteção de dados. Sua missão é advertir e remediar ameaças e vulnerabilidades.

Seu projeto bWAPP é uma "Aplicação Web 'Bugada'". Por ser uma aplicação intencionalmente insegura, todas as mais conhecidas vulnerabilidades da atualidade estão presentes em seu sistema. Isto permite que seus usuários, em grande maioria entusiastas e equipes de segurança, possam de uma forma segura e controlada descobri-las, entender

como pode funcionar um ataque ("*think like an attacker*") e com isso pensar em maneiras e estratégias de prevenir que estas ameaças se encontrem presentes nas suas aplicações Web [Mesellem 2014].

A arquitetura utilizada na bWAPP é a LAMP. LAMP é um modelo muito comum de pilha de serviços Web, onde sua sigla vem dos nomes originais dos quatro componentes *open-source* utilizados: sistema operacional Linux, servidor HTTP Apache, sistema de gerenciamento de banco de dados relacional MySQL e a linguagem de programação PHP. Esta pilha composta por estes componentes é altamente recomendada para construir *sites* e aplicações Web. A justificativa para utilização desses serviços, ao invés de outros, como alguns da Microsoft, envolve segurança, desempenho e popularidade [Timberlake 2002].

A preferência do Linux é resultado de sua utilização da política GPL - GNU *Public License*, onde os *softwares* que estão sob esta política são livres e continuarão livres independentemente de quem modificar ou distribuir o programa [Smith 2014].

Em relação ao Apache, foi escolhido por estar presente em mais da metade dos servidores Web do mundo, além de não ter problemas históricos com segurança de vírus, necessitando frequentemente de atualizações.

Já o MySQL é recomendado para aplicações mais simples, devido a seu desempenho mais rápido.

Por último, foi escolhido o PHP devido ao fato de ser uma linguagem para manipulação de HTML e criação de páginas Web "*on the fly*". O que o difere de outras linguagens similares, como o ASP.Net, é o fato desta última fazer parte do .Net *framework*, a tornando mais pesada que o PHP.

A principal característica da bWAPP é ser amigável ao usuário. Devido à sua interface simples e fácil de usar, o usuário é capaz de entender o propósito de cada funcionalidade e, então, fazer uma correta e intuitiva utilização da máquina. Possui seu código PHP bem estruturado e documentado, possibilitando um melhor entendimento das funções ao usuário avançado.

Uma outra característica da bWAPP é sua possibilidade de edição por parte do usuário. Possibilita-lhe definir o nível de segurança e o fator de dificuldade para cada vulnerabilidade que será testada (em testes manuais). A máquina também possui funções secundárias e opcionais de configurações, como a criação de novos usuários e redefinição

do banco de dados e de teste em emails. Quanto às funcionalidades relacionadas a arquivos, permite que o usuário realize configurações locais PHP, além de possuir um modo nomeado de "Evil Bee" que tem como objetivo atravessar as confirmações de segurança dos arquivos. Para auxílio em métodos de ataques, a máquina é equipada com um diretório chamado de "Evil", que possui scripts para este fim.

O motivo pelo qual a bWAPP se destaca num mercado tão saturado de máquinas com foco em treinamentos e aprimoramento de conhecimento dos profissionais e entusiastas de segurança, é o fato dela conter mais de 100 vulnerabilidades da Web. É uma cobertura de todas as maiores, e mais conhecidas, vulnerabilidades que se tornam alvos de diversos ataques que prejudicam o negócio de várias organizações nos dias de hoje. Além destas, a bWAPP também contempla todos os riscos expostos no OWASP Top 10, sendo uma peça chave, e fundamental, no treinamento de uma equipe de segurança que objetive a estruturação de um plano de segurança eficiente. Como no cenário real diversos ataques dependem de outros para serem bem sucedidos, devido à sua ampla quantidade de vulnerabilidades, esta máquina consegue preparar os profissionais para compreenderem e simularem ataques que não focam em somente uma vulnerabilidade.

Das mais de 100 vulnerabilidades Web contidas na bWAPP, algumas merecem destaque devido a sua prevalência nas aplicações atuais. No contexto das injeções, possui as de SQL, HTML, LDAP, SMTP, entre outras. Quanto aos bugs existentes intra-sites, ela contempla os três principais, sendo estes XSS, XST e CSRF. Problemas relacionados a AJAX e serviços Web, como jQuery, JSON e XML também podem ser testados pelos usuários da máquina. Quanto a arquivos, a bWAPP contempla falhas de uploads, arquivos de backdoor, acesso a arquivos arbitrários, inclusão de arquivos locais e remotos (LFI/RFI). Nas partes relacionadas às sessões, pode-se testar problemas relacionados à autenticação e autorização. Os erros de configuração estão representados pelo Man-in-the-Middle, vazamento de informações, incorretas configurações dos protocolos SNMP e Samba. Outros problemas em aplicações Web, como ClickJacking, Cross-Origin Resource Sharing, Server Side Request Forgery e ataques XML External Entity também são encontrados nesta máquina vulnerável. Para finalizar a enumeração dos principais bugs presentes, pode-se destacar vulnerabilidades como Heartbleed e Shellshock, ataques DoS, envenenamento de cookie e reset de senhas.

A bWAPP conta com o modo chamado de A.I.M (Authentication Is Missing), intencionalmente desenvolvido para testar Web scanners e crawlers. Trata-se de um modo que não requer autenticação é acessado a partir do endereço http://[IP]/bWAPP/aim.php [Mesellem 2014].

A máquina bWAPP não funciona sozinha, ela apenas serve para simular uma máquina exposta, contendo vulnerabilidades. Mas para que essas vulnerabilidades consigam ser corretamente exploradas, há a necessidade de ferramentas externas, localizadas em outras máquinas. As ferramentas para testes, chamadas de ferramentas de testes de invasão, podem ser encontradas separadamente nos sites de seus fabricantes, o que levaria a uma perda de tempo desnecessária baixar, configurar e instalar cada uma. Por conta disso, alguns sistemas operacionais baseados em Linux são projetados especialmente para esse tipo de cenário de testes de invasão e segurança, onde sua instalação padrão já possui todas as principais ferramentas de segurança utilizadas pelos profissionais da área. Estas distribuições contam com ferramentas de invasão e forense, todas open source e agrupadas por categorias. Esta organização das ferramentas auxilia o trabalho do auditor de segurança, mostrando-lhe as diferentes opções de ferramentas disponíveis para cada parte de um teste de invasão.

As principais distribuições do sistema operacional Linux existentes no mercado atualmente são: Kali, BackBox, NodeZero, Blackbuntu e Samurai WTF. Entre estas, a mais utilizada e que, consequentemente, será utilizada como máquina atacante neste trabalho, será o Kali Linux.

O Kali é uma distribuição baseada em Debian, projetada para forense digital e testes de invasão, mantido e criado pela Offensive Security. Vem pré-instalado com uma grande quantidade de ferramentas, incluindo Aircrack-ng, Ettercap, John the Ripper, Metasploit, Nmap, Wireshark, entre outros. Além disso, esta distribuição possui algumas das principais ferramentas de varredura de vulnerabilidades, como o Vega.

4 Avaliação dos scanners de vulnerabildiade

4.1 Scanners de vulnerabilidade

Sempre que fabricantes de *software* criam uma nova aplicação, inúmeras linhas de código são escritas. Mesmo que eles desenvolvam o código utilizando as melhores práticas, enganos são cometidos e erros consequentemente aparecem. Em diversas ocasiões, não é possível notar-se que o modo que o código foi escrito é errado. Este fato só se torna perceptível depois, quando alguém descobre uma maneira de manipular o código para realizar outra tarefa, diferente da que ele foi planejado para executar.

Quando um *software* é liberado para uso público, ele pode levar a descobertas. Pode-se, por exemplo, fazer algo totalmente inesperado com ele e receber-se de volta um resultado mais inesperado ainda. Algumas vezes, este resultado não é somente inesperado, mas também indesejável e atacantes, ou *hackers* maliciosos, podem ganhar acesso a máquinas ao utilizarem essa falha do software.

Neste cenário, existem ferramentas apropriadas para minimizar a quantidade de erros presentes em uma aplicação. São os *scanners* de vulnerabilidade.

Scanners de vulnerabilidade para aplicação Web são ferramentas automáticas, comerciais ou *open sources*, externas à aplicação, que tem como função varre-la à procura de vulnerabilidades de segurança, como *Cross-site Scripting*, SQL *Injection*, *Command Injection*, Diretórios Transversos, configurações inseguras do servidor, entre diversas outras. Sua importância está na identificação e recomendações para conserto de brechas de segurança, prevenindo ataques ao sistema.

Escanear vulnerabilidades envolve executar um programa (uma ferramenta de scan de vulnerabilidade) em uma máquina e conectá-la, através da rede, a uma máquina alvo, a ser testada. Implica, portanto, em tentar uma grande quantidade de eventos para ver quais serão bem sucedidos, em relação ao alvo.

As ferramentas de *scan* de vulnerabilidades, *Vulnerability Scanning Device* (VSD), funcionam sem possuir um conhecimento prévio dos sistemas, além de poderem ter conhecimento, ou não, de seus endereços IPs. Não há necessidade de carregar ou manter qualquer *software* cliente nos servidores, tudo é feito pela rede [Houghton 2003].

A partir do momento que é determinada a necessidade de automatizar o teste nas aplicações Web de um servidor, torna-se necessário identificar as ferramentas disponíveis e suas limitações. Existe um grande número de diferentes produtos que realizam essa tarefa em um grau maior ou menor de profundidade.

