Proposta e Implementação de um Firewall para Aplicações Web Denominado UniscanWAF

Alfredo Del Fabro Neto, Rogério Turchetti,
Celio Trois, Walter Priesnitz Filho
Universidade Federal de Santa Maria - CTISM/UFSM
{alfredodfn, turchetti, celio.trois,
walter}@redes.ufsm.br

Douglas Rocha, Diego Kreutz
Grupo de Pesq. em Sist. de Informação – GPSI
Universidade Federal do Pampa – UNIPAMPA
douglas.poerschke@gmail.com,
kreutz@unipampa.edu.br

***Resumo*—Os sistemas Web predominam na maioria dos segmentos da sociedade. Ao mesmo tempo que cresce o número de aplicações e recursos online, aumentam também as preocupações relacionadas à segurança da informação. Os sistemas Web são os maiores alvos dos atacantes. Buscando contribuir com a segurança das aplicações Web, este trabalho apresenta uma ferramenta, denominada UniscanWAF, que tem por objetivo monitorar em tempo real os acessos a sistemas Web. Sua principal finalidade é detectar, interceptar e bloquear tentativas de ataque às aplicações online. Para isso, a ferramenta utiliza mecanismos e recursos de detecção do Uniscan, tornando possível a interceptação de ataques que visem explorar vulnerabilidades como *XSS*, *LFI*, *RFI*, *RCE* e *SQL-Injection*.**

I. Introdução

A maioria das vulnerabilidades encontradas em aplicações Web são derivadas de uma fraca implementação do ciclo de desenvolvimento de software (SDLC - *Software Development Life Cycle*) [1]. Além disso, sistemas Web são alvos constantes de ataques. Um filtro de pacotes convencional, que trabalha nas camadas de redes e transporte, não detecta ou evita ataques que exploram vulnerabilidades em nível de aplicação. Uma das soluções existentes para o nível de aplicação são os *Web Application Firewall* (WAF) [2]. Um exemplo de WAF é o *ModSecurity* [3]. Este foi projetado para atuar com o servidores Web Apache.

No intuito de minimizar o efeito e as ações de ataques a sistemas Web, este trabalho descreve e avalia o UniscanWAF, uma ferramenta capaz de detectar comportamentos anômalos e interceptar, com base em regras pré-definidas, ações maliciosas contra sistemas Web. A ferramenta proposta representa uma extensão ao Uniscan [4], o qual é um scanner convencional de vulnerabilidades em sistemas Web. O Uniscan fornece um diagnóstico do sistema testado, identificando todas as páginas ou URLs problemáticas do sistema. Ele está disponível no SourceForge [5] e na distribuição *Linux BackTrack*.

O UniscanWAF trabalha com as mesmas regras (*plugins* de detecção de vulnerabilidades) do Uniscan. A principal diferença reside no fato de a ferramenta proposta atuar de forma similar a um *proxy* com filtro de pacotes, ou seja, é capaz de analisar e interceptar ações maliciosas em tempo de execução, protegendo a aplicação Web contra ataques conhecidos (regras). Essa característica é especialmente interessante para proteger sistemas Web que estão em constante desenvolvimento ou evolução. Segundo estudos, muitas aplicações Web entram nessa classificação [6].

A ferramenta proposta encontra-se em fase de desenvolvimento. Neste sentido, o principal objetivo deste trabalho é verificar sua viabilidade de utilização, avaliando as funcionalidades de análise e interceptação de requisições maliciosas destinadas à aplicações Web.

O restante deste trabalho está estruturado da seguinte forma: a seção II apresenta a arquitetura geral do UniscanWAF. Na seção III são apresentadas as vulnerabilidades detectadas pela ferramenta, bem como códigos vulneráveis utilizados no cenário de testes para cada uma das vulnerabilidades. A seção IV apresenta os testes realizados para validar as funcionalidades do UniscanWAF. Por fim, a seção V apresenta as considerações finais.

II. Arquitetura do UniscanWAF

A arquitetura do UniscanWAF foi dividida em dois módulos distintos: o módulo de Inicialização e o módulo de Detecção de Vulnerabilidades. O módulo de Inicialização compreende a inicialização da ferramenta de acordo com os parâmetros configurados no arquivo de configuração. Algumas das possibilidades de configuração são: a ação tomada pelo módulo de Detecção de Vulnerabilidades no momento da detecção, que pode ser configurado para encerrar a conexão ou executar um *script*, por exemplo, gerar um relatório técnico, colocar o host de origem em quarentena, ou mesmo obter maiores informações com a execução de outras ferramentas que possibilite um diagnóstico detalhado. Outros exemplos de parâmetros de configuração incluem a localização do arquivo de *log* e a porta que o UniscanWAF utilizará para comunicação.

