UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Lucas Alberto Tomasi

GERAÇÃO DE RELATÓRIO PRELIMINAR PARA ANÁLISE DE VULNERABILIDADES WEB DE FORMA AUTOMATIZADA: NO ESCOPO DO OWASP TOP TEN PROJECT

Florianópolis

2015

Lucas Alberto Tomasi

GERAÇÃO DE RELATÓRIO PRELIMINAR PARA ANÁLISE DE VULNERABILIDADES WEB DE FORMA AUTOMATIZADA: NO ESCOPO DO OWASP TOP TEN PROJECT

Trabalho de Conclusão de Curso submetido ao Curso de Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Jean Everson Martina

Florianópolis

2015

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

A ficha catalográfica é confeccionada pela Biblioteca Central.

Tamanho: 7cm x 12 cm

Fonte: Times New Roman 9,5

Maiores informações em:
http://www.bu.ufsc.br/design/Catalogacao.html

À minha família; ao LabSEC; e aos meus amigos.

AGRADECIMENTOS

Aos meus pais, Sadir e Graça, aos meus irmãos, Fabrício e Juliane, e aos cunhados, Andrelise e Norton, por todo o apoio, paciência e estímulo oferecidos durante toda a minha vida. Que não mediram esforços para que eu tivesse uma vida confortável e corresse atrás de todos os meus sonhos. Muito obrigado!

Ao colegas do LabSEC, que fizeram parte do meu dia-a-dia por quase toda a minha graduação, onde encontrei um ambiente fantástico que deu-me a oportunidade de aprofundar os conhecimentos em segurança da computação. Em especial ao Armindo Guerra e Felipe Werlang que realizaram a seleção de bolsistas em que entrei no laboratório e guiaram meus primeiros passos, tanto dentro, como fora do laboratório.

Ao professor Jean Martina pela sugestão do tema, por acreditar no trabalho e pela paciência pela demora de concluir esse projeto.

À Thaís Idalino por ter me orientado no projeto quando era outro tema. Suas dicas de escrita foram muito importantes!

À Bridge, em especial a equipe do SISMOB pela paciência e apoio nos momentos finais da execução deste trabalho.

Por fim, agradeço a este trabalho pela compreensão por tantas horas em que não o escrevi porque estava dedicado à minha família e aos amigos.

Avalia-se a inteligência de um indivíduo pela quantidade de incertezas que ele é capaz de suportar.

Immanuel Kant

RESUMO

Aplicações web estão cada vez mais presentes na internet e assim também cresce o número de crimes cibernéticos. Nesse cenário, os testes de penetração são essenciais como medida de segurança, identificando as possíveis vulnerabilidades pela ótica de um atacante. No entanto, a maioria das instituições preocupam-se com a sua segurança quando seus sistemas já estão em produção, ou ainda pior, ao se tornarem vítimas de ataques. Pela natureza analítica de um teste de penetração, sua execução pode levar algumas semanas. Assim, este trabalho apresenta uma forma de entregar rapidamente um relatório preliminar sobre a segurança de um sistema. Dessa maneira, a instituição já pode começar a trabalhar para resolver os principais problemas de segurança enquanto os testes manuais e minuciosos são executados.

Palavras-chave: Segurança, Teste de penetração, Vulnerabilidade, OWASP, W3AF, Análise de vulnerabilidade.

ABSTRACT

Web applications are increasingly present on the internet and with it the number of cyber crimes has been growing as well. In this scenario, penetration tests are essential for safety, identifying potential vulnerabilities through the eyes of an attacker. However, most institutions are only concerned about their safety when their systems are already in production mode, or even worse, when they become victims of attacks. Due to its own analytical nature, penetrations tests execution may take few weeks. So, this work presents a way to quickly deliver a preliminary report on the security of a system. Using this tool, the institution can start working to solve the major security issues, while the manual and detailed tests are performed.

Keywords: Security, Penetration test, Vulnerability, OWASP, W3AF, Vulnerability analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

A partir da segunda metade de 1990, com a popularização da internet, as empresas começaram gradualmente a migrar para o novo ambiente virtual. Inicialmente, de forma tímida, elas desfrutavam dos benefícios de digitalizar seus processos e armazenar informações em locais remotos, que poderiam ser acessados de qualquer lugar através da internet.

Com o passar dos anos e com o avanço da tecnologia, facilitando e barateando o acesso à internet, as instituições e pessoas gradativamente migraram as atividades cotidianas para o ambiente eletrônico. Dois exemplos clássicos são o uso do internet *banking* para consultas e pagamentos, e o envio do imposto de renda pela internet, o que facilita a vida dos usuários, que não precisam se deslocar a um estabelecimento físico para realizar tais operações. Recentemente, podemos destacar o aumento da participação das lojas virtuais na economia mundial. Por exemplo, as lojas que vendem produtos pela internet faturaram no Brasil em 2014, 43 bilhões de reais, um crescimento de 14% em relação a 2013 (CARNETTI, 2015).

A alta oferta de aplicações na internet, aliada com valor dos dados que nela trafegam, sejam informações sensíveis ou operações financeiras, seduziram pessoas mal intencionadas a realizar ataques nesses sistemas para obter vantagens ilícitas. Em virtude da preocupação sobre o tema, foram criadas diversas iniciativas com o objetivo de avaliar as questões de segurança de uma aplicação web, como por exemplo, o *Open Web Application Security Project* (OWASP).

Assim, as instituições precisam proteger seus dados e sistemas, já que queixas relacionadas com crimes cibernéticos aumentaram mais que 15 vezes de 2000 até 2013 (IC3, 2013) e o prejuízo estimado com esse tipo de crime no Brasil foi de R$ 15,9 bilhões em 2012 e US$ 338 bilhões no mundo (PATCHER, 2014). Outros números que evidenciam esse problema são apresentados pelo Imperva (2010), em que num intervalo de seis meses, de Julho a Dezembro de 2011, foi registrado, para 40 aplicações web monitoradas, um aumento de 25.000 para 38.000 ataques por hora, ou 10 tentativas a cada segundo.

Nesse contexto, destaca-se o teste de penetração como um método para avaliar a segurança de um sistema computacional. As atividades envolvidas nesse método tentam descobrir e explorar fraquezas dos sistemas, como acessar regiões sem acesso autorizado, vazamento de dados, indisponibilidade do serviço, e etc. Tal diagnóstico é regis-

trado conforme os testes são executados em um relatório e ao comprovar vulnerabilidades, são fornecidas estratégias de mitigação. Esse tipo de teste tenta simular as possibilidades que um atacante real empregaria para quebrar a segurança de um sistema.

1.1 JUSTIFICATIVA

A indústria de segurança da informação tinha como costume enumerar as vulnerabilidades sobre a infraestrutura de redes de computadores. Contudo, com o avanço da tecnologia, fazendo com que a internet ficasse onipresente na vida das instituições e pessoas, o foco do atacante foi significativamente alterado, sendo as falhas em aplicações web o alvo predileto dos atacantes. Segundo Cross (2007), segurança passou a ser um requisito relevante não somente às redes de computadores, mas ao nível de aplicação.

Aplicações web, por sua natureza, costumam ser publicamente acessíveis e assim é também com os servidores que as tornam disponíveis aos usuários legítimos. Em consequência disso, há maiores chances de comprometimento dessa classe de sistemas de computador (GRAVES, 2010).

Infelizmente a maioria das organizações preocupam-se com a segurança quando os sistemas já estão disponíveis em produção, ou ainda mais tarde, quando se tornam vítimas de ataques. Esse comportamento típico é arriscado, pois exige uma rápida resposta para a manutenção dos serviços e para a correção das vulnerabilidades exploradas. Dessa maneira, as organizações carecem de uma análise de segurança o mais rápido possível. No entanto, constatar o nível de segurança de um sistema através de um teste de penetração é uma tarefa onerosa, levando semanas para ser concluída. Apesar do avanço tecnológico no desenvolvimento de ferramentas que realizam testes de segurança de forma automatizada, elas jamais substituirão por completo a análise manual (ALLAN, 2008), pois tais ferramentas podem interpretar os dados equivocadamente, não estando preparadas para as especificidades de cada sistema ou para pertubações externas.

Finalmente, diante do cenário apresentado, incorre-se a necessidade de elaborar um meio para gerar rapidamente um relatório sobre o estado da aplicação. Nesse caso, poderia ser gerado um relatório preliminar verificando o nível de segurança da aplicação, através de ferramentas que realizam testes automatizados, com o mínimo de interferência humana, pois aqui o tempo é o fator primordial. Enquanto

o analista procede a execução dos testes minuciosos, a instituição que requeriu o teste de penetração, em posse do relatório preliminar, pode começar a agir verificando os possíveis vetores de ataque reportados.

1.2 OBJETIVOS

Abaixo serão apresentados os objetivos deste trabalho.

1.2.1 Objetivo geral

O objetivo deste trabalho é realizar a geração de um relatório preliminar do teste de penetração em uma aplicação web de forma automatizada. O fator central do trabalho é utilizar uma ferramenta que executa testes de segurança automatizados, e com o resultado dos testes, gerar um relatório legível com as vulnerabilidades encontradas.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Demonstrar a utilização de uma ferramenta que executa testes automatizados de segurança em aplicações web;

- Desenvolver uma aplicação que seja capaz de traduzir a saída gerada pelos testes em um relatório legível;

1.3 LIMITAÇÕES

No presente trabalho será utilizada a ferramenta *Web Application Security Project* (W3AF) para realizar os testes nas aplicações web, sendo consideradas as vulnerabilidades relacionadas às falhas descritas na última versão do projeto OWASP TOP 10, lançado em 2013. Contudo, é necessário uma amostra de como as vulnerabilidades são reportadas pelo W3AF para o correto *parser*, assim, será utilizada duas aplicações web com falhas destinadas à prática do teste de penetração para levantar essas amostras: *Mutillidae II* e *Damn Vulnerable Web Application* (DVWA).

1.4 MÉTODO DE PESQUISA

A fim de atingir os objetivos propostos neste trabalho, o desenvolvimento deu-se início com o estudo de como utilizar a ferramenta W3AF para que ela abordasse as vulnerabilidades do OWASP TOP 10. Em seguida, foi analisado como os relatórios poderiam ser gerados para facilitar a interpretação pela aplicação que será desenvolvida. Posteriormente foi verificado como gerar os relatórios que serviriam como base para entender a estrutura da descrição das falhas reportadas pelo W3AF.

Após isso, foi implementado uma ferramenta de fácil customização, tanto para adicionar novas vulnerabilidades, como para alterar o *template* do relatório que é gerado.

1.5 ESTRUTURA DO TRABALHO

O presente trabalho é divido em cinco partes, e inicialmente no capítulo 2 são expostos os conceitos teóricos necessários à compreensão da ferramenta que será implementada. São tratados os conceitos de segurança da informação, do teste de penetração e sobre a metodologia OWASP.

O capítulo 3 apresenta as ferramentas utilizadas, sendo elas o W3AF para realizar o teste automatizado, e o DVWA e Mutillidae para aprendermos a manipular a W3AF.

No capítulo 4, por sua vez, são apresentadas as ferramentas e tecnologias utilizadas para desenvolver a aplicação e em seguida descreve como foi a implementação e a motivação das decisões tomadas.

Por fim, o capítulo 5 encerra o trabalho trazendo as considerações finais e a conclusão, fazendo a comparação com os objetivos propostos e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 SEGURANÇA DA INFORMAÇÃO

Uma definição clássica para segurança computacional é que ela permeia três características essenciais: confidencialidade, integridade e disponibilidade (BISHOP, 2004). As interpretações desses três aspectos variam, bem como os contextos em que eles surgem, determinado pelo ambiente em que ela está inserida. De forma geral, confidencialidade assegura que apenas usuários autenticados e autorizados tenham acesso a determinada informação. A integridade por sua vez, refere-se a confiabilidade dos dados, provendo mecanismos para prevenir e detectar modificações sem autorização. Por fim, temos a disponibilidade, que refere-se à capacidade de utilizar a informação ou recurso desejado a qualquer momento.

Neste trabalho será adotada a definição proposta por Stallings (2015), em que ele estende a definição anterior incluindo duas novas propriedades, a autenticidade e o não-repúdio. A autenticidade é descrita como um mecanismo para garantir a identificação das partes envolvidas em uma operação, e o não-repúdio como uma forma para que uma entidade não negue a autoria de uma processo executado por ela.

A segurança de sistemas distribuídos, por exemplo um sistema web, é referida em Belapurkar et al. (2009), como o desafio mais atraente relacionado a essa família de sistemas. A complexidade desta questão está no fato dessas aplicações possuírem diferentes pontos de vulnerabilidade. Tradicionalmente os pontos óbvios que precisam ser protegidos são os nodos de processamento, os clientes e a transmissão por onde ocorre a comunicação entre as partes. Com o crescimento das aplicações distribuídas, sendo concebidas em diferentes camadas heterogêneas, a complexidade de garantir a segurança aumenta consideravelmente.