Eles podem ser classificados em duas famílias, as Ferramentas *Policy Compliance* e os *Scanners* de Vulnerabilidade. As primeiras possuem uma parte que funciona como cliente e outra como servidor ou controlador, de modo que o cliente, localizado internamente na máquina alvo, realiza uma varredura desta e o servidor recolhe o resultado. Já os *Scanners* de Vulnerabilidade varrem o servidor sem a existência de um cliente e testam um vasto catálogo de eventos com a finalidade de obter respostas inesperadas.

Cabe esclarecer que esses diferentes tipos de ferramentas possuem apenas pequenas diferenças em seus funcionamentos, porém os passos executados para realizar a busca de vulnerabilidades em um grupo de servidores que pertencem à mesma subrede são iguais.

A primeira exigência necessária ao funcionamento dessas ferramentas é uma rota de rede entre elas e os servidores alvos a serem escaneados. No caso de uma varredura de vulnerabilidades de servidores Web, o melhor lugar para realizar a execução de uma ferramenta é a Internet, pois ela proporciona uma visão universal do servidor, isto é, se é possível que um auditor veja uma vulnerabilidade, todos que a procurarem cuidadosamente também conseguirão vê-la [Palmaers 2013].

Há a possibilidade de se fazer varreduras tanto externas quanto internas na rede. Entretanto, somente as máquinas que têm um endereço IP externo são visíveis na Internet e, consequentemente, para as varreduras externas.

O primeiro passo de uma varredura, normalmente, é realizar uma varredura de portas limitada para cada IP, como forma de verificar se o servidor responde naquele endereço e se aceita conexões.

De posse da lista dos servidores pertencentes à rede, o scanner volta à cada um dos endereços e faz uma varredura mais completa, para determinar quais portas estão abertas. Conhecendo as portas aptas a receberem conexão, o scanner roda um conjunto de verificações de vulnerabilidades contra cada uma delas, para ver se alguma irá retornar um comportamento inesperado, caracterizando um possível risco.

Uma vez que o *scanner* executou por completo, ele gera uma saída contendo os eventos que ele acredita que mereçam atenção. Desses dados, as ferramentas de varredura de vulnerabilidades criam um relatório e categorizam os resultados de acordo com o script que foi executado para encontrar determinada vulnerabilidade.

É importante destacar que não se pode afirmar que os resultados encontrados e apresentados no relatório se tratam efetivamente de riscos. Eles são considerados respostas que precisam de uma investigação mais aprofundada. O motivo pelo qual não podemos declarar que toda a saída se trata de uma ameaça, é o fato dos *scanners* de vulnerabilidade somente verem as respostas (ou sua falta) do servidor para um evento específico.

4.2 Ferramentas de detecção de vulnerabilidade

As ferramentas usadas no presente estudo foram o Arachini, o Nessus e o Vega. Sua escolha atendeu a motivos diferentes. O Arachini e o Vega foram aleatoriamente selecionados por serem ferramentas públicas de varredura de vulnerabilidades em aplicações *Web* bastante utilizadas e compatíveis com o Kali Linux. Já o Nessus foi escolhido por ser uma ferramenta de mercado que oferece uma versão *home*, grátis, com apenas algumas funcionalidades, porém, suficientes para o escopo deste estudo. É também compatível com o Kali Linux.

4.2.1 O Arachini

O Arachini é um *scanner* de vulnerabilidades modular e com diversos recursos, voltado para ajudar auditores a avaliarem a segurança das aplicações *Web*. É grátis e seu código fonte é público.

Diferentemente dos outros *scanners*, ele leva em conta a natureza dinâmica das aplicações *Web*, podendo detectar as alterações causadas enquanto percorre os caminhos dessas aplicações e se adaptar às alterações. Portanto, ele é inteligente, analisa e aprende com o comportamento da aplicação *Web*, se aprimorando à medida que a varredura vai sendo concluída. Assim, as ameaças que poderiam, em outras circunstâncias, serem indetectáveis pelas máquinas, podem ser tratadas sem problemas. A análise dos resultados encontrados considera uma série de fatores que visam maximizar sua confiabilidade, minimizando os falsos positivos.

É capaz de se integrar com o navegador e, portanto, pode auditar códigos *client-side* e suportar aplicações *Web* elaboradas, que fazem uso massivo de tecnologias como JavaScript, HTML, *Document Object Model* e AJAX.

Além disso, é versátil suficiente para cobrir diversas demandas, desde uma simples varredura utilizando linha de comando até uma rede global de *scanners* de alto desempenho.

O Arachini é estável, eficiente e com alto desempenho. Apesar de algumas de suas partes serem relativamente complexas, o usuário nunca terá que lidar diretamente com elas. Tudo lhe é apresentado de maneira simples e direta, desde a função de um simples scanner em linha de comando até a plataforma *Web*, fornecendo-lhe desempenho, flexibilidade, diversas orientações e *feedbacks*.

Em linhas gerais, o Arachini só necessita que lhe seja passada a URL do *site* que será alvo, para que o *scan* seja realizado.

Ele possui diversas funcionalidades, sejam para adicionar mais características ao scanner, quanto para permitir personalizações específicas. Podem ser destacados, para o primeiro caso a possibilidade de realizar *User Agent Spoofing*, detectar automaticamente *logouts*, detectar páginas customizadas de erro 404, entre outras. Já, em relação à segunda, o auditor tem a possibilidade de realizar a customização do cabeçalho das requisições, especificar valores de *cookie-string* e *cookie-jar*, pausar e retomar uma varredura, entre outras.

Uma opção disponível na configuração da varredura é a detecção e abstração das páginas redundantes de forma automática pelo *scanner*. Ele é capaz de utilizar filtros para incluir ou excluir URLs do escopo através de expressões regulares, além de também utilizá-las para criar filtros que excluam páginas de acordo com seu conteúdo. Pode também forçar navegações somente sobre caminhos HTTPS impedindo que sejam modificados para HTTP, além da opção de seguir subdomínios. Permite ainda o ajuste do limite da contagem de páginas, de redirecionamentos e da profundidade de diretórios.

O Arachini é um sistema altamente modular, utilizando vários componentes de diferentes tipos para realizar suas atividades. Além de configurar componentes visando ajustar o comportamento e os recursos utilizados do sistema durante uma varredura, sua configuração pode ser estendida à adição de componentes criados pelo usuário para atender suas necessidades.

Para fazer um uso eficiente da largura de banda disponível, ele realiza um *fingerprinting* dos serviços que estão atuando na aplicação *Web* e adapta o processo de

auditoria às tecnologias implementadas na aplicação, usando somente os *payloads* apropriados. Nos sistemas operacionais, o Arachini é capaz de realizar detecção de BSD, Linux, Unix, Windows e Solaris. Nos quesitos dos servidores *Web*, a ferramenta consegue identificar Apache, IIS, Nginx, Tomcat, Hetty e Gunicorn. As linguagens de programação PHP, ASP, ASPX, Java, Python e Ruby são reconhecidas pelo *fingerprint*. E, por último, o Arachini reconhece alguns *frameworks*, como Rack, CakePHP, Rails, Django, ASP.NET MVC, JSF, CherryPy, Nette e Symfony. Porém, o usuário tem a opção de especificar plataformas extras, como servidores de banco de dados, para tornar o sistema a ser mais eficiente ainda.

Os componentes do sistema que executam avaliações de segurança e problemas de log são chamados de *checks*, podendo se enquadrar em dois tipos: ativos e passivos.

Os primeiros auditam as aplicações *Web* a partir de suas entradas. No contexto das injeções SQL, o Arachini é compatível com os principais bancos de dados, como Oracle, PostgreSQL e MySQL, além de permitir testes de blind SQL Injection. Já, nos bancos de dados não-relacionais, ele permite NoSQL injection em MongoDB e blind NoSQL injection. Nas injeções de código, pode-se testar em PHP, Ruby, Python, Java e ASP. Dentre as várias outras vulnerabilidades que podem se checadas pelo Arachini, vale ressaltar o Cross-Site Request Forgery, path traversal, injeção de LDAP e diversos tipos de XSS, como XSS em tags HTML e em DOM.

Quanto às avaliações passivas, procuram a existência de arquivos, pastas e assinaturas. Totalizam 34, porém alguns testes merecem ser destacados devido à sua importância, como é o caso do “allowed HTTP methods”, “credit card number disclosure”, “insecure cookies”, “missing strict-transport-security” e “missing X-Frame-Options”.

Com o objetivo de adicionar funcionalidades extras ao sistema de uma maneira modular, o Arachini conta com 21 *plugins* dos mais variados, contendo desde ataques de dicionário para HTTP Auth, até um limitador de taxas, que serve para limitar as requisições HTTP. Outros, porém, podem ser destacados como o coletor de *cookie*, que rastreia e cria uma linha do tempo das mudanças de valores de um *cookie* e o detector de WAF (Web Application Firewall).

Existem também os *plugins* padrões, executados em todas as varreduras, o AutoThrottle e o Healthmap. O primeiro ajusta a taxa de transferência HTTP durante a varredura para obter a máxima utilização da largura de banda. Já o segundo cria um mapa de *site* que mostra o estado de cada URL auditada.

Ao acessar o Arachini, o usuário se depara com sua interface gráfica. Sua página principal funciona como um *dashboard* bem simples, mostrando um gráfico de fácil entendimento com o progresso das vulnerabilidades encontradas na varredura. Ao criar um novo *scan*, o usuário deve escolher entre os três perfis existente, SQL injection, XSS e Default, um para guiar sua busca por vulnerabilidades. Além disto, o usuário tem total autonomia para criar um novo que se direcione melhor aos seus objetivos [Arachini 2010].