O módulo de Detecção de Vulnerabilidades é responsável por realizar a análise de todas as requisições dos clientes que tenham como destino o servidor Web a ser protegido. Para cada regra há um arquivo correspondente, que determina as ações a serem executadas pela ferramenta para detectar a existência ou não de uma vulnerabilidade na requisição que está sendo analisada. Com isso, novas regras podem ser adicionadas ao sistema sob-demanda, conferindo flexibilidade e simplicidade ao sistema. Na versão atual, o UniscanWAF é capaz de analisar e processar a maioria das vulnerabilidades implementadas e detectadas pelo Uniscan, incluindo *XSS* (*Cross-site Scripting*), *LFI* (*Local File Include*), *RFI* (*Remote File Include*), *RCE*

(*Remote Command Execution*) e *SQL-Injection*. A figura 1 ilustra a arquitetura do UniscanWAF.

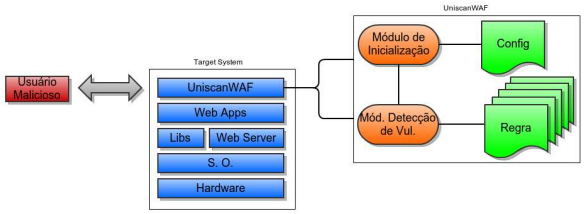

Figura 1. Arquitetura do UniscanWAF

O UniscanWAF atua como um intermediário, similar a um proxy, entre o usuário e o servidor Web. Sendo assim, ele é capaz de analisar e repassar ou não as requisições para a aplicação Web, conforme as regras habilitadas para detecção.

III. Vulnerabilidades Web Detectadas pelo UniscanWAF

Para validar a eficiência do detector de vulnerabilidades proposto, foi criada uma aplicação teste contendo as cinco vulnerabilidades reconhecidas pela ferramenta. Cada vulnerabilidade, acompanhada de um exemplo de código vulnerável, é descrita a seguir.

A. Cross-site Scripting

A vulnerabilidade *XSS* ocorre quando uma aplicação inclui dados fornecidos pelo usuário sem a validação desses dados [7]. O impacto desse tipo de ataque inclui o eventual roubo de sessão. Aplicações que não tratam os dados de entrada adequadamente estão sujeitas a esse tipo de ataque. Esse tipo de vulnerabilidade passa, com uma certa frequência, desapercebido pelos desenvolvedores. De maneira a amenizar esse problema, estratégias de segurança presentes no SDLC são recomendadas [8].

```
<?php
print "Pagina: " . urldecode($_SERVER["REQUEST_URI"]);
?>
```

Figura 2. Código-fonte PHP vulnerável a *XSS*.

Na figura 2 é apresentado um exemplo de código-fonte PHP vulnerável ao ataque *XSS*. No caso em questão, o trecho de código imprime o recurso que foi requisitado. Consequentemente, ao requisitar uma URL como `"http://sistemaWeb.com.br/index.php?name=teste"`, será exibido `"Pagina: /index.php?name=teste"`. Entretanto, há outras possibilidades de ações, como no caso de uma URL similar a `"http://sistemaWeb.com.br/index.php?name=<script>alert('XSS')</script>"`, que, no caso, irá criar uma caixa com o conteúdo `"XSS"` e um botão `"OK"`. Isso significa que um atacante consegue executar ações e códigos diversos, tornando o ataque potencialmente comprometedor para o sistema Web, ou ambiente alvo, como o sistema hospedeiro do servidor Web.

B. Local File Include

A vulnerabilidade *LFI* torna possível a inclusão de arquivos locais. Este ataque pode expor arquivos e dados dos sistemas hospedeiros da aplicação Web.

```
<?php
$arquivo = $_GET['arg'];
include($arquivo);
?>
```

Figura 3. Código-fonte PHP vulnerável a *LFI* e *RFI*.

Na figura 3 é ilustrado um exemplo de código-fonte PHP vulnerável ao ataque *LFI*. Pode ser observado que o código recebe um parâmetro de nome `"arg"` via método GET. O valor é atribuído à variável `"$arquivo"`, que, logo após, é incluída no código sem nenhuma verificação. Desse modo, um usuário poderia requisitar uma URL como `"/arquivo.php?arg=/etc/passwd"`. Neste caso, o arquivo `"/etc/passwd"` será exibido para o usuário.