O *software* é um produto extremamente complexo que abrange inúmeros fatores imprevisíveis e de difícil controle, como mudança nos requisitos e inovações tecnológicas. A complexidade inerente ao processo de desenvolvimento de um *software* faz com que grande parte dos projetos ultrapasse o orçamento e prazos previstos, além de não atender completamente as necessidades do cliente. Da mesma forma que os riscos inerentes ao desenvolvimento de um *software* devem ser mitigados, os riscos relacionados à segurança das aplicações devem ser analisados. Para qualificar e entender melhor a gravidade de uma falha em

um sistema, faz-se necessário tomar emprestado alguns conceitos utilizados no gerenciamento de riscos. NIST (2012) trata exclusivamente do gerenciamento dos riscos e apresenta exaustivamente os conceitos descritos a seguir:

2.1.1 Risco

Risco é uma medida em que uma entidade está ameaçada por um potencial evento ou circunstância e é uma função em relação a dois fatores: (i) os impactos que surgiriam se ocorressem os eventos ou circunstâncias; e (ii) a chance de ocorrência. Existem dois grupos que podem ser classificados como riscos em segurança da informação. O primeiro, são os riscos que surgem com a perda de pelo menos uma das características da segurança da informação: confidencialidade, integridade, disponibilidade, autenticidade e não-repúdio. O segundo, são os riscos que podem gerar impactos adversos nas instituições como manchar a imagem e reputação.

O processo para gerenciar a segurança da informação é fundamental para manter a saúde de uma instituição. Os mecanismos de segurança são implementados para assegurar a infraestrutura tanto das falhas conhecidas, como das desconhecidas, que são causadas pelos altos riscos. Os especialistas em segurança devem identificar os riscos enfrentados, o potencial impacto desses riscos e se o impacto faz com que o risco seja qualificado como uma ameaça (KBAR, 2009). Segundo Schneier (2006), avaliar os riscos de segurança é um processo subjetivo que é afetado por diversas variáveis, sendo essas os requisitos estabelecidos pelo especialista responsável e a capacidade do sistema para ameaças específicas.

2.1.2 Ameaça

Uma ameaça é uma potencial violação de segurança que pode afetar as instituições, ativos e/ou indivíduos através de acesso não autorizado, destruição, divulgação ou modificação de informações, e negação de serviços. A violação não precisa efetivamente ocorrer para que se exista uma ameaça e com a possibilidade que uma violação ocorra, ações devem ser tomadas para proteger-se (BISHOP, 2004). A ocorrência de uma violação é denominada ataque e o seu executor, o atacante.

De acordo com Shirey (2007), uma ameaça pode ser intencional

ou não: a ameaça intencional é quando o ataque é realizado por uma entidade inteligente (por exemplo, um cracker ou uma organização); e a ameaça acidental é devido ao erro humano ou omissão, mal funcionamento de equipamentos ou desastres naturais. Ainda em Shirey (2007), as consequências das ameaças são dividas em 4 categorias: acesso não autorizado à informação; aceitação de dados falsos; interrupção dos serviços; e controle não autorizado de alguma parte de um sistema.

2.1.3 Vulnerabilidade

Vulnerabilidade é uma fraqueza de um sistema que pode ser explorada por uma fonte de ameaça. A maioria das vulnerabilidades estão associadas com a não implantação de controles de segurança (intencionalmente ou não), ou que tenham sido implementados de forma errada ou incompleta. Além disso, as instituições devem ficar atentas, pois ao longo do tempo com as novas funcionalidades, novos ambientes e tecnologias, podem surgir novas vulnerabilidades.

As vulnerabilidades em um sistema podem ocorrer na arquitetura, na implementação, ou ainda, na cerimônia de um processo (SHIREY, 2007). A maioria dos sistemas possuem vulnerabilidades, mas isso não significa que os sistemas são muito inseguros para serem utilizados. Nem toda ameaça resulta em um ataque, e nem todo ataque é bem-sucedido. O sucesso de um ataque depende da vulnerabilidade, da força do ataque e dos mecanismos de defesa. Se o ataque para explorar uma vulnerabilidade for muito difícil para realizar, então a vulnerabilidade pode ser tolerada; caso o benefício que um atacante teria for pequeno, a vulnerabilidade também pode ser tolerada.

2.1.4 Impacto

O impacto é a magnitude de um dano causado quando uma ameaça se concretiza. Segundo Peltier (2005), o nível de impacto pode ser divido em três categorias:

- Alto impacto: Comprometimento dos serviços fundamentais que levam a uma perda significativa, como reputação, ou lucro;
- Médio impacto: Comprometimento de um serviço fundamental com uma perda financeira limitada;
- Baixo impacto: Comprometimento sem prejuízo financeiro.

2.1.5 Ataque

De acordo com Shirey (2007), o ataque é um ato intencional pelo qual uma entidade tenta burlar mecanismos de segurança, ou violar as políticas de um sistema. Eles podem ser divididos em duas categorias, ataques passivos e ativos. Os passivos tem como objetivo capturar informações sem afetar os recursos dos sistemas. Stallings (2015) cita que os ataques passivos têm como finalidade a espionagem ou o monitoramento do tráfego. Além disso, é uma forma de um atacante obter informações sobre uma aplicação alvo, analisá-las e preparar novos ataques no alvo. Já os ataques ativos, podem ter os mesmos objetivos dos passivos, mas alteram os recursos do sistema ou afetam sua operação.

Outra classificação é realizada tendo como eixo a origem do ataque: os ataques internos são cometidos por entidades que estão dentro do perímetro de segurança, como por exemplo, um usuário que possui acesso aos recursos de um sistema, mas não os usa de forma adequada por quem lhe cedeu a autorização; já os ataques externos são iniciados por entidades fora do perímetro de segurança, por exemplo, um usuário não autorizado. Assim, qualquer entidade conectada na internet pode se tornar um atacante externo, desde um adolescente fazendo por diversão, até criminosos internacionais e governos hostis.

2.2 TESTE DE PENETRAÇÃO

Em Shirey (2007), o teste de penetração é descrito como um teste de sistema, que tem como intuito certificá-lo, em que os avaliadores tentam contornar a sua segurança. Já conforme Engebretson (2011), que restringe esse conceito, um teste de penetração pode ser definido como tentativa legal e autorizada para localizar e explorar uma vulnerabilidade, para que tal sistema possa ter sua segurança aprimorada. Desse maneira, a exploração serve como prova de conceito para demonstrar que uma vulnerabilidade é real. O teste de penetração gera como artefato final um documento que descreve as vulnerabilidades encontradas e como corrigí-las, para que os sistemas se tornem seguros a futuros ataques da mesma natureza.

Na prática dos testes de penetração são conhecidos dois personagens distintos (ENGEBRETSON, 2011): o profissional que executa o teste de penetração, denominado como hacker ético ou *white hat hacker*; e o usuário que tenta invadir os sistemas computacionais com motivos maliciosos, nomeado como atacante malicioso, *black hat hacker*, ou *cracker*.

O mesmo autor ressalta que os hackers éticos executam a maioria das atividades e utilizam a maioria das ferramentas que um atacante malicioso. Desta forma, um hacker ético deve se esforçar para agir e pensar como um atacante malicioso. Quanto mais próximo o teste de penetração simula um ataque do mundo real, maior será a relevância dos resultados obtidos.

A principal diferença entre esses personagens está centrada em três pontos: autorização, motivação e intenção. Os hackers éticos precisam obter autorização antes de executar qualquer tipo de teste, e após obtida, definem o escopo do teste, combinando com o cliente quais serviços e recursos devem ser considerados. Nessa questão, os atacantes maliciosos não tem qualquer restrição. Outro ponto é a motivação, os hackers éticos tem o propósito de ajudar a organização a aprimorar sua segurança. Os atacantes maliciosos, podem ter qualquer outro motivo, como por exemplo, ganho pessoal, extorsão, vingança, fama e diversão. Por fim, se o atacante oferece a simulação de um ataque real, preocupando-se em descobrir vulnerabilidades e como mitigá-las, é considerado um hacker ético. Já um atacante malicioso caracteriza-se por ter a intenção de algum ganho pessoal, por exemplo, com o vazamento das informações, ou com a negação dos serviços.

Segundo Regalado et al. (2015), os testes de penetração podem ser classificados em três tipos: caixa preta, cinza e branca. A diferença entre eles é a quantidade de conhecimento disponível para quem realiza o teste. O teste de caixa preta é o que melhor simula a ação de um ataque real, pois o hacker ético tem pouquíssimas informações sobre o alvo, como por exemplo, uma lista de IPs. Por sua vez, os testes de caixa branca são o oposto. Nele, o hacker ético tem acesso completo à aplicação, inclusive código fonte, e a diagramas. Por fim, o caixa cinza caracteriza-se pelo fato o hacker ético ter à sua disposição um pouco de conhecimento sobre a aplicação, sendo um meio termo entre o caixa preta e branca.

2.2.1 Metodologia e Fases do Teste

A utilização de uma metodololgia é importante para fornecer consistência e estrutura aos testes, minimizando os riscos e limitando os recursos relacionados com o teste (SCARFONE et al., 2008). A disponibilidade dos recursos é o principal gargalo nesse tipo de teste, pois ele consome bastante tempo, pessoal, *software* e *hardware*. Assim, a utilização de uma metodolgia propicia a reutilização de recursos pré-

estabelecidos, tais como pessoal treinado e plataformas de ensaio normalizados; diminui o tempo necessário para realizar a avaliação e a necessidade de adquirir equipamentos de teste e sofware; e reduz os custos globais da avaliação. O mesmo autor avalia que uma metodologia baseada em fases oferece muitas vantagens como uma estrutura fácil de seguir e e bem definida caso precise revezar a equipe. Engebretson (2011) enfatiza a indispensabilidade de utilizar uma metodologia para executar os testes de penetração, em virtude da complexidade inerente ao processo, e cita a existência de tarefas menores e mais gerenciáveis como um dos benefícios de se utilizar metodologia com fases.

Para Scarfone et al. (2008), as metodologias devem possuir no mínimo três fases bem definidas:

- **Planejamento:** fase em que ocorre a coleta de informações necessárias para a execução dos testes. Essas informações são os recursos que devem ser avaliados, as ameaças de interesse e o controle de segurança para desenvolver a abordagem de avaliação;
- **Execução:** o propósito dessa etapa é identificar as vulnerabilidades e validá-las. Embora as atividades específicas para esta fase diferem por tipo de avalição, após o fim desta etapa os avaliadores terão mapeado o sistema, rede e as vulnerabilidades encontradas;
- **Pós-execução:** as vulnerabilidades encontradas são avaliadas, determinando a causa e o impacto, e estabelecendo as recomendações para mitigação. Nessa etapa, também é elaborado o relatório final.

O escopo deste trabalho está concentrado na execução da última fase, preocupando-se com a geração do relatório.

2.2.2 Análise de vulnerabilidades

Segundo Engebretson (2011), existe uma importante diferença entre o teste de penetração e a análise de vulnerabilidade, que é motivo de engano por boa parte da comunidade. A análise de vulnerabilidade é o processo que apenas revisa os recursos e o sistema a fim de encontrar potenciais problemas de segurança, enquanto um teste de penetração tem por natureza explorar efetivamente as vulnerabilidades encontradas, apresentando provas de conceito que atestam que a falha de segurança existe. Assim, um teste de penetração é considerado um

passo posterior à análise de vulnerabilidades, simulando a atividade de um hacker malicioso.

2.3 OWASP

A *Open Web Application Security Project* (OWASP) é uma organização mundial, sem fins lucrativos, projetada para o aprimoramento da segurança de software (OWASP, 2015). Sua fundação foi em dezembro de 2001, mas apenas estabeleceu-se como uma organização em abril de 2004, com o objetivo de ser uma comunidade aberta dedicada para que as instituições possam criar, desenvolver, adquirir, operar e manter aplicações seguras. A sua principal estratégia é aumentar a visibilidade da segurança, em escala mundial, permitindo que indivíduos e instituições tomem decisões mais acertadas em relação aos riscos das aplicações. Para tanto, ela possui capítulos locais em todos os continentes que ajudam a disseminar os conhecimentos, fixando regionais em torno do assunto de segurança em aplicações web.

A OWASP (2015) se descreve como um novo tipo de organização. Ela não possui pressões econômicas, pois não é afiliada a nenhuma empresa de tecnologia, apesar de apoiar o uso de algumas tecnologias comerciais. Esse fato permite à OWASP fornecer informação imparcial e prática sobre segurança de sistemas, produzindo diferentes tipos de materiais de forma colaborativa e aberta. Ela está amparada por 4 valores principais:

- **Aberta:** Tudo é transparente, desde as finanças aos projetos;
- **Inovação:** Apoia e incentiva a inovação e as experiências de soluções para os desafios de segurança;
- **Global:** Qualquer pessoa ao redor do mundo é encorajada a participar da comunidade;
- **Integridade:** é neutra e fornece as informações de forma honesta e verdadeira.

2.3.1 Principais projetos OWASP

Dentre os projetos da fundação OWASP, destacam-se:

- ***OWASP Zed Attack Proxy (ZAP) Project*:** ferramenta de testes de penetração, que permite a descoberta de vulnerabilida-

des em aplicações web; fornece tanto scanners automáticos como um conjunto de funcionalidades que permitem a realização de testes manuais;

- ***OWASP WebGoat Project*:** plataforma de treinamento para realização de testes de segurança;
- ***OWASP Mantra Security Framework*:** navegador com plugins pré-instalados para a execução do teste de penetração; além disso, é facilmente configurado para utilizar o ZAP;
- ***OWASP Mobile Security Project*:** projeto que centraliza iniciativas para a segurança de *mobiles*, desde boas práticas, ferramentas para teste, como se proteger e as mais significativas falhas de aplicativos;
- ***OWASP Testing Guide Project*:** descreve os procedimentos e tarefas para testes completos de segurança em aplicações web; atualmente está na versão 4, que foi lançada em 2015;
- ***OWASP Top Ten Project*:** documento que revela as dez mais significativas vulnerabilidades em aplicações web; ele é atualizado a cada 3 anos e sua última versão data de 2013; este projeto é melhor detalhado na próxima seção.