4.2.2 O Nessus

O Nessus é uma solução paga de avaliação de vulnerabilidade em aplicações, amplamente utilizada. Sua função é ajudar a reduzir a superfície de ataque de uma organização. Para concluir esta tarefa com maior eficiência, oferece varreduras mais específicas, sendo estas de sistemas operacionais, de dispositivos de rede, de aplicações *Web* e infraestruturas críticas atrás de vulnerabilidades, e violações de *compliance*. Como forma de otimizar o *scan*, conta com o reconhecimento de ativos de alta velocidade, auditoria de configuração, criação de perfis de segmentação, entre outros recursos.

Com uma biblioteca de vulnerabilidades constantemente atualizada, o Nessus é uma das referências em velocidade e precisão em *scan* de vulnerabilidades.

Seus módulos focados na detecção de vulnerabilidades são chamados de *Plugins* e são escritos em uma linguagem própria, a Nessus *Attack Scripting Language* (NASL). Os *Plugins* contêm informação de vulnerabilidade, um conjunto genérico de ações remediativas e um algoritmo que testam a presença destes problemas de segurança na aplicação. Também são utilizados para obter informações da configuração dos *hosts* autenticados, servindo para detectar a não utilização de boas práticas de segurança. Por padrão, são configurados e atualizados automaticamente nas atualizações do Nessus que ocorrem diariamente.

Além disso, o Nessus suporta o CVSS e valores de suas versões 2 e 3. Na situação em que ambos os atributos, CVSS2 e CVSS3, estiverem presentes, as duas pontuações serão contadas, entretanto na determinação do atributo "Fator de Risco" presente no relatório final, a pontuação da CVSS2 terá prioridade.

Ao se logar na interface *Web* do Nessus, o usuário é apresentado a uma página simples, porém funcional. Além das opções de configuração de conta e do sistema, duas outras são encontradas em seu menu: *scans* e *policies*.

O item *scans* leva o usuário à página *My Scans*, que lista todos os *scans* criados, concluidos, agendados e em execução pelo usuário. Já o segundo item, leva à página *All Policies*, que lista todas as políticas de varredura criadas pelo usuário.

O início da varredura de vulnerabilidades em aplicações *Web* pelo Nessus dá-se com a seleção de um template, entre os disponíveis, pelo auditor. Para que seja possível sua execução, o usuário deve configurá-lo corretamente, de modo a assegurar-se de que todo o ativo será varrido pelo Nessus.

A parte técnica das varreduras, como tempo limite, número de *hosts*, tipo de *scanner* de porta, entre outros, é controlada por parâmetros configurados na página dos *scans*. Nela, também são inseridas credenciais para *scanners* locais que possuam autenticação, como: SSH, *scans* autenticados em bancos de dados Oracle, e *scanners* em protocolos do tipo HTTP, FTP, POP, IMAP ou Kerberos.

O scan do Nessus conta com a funcionalidade de *Discovery*, responsável por controlar opções relacionadas à descoberta de ativos e varredura de portas. Cabe ao auditor especificar quais portas devem ser escaneadas e qual o tipo de varredura que será executada, de modo a chegar a uma conclusão sobre o seu estado, se aberta ou fechada. Por padrão, no Nessus é utilizado o SYN Scan em aproximadamente 4790 portas TCP.

Ainda quanto ao *Discovery*, o Nessus conta com o *Service Discovery*. Ele define opções que tentam mapear cada porta que se encontra aberta com um possível serviço que ela pode estar executando.

Outro item a ser configurado no *scan* são as avaliações (*Assessment*), visando-se a exatidão através da sobreescrição da precisão padrão de uma varredura. Quando habilitado, o Nessus é capaz de minimizar, ainda mais, a quantidade de falsos positivos e produzir um resultado ainda mais completo. Por exemplo, ao encontrar um servidor de arquivos, o Nessus poderá analisar até 3 níveis de profundidade de diretórios, porém os plugins irão trabalhar com maior esforço, consumindo mais recurso de banda.

O Nessus também conta com funcionalidade de força bruta. É capaz de realizar testes específicos deste tipo de ataque na aplicação através da integração com um programa muito utilizado para este fim, o Hydra.

Por fim, o auditor pode personalizar certas funções de comportamento do *scan*: configurações pontuais de como será feito o *crawling* (por exemplo, escolha do diretório onde o *crawling* será iniciado), definição do máximo de páginas a serem visitadas e das

páginas a serem excluídas do escopo do *crawling* e inserção de características específicas como o *User-Agent* das requisições, para evitar que sejam bloqueadas por determinados WAF. Além de outras opções disponíveis, ele também pode testar: tanto os métodos GET quanto o POST na aplicação e a poluição de parâmetros HTTP - uma espécie de ataque que tenta atravessar mecanismos de filtragem de inputs enviados pelo usuário, injetando um conteúdo inválido numa variável dos parâmetros HTTP, ao mesmo tempo que envia esta mesma variável com valores válidos para a aplicação. Quanto aos relatórios, pode ser modificada a maneira que o Report é gerado, de forma a possuir mais ou menos informações sobre a varredura.

Com o objetivo de reduzir o impacto causado na rede, o Nessus tem a opção de diminuir o fluxo do scan, caso seja detectado congestionamento na rede [Tenable Network Security 2016].

4.2.3 O Vega

O Vega é um *scanner* de vulnerabilidade *open source* que possui dois modos, o de *scanner* e o de *proxy*. Quando a ferramenta é inicializada, automaticamente abre o primeiro, modo este que visa a procura por vulnerabilidades em uma máquina. O *scanner* do Vega é uma ferramenta automática de teste de segurança que realiza um *crawling* em *Websites*, visando uma análise do seu conteúdo para buscar *links*, formulários, pontos de injeção e outros. Além disto, executa módulos JavaScript para testar o conteúdo descoberto.

Esta ferramenta realiza a varredura dos *sites* de maneira recursiva. Primeiro, ela cria uma representação interna do site em uma estrutura de árvore composta por entidades conhecidas como nós. Estes nós podem ser os mais variados possíveis, como diretórios, arquivos em geral, ou arquivos mais específicos com parâmetros GET ou POST. Quanto mais complexo for um *site*, maior será a varredura realizada. Por isso, o Vega disponibiliza nas preferências do *scan* uma opção de limite no escopo da varredura, caso seja de interesse do auditor.

Existem dois tipos de configurações ligadas às varreduras, as configurações do *scan* e as de *debug*.

As primeiras servem para limitar uma varredura e fornece ao usuário seis opções para o ajuste do *scan* a seu gosto.

É possível determinar o número total de descendentes de um nó que será visitado. Os descendentes tanto podem ser os subdiretórios deste nó, quanto seus parâmetros. Além disso, o usuário conta com a opção de limitar o total de descendentes que serão visitados, a hierarquia do nó, através da profundidade máxima de um caminho. Tomando-se como exemplo a profundidade máxima de tamanho quatro, o *scan* contemplaria o /nível1/nível2/nível3/nível4. O Vega também conta com a opção de configurar o número máximo de elementos duplicados em um caminho, por exemplo, /imagens/imagens/imagens. Por último, pode-se configurar o número máximo de requisições que a ferramenta envia por segundo. O ajuste da velocidade da varredura que o Vega irá executar evita um possível congestionamento de rede.

As configurações *Scanner Debugging* são responsáveis por ajustar as preferências que serão utilizadas durante uma depuração. Para isto, o usuário conta com duas opções.

A primeira consiste na realização de um *log* de todas as requisições do *scanner*. Por padrão, somente as requisições e respostas que geram algum tipo de alerta no banco de dados da ferramenta são salvas pelo Vega. Ao habilitar esta opção, todos os pares requisição-resposta têm um *log* gerado.

A segunda opção é a exibição da saída do *debug* no console. Esta faz com que o Vega mostre, detalhadamente, todos os *logs* no console da ferramenta.

Uma vantagem do Vega é o fato de sua página principal ser um *dashboard* montado de acordo à vontade do usuário, onde ele, conhecendo as informações mais relevantes para si, pode escolher quais devem aparecer na página inicial.

De posse do conhecimento sobre a arquitetura do *scan* no Vega e algumas de suas preferências disponíveis, cabe um aprofundamento na configuração de uma nova varredura.

Para iniciar uma varredura, o usuário deve inserir a URL base do alvo ou editar o escopo, maneira na qual o usuário insere uma lista de URLs alvo.

Posteriormente, é necessário configurar os módulos que serão habilitados no *scan*. O Vega suporta dois tipos de módulos: os módulos básicos, que são executados em cada nó e os módulos processadores de respostas, que são executados em todas as respostas recebidas do servidor. Após a seleção do módulo que adicionará funcionalidades ao *scan*, apresentam-se as opções relacionadas à autenticação e *cookies*. A primeira torna possível a execução de *scans* autenticados e a segunda possibilita que sejam inseridos *cookies* que

serão enviados em todas as requisições do *scanner*. Com as configurações estabelecidas, o próximo passo do auditor é executar o *scan*.

A primeira ação tomada pela ferramenta, ao começar a varredura, é disparar um *crawling* na aplicação *Web* alvo, enviando diversas requisições. Isto ocorre porque, além de analisar o conteúdo da página, o mecanismo de *crawling* realiza diversos testes em potenciais caminhos desta página, para determinar se são arquivos ou diretórios.

O comportamento do *crawling* realizado pela ferramenta é dinâmico, ou seja, a quantidade de *links* que sofrem o processo de *crawling* irá aumentar à medida que novos *links* forem descobertos.

Os *scans* do Vega também comparam uma página com outra, para assim aprimorar sua qualidade na detecção de páginas de error que não têm como retorno respostas HTTP 4XX.

Com o *crawling* e teste de vulnerabilidades terminados em todo o escopo do teste, o Vega apresenta o resultado do *scan*. São listados os tipos de alertas e seus níveis de severidade, do maior para o menor, e qual nó cada um representa.

Por fim, cabe destacar uma característica dos alertas. Eles gravam o par requisição-resposta para cada nó dado como vulnerável. Assim, é possível ao auditor compreender melhor o comportamento da aplicação e a causa de sua vulnerabilidade [Subgraph 2014].

