Computer Science and Engineering July 2012, Special Issue: 55-65
DOI: 10.5923/j.computer.20120001.08

Uma Ferramenta Livre e ExtensÍVel Para Detecção de Vulnerabilidades em Sistemas Web

Douglas Rocha[1,*], Diego Kreutz[1], Rogério Turchetti[2]

[1]Grupo de Pesquisa em Sistemas de Informação , GPSI Universidade Federal do Pampa , UNIPAMPA Alegrete, RS, Brasil
[2]Colégio Técnico Industrial de Santa Maria , CTISM Universidade Federal de Santa Maria , UFSM Santa Maria, RS, Brasil

Abstract The increasing number of intrusions and data thefts on online systems is one of the triggers of the growing concern about security inside organizations. Nowadays, dynamic and extensible detection tools are required and critical to detect and diagnose vulnerabilities in Web systems. In this paper we present the development and evaluation of a vulnerability scanner for online systems. Unlike most existing tools, it is free and open source, available at SourceForge, and has a modular and extensible architecture. The achieved results show that the proposed tool, called Uniscan, is able to better detect and diagnose vulnerabilities such as LFI, RFI and RCE.

Keywords Vulnerability Detection, Security, Web Systems, Free , Open Source Tools, Flexible , Extensible Architecture

1. Introdução

Sistemas Web (online) são alvos fáceis e frequentes para os atacantes,principalmente por oferecerem serviços ligados diretamente à Internet[1, 2, 3, 4,5].Algumas pesquisas indicam que o número de vulnerabilidades e ataques é cada vez maior e mais frequente em serviços e sistemas Web[3,4, 5,6,7]. Mesmo aqueles sistemas que oferecem serviços onlinemenos atraentes são alvo de ataques, pois os servidores que hospedam os sistemas online vulneráveis servem para os atacantes como pontes para consequentesataques às vítimas mais visadas, como aquelas que possuem dados e informações financeiras de clientes. Logo, é visível a crescente necessidade de boas práticas de segurança no contexto de sistemas Web[8, 9]. Isso inclui, com especial atenção, a detecção e o diagnóstico de vulnerabilidades nos sistemas. Este é o primeiro passo para um bom ciclo de desenvolvimento, evolução e manutenção dos sistemas online. A detecção deve ainda considerar, nos processos de testes de segurança, casos em que o comportamento da aplicação seja imprevisível aos dados de entrada[10]. Esta característica torna os testes de segurança uma tarefa não trivial.

Testar manualmente todas as possíveis vulnerabilidades de sistemas Web é uma tarefa complexa e muito dispendiosa, sendo considerada como impraticável. Devido a esse fato pode-se observar, tanto no contexto científico quanto pro-fissional, a eminente necessidade de ferramentas de análise de segurança que executem testes automáticos, como os diferentes scanners de vulnerabilidades[1, 3, 5, 11, 12]. Boa parte dessas soluções são baseadas em banco de dados de vulnerabilidades conhecidas e utilizam rastreadores, também denominados de*crawlers*, para encontrar os *links* e subsistemas do sitio Web a ser analisado e diagnosticado.

A maioria das ferramentas existentes não é capaz de encontrar todas as vulnerabilidades dos sistemas online[11]. Os próprios resultados da varredura para um mesmo sistema alvo, na forma de dados estatísticos e relatórios, variam entre os diferentes*scanners* de vulnerabilidades[11, 12, 13]. Algumas dessas ferramentas detectam mais vulnerabilidades que outras, tanto em termos quantitativos para uma mesma vulnerabilidade, com múltiplas e diversificadas incidências num mesmo sistema, quanto em relação a variabilidade de defeitos detectáveis. Adicionalmente, estudos recentes demonstram que a média de detecção de vulnerabilidades das principais soluções existentes é relativamente baixa[11].

Boa parte dos *scanners* de vulnerabilidades atua na detecção devulnerabilidadescomo *SQL Injection* e *Cross-site scripting* (XSS)[2, 3, 14].Não obstante, a maior parte das ferramentasexistentessãocomerciais, levando a uma carência de ferramentaslivres/gratuitas eque sejam capazes de contribuir efetiva e eficazmente na tarefa de diagnosticar e melhorar a segurança dos sistemas Web[3,11].