C. Remote File Include

Esta vulnerabilidade é similar a LFI. A principal diferença reside no fato de incluir um arquivo hospedado em outro servidor Web ao invés do sistema local. Isso representa um risco de segurança maior, pois o arquivo remoto pode ter sido intencionalmente criado e preparado pelo atacante, aumentando suas possibilidades de ações maliciosas de forma relativamente simples e prática.

A vulnerabilidade RFI ocorre devido a falta de validação e tratamento correto dos dados de entrada. O código ilustrado na figura 3 permite a inclusão remota de arquivos. Um exemplo de URL maliciosa poderia ser `"/arquivo.php?arg=http://servidorWebDoAtacante.com.br/comandos.txt"`. Este arquivo pode conter qualquer sequência de comandos e construções reconhecidas pela linguagem PHP e pelo sistema hospedeiro da aplicação alvo. Consequentemente, uma vulnerabilidade RFI é crítica e permite ao atacante realizar diferentes tipos de ações no sistema alvo.

D. Remote Command Execution

A vulnerabilidade de *RCE* permite a um atacante executar remotamente comandos no sistema alvo. Num código PHP, através do comando *system()* sem o devido tratamento dos dados inseridos pelo usuário, é possível executar comandos do sistema operacional.

```
<?php
$user = $_GET['arg'];
system("echo $user >> users_forum.txt");
?>
```

Figura 4. Código-fonte PHP vulnerável a *RCE*.

A figura 4 apresenta um exemplo de código-fonte PHP vulnerável. O objetivo do código é inserir usuários no final do arquivo `"users_forum.txt"`, como `"/rce.php?arg=user4"`. Neste caso, apenas o usuário `"user4"` será acrescentado no arquivo indicado. Porém, um atacante pode ir muito além. A chamada de

sistema *system()* invoca um shell para a execução do comando. Como consequência, qualquer composição de comandos possível em um shell, utilizando o separador `";"`, pode ser invocada pelo atacante. Este, por exemplo, poderia requisitar a URL `"/rce.php?arg=user4;cat /etc/passwd;"`, o que provocaria a inclusão do `"user4"` no arquivo `"users_forum.txt"`, bem como a leitura do conteúdo do arquivo `"/etc/passwd"`.

O risco de segurança aumenta em situações onde os servidores Web são executados em modo *root*. Nesses casos, os sistemas hospedeiros podem ser facilmente comprometidos por um atacante, extendendo as implicações de segurança para além da aplicação Web vulnerável.

E. Sql Injection

SQL Injection pode ser definida como a inserção de código SQL (*Structured Query Language*) malicioso através de dados de entrada de uma aplicação. Se bem sucedido, esse tipo de ataque pode obter acesso aos dados do banco de dados, modificar esses dados, eventualmente executar operações de administrador de banco de dados, e, em alguns casos, executar comandos do sistema operacional [9]. O tratamento desses dados, por parte de programadores, pode evitar esse tipo de ataque.

```
<?php
$par1 = $_GET['username'];
$par2 = $_GET['password'];
$query = "SELECT * FROM usuario WHERE username= '$par1'".
                                    "AND password= '$par2'";
?>
```

Figura 5. Código-fonte PHP vulnerável a *SQL Injection*.

A figura 3 apresenta um exemplo de código PHP vulnerável a *SQL Injection*. No exemplo, caso a URL de requisição for `"/show.php?username=user&password=12345"`, serão apresentadas as informações do usuário `user`, com senha `12345`, cadastradas no banco de dados. Entretanto, se a URL apresentar o formato `"/show.php?username=user&password=1' OR '1"`, serão listadas informações de todos os usuários presentes no banco de dados da aplicação Web. Cabe observar que os parâmetros `username` e `password` não sofrem nenhum tipo de tratamento e são inseridos diretamente na pesquisa (*query*) SQL. Deste forma, a operação `"1' OR '1"` é sempre verdadeira, retornando os dados de todos os usuários registrados.

IV. Testes Efetuados

O objetivo é apresentar os primeiros resultados de avaliação da funcionalidade de detecção de vulnerabilidades do UniscanWAF. E, também, comparar os resultados com o *ModSecurity* [3], um WAF *open source* projetado para trabalhar junto a servidores Web Apache.