4.3 Testes e resultados

Os resultados obtidos na avaliação de vulnerabilidade dos *scanners* na aplicação alvo, a bWAPP, não se limitou somente aos dez riscos mais críticos de segurança. Entretanto, tendo em vista o objetivo deste estudo, a análise dos dados resultantes dos testes realizados utilizando-se as ferramentas Arachini, Nessus e Vega, está centrada nas vulnerabilidades que integram as categorias descritas pelo OWASP Top Ten. Devido a configuração da máquina possuir, intencionalmente, falhas em inúmeros pontos, algumas vulnerabilidades encontradas pelas ferramentas serão propositalmente ignoradas por este trabalho, como é o caso da propensão a ataques de *Click-Jacking* (vulnerabilidade de falta do cabeçalho *X-Frame-Option* nas respostas HTTP) detectada pelo Nessus.

Para um melhor entendimento dos resultados, os relatórios dos três *scanners* de vulnerabilidade se encontram no Apêndice deste trabalho e as tabelas com as

vulnerabilidades presentes na bWAPP, junto com as detecções feitas pelas ferramentas, estão ao longo do texto.

A categoria A1 da lista reúne diversos tipos de injeção de código. A bWAPP possui algumas vulnerabilidades que se enquadram nesta categoria, como é o caso de SQL *Injection*, LDAP *Injection*, OS *Command Injection*, HTML *Injection* e XML *Injection*. Devido à presença destas diversas formas de injeção, espera-se que as ferramentas consigam detectar algum destes riscos dentro da aplicação. Dentre estas, será destacada a vulnerabilidade SQL *Injection*, pelo fato de ser mais comum e conhecida. Ela trata do envio de códigos maliciosos, na linguagem de banco de dados SQL, para qualquer área da aplicação que receba e trate os dados requisitados pelo usuário, como observado na figura 8 abaixo. Nenhum outro tipo de injeção, além da de código SQL, foi encontrada pelas ferramentas. Além disso, o Vega, foi o único a detectar uma ocorrência desta vulnerabilidade na página esperada da aplicação, como pode ser visto na tabela 1 abaixo.

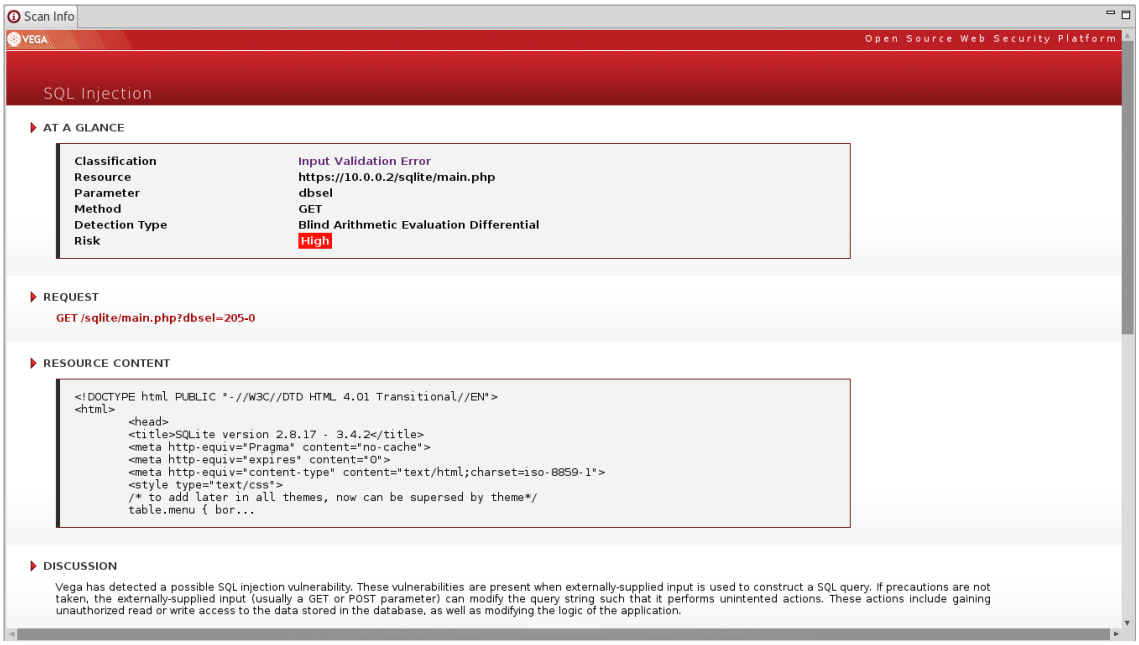


Figura 8. Detecção da vulnerabilidade SQL Injetction pelo Vega

Fonte: Vega

Tabela 1. Vulnerabilidades da bWAPP pertencentes a categoria A1 do Top Ten

	HTML Injection	LDAP Injection	OS Command Injection	SQL Injection	XML Injection
Arachini	-	-	-	-	-
Nessus	-	-	-	-	-
Vega	-	-	-	X	-

A segunda categoria, A2, refere-se aos riscos que afetam as quebras de autenticação e gerenciamento de uma sessão ativa pela aplicação *Web*. Diferentemente da citada anteriormente, esta categoria agrupa vulnerabilidades distintas e com diferentes formas de detecção. A máquina alvo é programada, intencionalmente, com as seguintes vulnerabilidades: formulários de *login* inseguros; gerenciamento de finalização de sessão; ataques a senha; políticas fracas de senhas; portais administrativos de acesso público; atributo de segurança de *cookies*; e URLs contendo ID de sessão. Por ser a vulnerabilidade que apresentou maior detecção entre as demais, será destacada a de atributos de segurança de *cookies*, como apresentado na figura 9. Ela se refere à falta de pelo menos um dos parâmetros presentes nos *cookies* que contêm informações da sessão - HTTP *Only* e *Secure*. Os *cookies*, ao serem enviados da aplicação para o usuário, devem sempre conter esses dois atributos que prezem por sua segurança enquanto estiver em tráfego. O primeiro tem a função de impedir que ele seja acessado por *scripts* do lado do usuário, como o JavaScript, protegendo-lhe de ser roubado por ataques como XSS. Já o outro instrui ao navegador a somente enviar o *cookie* se a requisição estiver sendo feita por um canal encriptado, como o HTTPS, evitando que este seja interceptado por um atacante que tenha acesso ao tráfego. Dentre as três ferramentas, o Nessus encontrou, em 12 páginas diferentes, *cookies* suscetíveis a esse risco, e o Vega em apenas duas, como apresentado na tabela 2. Somente o Arachini não detectou a presença desta vulnerabilidade.

85601 (6) - Web Application Cookies Not Marked HttpOnly

Synopsis

HTTP session cookies might be vulnerable to cross-site scripting attacks.

Description

The remote web application sets various cookies throughout a user's unauthenticated and authenticated session. However, one or more of those cookies are not marked 'HttpOnly', meaning that a malicious client-side script, such as JavaScript, could read them. The HttpOnly flag is a security mechanism to protect against cross-site scripting attacks, which was proposed by Microsoft and initially implemented in Internet Explorer. All modern browsers now support it.

Note that this plugin detects all general cookies missing the HttpOnly cookie flag, whereas plugin 48432 (Web Application Session Cookies Not Marked HttpOnly) will only detect session cookies from an authenticated session missing the HttpOnly cookie flag.

See Also

https://www.owasp.org/index.php/HttpOnly

Solution

Each cookie should be carefully reviewed to determine if it contains sensitive data or is relied upon for a security decision.

If possible, add the 'HttpOnly' attribute to all session cookies and any cookies containing sensitive data.

Risk Factor

None

References

XREF	CWE:20
XREF	CWE:74

85602 (6) - Web Application Cookies Not Marked Secure

Synopsis

HTTP session cookies might be transmitted in cleartext.

Description

The remote web application sets various cookies throughout a user's unauthenticated and authenticated session. However, there are instances where the application is running over unencrypted HTTP or the cookies are not marked 'secure', meaning the browser could send them back over an unencrypted link under certain circumstances. As a result, it may be possible for a remote attacker to intercept these cookies.

Note that this plugin detects all general cookies missing the 'secure'
cookie flag, whereas plugin 49218 (Web Application Session Cookies Not Marked Secure) will only detect session cookies from an authenticated session missing the secure cookie flag.

See Also

https://www.owasp.org/index.php/SecureFlag

Solution

Each cookie should be carefully reviewed to determine if it contains sensitive data or is relied upon for a security decision.

If possible, ensure all communication occurs over an encrypted channel and add the 'secure' attribute to all session cookies or any cookies containing sensitive data.

Risk Factor

None

References

XREF	CWE:522
XREF	CWE:718
XREF	CWE:724
XREF	CWE:928
XREF	CWE:930

Plugin Information:

Publication date: 2015/08/24, Modification date: 2015/08/24

Hosts

10.0.0.2 (tcp/80)

Figura 9. Detecção da falta dos atributos de segurança nos Cookies pelo Nessus

Fonte: Nessus

Tabela 2. Vulnerabilidades da bWAPP pertencentes a categoria A2 do Top Ten

	Formulários de Login Inseguros	Gerenciamento de Finalização de Sessão	Ataques a Senhas	Políticas Fracas de Senha	Portais Administrativos de Acesso Público	Atributos de Segurança dos Cookies	URL Contendo ID de Sessão
Arachini	-	-	-	-	X	-	-
Nessus	-	-	-	X	-	X	-
Vega	X	-	-	-	-	X	-

Dando continuidade à lista dos riscos da OWASP Top Ten, a categoria A3 aborda o *Cross-Site Scripting* - XSS. De maneira semelhante à categoria das Injeções, as vulnerabilidades descritas compartilham pontos em comum. Neste caso, códigos JavaScript maliciosos. A máquina bWAPP possui duas vulnerabilidades de *Cross-Site Scripting*, o XSS refletido e o XSS armazenado. Os primeiros, e também mais comuns, são, em sua grande maioria, *links* maliciosos manipulados pelo atacante, como apresentado na figura 10. Estes, devido ao código malicioso injetado, quando acessados pela vítima, instruem seu navegador a visitar o *site* vulnerável que irá refletir o ataque para o navegador do usuário. Todas as ferramentas testadas, como evidenciadas na tabela 3, obtiveram sucesso na detecção do XSS refletido. Já, na identificação do XSS armazenado, nenhuma foi eficiente.