2.3.2 *OWASP Top Ten Project*

A OWASP lançou em 2003 o projeto *OWASP Top Ten* como resposta ao crescente número de softwares inseguros (OWASP, 2013). O objetivo deste projeto é aumentar a consciência sobre a segurança das aplicações web, identificando os riscos mais críticos que as organizações estão expostas. A versão atual foi lançada em 2013, marcando o décimo ano do projeto, que é referenciado por inúmeros padrões, ferramentas e livros, incluindo MITRE[1], PCI[2], DISA[3] e FTC[4] (OWASP, 2013).

As versões dos anos 2003, 2004 e 2007 ordenam as vulnerabilidades apenas pela sua prevalência. A partir de 2010, a metodologia foi reformulada para dar prioridade também ao risco associado. A seguir serão descritas as características de cada uma das 10 categorias

[1] http://www.mitre.org/
[2] https://www.pcisecuritystandards.org/
[3] http://www.disa.mil/
[4] https://www.ftc.gov/

de vulnerabilidades sugeridas pelo projeto. O nome das classes foram mantidos em língua inglesa, em virtude da necessidade de preservação semântica:

- ***A1 - Injection*:** falhas de injeção, como injeção de SQL, de sistema operacional e LDAP (*Lightweight Directory Access Protocol*), surgem a partir do envio de dados não confiáveis utilizados na construção de um comando ou consulta. O atacante informa dados que enganam o interpretador e executa comandos que acessam dados sem a devida autorização;

- ***A2 - Broken Authentication and Session Management*:** as funcionalidades relacionadas com a autenticação e gerenciamento de sessão são geralmente implementadas de forma incorreta (comprometendo senhas, chaves, *tokens* de sessão), ou outras falhas que permitem o atacante assumir a identidade de outro usuário;

- ***A3 - Cross-Site Scripting (XSS)*:** falhas de XSS ocorrem quando a aplicação envia dados não confiáveis para o navegador sem o correto tratamento. XSS permite que os atacantes executem *scripts* nos navegadores da vítima, roubem sessões, ou redirecionem o usuário para outros sites;

- ***A4 - Insecure Direct Object References*:** essa vulnerabilidade acontece quando a referência para um objeto interno da implementação é exposta, normalmente por uma falha de um desenvolvedor, como a chave primária de um objeto no banco. Sem o devido controle de proteção, um atacante pode manipular esse valor e acessar dados para os quais não teria autorização;

- ***A5 - Security Misconfiguration*:** visando uma boa segurança, as aplicações devem respeitar as configurações adequadas dos *frameworks*, servidores, banco de dados e toda a infraestrutura que utiliza. As configurações padrão normalmente são inseguras e devem ser redefinidas, além de manter a infraestrutura atualizada;

- ***A6 - Sensitive Data Exposure*:** são as falhas relacionadas com a falta de segurança no armazenamento de dados confidenciais, como cartões de crédito, número de documentos e credenciais de autenticação. Os atacantes podem roubar ou modificar essas informações e realizar fraudes ou se passar pelo portador desses dados;

- ***A7 - Missing Function Level Acesss Control*:** as aplicações devem sempre verificar se os usuários tem permissão para executar uma ação específica antes de tornar essa funcionalidade disponível ao usuário, quando cada função é acessada. Caso não ocorra essa verificação no servidor, os atacantes podem forjar requisições e acessar funções sem a devida autorização;

- ***A8 - Cross-Site Request Forgery (CSRF)*:** nessa vulnerabilidade, o atacante força o navegador da vítima, estando ela autenticada em um serviço, a enviar uma solicitação HTTP forjada, incluindo suas informações de autenticação, como o valor do cookie para esse serviço. Assim, o navegador da vítima gera uma solicitação na aplicação vulnerável, que entende como um pedido legítimo da vítima;

- ***A9 - Using Components with Known Vulnerabilities*:** aplicações utilizam muitos componentes de terceiros para facilitar o desenvolvimento dos sistemas. Tais bibliotecas, módulos, podem possuir vulnerabilidades e ser alvo dos atacantes, minando as defesas da aplicação;

- ***A10 - Unvalidated Redirects and Forwards*:** redirecionamentos e encaminhamentos são funções comuns em sistemas web, conduzindo o usuário para outras partes do sistema. Sem a validação adequada, um atacante pode redirecionar vítimas para sites de *phishing* ou para um servidor com arquivos maliciosos.

No escopo deste projeto, serão consideradas apenas as vulnerabilidades relacionadas com os itens do *OWASP Top Ten Project*.

3 FERRAMENTAS UTILIZADAS

Atualmente existem diversas ferramentas disponibilizadas gratuitamente na internet que podem ser utilizadas para realizar um teste de penetração (ESPENSCHIED, 2008), promovendo a redução nos custos de um projeto dessa natureza. Seguindo essa tendência, neste capítulo serão apresentadas as ferramentas relacionadas ao teste de penetração que foram utilizadas na elaboração desse projeto.

3.1 APRENDIZADO

Conforme explica Engebretson (2011), todo hacker ético precisa de um local para praticar e explorar, dessa maneira, pode aprender sobre segurança e treinar o uso de ferramentas sem infringir leis ou atacar alvos sem autorização. As pessoas interessadas em aprimorar seus conhecimentos geralmente montam um laboratório (*hacking lab*), que é constituído por aplicações vulneráveis e é isolado em uma rede interna, assim nenhum tráfego pode atingir um alvo por descuido.

Portanto, iniciou-se a utilização de aplicações web vulneráveis para o treinamento e estudo das consequências das vulnerabilidades. Adicionalmente, essas aplicações são utilizadas para avaliação de ferramentas para teste de penetração, servindo de comparação os resultados obtidos pelas varreduras. As aplicações dessa família devem cobrir vários aspectos de segurança de aplicações web, de forma geral, seguindo as seguintes características (DRUIN, 2013):

- Simular aplicações reais utilizando *web services* e banco de dados;
- Fácil instalação;
- Disponível para vários sistemas operacionais;
- Pouca experiência em programação para utilizar;
- Voltar ao estado original após os ataques;
- Incluir dicas e tutoriais;
- Existência de múltiplos níveis de segurança;
- Ser de graça e de código-fonte aberto.

Nesse panorama, foram utilizadas duas aplicações sabidamente vulneráveis, a fim de aprender a utilizar a ferramenta W3AF e entender como ela reporta as vulnerabilidades. As aplicações utilizadas para treino foram a DVWA e o Mutillidae II que são descritas a seguir.

3.1.1 DVWA

A *Damn Vulnerable Web App*[1] (DVWA) é uma aplicação *open source* vulnerável em PHP/MySQL (DVWA, 2015). O objetivo principal é ajudar os profissionais de segurança a testar suas habilidades e ferramentas em um ambiente legal e ajudar os desenvolvedores a entenderem melhor como proteger suas aplicações web das principais ameaças (LEBEAU et al., 2013).

Lebeau et al. (2013) explica que a aplicação incorpora várias vulnerabilidades importantes, como as vistas na seção do *OWASP Top Ten Project*, incluindo *SQL Injection*, *Blind SQL Injection*, XSS refletido e persistido. Cada vulnerabilidade existente tem um item no menu que redireciona o usuário para a página dedicada à falha. Além disso, o DVWA possui três níveis de segurança que carregam mecanismos de proteção diferenciados. O nível baixo não oferece proteção, o nível médio é uma versão mais refinada, mas ainda bastante vulnerável. Por último, o nível mais alto implementa adequadamente as proteções de segurança. Os usuários podem escolher qual nível desejam enquanto usam a aplicação e também tem acesso ao código-fonte para cada nível de segurança.

3.1.2 Mutillidae II

Mutillidae II é uma aplicação gratuita, com código aberto, em que foram implementadas deliberadamente vulnerabilidades web para treinamento (DRUIN, 2013). O projeto iniciou-se em 2009 cobrindo apenas as vulnerabilidades do projeto *OWASP Top Ten Project*. Já em sua segundo versão, Mutillidae II foi totalmente reestruturado, adotando orientação a objeto, novas vulnerabilidades, maior divulgação, aprimoramento do sistema de ajuda e da documentação.

A aplicação é bastante similar ao DVWA, possuindo um menu na esquerda que contém itens para todas as páginas com vulnerabilidades, acesso fácil às dicas, tutoriais e a recuperação da aplicação. No entanto,

[1] http://www.dvwa.co.uk/

ela consegue abordar muito mais vulnerabilidades do que a anterior, indo além das descritas no *OWASP Top Ten Project*.

3.2 EXECUÇÃO

Nos últimos anos tem crescido o surgimento de ferramentas para varrer aplicações web procurando vulnerabilidades, com algumas estatísticas apontando que já existem mais de cem ferramentas com código aberto (QIANQIAN; XIANGJUN, 2014). Também segundo Qianqian e Xiangjun (2014), as ferramentas são dividas em dois grupos: as automáticas e semi-automáticas. As automáticas são extremamente simples de configurar, precisando no mínimo que a URL do alvo seja fornecida e em seguida ela executa uma varredura mapeando todas as páginas do site e trazendo o resultado. Por sua vez, as semi-automáticas exigem que seja configurado um *proxy* com o navegador e cada requisição pode ser analisada manualmente, além de também possuírem algoritmos que executam uma varredura autônoma.

Apesar de existirem inúmeras ferramentas que executam o teste automatizado, como a Arachni[2], Grendel-Scan[3], Skipfish[4] e OpenVas[5], para o desenvolvimento do projeto foi utilizado a W3AF por experiência prévia com a ferramenta, por ser código aberto, possuir comunidade ativa e ser totalmente automatizada.

3.2.1 W3AF

A ferramenta *Web Application Attack Audit Framework*, também conhecida como W3AF, é um projeto de código aberto criado por Andres Riacho com objetivos arrojados, pretendendo ser a melhor ferramenta para identificar e explorar vulnerabilidades. Atualmente ela está dividida em três partes principais (RIACHO, 2015): (i) o núcleo da aplicação, responsável por controlar o processo e fornecer as bibliotecas utilizadas pelos plugins; (ii) a interface do usuário, que permite a configuração e execução de varreduras; e (iii) os plugins, responsáveis por encontrar as vulnerabilidades.

De acordo com a função do plugin, ele pode pertencer a uma das

[2]Arachni - http://www.arachni-scanner.com/
[3]Grendel-Scan - http://sourceforge.net/projects/grendel/
[4]Skipfish - https://code.google.com/p/skipfish/
[5]http://www.openvas.org/

nove categorias[6]:

- ***Audit*:** responsáveis por identificar as vulnerabilidades nas aplicações e servidores web;
- ***Auth*:** encarregados de executar a autenticação no começo e o logout ao fim da varredura, além de verificar se a sessão está ativa regularmente, possibilitando a varredura em áreas protegidas;
- ***Bruteforce*:** executam operações de força bruta no login e sobre informações dos usuários;
- ***Crawl*:** incumbidos de descobrir novas URLs, formulários e qualquer tipo de recurso;
- ***Evasion*:** responsáveis por alterar as requisições para não serem detectados por sistemas de prevenção à intrusão;
- ***Grep*:** plugins que analisam toda requisição e resposta para encontrar erros, e-mails, comentários e qualquer informação;
- ***Infrastructure*:** possuem função para identificar as características da infraestrutura, como sistema operacional, servidor web, *firewall* e qualquer outra informação que não tenha relação com o código fonte;
- ***Mangle*:** responsáveis por alterar as requisições em tempo real;
- ***Output*:** os plugins desta categoria servem para exportar os resultados obtidos.

Conforme Qianqian e Xiangjun (2014), tanto a versão com interface gráfica como a por linha de comando requerem um conhecimento mínimo sobre segurança de aplicações web, em razão da complexidade para configurá-la. Apenas com a configuração padrão não é possível realizar uma varredura, alguns plugins podem demorar muito tempo para executar e outros são específicos para aplicações escritas em uma determinada linguagem ou que utilizem um determinado *framework*. Para contornar esse problema, a ferramenta disponibiliza nove *profiles*, que quando ativados, configuram a ferramenta com plugins pré-definidos:

- ***Empty profile*:** *profile* vazio que pode ser utilizado como base para a criação de outros;

[6] Os nomes dos plugins foram mantidos em inglês pela semântica.

- ***Bruteforce*:** destinado a fazer ataques de força bruta em formulários de autenticação utilizando credenciais padrão;
- ***Audit high risk*:** executa uma varredura que detecta apenas as vulnerabilidades de auto risco;
- ***OWASP TOP10*:** realiza uma varredura buscando pelas falhas listadas no *OWASP Top Ten Project*;
- ***Fast scan*:** processa uma rápida varredura com os plugins de rápida execução;
- ***Full audit*:** *profile* configurado para realizar uma varredura completa, utilizando apenas o plugin *web_spider* para executar o *crawl*;
- ***Full audit spider man*:** profile configurado para realizar uma varredura completa, utilizando apenas o plugin *spider_man* para executar o *crawl*;
- ***Sitemap*:** prepara a ferramenta para criar um *sitemap* de uma aplicação;
- ***Web infrastructure*:** utiliza todas as técnicas disponíveis na ferramenta para levantar as informações relacionadas à infraestrutura do servidor web.