Estudosmostramqueexistemdiferentestipos de ataques contra sistemas Web. Entre osmaisfrequentesestãoa inclusão remota de arquivos, SQL *Injection* e *cross-scripting*[15]. Não obstante, existem outros tipos de vulnerabilidades, comoformasavançadas e de segundaordem de *XSS* e SQL-i, variações de *Cross-Channel Scripting* e*Cross-Site Request Forgery*, que aindaestãodescobertaspelasferramentasexisten

* Corresponding author:
douglas.poerschke@gmail.com (Douglas Rocha)
Published online at http://journal.sapub.org/computer

tes[11].Osresultadosde diferentes pesquisas[8, 9, 11, 12, 13] apontamqueaindaháespaçoparainvestigação, inovação e desenvolvimentona área de *scanners* de vulnerabilidadesvoltados paraaplicações Web.

Este artigoapresenta o desenvolvimento e os resultados de um*scanner* de vulnerabilidades livre e gratuito, denominado Uniscan[16], para ser utilizado por profissionais responsáveis pelo desenvolvimento ou segurança de sistemas Web. O objetivo da ferramenta éautomatizar o processo de detecçãoe diagnósticode vulnerabilidades nesses ambientes. Comoprincipais pontos positivos do *scanner*podem ser destacados:

(a) uma ferramenta livre e gratuitamente disponível;

(b) uma arquitetura simples, modular e extensível, per-mitindo a rápida e dinâmica incorporação de novas funcio-nalidades;

(c) incorporação de *plug-ins* para detectar vulnerabilidades como RFI (*Remote File Inclusion*), LFI (*Local File Inclusion*), RCE (*Remote Command Execution*), SQL-i, XSS e Blind SQL-i;

(d) arquitetura *multi-thread* para acelerar o processo de análise e diagnóstico detalhado dos sitios Web alvos;

(e) configurabilidade, permitindo ao utilizador selecionar os *plug-ins*de análise desejados para cada sistema alvo;

(f) módulo de *stress* para testar a resistência das aplicações Web em diferentes níveis, como a configuração dos servidores Web responsáveis por manter os sistemasdisponíveis; e

(g) utilização de metodologia híbrida, através da combina-ção de testes dinâmicos e estáticos.

Vulnerabilidades como RFI, LFI e RCE são pouco exploradas e analisadas pela maioria das soluções existentes. Sendo assim, detectar eficazmente essas três faltas, que podem colocar sistemas Web em risco, foi um dos objectivos primários do desenvolvimento do Uniscan. Consequente-mente, o *scanner* pode ser considerado uma iniciativa que contribui com o desenvolvimento de detectores de vulnerabi-lidades capazes de eficientemente detectar vulnerabilidades como LFI, RFI e RCE.

A maioria dos *scanners* existentes, diferentemente da ferramenta desenvolvida, não possuem uma arquitetura modular e nem suporte a uma metodologia híbrida, utilizando tanto testes estáticos quanto dinâmicos na varredura dos sistemas alvo.A arquitetura modular, flexível e extensível do Uniscanextendea sua utilização e extensão para diferentescenários e casos de aplicação.

Os resultados atingidos demonstram como a ferramenta desenvolvida contribui com o processo de detecção de vulne-rabilidades em sistemas reais. O Uniscan está disponível para *download*em *http://sourceforge.net/projects/uniscan/*.A fer-ramenta nasceu e evoluiu como resultado de um trabalho acadêmico de pesquisa e desenvolvimento. A versão 5.3 é constituída por cinco grandes módulos e vinte *plug-ins*.Até o momento da escrita deste artigo, foram registrados mais de 6.000 *downloads*da ferramenta, tendo como origem mais de 100 paises[17].

O restante do artigo está organizado como segue. A próxima seção apresenta uma breve descrição e caracterização de vulnerabilidades e metodologias de detecção. Na sequência, seção III,sãodiscriminadas a arquitetura e a implementação da ferramenta.Os experimentos e resultados analíticos e estatísticos são apresentados e discutidos na seção IV. Por fim, as inferências finais compõe a última seção.

2. Vulnerabilidades e Metodologias de Detecção

Esta seção introduz as vulnerabilidades e metodologias de detecção. A primeira sub-seção apresenta exemplos concretos de vulnerabilidades em códigos PHP e Perl, que são alvos iniciais de investigação deste trabalho. A segunda sub-seção aborda as metodologias estática e dinâmica de varredura dos sitios alvo.