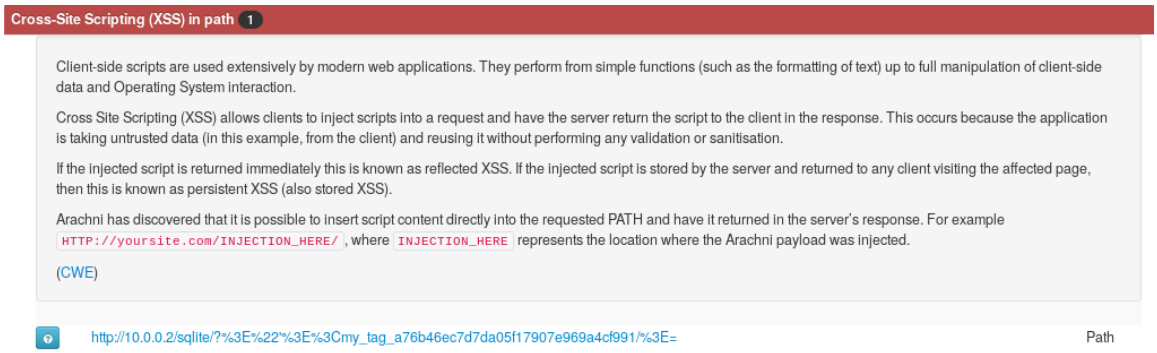


Figura 10. Identificação da vulnerabilidade Cross-Site Scripting pelo Arachini

Fonte: Arachini

Tabela 3. Vulnerabilidades da bWAPP pertencentes a categoria A3 do Top Ten

	A3 - Cross-Site Scripting	
	XSS Refletido	XSS Armazenado
Arachini	X	-
Nessus	X	-
Vega	X	-

O próximo item tratado são as referências inseguras e diretas a objetos. A categoria A4 não possui nenhuma vulnerabilidade especificada pela bWAPP porque

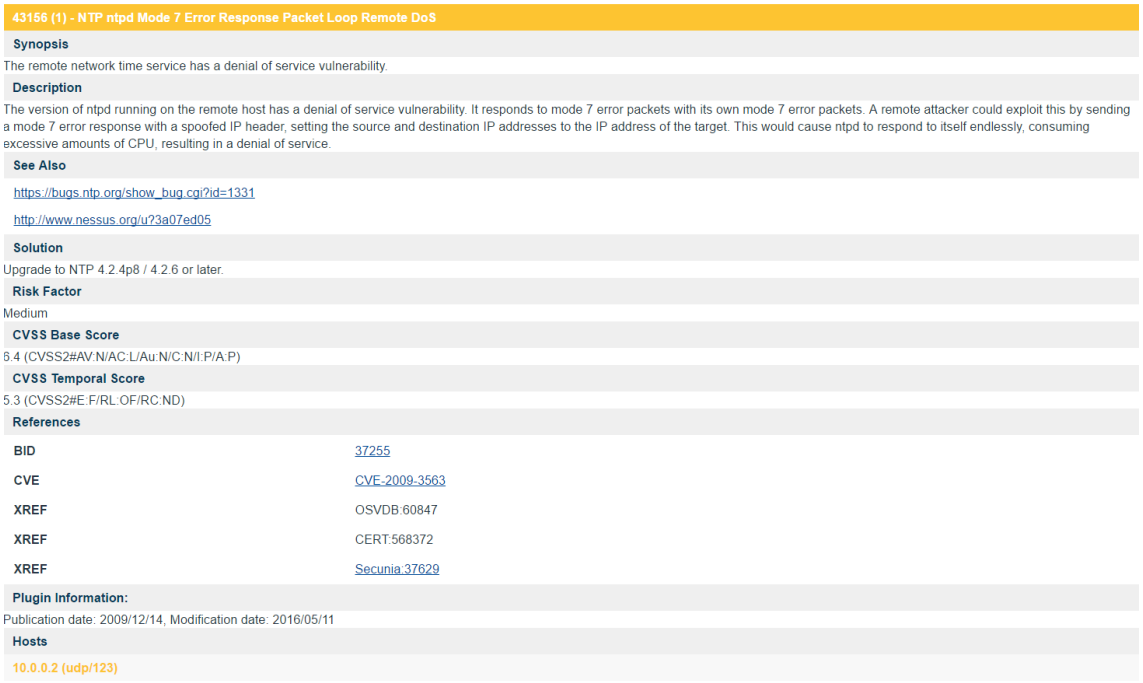
qualquer objeto interno à aplicação que seja de acesso controlado pode ser considerado propício a este ataque. Um bom exemplo para contextualizar esta vulnerabilidade se encontra nos comércios *online*. Ao concluir uma compra em um *site*, é comum que a aplicação envie ao usuário um *email* contendo um *link* que, quando clicado pelo comprador, apresenta a nota fiscal referente à sua compra. Quando a aplicação está vulnerável a referências inseguras e diretas a objetos, um usuário malicioso, que possua conhecimento sobre o endereço do objeto no qual deseja acessar, consegue acessá-lo ao digitar a referência direta deste objeto em seu navegador. Devido à condição de se conhecer previamente ao ataque o objeto alvo, nenhuma ferramenta obteve resultado positivo nessa detecção, como mostra a tabela 4. Para realizar este tipo de ataque, é necessário a interferência de um ser humano.

Tabela 4. Vulnerabilidades da bWAPP pertencentes a categoria A4 do Top Ten

	A4 - Referências Inseguras e Diretas a Objetos
Arachini	-
Nessus	-
Vega	-

A categoria A5 da lista apresenta as principais configurações incorretas de segurança. Tratam-se de serviços ou artefatos que, devido a algum problema em sua configuração, acabam por deixar parte da aplicação comprometida. Dentre as vulnerabilidades destacadas pela bWAPP para esta sessão, encontram-se: acesso a arquivos através do protocolo Samba; negação de serviço (DoS); *Cross-Site Tracing* (XST); configurações frágeis em protocolos FTP, SMB e SNMP; ataques *Man-in-the-Middle*; e arquivos de *backup*, antigos ou sem referência. Dos riscos citados, vale ressaltar o DoS. O ataque de negação de serviço tem como objetivo tornar algum recurso da aplicação indisponível ao seu propósito inicial. Somente o Nessus foi capaz de identificar este problema, encontrando três pontos vulneráveis. Um dos possíveis cenários causadores de indisponibilidade de serviço na aplicação, o *loop* do pacote de resposta de

erro *Mode*7 do NTP, encontra-se detalhado na figura 11. As outras ferramentas não obtiveram sucesso na detecção, como é apresentado na tabela 5.



43156 (1) - NTP ntpd Mode 7 Error Response Packet Loop Remote DoS

Synopsis

The remote network time service has a denial of service vulnerability.

Description

The version of ntpd running on the remote host has a denial of service vulnerability. It responds to mode 7 error packets with its own mode 7 error packets. A remote attacker could exploit this by sending a mode 7 error response with a spoofed IP header, setting the source and destination IP addresses to the IP address of the target. This would cause ntpd to respond to itself endlessly, consuming excessive amounts of CPU, resulting in a denial of service.

See Also

https://bugs.ntp.org/show_bug.cgi?id=1331

http://www.nessus.org/u?3a07ed05

Solution

Upgrade to NTP 4.2.4p8 / 4.2.6 or later.

Risk Factor

Medium

CVSS Base Score

6.4 (CVSS2#AV:N/AC:L/Au:N/C:N/I:P/A:P)

CVSS Temporal Score

5.3 (CVSS2#E:F/RL:OF/RC:ND)

References

BID	37255
CVE	CVE-2009-3563
XREF	OSVDB:60847
XREF	CERT:568372
XREF	Secunia:37629

Plugin Information:

Publication date: 2009/12/14, Modification date: 2016/05/11

Hosts

10.0.0.2 (udp/123)

Figura 11. Vulnerabilidade de negação de serviço (DoS) detectada pelo Nessus

Fonte: Nessus

Tabela 5. Vulnerabilidades da bWAPP pertencentes a categoria A5 do Top Ten

	A5 - Configuração Incorreta de Segurança					
	Acesso a Arquivos Através do Protocolo Samba	Negação de Serviço (DoS)	Cross-Site Tracing (XST)	Configurações Frágeis em Protocolos (FTP, SMB, SNMP)	Ataques Man-in-the-Middle	Arquivos de Backup, Antigos ou sem Referências
Arachini	-	-	X	-	-	X
Nessus	X	X	X	X	X	-
Vega	-	-	X	-	-	X

A sexta categoria representa a exposição de dados sensíveis. Esta categoria agrupa qualquer configuração frágil que, quando atacada, reflete na captura de dados sensíveis da aplicação para o atacante. Ela é composta por vulnerabilidades como ataques a cifras (utilização de cifras sujeitas à ataques de criptoanálise), credenciais trafegadas por canais não criptografados, POODLE (*Man-in-the-Middle* baseado no protocolo SSL 3.0), utilização de protocolos obsoletos (SSL 2.0), credenciais armazenadas em texto claro, entre outras. Uma vulnerabilidade encontrada na bWAPP por duas ferramentas foi a que

apontou credenciais trafegadas em canais não criptografados. Esta vulnerabilidade consiste em um formulário de *login* da aplicação que é trafegado via HTTP. Desta maneira, qualquer usuário que esteja controlando o tráfego da rede e tenha interceptado este pacote, é capaz de identificar as credenciais utilizadas, devido à falta de criptografia no conteúdo do pacote. Esta evidência está retratada na figura 12. As ferramentas que a detectaram foram o Nessus e o Vega. As vulnerabilidades encontradas e as detecções de cada ferramenta encontram-se na tabela 6.