Ainda assim, o usuário ao optar por utilizar um *profile*, pode alterar as configurações dos plugins e adicionar ou excluir outros. Deve-se ficar atento que caso o usuário deseje aplicar a ferramenta em uma parte restrita do site, um plugin de autenticação deve ser configurado após a escolha do *profile*. O mesmo ocorre com as configurações da exportação dos resultados, pois ao definir um *profile*, todas as configurações correntes são perdidas.

4 DESENVOLVIMENTO

Para solucionar o problema foi realizada a implementação de um sistema web utilizando a linguagem Java no *backend*, através do *framework PlayFramework*. A escolha deu-se por experiência prévia e comunidade extremamente ativa. Dentre os formatos de relatório que o W3AF pode gerar para exportar os resultados, optou-se pelo XML. Nesse formato o relatório é completo, contendo as vulnerabilidades encontradas e até as requisições e respostas HTTP. Além disso, o próprio autor da ferramenta recomenda esse formato para fazer a integração do relatório com outros aplicativos (RIACHO, 2015).

4.1 TECNOLOGIAS

Antes de apresentar o JAXB, faz-se necessário reforçar os conceitos de um documento XML e de seu esquema.

4.1.1 XML e XSD

XML é a sigla para *Extensible Markup Language*, uma recomendação da W3C para gerar linguagens de marcação, sendo capaz de descrever vários tipos de dados, facilitando o compartilhamento de informações por meio da internet. O XML pode representar diversas linguagens com uma infraestrutura única, devido a simplicidade e legibilidade, tanto para humanos quanto para computadores, sem restrição para criar tags e separação do conteúdo da formatação. Já o esquema de um XML, apesar de seguir as mesmas regras de um documento XML, é denominado *XML Schema Definition* (XSD). O XSD define a estrutura que um documento XML deve seguir, possibilitando a validação desse documento. Citando como exemplo, é possível definir os elementos permitidos, o seu tipo de dado e número de ocorrências. A título de exemplo, segue um simples XSD[1]:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="carro" type="Carro" />
  <xsd:complexType name="Carro">
```

[1] XSD retirado de http://blog.caelum.com.br/jaxb-xml-e-java-de-maos-dadas/

```
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"
          minOccurs="1" maxOccurs="1" nillable="
          false"/>
      <xsd:element name="portas" type="xsd:int"
          minOccurs="1" maxOccurs="1" nillable="
          false"/>
      <xsd:element name="motoristas" type="
          Motorista" minOccurs="0" maxOccurs="
          unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Motorista">
    <xsd:sequence>
      <xsd:element name="nome" minOccurs="1"
          maxOccurs="1" type="xsd:string" nillable=
          "false"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Código 1 – XSD de exemplo para um carro

Esse esquema define a estrutura de um arquivo em XML para representar um carro. Um carro precisa ter um nome, a quantidade de portas e não tem limitação em relação ao número de motoristas. Caso possua, cada motorista deve ter exatamente um nome. Seguindo essas restrições, temos abaixo um arquivo xml que segue esse padrão[2].

```
<?xml version="1.0" encoding="UTF-8"?>
<carro>
  <nome>Fusca</nome>
  <portas>2</portas>
  <motoristas>
    <motorista>
      <nome>Guilherme</nome>
    </motorista>
    <motorista>
      <nome>Leonardo</nome>
    </motorista>
  </motoristas>
```

[2]XML retirado de http://blog.caelum.com.br/jaxb-xml-e-java-de-maos-dadas/

```
</carro>
```

Código 2 – XML que respeita o padrão do XSD de um carro

4.1.2 JAXB

Java Architecture for XML Binding[3] (JAXB) fornece uma API e ferramentas para facilitar o uso de aplicações Java no processamento de dados em XML (JIANG et al., 2005), possibilitando mapear os objetos do Java para representações do XML. Assim, o desenvolvedor não precisa implementar funcionalidades específicas para carregar e salvar um XML. A relação direta entre os objetos Java com um documento XML permite a realização de três operações:

- Transformar o conteúdo do XML em representações de objetos em Java;
- Transformar objetos em Java em arquivo XML;
- Ler, modificar e validar se a representação em Java respeita as restrições de um esquema.

Sua utilização torna-se muito conveniente quando a especificação do XML é complexa ou varia muito. Nesse cenário, alterar regularmente a estrutura do XML e sincronizá-lo com as definições em Java seria muito custosa e propensa a erros. Para resolver essa questão, o JAXB disponibiliza uma ferramenta, a *xjc*, que pode ser usada para converter um XSD para representação em Java. A ferramenta cria classes e adiciona *annotations* relacionadas as classe e aos atributos que representam as restrições do esquema. A classe que representa o elemento principal do esquema é anotada com *@XmlRootElement* e as demais classes como um elemento normal, aplicando *@XmlElement*. Como exemplo, segue a definição de um atributo no XSD:

```
<xs:attribute name="start" use="required" type="
    xs:integer"/>
```

Código 3 – Definição do atributo *start* no XSD

De acordo com a definição, o atributo *start* é obrigatório e é do tipo *xs:integer*. Ao aplicar o *xjc* sobre o esquema, é gerado uma classe que contém um atributo do tipo *BigInteger*, por eles serem equivalentes.

[3]Site do projeto JAXB: https://jaxb.java.net/

Note que o atributo é anotado com *@XmlAttribute(name = "start", required = true)*, que definem o seu nome e restrição respectivamente.

```
@XmlAttribute(name = "start", required = true)
protected BigInteger start;
```

Código 4 – Definição do atributo *start* no Java

4.1.3 *PlayFramework*

PlayFramework[4] é um *framework* para aplicações web de código aberto escrito em Java e Scala. Seguindo o padrão MVC (*Model-View-Controller*), têm o objetivo de aumentar a produtividade dos desenvolvedores utilizando convenção ao invés de configuração, atualização instântanea do código e exibição dos erros no navegador. Ele não possui nenhuma restrição do Java EE e foi inspirado nos *frameworks* Ruby on Rails e Django, sendo muito familiar aos desenvolvedores destes. Outra característica é que ele já possui embutido um servidor web, mas mesmo assim, pode ser comprimido em um arquivo WAR para ser utilizado em servidores web que adotam o padrão Java EE.

Segue as caractertísticas mais relevantes em relação aos outros *frameworks*:

- *Stateless*: segue a arquitetura *RESTful*;
- *Framework* para testes unitários, integração e funcional embutido;
- *Asynchronous I/O*: consegue atender as requisições de forma assíncrona;
- Suporte nativo ao Scala;

4.2 FLUXO DE OPERAÇÃO

A utilização da ferramenta é bem simples e as etapas estão representadas na Fig. 1.

Inicialmente, o usuário deve configurar a ferramenta W3AF para realizar a análise na aplicação desejada, representado pela etapa 1. Na próxima etapa, após encerrar a varredura, o W3AF exporta o relatório

[4]Site do PlayFramework https://www.playframework.com/

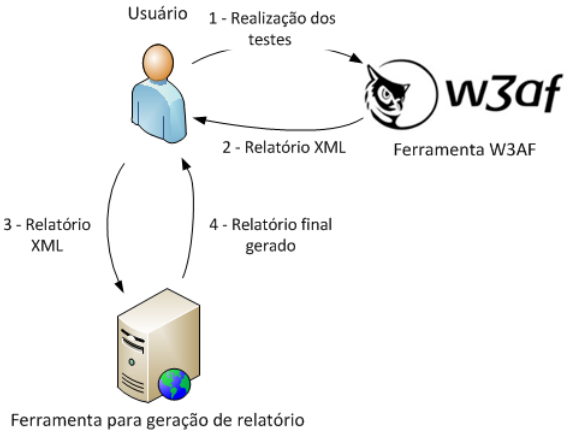

Figura 1 – Fluxo para utilização da ferramenta

com todos os dados da análise em XML. Estando o usuário em posse do arquivo XML, ele o submete na ferramenta, que faz a sua leitura, e por fim retorna um relatório com as vulnerabilidades encontradas.

4.3 OBTENÇÃO DAS AMOSTRAS DE RELATÓRIOS

O DVWA e o Mutillidae II foram instalados no ambiente *localhost*, sendo os respectivos endereços *http://127.0.0.1/dvwa* e *http://127.0.0.1:8080/mutillidae*. Ao tentar utilizar a ferramenta pela interface gráfica ela parava de responder e travava, por isso foi utilizada a versão por linha de comando. Desse modo, optou-se por configurá-la passando um arquivo de configuração através do parâmetro *-s*:

```
$ ./w3af -s arquivoDeConfiguracao.w3af
```

Código 5 – Comando para executar o W3AF

O primeiro sistema que passou pela análise foi o DVWA e o respectivo arquivo de configuração está completo no Apêndice A.1. Como o escopo desse trabalho são as vulnerabilidades descritas no *OWASP Top Ten Project*, foi utilizado o *profile* disponível especificamente para essas falhas, o *OWASP_TOP10*. Como explicado anteriormente, os *profiles*, quando utilizados, devem ser a primeira configuração ativada.

```
1 profiles
2 use OWASP_TOP10
```

```
3  back
4  http-settings
5  set timeout 30
6  back
```

Código 6 – Configuração do *profile* para o W3AF

Em seguida, foi definido o limite de 30 segundos por requisição para evitar erros de *timeout*. Feito isso, inicia a configuração dos plugins, primeiramente pelos que exportam os resultados.

```
8  output xml_file
9  output config console
10 set verbose False
```

Código 7 – Configuração da exportação dos resultados do W3AF

O trecho acima informa a ferramenta que desejamos exportar o relatório em XML e configuramos para que não seja impressa no terminal todas as informações no decorrer da análise.

O DVWA exige que o usuário esteja autenticado para poder utilizar a aplicação, então foi configurado um plugin para esse fim.

```
12 auth detailed
13 auth config detailed
14 set username admin
15 set password password
16 set method POST
17 set auth_url http://127.0.0.1/dvwa/login.php
18 set username_field username
19 set password_field password
20 set check_url http://127.0.0.1/dvwa/index.php
21 set check_string 'admin'
22 set data_format %u=%U&%p=%P&Login=Login
```

Código 8 – Configuração para autenticação do W3AF

Na linha 12 definimos a utilização do plugin e nas demais realizamos sua configuração. As credenciais do usuário que foram utilizadas são definidas nas linhas 14 e 15, através dos atributos *username* e *password*. Analisando a aplicação alvo, verificou-se que o formulário de autenticação gera uma requisição HTTP para o endereço *http://127.0.0.1/dvwa/login.php* com o método POST e que o atributo *name* dos *inputs* são *username* e *password*, sendo essas configurações registradas entre as linhas 16 e 19. Nas linhas 20 e 21 são as informações que o W3AF utiliza para assumir que a autenticação foi bem

sucedida. O parâmetro *check_ url* verifica se após o *login* a ferramenta foi redirecionada para a URL informada e o *check_ string* tenta encontrar o padrão na página. Por fim, é definido como deve ser construído os parâmetros do POST. O *u* e *U* representam respectivamente os atributos *username_ field* e *username*, e *p* e *P* representam *password_ field* e *password*.

Logo depois está a configuração do plugin *web_ spider* que é reponsável por varrer a aplicação mapeando todas as URLs.

```
24 crawl config web_spider
25 set only_forward True
26 set ignore_regex (?i)(logout|disconnect|signout
     |exit|setup|security)+
```

Código 9 – Configuração do *plugin* de varredura do W3AF

O atributo *only_ forward*, quando verdadeiro, define que a varredura deve levar em consideração o endereço fornecido e apenas os diretórios nele contidos. Quando este atributo é falso, a varredura leva em consideração tudo que está dentro do domínio em que a aplicação está hospedada. Outro atributo essencial desse plugin é o *ignore_ regex*, em que a ferramenta não irá acessar as URLs que seguem o padrão da expressão regular fornecida. Isso é muito útil para varreduras em ambientes autenticados, pois impede que a aplicação faça *logout*, e para não acessar páginas de reinstalação.

Por fim, é definido o alvo, a linguagem utilizada e o sistema operacional. Assim a ferramenta não dispensa tempo tentando detectar essas informações para fazer os testes específicos de cada plataforma.

```
30 set target http://127.0.0.1/dvwa
31 set target_os unix
32 set target_framework php
```

Código 10 – Configuração do alvo do W3AF

O relatório gerado ao final da varredura possui 14Mb e lista 327 possíveis vulnerabilidades encontradas.

Analogamente foi realizada a análise na aplicação Mutillidae II, contudo, essa aplicação não precisa de autenticação, e assim o plugin para esse fim não foi ativado pelo arquivo de configuração. Além da configuração que define o endereço da aplicação alvo, a do atributo *ignore_ regex* do plugin *web_ spider* ficou diferente da configuração do W3AF, para que a ferramenta não tenha acesso a URLs do Mutillidae. O arquivo completo de configuração está disponível no Apêndice A.2. O relatório gerado possui 26Mb e informa 466 possíveis vulnerabilidades.