2.1. Vulnerabilidades em Códigos Web PHP e Perl

RFI, LFI e RCE são algumas das vulnerabilidades mais comuns que programadores tendem a inconscientemente implementar sem ter ciência dos riscos envolvidos tanto para as aplicações Web quanto para os dados e sistemas opera-cionais hospedeiros. Elas estão entre as vulnerabilidades presentes com relativa frequência em sistemas Web[15] e representam um nível de risco crítico[24].Conforme estatísticas do Zone-H[25], que registra ataques a servidores e sites na Internet, a maior parte dos servidores Web que foram invadidos em 2010 foi por meio de uma dessas três vulnera-bilidades. Isso evidencia e justifica a importância de detectar, diagnosticar e tratar, da melhor forma possível, tais vulnerabi-lidades.

A vulnerabilidade RFI ocorre quando a validação dos dados, em algumas funções e procedimentos dos códigos dos sistemas Web, inexiste ou está incompleta. Um exemplo de validação é o processo de verificar se o arquivo que está sendo incluído no código realmente está no diretório especificado pela aplicação Web. Arquivos, em códigos PHP, são comu-mentemente passados como parâmetros e utilizados em fun-çõescomo *include()*; *include_once()*; *require()*; e *require_once()*. Geralmente, quando alguma destas funções é uti-lizada, o objetivo é incluir um arquivo local, armazenado em algum diretório especifico, ao código fonte do PHP[24], agregando código externo.

A vulnerabilidade RFI pode ser explorada por um atacante quando os parâmetros do código PHP não estão sendo devi-damente verificados. Nesses casos, o atacante pode passarum arquivo qualquer como parâmetro. Como exemplo, considere o endereço eletrônico, conhecido como URL,*http://www.exemplo.com/index.php?inc=topo.php*.O atacante pode tentar manipular os dados que são passados como parâmetros para o arquivo *index.php*.Neste caso, ao invés de incluir o arquivo *topo.php*no parâmetro *inc*, o atacante poderia incluir um endereço Web de um código malicioso, como *inc=http://www.sitecracker.com/cmd.txt?c*

md=id.Logo,quando o inter-pretador PHP analizar o arquivo *index.php*, o arquivo *cmd.txt,* hospedado em *www.sitecracker.com*, será carregado e incluído no processamento do código. O próprio PHP cria e gerencia a conexão com o sitio malicioso para a *download* do arquivo remoto.

A Figura 1 ilustra o processo de exploração da vulnera-bilidade RFI. O *cracker*, atacante, pode utilizar tanto um navegador Web bem como *exploits* para conseguir explorar essa vulnerabilidade nos *scripts* PHP do site *www.example.com*, hospedados no servidor Web do respectivo domínio. Para isso ele necessitará de outro site (exemplo: *www.sitecracker.com*), hospedado em outro servidor Web, para disponibilizar o seu arquivo malicioso (exemplo: *cmd.txt*), que servirá para explorar a vulnerabilidade RFI. O resultado do processo é a abertura de uma linha de comando (uma espécie de *shell*) via navegador Web, onde, a partir da manipulação dos parâmetros, é possível a execução de comandos no sistema operacional do servidor Web a partir do código PHP.