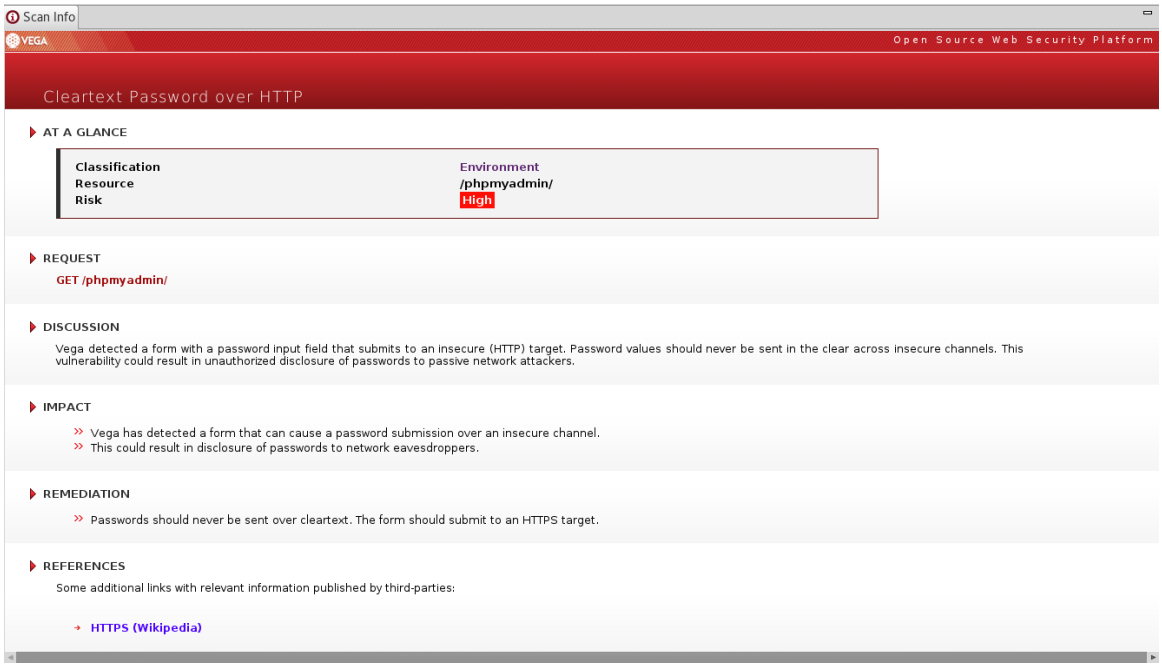


Figura 12. Achado de credenciais de login trafegadas via HTTP pelo Vega

Fonte: Vega

Tabela 6. Vulnerabilidades da bWAPP pertencentes a categoria A6 do Top Ten

	A6 - Exposição de Dados Sensíveis				
	Ataques a Cifras	Credenciais Trafegadas por Canais Não Criptografados (HTTP)	POODLE (Man-in-the-Middle em SSL 3.0)	Utilização de Protocolo SSL 2.0	Contas Armazenadas em Arquivos de Texto
Arachini	-	-	-	-	-
Nessus	X	X	X	X	-
Vega	X	X	X	X	-

Após a categoria de exposição de dados sensíveis, encontra-se a de falta de função para controle do nível de acesso. Esta categoria é composta por vulnerabilidades que

permitem a um usuário não autorizado, devido à falta de uma função na aplicação que controle o acesso e permissões impostos a ele, acessar ou modificar arquivos ou pastas restritas. As três vulnerabilidades encontradas na bWAPP e que pertencem a esta categoria são: acesso a pastas críticas (utilizando *Directory Traversal*); inclusão local e remota de arquivos (LFI / RFI); e acesso a diretórios restritos. Vale destacar, destas, o acesso a pastas críticas utilizando *Directory Traversal*, problema mostrado na figura 13. O ataque de diretório ou caminho transverso objetiva acessar arquivos ou pastas que estão armazenados fora da pasta raiz da aplicação *Web*. Ele é feito através da manipulação de sequências de comandos Linux voltados para navegação em diretórios (../), ou utilizando caminhos absolutos de arquivos. Ao obter sucesso na sequência de comandos utilizados, pode-se acessar arquivos e diretórios inesperados pela aplicação. As ferramentas Nessus e Vega detectaram esta vulnerabilidade, como mostrado na tabela 7.

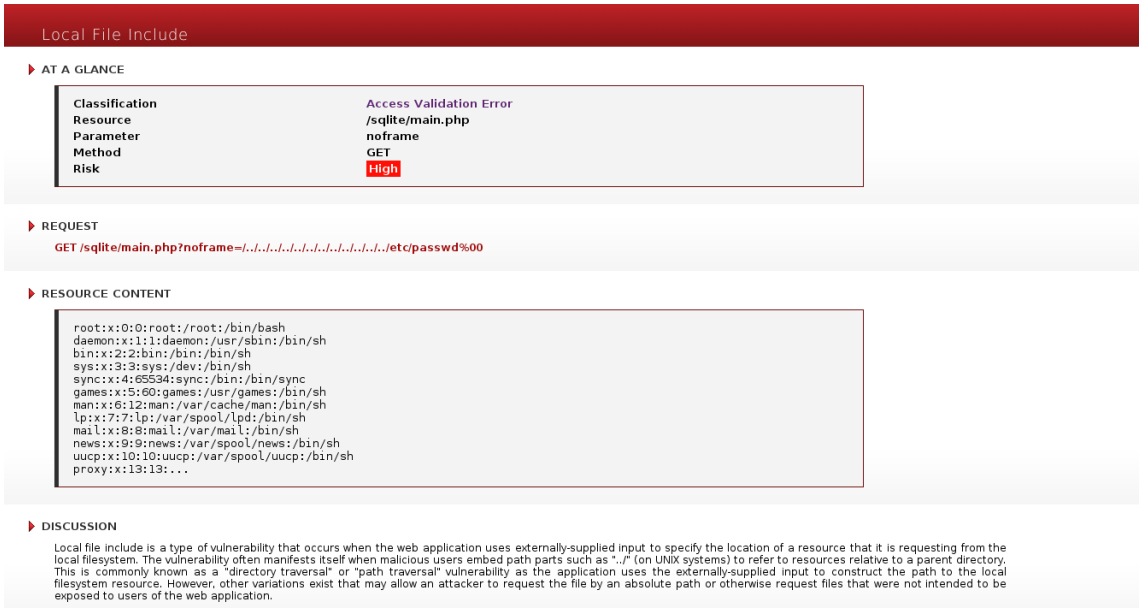


Figura 13. Detecção da vulnerabilidade de Directory Traversal pelo Vega

Fonte: Vega

Tabela 7. Vulnerabilidades da bWAPP pertencentes a categoria A7 do Top Ten

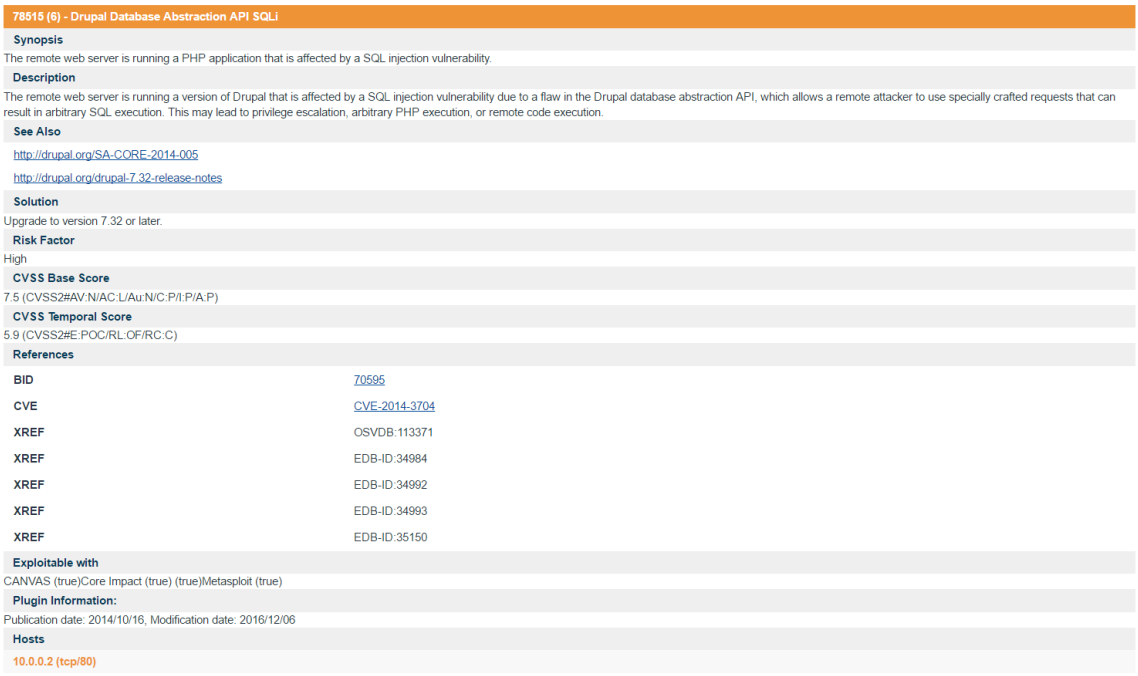
	A7 - Falta de Função para Controle do Nível de Acesso		
	Acesso a Pastas Críticas (Directory Traversal)	Inclusão Local e Remota de Arquivos (RFI/LFI)	Acesso a Diretórios Restritos
Arachini	-	-	X
Nessus	X	-	X
Vega	X	X	X