4.4 INTERPRETAÇÃO DO RELATÓRIO

4.4.1 Geração das classes Java

Em posse dos relatórios XML, foi preciso descobrir uma forma de manipulá-los em Java. No repositório do W3AF encontra-se o esquema do relatório XML[5] que define todas as suas restrições, e como visto anteriormente, o JAXB oferece a ferramenta *xjc*, que lê um arquivo XSD e gera as classes Java equivalentes com as anotações do JAXB. Desse modo, a ferramenta foi utilizada como segue:

```
$ xjc -d output -p models.
    securityvulnerabilities.vulnerabilities
    reportW3af.xsd
```

Código 11 – Execução da ferramenta *xjc*

O parâmetro *-d* indica em qual diretório serão criadas as classes, já o *-p* informa que as classes devem ser criadas com o valor fornecido na declaração *package* e por fim, o arquivo XSD que servirá como base. Após o fim da execução foram geradas 27 classes Java, cada uma representando um elemento XML do relatório.

4.4.2 Vulnerabilidades suportadas

Analisando os relatórios gerados anteriormente, verificou-se que muitas das vulnerabilidades eram do mesmo tipo de falha e outras eram de caráter informativo e não efetivamente falhas de segurança. Relacionando as possíveis falhas com os itens do *OWASP Top Ten Project*, foram destacados 8 tipos de vulnerabilidades, como mostra a Tab. 1. Na coluna da direita foram preservados os nomes das vulnerabilidades reportadas pela ferramenta, esse nome é o atributo *name* do elemento XML *vulnerability*, que identifica o tipo que uma vulnerabilidade pertence.

Abaixo segue uma rápida explicação sobre cada vulnerabilidade:

Sql injection. A falha permite o atacante injetar código SQL em uma query e conseguir modificar sua execução.

Blind SQL injection vulnerability. Muito similar à falha anterior,

[5] Esquema do relatório XML disponível em: https://github.com/andresriancho/w3af/blob/master/w3af/plugins/output/xml_file/report.xsd

OWASP Top Ten	Vulnerabilidade W3AF
A1	SQL injection Blind SQL injection vulnerability OS commanding vulnerability
A2	Cookie Cookie without HttpOnly
A3	Cross site scripting vulnerability
A5	Apache Server version
A8	CSRF vulnerability

Tabela 1 – Vulnerabilidades suportadas

a diferença é que nessa falha o atacante apenas consegue obter respostas verdadeiras e falsas do banco de dados, e com base nelas assume as informações no banco.

OS commanding vulnerability. Falha análoga à primeira, no entanto, é injetando comandos do sistema operacional.

Cookie. Essa falha indica que o valor de usuário utilizado para realizar a autenticação está trafegando em texto plano em um dos cookies da aplicação.

Cookie without HttpOnly. Indica que os cookies estão sendo definidos sem a flag *HttpOnly*, isso faz com que eles sejam acessíveis via scripts do lado cliente, como Javascript.

Cross site scripting vulnerability. Ver o item A3 na seção 2.3.2.

Apache Server version. Aponta que o servidor apache está com o módulo *server_status* ativado, sendo possível qualquer usuário visualizar as informações de configuração do Apache.

CSRF vulnerability. Ver o item A8 na seção 2.3.2.

4.4.3 Leitura das vulnerabilidades

Inicialmente, ao ser submetido o relatório gerado pelo W3AF na ferramenta desenvolvida, são instanciadas e populadas as classes Java equivalentes. Nesse momento também é verificado se o relatório segue as restrições determinadas pelo esquema.

```
1  JAXBContext jaxbContext = JAXBContext.
       newInstance(W3AfRun.class);
```

```
2 Unmarshaller unmarshaller = jaxbContext.
      createUnmarshaller();
3 W3AfRun report = this.unmarshalW3afReport(
      inputStream);
```

Código 12 – Transformação do relatório XML para Java

A classe *W3AfRun* foi gerada pela ferramenta *xjc* e representa o elemento *root* do XML. Assim, o relatório fica acessível na variável *report* e uma lista com todos os elementos que representam uma vulnerabilidade estão disponíveis pelo método *report.getVulnerability()*.

Para cada tipo de vulnerabilidade suportada foi criada uma classe específica que implementa a interface *SecurityVulnerability*, o que obriga as classes a implementar os métodos para decodificar e codificar a vulnerabilidade. O método para decodificar recebe como parâmetro um objeto que representa o elemento *vulnerability* do XML, dessa forma, cada classe obtém as informações do XML necessárias. O método para codificar será explicado na próxima seção. Além disso, a classe deve ser colocada no diretório *models.securityvulnerabilities.vulnerabilities* e anotada com *@Vulnerability("nomeDaVulnerabilidade")*, sendo o parâmetro o nome da vulnerabilidade utilizado pelo W3AF.

Ao iniciar o servidor de aplicação, uma fábrica[6] percorre o diretório citado buscando as classes que possuem a *annotation @Vulnerability* e salva em um mapa o nome da vulnerabilidade passado na *annotation*, relacionado com a classe Java referente a essa vulnerabilidade.

Logo após criar as classes Java do relatório XML, as vulnerabilidades retornadas pelo método *report.getVulnerability()* passam por um processo de iteração. Cada uma delas é enviada para a fábrica que verifica se o valor do seu atributo *name* existe no mapa gerado no processo de inicialização, se existir, a vulnerabilidade é criada, caso contrário, assume-se que tal vulnerabilidade não é suportada e segue-se para a próxima.

4.5 GERAÇÃO DO RELATÓRIO PRELIMINAR

Ao encerrar a geração das vulnerabilidades com as informações pertinentes do XML, inicia-se o processo para gerar o relatório no formato final, nesse trabalho foi adotado o LaTeX[7]. A lista com as vulnerabilidades encontradas é percorrida, sendo executado o método que as

[6] Factory Method - https://pt.wikipedia.org/wiki/Factory_Method
[7] LaTeX - https://www.latex-project.org/

codifica, ou seja, gera um texto em LaTeX mostrando as informações relevantes extraídas do relatório XML, como as URLs, parâmetros e métodos HTTP, além de conter uma descrição e mitigação da vulnerabilidade. Por último é realizado um *parsing* com o *template* do relatório que contém informações gerais sobre como ele foi gerado, o seu escopo e as considerações finais.

4.6 SUPORTE A NOVAS VULNERABILIDADES

Adicionar novas vulnerabilidades é um processo bem simples. O desenvolvedor deve criar uma classe e adicioná-la no pacote *models.securityvulnerabilities.vulnerabilities* e anotá-la com *@Vulnerability("nomeDaVulnerabilidade")*, sendo o valor do atributo, o nome da vulnerabilidade reportado pelo W3AF. Em seguida, implementa a interface *SecurityVulnerability* e os métodos para decodificar e codificar.

4.7 EXEMPLO DE USO

Como prova de conceito a ferramenta desenvolvida foi utilizada para gerar um relatório preliminar do sistema Mutillidae II. Em posse do relatório gerado pelo W3AF obtido na etapa para geração das amostras de relatórios, o sistema web desenvolvido foi acessado, cuja página inicial é mostrada na Fig. 2. Ela é composta por um botão no topo, que apresenta um *modal* com algumas informações básicas sobre a ferramenta. Já na parte central está localizado o formulário para informar o arquivo gerado pelo W3AF e um botão para efetuar a operação.

Em seguida, selecionamos o arquivo gerado pelo W3AF e clicamos no botão para obter o relatório preliminar. Em menos de 20 segundos é feito o upload do arquivo de 26 MB e o seu processamento pela ferramenta, que ao fim, apresenta o relatório preliminar em LaTeX para download, como mostra a Fig. 3.

O relatório gerado é composto por 4 capítulos que serão descritos a seguir. O Apêndice B contém na íntegra o relatório gerado para a aplicação Mutillidae II.

O capítulo de introdução inicia o relatório abordando a importância de avaliar a segurança de um sistema web e apresenta a aplicação alvo. No capítulo 2 é descrito como foi feita a avaliação automatizada utilizando a ferramenta W3AF.

No terceiro capítulo são listadas em seções as vulnerabilidades

Figura 2 – Página inicial da ferramenta

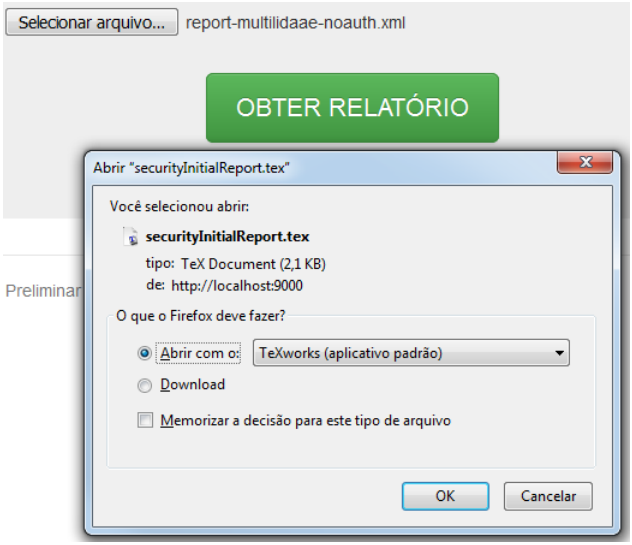

Figura 3 – Download do relatório preliminar gerado

encontradas na aplicação alvo que são suportadas pela ferramenta. Para cada vulnerabilidade é apresentada uma descrição, explicando do que ela se trata e seu impacto, a remediação e os vetores de ataque, que são como um atacante pode explorá-la. Por exemplo, para as falhas de *SQL Injection* são destacados a url, método HTTP e o parâmetro vulnerável. Por fim, o quarto capítulo traz as considerações finais, destacando que se trata de um relatório preliminar, não dispensando uma análise manual.

5 CONSIDERAÇÕES FINAIS

O trabalho conseguiu alcançar o objetivo de desenvolver uma aplicação para gerar um relatório a partir da saída de uma ferramenta de teste automatizado. Foram realizados dois testes de segurança automatizados em aplicações web com explicações de como replicá-los e desenvolvida uma ferramenta para a geração do relatório. Além disso, os conhecimentos em segurança de informação, análise de vulnerabilidades e sobre o projeto OWASP foram reforçados. Foi também uma excelente oportunidade para trabalhar com o *PlayFramework*, que vem ganhando muito destaque na comunidade.

O objetivo inicial era que a ferramenta desenvolvida gerasse um relatório final legível, entretanto, notou-se que seria mais interessante retornar o arquivo em texto LaTeX e não PDF, para que o usuário tivesse a oportunidade de alterá-lo sem dificuldade.

O escopo do trabalho era suportar as vulnerabilidades do *OWASP Top Ten Project*, o que de fato foi alcançado, contudo, notou-se que a ferramenta W3AF, escolhida para a realização do teste automatizado, reportou poucas vulnerabilidades existentes, visto que as aplicações eram sabidamente vulneráveis, mas ainda assim, o relatório gerado é preliminar e consegue transmitir a situação da aplicação alvo.

5.1 TRABALHOS FUTUROS

Como trabalhos futuros são sugeridas duas vertentes: (i) uma análise mais profunda sobre a ferramenta W3AF; e (ii) expandir as funcionalidades da ferramenta desenvolvida. A primeira opção visa verificar se o W3AF é realmente uma boa opção como ferramenta para realização de teste automatizado e se ela foi utilizada adequadamente. Caso contrário, apontar novos caminhos. Já na segunda, pode-se aprimorar a ferramenta para suportar mais vulnerabilidades, novas ferramentas de teste automatizado, ou ainda, para alimentar uma base com estatísticas para comparações futuras.

REFERÊNCIAS

ALLAN, D. *Web application security: automated scanning versus manual penetration testing*. [S.l.], 2008. <ftp://ftp.software.ibm.com/software/rational/web/whitepapers/r_wp_autoscan.pdf>. Acessado em 30/09/2015.

BELAPURKAR, A. et al. *Distributed Systems Security issues, processes, and solutions*. [S.l.]: Wiley, 2009. 307 p. ISBN 9780470519882.

BISHOP, M. *Introduction to Computer Security*. [S.l.]: Addison-Wesley Professional, 2004. 784 p. ISBN 9780321247445.

CARNETTI, K. *Comércio eletrônico fatura R$ 43 bi e registra crescimento de 26% nas vendas em 2014*. 2015. <http://info.abril.com.br/noticias/internet/2015/01/comercio-eletronico-fatura-r-43-bi-e-registra-crescimento-de-26-nas-vendas-em-2014.shtml>. Acessado em 30/10/2015.

CROSS, M. *Developer's Guide to Web Application Security*. 1. ed. [S.l.]: Syngress, 2007. 500 p. ISBN 9781597490610.

DRUIN, J. Introduction to the owasp mutillidae ii web pen-test training environment. *Software Testing, Verification and Validation Workshops (ICSTW)*, 2013.

DVWA: Damn vulnerable web application. 2015. <http://www.dvwa.co.uk/>. Acessado em 12/10/2015.

ENGEBRETSON, P. *The Basics of Hacking and Penetration Testing*. [S.l.]: Syngress, 2011. 159 p. ISBN 9781597496551.