Os testes foram divididos em duas baterias. Na primeira etapa (bateria I) foram realizados testes individuais, ou seja, com recursos de detecção, das ferramentas *ModSecurity* e UniscanWAF, especificamente habilitados para cada uma das vulnerabilidades. Na segunda etapa (bateria II), foram realizados testes de detecção com todos os recursos habilitados em ambas as ferramentas.

As vulnerabilidades *XSS*, *LFI*, *RFI*, *RCE* e *SQL Injection* foram implementadas em PHP. Foram utilizados os exemplos apresentados na seção III. A aplicação com os códigos vulneráveis foi hospedada em uma máquina virtual com sistema operacional *Linux Ubuntu 10.04.4 LTS*, servidor Web *Apache* versão 2.2.14, PHP versão 5.3.2-1ubuntu4.17, *Perl* versão 5.10.1, *ModSecurity Stable* versão 2.2.6 (com as regras de detecção *Core Rules* versão 2.2.5) e o UniscanWAF.

Para permitir a vulnerabilidade *RFI*, foram ativadas as seguintes variáveis de configuração do PHP: `register_globals`, `allow_url_fopen` e `allow_url_include`. Uma vez que a comunicação e integração entre sistemas é algo necessário e cada vez mais explorada por diferentes organizações, a ativação desses parâmetros é cada vez mais comum [10]. Questões de escalabilidade, como desempenho e tempo de resposta ao usuário, não foram levadas em consideração neste trabalho.

Em um primeiro momento, realizou-se testes no *ModSecurity*, e em seguida, no UniscanWAF. Os detalhes desse processo são descritos a seguir.

A. ModSecurity

Os resultados da primeira bateria de testes estão sumarizados na tabela I. A coluna Regra *ModSecurity* refere-se ao arquivo que habilita a regra, a coluna Vulnerabilidade refere-se a vulnerabilidade relacionada com o arquivo e a coluna Status indica se a vulnerabilidade foi ou não encontrada. Nota-se que o *ModSecurity* conseguiu detectar apenas duas das cinco vulnerabilidades, indicando que as regras referentes as vulnerabilidades não detectadas encontram-se deficientemente implementadas (para os arquivos referenciados pela coluna Regra *ModSecurity* da tabela I), ou não implementadas, no caso da vulnerabilidade *RCE*. A empresa *Trustwave* [11] apresenta regras adicionais que contemplam o ataque *RCE*, mas que devem ser adiquiridas comercialmente.

Tabela I
Testes individuais de vulnerabilidades

Regra ModSecurity	Vulnerabilidade	Status
modsecurity_crs_41 _xss_attacks.conf	*XSS*	✓
modsecurity_crs_46 _slr_et_lfi_attacks.conf	*LFI*	
modsecurity_crs_46 _slr_et_rfi_attacks.conf	*RFI*	
inexistente[1]	*RCE*	
modsecurity_crs_41 _sql_injection_attacks.conf	*SQL-Injection*	✓

A segunda bateria de testes foi realizada com todas as regras habilitadas no *ModSecurity*. O resultado dos testes pode ser visualizado na tabela II. Pode-se observar que a ferramenta detectou mais vulnerabilidades com todas as regras habilitadas. De fato, o arquivo

[1]Arquivo não encontrado para esta vulnerabilidade.

"modsecurity_crs_40_generic_attacks.conf" possibilita a detecção dos ataques de *LFI* e *RFI*, anteriormente não detectados. Este arquivo concentra vários ataques Web, entretanto, na documentção não fica claro quais tipos de ataques o arquivo em questão trata.

Mesmo assim, observou-se um comportamento diferente para diferentes entradas de parâmetros na URL para a vulnerabilidade *RFI*. Quando o formato da URL é similar a este "http://sistemaWeb.com.br/arquivo.php?arg=http://10.1.1.1/index.html", ou seja, com um endereço IP como parâmetro, o *ModSecurity* identifica a tentativa de exploração da vulnerabilidade. No entanto, quando o formato da URL é similar ao seguinte "http://sistemaWeb.com.br/arquivo.php?arg=http://www.dominiovalido.br/index.html", ou seja, sem um endereço IP como parâmetro, mas com uma URL válida, o *ModSecurity* não detecta a tentativa e explocação *RFI* do atacante. Entramos em contato com o suporte do ModSecurity para descobrir o motivo dos resultados, mas não obtivemos resposta até o momento.

B. UniscanWAF

Para o UniscanWAF foi utilizada a mesma metodologia de testes do (*ModSecurity*). Os resultados obtidos foram os mesmos nas duas baterias de testes e podem ser observados na tabela II.