O oitavo item descrito na lista dos dez riscos mais críticos destacados pelo OWASP para as aplicações *Web* é o *Cross-Site Request Forgery* - CSRF. Devido ao CSRF se tratar diretamente de um ataque, e não uma categoria em si, a vulnerabilidade é o próprio *Cross-Site Request Forgery*. Este ataque consiste em fazer com que uma vítima, que está autenticada em determinado sistema, execute ações indesejadas dentro dele. Não é focado no roubo de dados, mas sim em requisições de troca de estado dessa aplicação, uma vez que o atacante não é capaz de ver a resposta dada por ela. Um exemplo cabível é no cenário de *internet banking*. O usuário encontra-se autenticado em sua conta dentro do *site* de seu banco e o atacante manipula um *link* que contém uma requisição para este *site*. O *link*, ao ser acessado, emite uma requisição para que seja transferida certa quantidade de dinheiro da conta da vítima para a do usuário malicioso. Com a ajuda da engenharia social, o atacante a engana para que ela acesse a URL manipulada. A partir do momento em que a vítima recebe o *link* e o acessa, automaticamente a aplicação executará seus comandos, fazendo com que ela transfira dinheiro sem seu consentimento. Devido ao fato deste ataque necessitar a interação do alvo, nenhuma ferramenta foi capaz de detectá-lo, como mostra a tabela 8.

Tabela 8. Vulnerabilidades da bWAPP pertencentes a categoria A8 do Top Ten

	A8 - Cross-Site Request Forgery (CSRF)
Arachini	-
Nessus	-
Vega	-

A penúltima categoria, A9, é a utilização de componentes com vulnerabilidades conhecidas. Ela é composta por serviços ou componentes que, além de desatualizados, têm suas vulnerabilidades e formas de exploração conhecida por grande parte dos atacantes. A bWAPP possui cinco destes componentes, como o *Buffer-Overflow* no Samba, SQL *Injection no Drupal*, o *Heartbleed*, o XSS na *BBCode Tags* do *phpMyAdmin* e *SQLiteManager* vulnerável a XSS, injeção e LFI, evidenciados na figura 14. Cabe destacar a injeção de código SQL no banco de dados Drupal. A versão utilizada por ele é vulnerável a SQL *Injection* devido a uma falha na API de abstração do banco de dados do Drupal, que permite a um usuário malicioso explorá-la através de consultas SQL. Este ataque, se for bem sucedido, pode levar à elevação de privilégios, execução de código PHP arbitrário, ou execução remota de códigos. Apenas o Nessus foi capaz de detectar este risco na aplicação, como mostrado na tabela 9.



78515 (6) - Drupal Database Abstraction API SQLi

Synopsis

The remote web server is running a PHP application that is affected by a SQL injection vulnerability.

Description

The remote web server is running a version of Drupal that is affected by a SQL injection vulnerability due to a flaw in the Drupal database abstraction API, which allows a remote attacker to use specially crafted requests that can result in arbitrary SQL execution. This may lead to privilege escalation, arbitrary PHP execution, or remote code execution.

See Also

http://drupal.org/SA-CORE-2014-005

http://drupal.org/drupal-7.32-release-notes

Solution

Upgrade to version 7.32 or later.

Risk Factor

High

CVSS Base Score

7.5 (CVSS2#AV:N/AC:L/Au:N/C:P/I:P/A:P)

CVSS Temporal Score

5.9 (CVSS2#E:POC/RL:OF/RC:C)

References

BID	70595
CVE	CVE-2014-3704
XREF	OSVDB:113371
XREF	EDB-ID:34984
XREF	EDB-ID:34992
XREF	EDB-ID:34993
XREF	EDB-ID:35150

Exploitable with

CANVAS (true)Core Impact (true) (true)Metasploit (true)

Plugin Information:

Publication date: 2014/10/16, Modification date: 2016/12/06

Hosts

10.0.0.2 (tcp/80)

Figura 14. Identificação da vulnerabilidade SQL Injection no Drupal pelo Nessus

Fonte: Nessus

Tabela 9. Vulnerabilidades da bWAPP pertencentes a categoria A9 do Top Ten

	A9 - Utilização de Componentes com Vulnerabildiades Conhecidas				
	Buffer-Overflow (Samba)	SQL Injection (Drupal)	Heartbleed	phpMyAdmin BBCode Tag (XSS)	SQLiteManager (XSS / Injection / LFI)
Arachini	-	-	-	-	-
Nessus	X	X	X	X	X
Vega	-	-	-	-	-

A última categoria da lista são os redirecionamentos e encaminhamentos inválidos. A A10 não possui nenhuma vulnerabilidade explícita definida, assim, qualquer área da aplicação que possua atributos (tag href – HTML) de redirecionamento de página, pode sinalizar um potencial risco. Atacantes podem se beneficiar desta vulnerabilidade ao inserir neste campo uma URL manipulada por eles que (configurando uma URL *Injection*), com a utilização de engenharia social, faça que um usuário legítimo a acesse. Ao acessar a página que contém a URL maliciosa como redirecionamento e enviar uma solicitação para a aplicação, esta fará com que ele siga a URL e fique exposto ao conteúdo inserido pelo atacante, como apresentado na figura 15. As ferramentas Nessus e Vega foram capazes de identificar esse risco, como apresentado na tabela 10.

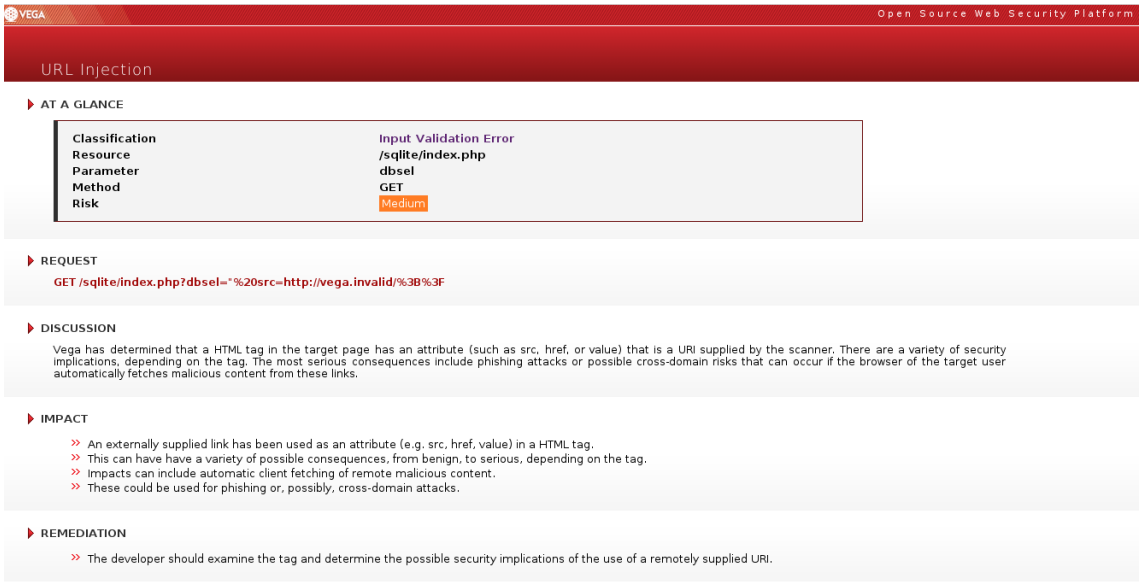


Figura 15. Identificação de área suscetível à injeção de URL pelo Vega

Fonte: Vega

Tabela 10. Vulnerabilidades da bWAPP pertencentes a categoria A10 do Top Ten

	A10 - Redirecionamentos e Encaminhamentos Inválidos
Arachini	-
Nessus	X
Vega	X

Para a avaliação da eficácia dessas ferramentas na detecção dos 10 maiores riscos do OWASP *Top Ten* foi utilizado o teste do Qui-Quadrado, conforme ilustrado na figura 16 a seguir.