ESPENSCHIED, J. *Five free pen-testing tools*: The best things in life are ... 2008. <http://www.computerworld.com/article/2536045/endpoint-security/five-free-pen-testing-tools.html>. Acessado em 12/10/2015.

GRAVES, K. *CEH: Certified Ethical Hacker Study Guide*. [S.l.]: Sybex, 2010. 439 p.

IMPERVA. *Imperva's Web Application Report*. 2010. <http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed2.pdf>. Acessado em 30/09/2015.

INTERNET CRIME COMPLAINT CENTER. *2013 Internet Crime Report*. 2013. <http://www.ic3.gov/media/annualreport/2013_IC3Report.pdf>. Acessado em 30/09/2015.

JIANG, H. et al. Xml views based approach for web service. *Information Reuse and Integration*, p. 458–463, 2005.

KBAR, G. Security risk analysis for asset in relation to vulnerability, probability of threats and attacks. *Computer Systems and Applications*, p. 874–879, 2009.

LEBEAU, F. et al. Model-based vulnerability testing for web applications. *Software Testing, Verification and Validation Workshops (ICSTW)*, p. 445–452, 2013.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Guide for Conducting Risk Assessments*. [S.l.], 2012. <http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf>. Acessado em 08/10/2015.

OPEN WEB APPLICATION SECURITY PROJECT. *OWASP Top 10 - 2013*: The ten most critical web application security risks. [S.l.], 2013.

OPEN WEB APPLICATION SECURITY PROJECT. *About The Open Web Application Security Project*. 2015. <https://www.owasp.org/index.php/About_OWASP>. Acessado em 30/09/2015.

PATCHER, J. *Desenvolvimento de um guia para testes de penetração através de exemplos práticos*. Tese (Doutorado) — Universidade Federal de Santa Catarina, Santa Catarina, Brasil, 2014.

PELTIER, T. R. *Information Security Risk Analysis*. [S.l.]: Auerbach, 2005. 360 p. ISBN 9781420031195.

QIANQIAN, W.; XIANGJUN, L. Research and design on web application vulnerability scanning service. *Software Engineering and Service Science (ICSESS)*, p. 671–674, 2014.

REGALADO, D. et al. *Gray Hat Hacking*: The ethical hacker's handbook. 4. ed. [S.l.]: McGraw-Hill Education, 2015. 656 p. ISBN 9780071832380.

RIACHO, A. *W3AF - Open Source Web Application Security Scanner*. 2015. <http://w3af.org/>. Acessado em 12/10/2015.

SCARFONE, K. et al. *Technical guide to Information Security Testing and Assessment*: Recommendations of the national institute of standards and technology. [S.l.], 2008. <http://csrc.nist.gov/publications/nistpubs/800-115/SP800-115.pdf>. Acessado em 08/10/2015.

SCHNEIER, B. *Beyond Fear*. [S.l.]: Wiley, 2006. 296 p. ISBN 9780387026206.

SHIREY, R. *Internet Security Glossary, Version 2*: Request for comments 4949. [S.l.], 2007. <https://tools.ietf.org/html/rfc4949>. Acessado em 08/10/2015.

STALLINGS, W. *Criptografia e segurança de redes Princípios e práticas*. 6. ed. São Paulo, Brazil: PEARSON, 2015. 576 p. ISBN 9788543005898.

APÊNDICE A – Arquivos de configuração do W3AF

A.1 CONFIGURAÇÃO PARA O DVWA

```
profiles
use OWASP_TOP10
back
http-settings
set timeout 30
back
plugins
output xml_file
output config console
set verbose False
back
auth detailed
auth config detailed
set username admin
set password password
set method POST
set auth_url http://127.0.0.1/dvwa/login.php
set username_field username
set password_field password
set check_url http://127.0.0.1/dvwa/index.php
set check_string 'admin'
set data_format %u=%U&%p=%P&Login=Login
back
crawl config web_spider
set only_forward True
set ignore_regex (?i)(logout|disconnect|signout
    |exit|setup|security)+
back
back
target
set target http://127.0.0.1/dvwa
set target_os unix
set target_framework php
back
cleanup
start

```

A.2 CONFIGURAÇÃO PARA O MUTILLIDAE II

```
profiles
use OWASP_TOP10
back
http-settings
set timeout 60
back
plugins
output xml_file
output config console
set verbose False
back
crawl config web_spider
set only_forward True
set ignore_regex (?i)(logout|disconnect|signout
    |exit|setup|security|set-up-database|toggle)
    +
back
back
target
set target http://127.0.0.1/mutillidae
set target_os unix
set target_framework php
back
cleanup
start
```

APÊNDICE B – Relatório preliminar para a ferramenta Mutillidae II

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Laboratório de Segurança em Computação

ANÁLISE DO MUTIILIDAE II: ANÁLISE DE VULNERABILIDADES WEB

Florianópolis

2015

SUMÁRIO

1 INTRODUÇÃO

Este documento refere-se à execução de testes de penetração no projeto da Bry. Os testes que serão aplicados no alvo (comumente conhecidos como PenTest, ou Penetration Tests) são de extrema importância, pois têm como objetivo avaliar a segurança do mesmo através da simulação de um ataque por uma fonte maliciosa. Durante o teste do sistema e o processo de avaliação, descrito nas próximas sessões, serão detalhadas as possíveis vulnerabilidades encontradas.

2 AVALIAÇÃO AUTOMATIZADA

2.1 METODOLOGIA OWASP

Para realizar os testes, tomaremos como base documentos da OWASP (Open Web Application Security Project). Esse projeto é bastante focado em auxiliar as empresas a desenvolverem, operarem e manterem aplicações cada vez mais seguras e confiáveis, e é altamente utilizado como guia para testes, uma vez que distribui ferramentas e tutoriais detalhados abertamente. Um dos documentos publicados a cada três anos pela OWASP é o Top Ten.

2.2 FERRAMENTA W3AF

Antes de aplicarmos qualquer teste, fizemos um reconhecimento automatizado utilizando a ferramenda Zed Attack Proxy(ZAP). Esta plataforma é um projeto da OWASP criado para facilitar a realização de PenTest, possibilitando que estes testes sejam feitos de maneira automatizada e dinâmica.

Utilizando o ZAP, é possível efetuar uma varredura na infraestrutura da aplicação e procurar falhas comuns, principalmente as relacionadas ao OWASP Top Ten, que, como dito anteriormente, serviu de base para nossos testes. A ferramenta trabalha atuando como proxy, interceptando requisições enviadas pelo navegador para a aplicação, possibilitando a análise de todo o conteúdo acessado, permitindo aplicação de filtros para procurar por scripts mal formados e outros erros na aplicação que podem levar ao comprometimento da mesma.

3 POTENCIAIS VULNERABILIDADES

3.1 COOKIE COM INFORMAÇÕES DE LOGIN

3.1.1 Descrição

Algum cookie possui no seu valor a informação de login do usuário.

3.1.2 Impacto

Um atacante pode estar monitorando a rede, ou essa informação pode ficar no cache do navegador, e ter acesso ao login de um usuário.

3.1.3 Remediação

A informação de login não deve existir no cookie, ou então, cifrar o valor do cookie.

3.1.4 Vetores de ataque

- **Url:** http://127.0.0.1/mutillidae/index.php;

3.2 COOKIE SEM HTTPONLY

3.2.1 Descrição

O atributo *HTTPOnly* não está sendo definido no cookie.

3.2.2 Impacto

Sem a definição do atributo *HTTPOnly*, *client side scripts*, como JavaScript, tem acesso ao cookie, assim, caso um atacante consiga injetar código JavaScript na aplicação, ele poderá ter acesso aos cookies.

3.2.3 Remediação

Devem ser definidos para todos os cookies o atributo *HTTPOnly*

3.2.4 Vetores de ataque

- **Url:** http://127.0.0.1/mutillidae/index.php;

3.3 SQL INJECTION

3.3.1 Descrição

As falhas de injeção SQL surgem com o envio de dados não confiáveis que são interpretados como parte de uma consulta SQL. O atacante informa dados que enganam o interpretador e executam comandos que acessam dados sem a devida autorização.

3.3.2 Impacto

Dependendo do escopo da vulnerabilidade o atacante pode ler informações sensíveis do banco, modificá-las e executar operações administrativas.

3.3.3 Remediação

Todas as queries devem ser codificadas utilizando queries parametrizadas, ou utilizar *stored procedures*, ou por último, realizar um filtro nos dados fornecidos pelos usuários, bloqueado caracteres especiais do SQL.

3.3.4 Vetores de ataque

- **Url:** http://127.0.0.1/mutillidae/includes/pop-up-help-context-generator.php; **Parâmetro:** pagename; **Método HTTP:** GET;

- **Url:** http://127.0.0.1/mutillidae/webservices/rest/ws-user-account.php;

Parâmetro: username; **Método HTTP:** GET;

3.4 CROSS SITE REQUEST FORGERY (CSRF)

3.4.1 Descrição

Este tipo de vulnerabilidade força um usuário final a executar ações indesejadas em uma aplicação web em que ele está atualmente autenticado, através de técnicas, como o envio de um link por e-mail.

3.4.2 Impacto

O atacante pode forçar os usuários de uma aplicação a executarem ações de vontade do atacante. Caso este ataque atinja um usuário administrador (com privilégios), pode comprometer toda a aplicação.

3.4.3 Remediação

Ao receber as requisições a aplicação deve ser capaz de identificar se ela partiu do próprio site. Isso pode ser feito através de um parâmetro *hidden* no formulário com um valor aleatório, que deve ser verificado antes de processar a resposta da requisição

3.4.4 Vetores de ataque

- **Url:** http://127.0.0.1/mutillidae/index.php; **Método HTTP:** GET;
- **Url:** http://127.0.0.1/mutillidae/webservices/soap/ws-hello-world.php; **Método HTTP:** GET;
- **Url:** http://127.0.0.1/mutillidae/webservices/soap/ws-user-account.php; **Método HTTP:** GET;
- **Url:** http://127.0.0.1/mutillidae/webservices/soap/ws-lookup-dns-record.php; **Método HTTP:** GET;
- **Url:** http://127.0.0.1/mutillidae/webservices/rest/ws-user-account.php; **Método HTTP:** GET;

- **Url:** http://127.0.0.1/mutillidae/documentation/Mutillidae-Test-Scripts.txt; **Método HTTP:** POST;

3.5 APACHE SERVER VERSION

3.5.1 Descrição

O servidor Apache está com o módulo de status habilitado.

3.5.2 Impacto

Qualquer usuário pode acessar as informações de configuração do Apache.

3.5.3 Remediação

O módulo *mod_status* deve ser desativado.

3.5.4 Vetor de ataque

Através da url http://127.0.0.1/server-status.

3.6 CROSS SITE SCRIPTING (XSS)

3.6.1 Descrição

Esta falha ocorre quando a aplicação envia dados não confiáveis para o navegador sem o correto tratamento, sendo esses dados códigos que são executados pelo navegador da vítima.

3.6.2 Impacto

O XSS permite que os atacantes executem *scripts* nos navegadores da vítima, roubem sessões, ou redirecionem o usuário para outros sites.

3.6.3 Remediação

Todos os dados informados pelos usuários e os obtidos em uma fonte de dados devem passar por um filtro. O filtro principal é transformar os dados em entidades do HTML.

3.6.4 Vetores de ataque

- **Url:** http://127.0.0.1/mutillidae/includes/pop-up-help-context-generator.php; **Parâmetro:** pagename; **Método HTTP:** GET;
- **Url:** http://127.0.0.1/mutillidae/webservices/rest/ws-user-account.php; **Parâmetro:** username; **Método HTTP:** GET;

4 CONSIDERAÇÕES FINAIS

Apesar do avanço tecnológico no desenvolvimento de ferramentas que realizam testes de segurança de forma automatizada, elas jamais substituirão por completo a análise manual, pois tais ferramentas podem interpretar os dados equivocadamente, não estando preparadas para as especificidades de cada sistema ou para pertubações externas. Assim, esse relatório é o primeiro passo para que a empresa solicitante comece a mitigar os pontos de falhas reportados neste trabalho, enquanto a análise manual é realizada.

APÊNDICE C – Artigo

Geração de Relatório Preliminar para Análise de Vulnerabilidades Web de Forma Automatizada: No Escopo do OWASP Top Ten Project

Lucas Alberto Tomasi[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

lucastomasi@gmail.com

Abstract. *Web applications are increasingly present on the internet and with it the number of cyber crimes has been growing as well. In this scenario, penetration tests are essential for safety, identifying potential vulnerabilities through the eyes of an attacker. However, most institutions are only concerned about their safety when their systems are already in production mode, or even worse, when they become victims of attacks. Due to its own analytical nature, penetrations tests execution may take few weeks. So, this work presents a way to quickly deliver a preliminary report on the security of a system. Using this tool, the institution can start working to solve the major security issues, while the manual and detailed tests are performed.*

Resumo. *Aplicações web estão cada vez mais presentes na internet e assim também cresce o número de crimes cibernéticos. Nesse cenário, os testes de penetração são essenciais como medida de segurança, identificando as possíveis vulnerabilidades pela ótica de um atacante. No entanto, a maioria das instituições preocupam-se com a sua segurança quando seus sistemas já estão em produção, ou ainda pior, ao se tornarem vítimas de ataques. Pela natureza analítica de um teste de penetração, sua execução pode levar algumas semanas. Assim, este trabalho apresenta uma forma de entregar rapidamente um relatório preliminar sobre a segurança de um sistema. Dessa maneira, a instituição já pode começar a trabalhar para resolver os principais problemas de segurança enquanto os testes manuais e minuciosos são executados.*

1. Introdução

A alta oferta de aplicações na internet, aliada com valor dos dados que nela trafegam, sejam informações sensíveis ou operações financeiras, seduziram pessoas mal intencionadas a realizar ataques nesses sistemas para obter vantagens ilícitas. Em virtude da preocupação sobre o tema, foram criadas diversas iniciativas com o objetivo de avaliar as questões de segurança de uma aplicação web, como por exemplo, o *Open Web Application Security Project* (OWASP).