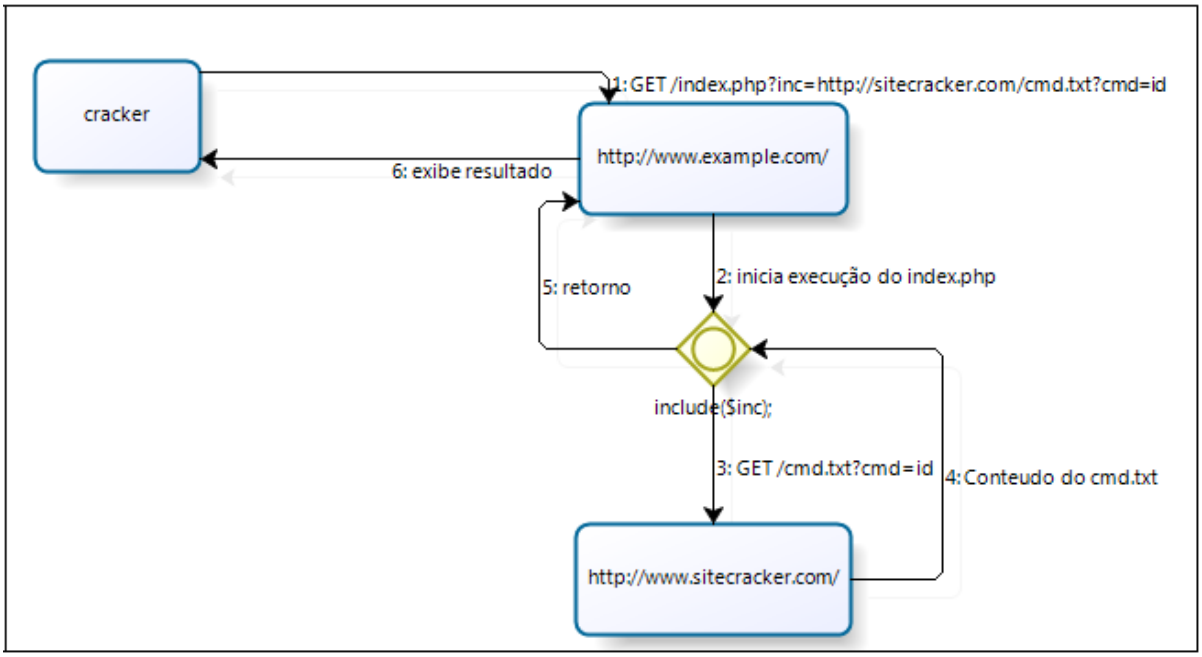

Figura 1. P rocesso de exploração da vulnerabilidade RFI

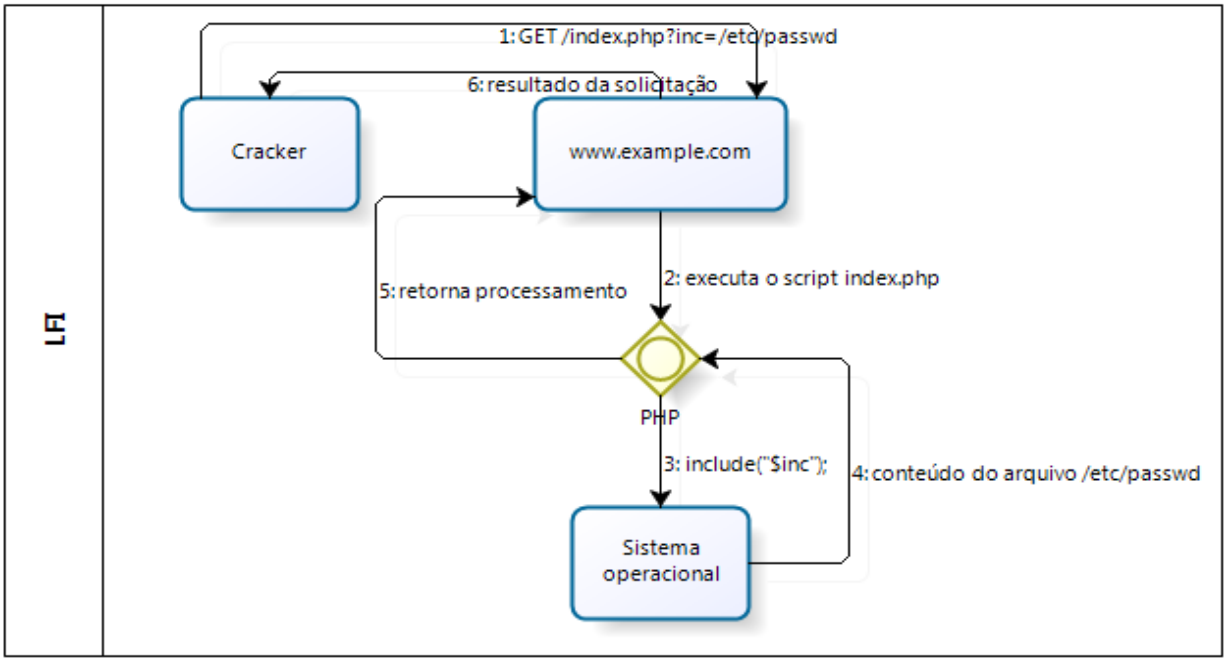

Figura 2. Processo de exploração da vulnerabilidade LFI

Com frequência, os servidores Web executam em modo usuário, ou seja, não possuem privilégios para fazer qualquer coisa no sistema operacional hospedeiro. Mesmo assim, o atacante terá acesso a todas as informações que os níveis de permissão do servidor Web permitem ter acesso. Entretanto, no caso de o servidor Web estar executando em modo super-usuário (*root*) a situação se torna ainda mais crítica, pois todos os comandos que o atacante utilizar serão executados com permissões de *root*, tendo acesso completo e irrestrito ao sistema operacional. Em ambos os casos o atacante pode utili-zar o sistema local para atacar outros sistemas internos ou externos aos domínios da entidade sendo atacada.