C. Resultados

A tabela II apresenta o resultado dos testes efetuados com o UniscanWAF e o *ModSecurity* (com todas as regras habilitadas) no cenário proposto. Como pode ser observado, o UniscanWAF conseguiu detectar todas as vulnerabilidades apresentadas, enquanto que o *ModSecurity* não detectou a vulnerabilidade *RCE* e teve problemas com a vulnerabilidade *RFI*, conforme descrito anteriormente.

Tabela II
COMPARAÇÃO ENTRE *ModSecurity* E UNISCANWAF

Ferramenta	XSS	LFI	RFI	RCE	SQL-Injection
UniscanWAF	✓	✓	✓	✓	✓
ModSecurity	✓	✓	✓[2]		✓

Os resultados apresentados permitem concluir que o UniscanWAF é capaz de detectar em tempo real requisições que visam explorar vulnerabilidades das aplicações Web. Além disso, detecta mais vulnerabilidades que ferramentas similares, como é o caso do *ModSecurity* e possui uma arquitetura modular e flexível idêntica a do Uniscan, um scanner de vulnerabilidades *open source* com mais de 7.000 downloads de mais de 130 países [5].

V. CONSIDERAÇÕES FINAIS

Este artigo apresentou o UniscanWAF, uma ferramenta para detectar e interceptar em tempo real a exploração de vulnerabilidades, por parte dos atacantes, endereçadas à aplicações Web. Os resultados mostram a funcionalidade do mecanismo mais essencial da ferramenta, a detecção das vulnerabilidades. Além disso, foi estabelecida uma breve comparação com o *ModSecurity*, demonstrando que a solução proposta é capaz de detectar mais vulnerabilidades, dentro do conjunto analisado. Complementarmente, o UniscanWAF foi concebido como uma extensão do Uniscan, dentro dos conceitos de flexibilidade, simplicidade e extensibilidade desta ferramenta, um scanner de vulnerabilidades de sistemas Web largamente utilizado.

Entre os próximos passos podem ser incluídos a criação de *white list*, ou seja, desenvolver uma interface que obrigue o programador a validar as atualizações realizadas na aplicação , para que então as novas funcionalidades sejam aplicadas e utilizadas. Esta tarefa faz com que o ciclo de testes de segurança, mesmo com a aplicação já em uso, seja sempre executado. Por último, testes de desempenho em um ambiente de produção serão realizados para avaliar a ferramenta proposta.

[2]Detectou parcialmente a vulnerabilidade.

REFERÊNCIAS

[1] N. Teodoro and C. Serrao, "Web application security: Improving critical web-based applications quality through in-depth security analysis," in *International Conference on Information Society (i-Society)*, 2011, pp. 457–462.

[2] H. Takahashi, H. F. Ahmad, and K. Mori, "Application for autonomous decentralized multi layer cache system to web application firewall," in *10th International Symposium on Autonomous Decentralized Systems (ISADS)*, vol. 1, 2011, pp. 113–120.

[3] (2012, Jul.) Modsecurity: Open source web application firewall. [Online]. Available: http://www.modsecurity.org/

[4] D. Rocha, D. Kreutz, and R. Turchetti, "Uma ferramenta livre e extensível para detecção de vulnerabilidades em sistemas web," in *Actas de la 7a Conferencia Ibérica de Sistemas y Tecnologías de Información*, 2011, pp. 747–752.

[5] D. Rocha. (2012) Uniscan - um scanner de vulnerabilidades para sistemas web. [Online]. Available: http://uniscan.sourceforge.net/

[6] T. Oreilly, "What is web 2.0: Design patterns and business models for the next generation of software," in *Communications & Strategies*, vol. 1, 2007, pp. 17–37.

[7] (2012, Jul.) The open web application security project. [Online]. Available: https://www.owasp.org/index.php/Top_10_2010-A2

[8] B. Chess and G. McGraw, "Static analysis for security," *IEEE SECURITY & PRIVACY*, vol. 2, no. 6, pp. 76–79, Nov-Dec 2004.

[9] (2012, Jul.) The open web application security project. [Online]. Available: https://www.owasp.org/index.php/SQL_Injection

[10] D. Rocha, "Uniscan: Um scanner de vulnerabilidades para sistemas web," in *Trabalho de Conclusão de Curso apresentado à Universidade Federal do Pampa (UNIPAMPA)*, 2012.

[11] (2012, Jul.) Trustwave. [Online]. Available: https://www.trustwave.com/