As instituições precisam proteger seus dados e sistemas, já que queixas relacionadas com crimes cibernéticos aumentaram mais que 15 vezes de 2000 até 2013 e o prejuízo estimado com esse tipo de crime no Brasil foi de R$ 15,9 bilhões em 2012 e US$ 338 bilhões no mundo [Patcher 2014].

Nesse contexto, destaca-se o teste de penetração como um método para avaliar a segurança de um sistema computacional. As atividades envolvidas nesse método tentam descobrir e explorar fraquezas dos sistemas, como acessar regiões sem acesso autorizado, vazamento de dados, indisponibilidade do serviço, e etc. Tal diagnóstico é registrado conforme os testes são executados em um relatório e ao comprovar vulnerabilidades, são fornecidas estratégias de mitigação. Esse tipo de teste tenta simular as possibilidades que um atacante real empregaria para quebrar a segurança de um sistema.

Infelizmente a maioria das organizações preocupam-se com a segurança quando os sistemas já estão disponíveis em produção, ou ainda mais tarde, quando se tornam vítimas de ataques. Esse comportamento típico é arriscado, pois exige uma rápida resposta para a manutenção dos serviços e para a correção das vulnerabilidades exploradas. Dessa maneira, as organizações carecem de uma análise de segurança o mais rápido possível. No entanto, constatar o nível de segurança de um sistema através de um teste de penetração é uma tarefa onerosa, levando semanas para ser concluída. Apesar do avanço tecnológico no desenvolvimento de ferramentas que realizam testes de segurança de forma automatizada, elas jamais substituirão por completo a análise manual [Allan 2008], pois tais ferramentas podem interpretar os dados equivocadamente, não estando preparadas para as especificidades de cada sistema ou para pertubações externas.

Finalmente, diante do cenário apresentado, incorre-se a necessidade de elaborar um meio para gerar rapidamente um relatório sobre o estado da aplicação. Nesse caso, poderia ser gerado um relatório preliminar verificando o nível de segurança da aplicação, através de ferramentas que realizam testes automatizados, com o mínimo de interferência humana, pois aqui o tempo é o fator primordial. Enquanto o analista procede a execução dos testes minuciosos, a instituição que requiriu o teste de penetração, em posse do relatório preliminar, pode começar a agir verificando os possíveis vetores de ataque reportados.

O presente trabalho é dividido em cinco partes, e inicialmente no segundo capítulo são expostos conceitos básicos de segurança e sobre testes de penetração. O capítulo 3 apresenta uma breve explicação sobre a organização OWASP e o projeto OWASP Top Ten Project. No capítulo 4, por sua vez, apresentadas as ferramentas utilizadas, e no quinto capítulo é descrito a implementação do projeto. Por fim, o sexto capítulo encerra o trabalho trazendo as considerações finais e as sugestões dos trabalhos futuros.

2. Conceitos em Segurança da Informação

O *software* é um produto extremamente complexo que abrange inúmeros fatores imprevisíveis e de difícil controle, como mudança nos requisitos e inovações tecnológicas. A complexidade inerente ao processo de desenvolvimento de um *software* faz com que grande parte dos projetos ultrapasse o orçamento e prazos previstos, além de não atender completamente as necessidades do cliente. Da mesma forma que os riscos inerentes ao desenvolvimento de um *software* devem ser mitigados, os riscos relacionados à segurança das aplicações devem ser analisados. Para qualificar e entender melhor a gravidade de uma falha em um sistema, faz-se necessário tomar emprestado alguns conceitos utilizados no gerenciamento de riscos [Stoneburner et al. 2012]:

Risco. Medida em que uma entidade está ameaçada por um potencial evento ou circunstância e é uma função em relação a dois fatores: (i) os impactos que surgiriam se

ocorressem os eventos ou circunstâncias; e (ii) a chance de ocorrência.

Ameaça. Potencial violação de segurança que pode afetar as instituições, ativos e/ou indivíduos através de acesso não autorizado, destruição, divulgação ou modificação de informações, e negação de serviços. A violação não precisa efetivamente ocorrer para que se exista uma ameaça e com a possibilidade que uma violação ocorra, ações devem ser tomadas para proteger-se [Bishop 2004].

Vulnerabilidade. Fraqueza de um sistema que pode ser explorada por uma fonte de ameaça. A maioria das vulnerabilidades estão associadas com a não implantação de controles de segurança (intencionalmente ou não), ou que tenham sido implementados de forma errada ou incompleta.

Impacto. O impacto é a magnitude de um dano causado quando uma ameaça se concretiza e nível de impacto pode ser dividido em três categorias [Peltier 2005]: (i) alto impacto, comprometendo serviços fundamentais com perda significativa de reputação ou lucro; (ii) médio impacto, comprometendo serviços com perda financeira limitada; e (iii) baixo impacto, comprometimento sem prejuízo financeiro.

Ataque. De acordo com Shirley (2007), o ataque é um ato intencional pelo qual uma entidade tenta burlar mecanismos de segurança, ou violar as políticas de um sistema.

2.1. Teste de penetração

O teste de penetração é um tentativa legal e autorizada para localizar e explorar uma vulnerabilidade, para que tal sistema possa ter sua segurança aprimorada [Engebretson 2011]. Desse maneira, a exploração serve como prova de conceito para demonstrar que uma vulnerabilidade é real. O teste de penetração gera como artefato final um documento que descreve as vulnerabilidades encontradas e como corrigí-las, para que os sistemas se tornem seguros a futuros ataques da mesma natureza.

Segundo [Regalado et al. 2015], os testes de penetração podem ser classificados em três tipos: caixa preta, cinza e branca. A diferença entre eles é a quantidade de conhecimento disponível para quem realiza o teste. O teste de caixa preta é o que melhor simula a ação de um ataque real, pois o hacker ético tem pouquíssimas informações sobre o alvo, como por exemplo, uma lista de IPs. Por sua vez, os testes de caixa branca são o oposto. Nele, o hacker ético tem acesso completo à aplicação, inclusive código fonte, e a diagramas. Por fim, o caixa cinza caracteriza-se pelo fato o hacker ético ter à sua disposição um pouco de conhecimento sobre a aplicação, sendo um meio termo entre o caixa preta e branca.

3. OWASP

A *Open Web Application Security Project* (OWASP) é uma organização mundial, sem fins lucrativos, projetada para o aprimoramento da segurança de software. Sua fundação foi em dezembro de 2001, mas apenas estabeleceu-se como uma organização em abril de 2004, com o objetivo de ser uma comunidade aberta dedicada para que as instituições possam criar, desenvolver, adquirir, operar e manter aplicações seguras. A sua principal estratégia é aumentar a visibilidade da segurança, em escala mundial, permitindo que indivíduos e instituições tomem decisões mais acertadas em relação aos riscos das aplicações. Para tanto, ela possui capítulos locais em todos os continentes que ajudam

a disseminar os conhecimentos, fixando regionais em torno do assunto de segurança em aplicações web.

3.1. OWASP Top Ten Project

A OWASP lançou em 2003 o projeto *OWASP Top Ten* como resposta ao crescente número de softwares inseguros. O objetivo deste projeto é aumentar a consciência sobre a segurança das aplicações web, identificando os riscos mais críticos que as organizações estão expostas. A versão atual foi lançada em 2013 e a seguir serão descritas as características de cada uma das 10 categorias de vulnerabilidades sugeridas pelo projeto. O nome das classes foram mantidos em língua inglesa, em virtude da necessidade de preservação semântica:

- ***A1 - Injection*:** falhas de injeção, como injeção de SQL, de sistema operacional e LDAP (*Lightweight Directory Access Protocol*), surgem a partir do envio de dados não confiáveis utilizados na construção de um comando ou consulta. O atacante informa dados que enganam o interpretador e executa comandos que acessam dados sem a devida autorização;
- ***A2 - Broken Authentication and Session Management*:** as funcionalidades relacionadas com a autenticação e gerenciamento de sessão são geralmente implementadas de forma incorreta (comprometendo senhas, chaves, *tokens* de sessão), ou outras falhas que permitem o atacante assumir a identidade de outro usuário;
- ***A3 - Cross-Site Scripting (XSS)*:** falhas de XSS ocorrem quando a aplicação envia dados não confiáveis para o navegador sem o correto tratamento. XSS permite que os atacantes executem *scripts* nos navegadores da vítima, roubem sessões, ou redirecionem o usuário para outros sites;
- ***A4 - Insecure Direct Object References*:** essa vulnerabilidade acontece quando a referência para um objeto interno da implementação é exposta, normalmente por uma falha de um desenvolvedor, como a chave primária de um objeto no banco. Sem o devido controle de proteção, um atacante pode manipular esse valor e acessar dados para os quais não teria autorização;
- ***A5 - Security Misconfiguration*:** visando uma boa segurança, as aplicações devem respeitar as configurações adequadas dos *frameworks*, servidores, banco de dados e toda a infraestrutura que utiliza. As configurações padrão normalmente são inseguras e devem ser redefinidas, além de manter a infraestrutura atualizada;
- ***A6 - Sensitive Data Exposure*:** são as falhas relacionadas com a falta de segurança no armazenamento de dados confidenciais, como cartões de crédito, número de documentos e credenciais de autenticação. Os atacantes podem roubar ou modificar essas informações e realizar fraudes ou se passar pelo portador desses dados;
- ***A7 - Missing Function Level Acesss Control*:** as aplicações devem sempre verificar se os usuários tem permissão para executar uma ação específica antes de tornar essa funcionalidade disponível ao usuário, quando cada função é acessada. Caso não ocorra essa verificação no servidor, os atacantes podem forjar requisições e acessar funções sem a devida autorização;
- ***A8 - Cross-Site Request Forgery (CSRF)*:** nessa vulnerabilidade, o atacante força o navegador da vítima, estando ela autenticada em um serviço, a enviar uma solicitação HTTP forjada, incluindo suas informações de autenticação, como o valor do cookie para esse serviço. Assim, o navegador da vítima gera uma solicitação na aplicação vulnerável, que entende como um pedido legítimo da vítima;

- ***A9 - Using Components with Known Vulnerabilities*:** aplicações utilizam muitos componentes de terceiros para facilitar o desenvolvimento dos sistemas. Tais bibliotecas, módulos, podem possuir vulnerabilidades e ser alvo dos atacantes, minando as defesas da aplicação;
- ***A10 - Unvalidated Redirects and Forwards*:** redirecionamentos e encaminhamentos são funções comuns em sistemas web, conduzindo o usuário para outras partes do sistema. Sem a validação adequada, um atacante pode redirecionar vítimas para sites de *phishing* ou para um servidor com arquivos maliciosos.

4. Ferramentas Utilizadas

Atualmente existem diversas ferramentas disponibilizadas gratuitamente na internet que podem ser utilizadas para realizar um teste de penetração, promovendo a redução nos custos de um projeto dessa natureza. Seguindo essa tendência, neste capítulo serão apresentadas as ferramentas relacionadas ao teste de penetração que foram utilizadas na elaboração desse projeto.

4.1. DVWA e Mutillidae II

Conforme explica Engebretson (2011), todo hacker ético precisa de um local para praticar e explorar, dessa maneira, pode aprender sobre segurança e treinar o uso de ferramentas sem infringir leis ou atacar alvos sem autorização. As pessoas interessadas em aprimorar seus conhecimentos geralmente montam um laboratório (*hacking lab*), que é constituído por aplicações vulneráveis e é isolado em uma rede interna, assim nenhum tráfego pode atingir um alvo por descuido.

Portanto, iniciou-se a utilização de aplicações web vulneráveis para o treinamento e estudo das consequências das vulnerabilidades. Adicionalmente, essas aplicações são utilizadas para avaliação de ferramentas para teste de penetração, servindo de comparação os resultados obtidos pelas varreduras. As aplicações dessa família devem cobrir vários aspectos de segurança de aplicações web, de forma geral, seguindo as seguintes características [Druin 2013]:

- Simular aplicações reais utilizando *web services* e banco de dados;
- Fácil instalação;
- Disponível para vários sistemas operacionais;
- Pouca experiência em programação para utilizar;
- Voltar ao estado original após os ataques;
- Incluir dicas e tutoriais;
- Existência de múltiplos níveis de segurança;
- Ser de graça e de código-fonte aberto.

A *Damn Vulnerable Web App*[1] (DVWA) é uma aplicação *open source* vulnerável em PHP/MySQL. O objetivo principal é ajudar os profissionais de segurança a testar suas habilidades e ferramentas em um ambiente legal e ajudar os desenvolvedores a entenderem melhor como proteger suas aplicações web das principais ameaças [Lebeau et al. 2013].