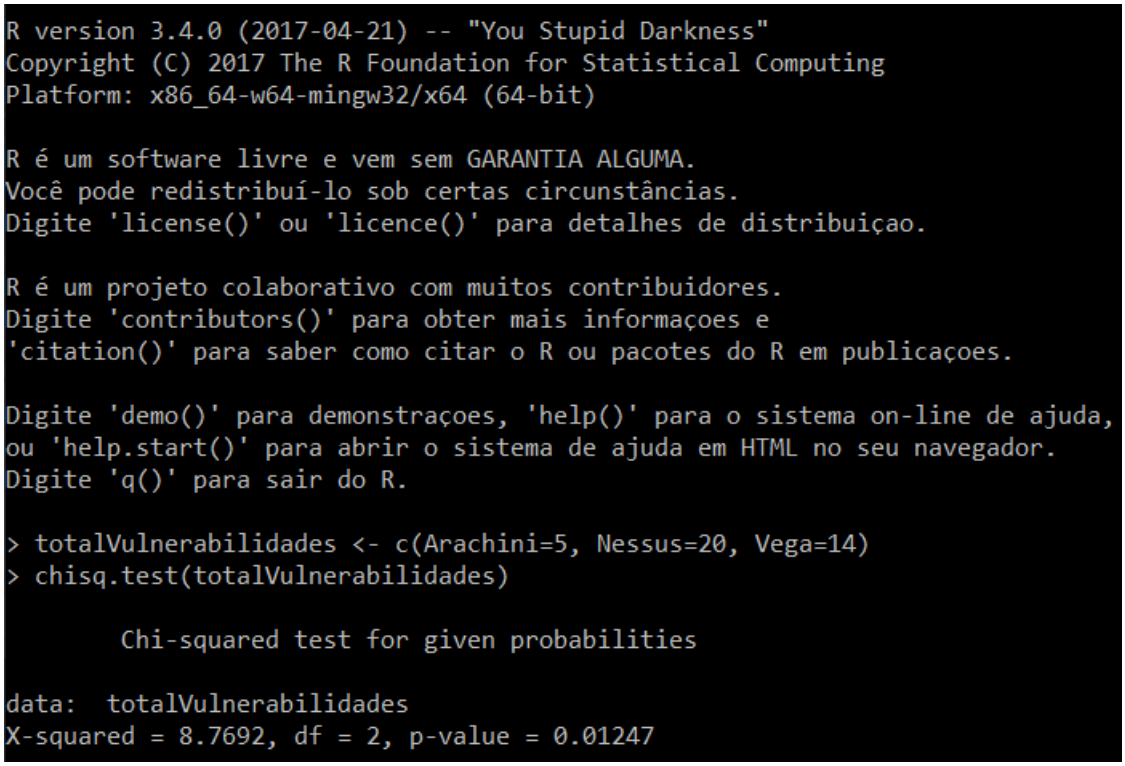


Figura 16. Resultado do Teste Qui-Quadrado no software R

Fonte: Software R

O valor obtido, X^2 = 8,76 (df=2, p-value=0,01), indica que os desvios apresentados entre as ferramentas foram significativos, e não ao acaso. Então, neste estudo, o Nessus mostrou-se como a ferramenta capaz de detectar mais vulnerabilidades seguido, em ordem decrescente, pelo Vega e pelo Arachini.

Considero que sua eficácia deva-se ao fato de ser uma ferramenta que dispõe de mais funcionalidades que as concorrentes avaliadas. O Nessus dispõe de mais modos de escaneamento (como um *scan* de vulnerabilidades para PCI DSS), mais recursos externos integrados ao seu *scan* (o Hydra, por exemplo, para a realização de testes de força bruta), entre outras.

Apesar deste estudo ter sido desenvolvido com duas ferramentas públicas e uma de mercado, este critério de categorização não foi considerado para fins de análise. Esta restringiu-se a verificar a eficácia das ferramentas na detecção dos 10 riscos mais comuns destacados na lista da OWASP.

5 Conclusão

As ferramentas Arachini, Nessus e Vega situam-se entre os diversos *scanners* utilizados para auditar aplicações *Web*. Destacando-se os dez riscos mais críticos de segurança levantados pelo OWASP, foi visto neste estudo que especialmente o Nessus destaca-se como um *scanner* eficaz para encontrar as vulnerabilidades que pertencem às categorias do OWASP *Top Ten*.

Considerando-se que o Arachini e o Vega são ferramentas públicas e o Nessus é de mercado, sugere-se que outros estudos sejam desenvolvidos com o objetivo de esclarecer se há diferenças significativas entre soluções *open source* e pagas quanto à eficácia na detecção de vulnerabilidades.

Recomenda-se, também, que sejam conduzidas pesquisas que abordem aspectos qualitativos relacionados à avaliação de *scanners*, tal como a questão levantada a seguir. Levando-se em conta os escores impostos pelo CVSS, uma ferramenta que detecte apenas uma vulnerabilidade, mas de nível crítico, poderia ser considerada como de melhor eficácia do que outra que, auditando o mesmo escopo, encontre diversas vulnerabilidades de menores riscos à aplicação?

Mas as recomendações não devem se restringir ao campo de pesquisas. Sabe-se que com o contínuo crescimento da *Web* e com as novas tecnologias nela presente, como a *Internet* das Coisas (IOT), as investigações sobre ferramentas de vulnerabilidades e seus aprimoramentos são indispensáveis. Entretanto, igualmente importante, é a segurança de ativos. Afinal, a invasão de máquinas é facilitada quando há falta ou baixo investimento em segurança digital.

Referências Bibliográficas

Acunetix. (2017). “Website security”, https://www.acunetix.com/Websitesecurity/Web-applications, May.

Arachini. (2010). “Arachni Web Application Security Scanner Framework”, http://www.arachni-scanner.com/, May.

Baan, S. and Chesney, B. (2017). “OWASP Developer Guide Reboot”, https://github.com/OWASP/DevGuide, May.

Booth, H., Witte, G. and Rike, D. (2013). “ITL Bulletin – The National Vulnerability Database (NVD): Overview”, https://www.hsdl.org/?abstract&did=797329, May.

Brigjaj, J.D. (2016). “What’s New in CVSS Version 3”, https://www.acunetix.com/blog/articles/whats-new-in-cvss-version-3/, May.

Bryant, M. (1999). “Web 101 for Dummies or Everything You Wanted to Know About the Web But Were Afraid to Ask!”, https://net.educause.edu/ir/library/html/cmr9939/cmr9939.html, April.

CERN. (2017). “CERN Accelerating Science”, https://home.cern/, May.

CERN. (2017). “Tim Berners-Lee submits a proposal for a distributed information system at CERN”, https://timeline.Web.cern.ch/timelines/The-birth-of-the-World-Wide-Web/overlay, March.

Cima, S. (2001). “Vulnerability Assessment”, https://www.sans.org/reading-room/whitepapers/basics/vulnerability-assessment-421, July.

Cockburn, C. and Wilson, T. D. (1996). Business Use of the World-WideWeb. *International Journal of Information Management*, 16(2):83-102.

Curphey, M and Araujo, R. (2006). Web Application Security Assessment Tools. *IEEE SECURITY & PRIVACY*, 4(4):32-41.

CVE. (2017). “The Standard for Information Security Vulnerability Names”, https://cve.mitre.org/about/, May.

Dwoskin, E. and Adam, K. (2017). “More than 150 countries affected by massive cyberattack : Europol says”, https://www.washingtonpost.com/business/economy/more-than-150-countries-affected-by-massive-cyberattack-europolsays/2017/05/14/

5091465e-3899-11e7-9e48-c4f199710b69_story.html?utm_term=.6cca249a09a7, May.

Fong E. and Okun, V. (2007). "Web application scanners: definitions and functions", *Proceedings of the 40th annual Hawaii International conference on system sciences*, p.280b.

Forouzan, B. A. (2007). "WWW and HTTP", Data Communications and Networking, New York, McGraw-Hill Companies, 4th ed, p. 851.

Github. (2017). "The OWASP guide", https://github.com/OWASP/DevGuide, May.

Houghton, K. (2003). "Vulnerabilities & Vulnerability Scanning", https://www.sans.org/reading-room/whitepapers/threats/vulnerabilities-vulnerability-scanning-1195, May.

Infosec Institute. (2017). "CIA Triad", http://resources.infosecinstitute.com/cia-triad/#gref, May.

Kurose, J.F. and Ross, K.W. (2013). "Computer Networks and the Internet", in Computer Networking : a Top-Down Approach, New Jersey, Pearson Education, Inc., 6th ed, p.1-80.

Mesellem, M. (2014). "What is Bwapp? Introducing an Extremely Buggy Web Application",http://www.mmebvba.com/sites/default/files/downloads/bWAPP_intro.pdf, May.

Newton, A. (2016). "Five of the Biggest Cyber-Attacks in History", https://superfast-it.com/five-biggest-cyber-attacks-history/, May.

OWASP. (2016). "Industry: Citations", https://www.owasp.org/index.php/Industry:Citations, May.

OWASP. (2017). "OWASP Code Review Guide", https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project, May.

OWASP. (2017). "OWASP Project", https://www.owasp.org/index.php/Category:OWASP_Project, May.

OWASP. (2017). "OWASP Testing Guide v4", https://www.owasp.org/index.php/OWASP_Testing_Project, May.

OWASP. (2017). "OWASP Top Ten Cheat Sheet", https://www.owasp.org/index.php/

OWASP_Top_Ten_Cheat_Sheet, May.

OWASP. (2017). "OWASP Top Ten Project", https://www.owasp.org/index.php/Category: OWASP_Top_Ten_Project, May.

Palmaers, T. (2013). "Implementing a Vulnerability Management Process", https://www.sans.org/reading-room/whitepapers/threats/implementing-vulnerability-management-process-34180, May.

PCI. (2017). "PCI Security", https://www.pcisecuritystandards.org/pci_security/, May.

Rouse, M. (2008). "Principle of Least Privilege", http://searchsecurity.techtarget.com/definition/principle-of-least-privilege-POLP, May.

Smith, B. (2014). "A Quick Guide to GPLv3", https://www.gnu.org/licenses/quick-guide-gplv3.html, May.

Subgraph. (2014). "VEGA Vulnerability Scanner", https://subgraph.com/vega/, May.

Sullivan, B. (2008). "SDL and the OWASP Top 10", https://blogs.microsoft.com/microsoftsecure/2008/05/01/sdl-and-the-owasp-top-ten/, May.

Tenable Network Security. (2016). "Nessus 6.8 User Guide", http://static.tenable.com/prod_docs/Nessus_6.8_User_Guide.pdf, May.

The Associated Press. (2017). "10 Things to know on Monday", https://www.nytimes.com/aponline/2017/05/14/us/ap-10 - things - to - know - monday .html, May.

Timberlake, B. (2002). "Building a LAMP Server", http://lamphowto.com/, May.

Yeomans, J. (2017). "Revealed: the biggest companies in the world in 2016", http://www.telegraph.co.uk/business/2016/07/20/revealed-the-biggest-companies-in-the-world-in-2016/, May.