[1] http://www.dvwa.co.uk/

Mutillidae II é uma aplicação gratuita, com código aberto, em que foram implementadas deliberadamente vulnerabilidades web para treinamento [Druin 2013]. O projeto iniciou-se em 2009 cobrindo apenas as vulnerabilidades do projeto *OWASP Top Ten Project*. Já em sua segundo versão, Mutillidae II foi totalmente reestruturado, adotando orientação a objeto, novas vulnerabilidades, maior divulgação, aprimoramento do sistema de ajuda e da documentação.

4.2. W3AF

Nos últimos anos tem crescido o surgimento de ferramentas para varrer aplicações web procurando vulnerabilidades, com algumas estatísticas apontando que já existem mais de cem ferramentas com código aberto [Qianqian and Xiangjun 2014]. As ferramentas são dividas em dois grupos: as automáticas e semi-automáticas. As automáticas são extremamente simples de configurar, precisando no mínimo que a URL do alvo seja fornecida e em seguida ela executa uma varredura mapeando todas as páginas do site e trazendo o resultado. Por sua vez, as semi-automáticas exigem que seja configurado um *proxy* com o navegador e cada requisição pode ser analisada manualmente, além de também possuírem algoritmos que executam uma varredura autônoma.

Apesar de existirem inúmeras ferramentas que executam o teste automatizado, como a Arachni[2], Grendel-Scan[3], Skipfish[4] e OpenVas[5], para o desenvolvimento do projeto foi utilizado a W3AF por experiência prévia com a ferramenta, por ser código aberto, possuir comunidade ativa e ser totalmente automatizada.

A ferramenta *Web Application Attack Audit Framework*, também conhecida como W3AF, é um projeto de código aberto criado por Andres Riacho com objetivos arrojados, pretendendo ser a melhor ferramenta para identificar e explorar vulnerabilidades. Atualmente ela está dividida em três partes principais: (i) o núcleo da aplicação, responsável por controlar o processo e fornecer as bibliotecas utilizadas pelos plugins; (ii) a interface do usuário, que permite a configuração e execução de varreduras; e (iii) os plugins, responsáveis por encontrar as vulnerabilidades.

5. Desenvolvimento

A utilização da ferramenta é bem simples e as etapas estão representadas na Fig. 1.

Inicialmente, o usuário deve configurar a ferramenta W3AF para realizar a análise na aplicação desejada, representado pela etapa 1. Na próxima etapa, após encerrar a varredura, o W3AF exporta o relatório com todos os dados da análise em XML. Estando o usuário em posse do arquivo XML, ele o submete na ferramenta, que faz a sua leitura, e por fim retorna um relatório com as vulnerabilidades encontradas.

Para gerar as amostras de relatórios da ferramenta W3AF, o DVWA e o Mutillidae II foram instalados no ambiente *localhost*, sendo os respectivos endereços *http://127.0.0.1/dvwa* e *http://127.0.0.1:8080/mutillidae*. A W3AF parava de responder ao ser utilizada pela interface gráfica, por isso foi utilizada a versão por linha de comando.

[2]Arachni - http://www.arachni-scanner.com/
[3]Grendel-Scan - http://sourceforge.net/projects/grendel/
[4]Skipfish - https://code.google.com/p/skipfish/
[5]http://www.openvas.org/

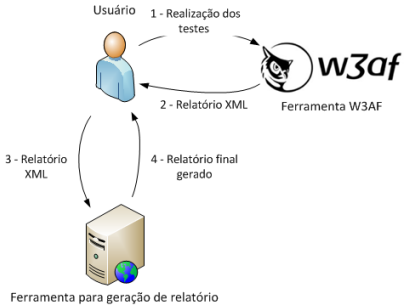

Figura 1. Fluxo para utilização da ferramenta

Desse modo, optou-se por configurá-la passando um arquivo de configuração através do parâmetro *-s*:

```
$ ./w3af -s arquivoDeConfiguracao.w3af
```

Código 1. Comando para executar o W3AF

Como o escopo desse trabalho são as vulnerabilidades descritas no *OWASP Top Ten Project*, foi configurado o *profile* disponível especificamente para essas falhas, o *OWASP_TOP10*. Também foi ativado a geração do relatório em XML e especificamente para o DVWA, foi configurado o plugin de autenticação.

Em posse dos relatórios XML, foi preciso descobrir uma forma de manipulá-los em Java. No repositório do W3AF encontra-se o esquema do relatório XML[6] que define todas as suas restrições, e sabendo que o JAXB (*Java Architecture for XML Binding*) oferece a ferramenta *xjc*, que lê um arquivo XSD e gera as classes Java equivalentes com as anotações do JAXB. Desse modo, a ferramenta foi utilizada como segue:

```
$ xjc -d output -p models.securityvulnerabilities.
    vulnerabilities reportW3af.xsd
```

Código 2. Execução da ferramenta *xjc*

O parâmetro *-d* indica em qual diretório serão criadas as classes, já o *-p* informa que as classes devem ser criadas com o valor fornecido na declaração *package* e por fim, o arquivo XSD que servirá como base. Após o fim da execução foram geradas 27 classes Java, cada uma representando um elemento XML do relatório.

Analisando os relatórios gerados, verificou-se que muitas das vulnerabilidades eram do mesmo tipo de falha e outras eram de caráter informativo e não efetivamente falhas de segurança. Relacionando as possíveis falhas com os itens do *OWASP Top Ten Project*, foram destacados 8 tipos de vulnerabilidades, como mostra a Tab. 1. Na coluna da direita foram preservados os nomes das vulnerabilidades reportadas pela ferramenta, esse nome é o atributo *name* do elemento XML *vulnerability*, que identifica o tipo que a vulnerabilidade pertence.

[6]Esquema do relatório XML disponível em: https://github.com/andresriancho/w3af/blob/master/w3af/plugins/output/xml_file/report.xsd

Tabela 1. Vulnerabilidades suportadas

OWASP Top Ten	Vulnerabilidade W3AF
A1	SQL injection Blind SQL injection vulnerability OS commanding vulnerability
A2	Cookie Cookie without HttpOnly
A3	Cross site scripting vulnerability
A5	Apache Server version
A8	CSRF vulnerability

Abaixo segue uma rápida explicação sobre cada vulnerabilidade:

Sql injection. A falha permite o atacante injetar código SQL em uma query e conseguir modificar sua execução.

Blind SQL injection vulnerability. Muito similar à falha anterior, a diferença é que nessa falha o atacante apenas consegue obter respostas verdadeiras e falsas do banco de dados, e com base nelas assume as informações no banco.

OS commanding vulnerability. Falha análoga à primeira, no entanto, é injetando comandos do sistema operacional.

Cookie. Essa falha indica que o valor de usuário utilizado para realizar a autenticação está trafegando em texto plano em um dos cookies da aplicação.

Cookie without HttpOnly. Indica que os cookies estão sendo definidos sem a flag *HttpOnly*, isso faz com que eles sejam acessíveis via scripts do lado cliente, como Javascript.

Cross site scripting vulnerability. Ver o item A3 na seção 3.1.

Apache Server version. Aponta que o servidor apache está com o módulo *server_status* ativado, sendo possível qualquer usuário visualizar as informações de configuração do Apache.

CSRF vulnerability. Ver o item A8 na seção 3.1.

Inicialmente, ao ser submetido o relatório gerado pelo W3AF na ferramenta desenvolvida, são instanciadas e populadas as classes Java equivalentes. Nesse momento também é verificado se o relatório segue as restrições determinadas pelo esquema.

Para cada tipo de vulnerabilidade suportada foi criada uma classe específica que implementa a interface *SecurityVulnerability*, o que obriga as classes a implementar os métodos para decodificar e codificar a vulnerabilidade. O método para decodificar recebe como parâmetro um objeto que representa o elemento *vulnerability* do XML, dessa forma, cada classe obtém as informações do XML necessárias. Além disso, a classe deve ser colocada no diretório *models.securityvulnerabilities.vulnerabilities* e anotada com *@Vulnerability("nomeDaVulnerabilidade")*, sendo o parâmetro o nome da vulnerabilidade utilizado pelo W3AF.

Ao iniciar o servidor de aplicação, uma fábrica[7] percorre o diretório citado buscando as classes que possuem a *annotation @Vulnerability* e salva em um mapa o nome da vulnerabilidade passado na *annotation*, relacionado com a classe Java referente a essa vulnerabilidade.

Logo após criar as classes Java do relatório XML, as vulnerabilidades retornadas pelo método *report.getVulnerability()* passam por um processo de iteração. Cada uma delas é enviada para a fábrica que verifica se o valor do seu atributo *name* existe no mapa gerado no processo de inicialização, se existir, a vulnerabilidade é criada, caso contrário, assume-se que tal vulnerabilidade não é suportada e segue-se para a próxima.

Ao encerrar a geração das vulnerabilidades com as informações pertinentes do XML, inicia-se o processo para gerar o relatório no formato final, nesse trabalho foi adotado o LaTeX[8]. A lista com as vulnerabilidades encontradas é percorrida, sendo executado o método que as codifica, ou seja, gera um texto em LaTeX mostrando as informações relevantes extraídas do relatório XML, como as URLs, parâmetros e métodos HTTP, além de conter uma descrição e mitigação da vulnerabilidade. Por último é realizado um *parsing* com o *template* do relatório que contém informações gerais sobre como ele foi gerado, o seu escopo e as considerações finais.

O relatório gerado é uma sugestão e é composto por 4 capítulos que serão descritos a seguir. O capítulo de introdução inicia o relatório abordando a importância de avaliar a segurança de um sistema web e apresenta a aplicação alvo. No capítulo 2 é descrito como foi feita a avaliação automatizada utilizando a ferramenta W3AF, para que o solicitante da análise possa reproduzir a varredura.

No terceiro capítulo são listadas em seções as vulnerabilidades encontradas na aplicação alvo que são suportadas pela ferramenta. Para cada vulnerabilidade é apresentada uma descrição, explicando do que ela se trata e seu impacto, a remediação e os vetores de ataque, que são como um atacante pode explorá-la. Por fim, o quarto capítulo traz as considerações finais, destacando que se trata de um relatório preliminar, não dispensando uma análise manual.

6. Considerações Finais

O trabalho conseguiu alcançar o objetivo de desenvolver uma aplicação para gerar um relatório a partir da saída de uma ferramenta de teste automatizado. Foram realizados dois testes de segurança automatizados em aplicações web com explicações de como replicá-los e desenvolvida uma ferramenta para a geração do relatório. Inicialmente, tinha-se a intenção de que a ferramenta desenvolvida gerasse um relatório final legível, entretanto, notou-se que seria mais interessante retornar o arquivo em texto LaTeX e não PDF, para que o usuário tivesse a oportunidade de alterá-lo sem dificuldade.

O escopo do trabalho era suportar as vulnerabilidades do OWASP Top Ten Project, o que de fato foi alcançado, contudo, notou-se que a ferramenta W3AF, escolhida para a realização do teste automatizado, reportou poucas vulnerabilidades existentes, visto que as aplicações eram sabidamente vulneráveis, mas ainda assim, o relatório gerado é preliminar e consegue transmitir a situação da aplicação alvo.

[7]Factory Method - https://pt.wikipedia.org/wiki/Factory_Method
[8]LaTeX - https://www.latex-project.org/

6.1. Trabalhos Futuros

Como trabalhos futuros são sugeridas duas vertentes: (i) uma análise mais profunda sobre a ferramenta W3AF; e (ii) expandir as funcionalidades da ferramenta desenvolvida. A primeira opção visa verificar se o W3AF é realmente uma boa opção como ferramenta para realização de teste automatizado e se ela foi utilizada adequadamente. Caso contrário, apontar novos caminhos. Já na segunda, pode-se aprimorar a ferramenta para suportar mais vulnerabilidades, novas ferramentas de teste automatizado, ou ainda, para alimentar uma base com estatísticas para comparações futuras.

Referências

Allan, D. (2008). *Web application security: automated scanning versus manual penetration testing*. IBM.

Bishop, M. (2004). *Introduction to Computer Security*. Addison-Wesley Professional.

Druin, J. (2013). Introduction to the owasp mutillidae ii web pen-test training environment. *Software Testing, Verification and Validation Workshops (ICSTW)*.

Engebretson, P. (2011). *The Basics of Hacking and Penetration Testing*. Syngress.

Lebeau, F., Legeard, B., Peureux, F., and Vernotte, A. (2013). Model-based vulnerability testing for web applications. *Software Testing, Verification and Validation Workshops (ICSTW)*, pages 445–452.

Patcher, J. (2014). *Desenvolvimento de um guia para testes de penetração através de exemplos práticos*. PhD thesis, Universidade Federal de Santa Catarina, Santa Catarina, Brasil.

Peltier, T. R. (2005). *Information Security Risk Analysis*. Auerbach.

Qianqian, W. and Xiangjun, L. (2014). Research and design on web application vulnerability scanning service. *Software Engineering and Service Science (ICSESS)*, pages 671–674.

Regalado, D., Harris, S., Harper, A., Eagle, C., Ness, J., Spasojevic, B., Linn, R., and Sims, S. (2015). *Gray Hat Hacking*. McGraw-Hill Education, 4 edition.

Stoneburner, G., Goguen, A., and Feringa, A. (2012). *Guide for Conducting Risk Assessments*. National Institute of Standards and Technology.