A vulnerabilidade LFI, assim como a RFI, também é explorada através dos parâmetros de funções como *include()*, *include_once()*, *require()*, e *require_once()*. Contudo, diferen-temente da RFI, a inclusão é de arquivos locais e não de arqui-vos remotos. O resultado desta vulnerabilidade éo acesso a arquivos locais ao sistema operacional hospedeiro, como o arquivo */etc/passwd*, que contém informações dos usuários do sistema GNU/Linux (*index.php?inc=/etc/passwd*).

A Figura 2 apresenta o processo de exploração da vulnera-bilidade de LFI. O atacante pode utilizar um navegador Web ou um *exploit* para explorar essa vulnerabilidade nos *scripts* PHP do site *www.example.com*. O *cracker* faz uma requisição *GET index.php?inc=/etc/passwd* para o endereço *www.example.com*, o que provoca a inclusão do arquivo *passwd*na execução do código PHP.

Através do processo ilustrado na Figura 2, um atacante pode explorar a vulnerabilidade de inclusão local de arquivos através de um simples navegador, digitando uma URL como*http://www.example.com/index.php?inc=/etc/passwd*. Com es-sa simples ação o conteúdo do arquivo */etc/passwd* será incluído no arquivo *index.php*e será exibido no navegador Web do atacante. Esses dados podem permitir, por exemplo, a um atacante lançar um ataque de força bruta para tentar descobrir (decifrar através de tentativa e erro) a senha de cada usuário listado no arquivo */etc/passwd*.

Outro exemplo de como explorar a vulnerabilidade LFI é através da utilização dos recursos e arquivos de registro de atividades (logs) do servidor Web. O servidor Web Apache, assim como outros, pode registraros logs dos acessos. Uma forma de utilizar esse recurso é forçar a gravação de logs (comandos) que levarão, posteriormente, a execução de comandos pela inclusão do arquivo de log no cógido PHP. Uma forma de forçar o registro de comandos é pelo simples método de tentar acessar páginas inexistentes com os parâmetros desejados (exemplo: */arquivoXYZ.php?id=<?php%20system("uname -a");?>*).Isto fará com que a requisição seja gravada no *access_log* do apache. A partir deste ponto,basta incluir o arquivo *access_log* na página vulnerável para que o comando *uname -a* seja executado no sistema hospe-deiro. A URL para exploração da vulnerabilidade ficaria da seguinte forma: *http://www.example.org/index.php?inc=/var/log/apache/access_log*.Quando incluído no código PHP o arquivo *access_log* do Apache, este executará o comando *uname –a* no sistema operacional e mostrará o resultado do comando no navegador do atacante.

Na execução remota de comandos (RCE) é possível executar chamadas ao sistema hospedeiro através de funções como *system* e *exec*. Aslinguagem *Perl*e PHP, em especial, podem apresentar vulnerabilidades quando existiremparâ-metros sem as devidas validações e verificações. As funções *system*, *open* e *exec* da linguagem *Perl* podem ser classificadas entre as maiores potencializadoras deste tipo de vulnera-bilidade. Em síntese, o problema reside na possibilidade que o código fonte fornece para executar comandos de chamadas ao sistema sem o tratamento adequado dos parâmetros passados à essas funções.

Uma simples linha de código, como a ilustrada a seguir, utilizando a função *open()* do Perl, pode levar a uma vulne-rabilidade crítica. O uso da função é geralmente destinado à leitura e escrita em arquivos através de operadores como "<", para leitura, e ">", para escrita. No entanto, o uso da função sem os operadores podem levar a outros problemas, como a execução de comandos quaisquer.

Exemplo de código Perl vulnerável:

open($arq, "$arquivo");

No exemplo apresentado, a falta do operador de leitura ("<") ou escrita (">") permite utilizar outros operadores, como *pipe* ("|"), pois o nome do arquivo será recebido por parâmetro. O operador *pipe* serve para mudar a entrada padrão da função *open()*. No caso, a entrada padrão da função *open()* pode ser alterada para a saída (da execução) padrão de qual-quer outro comando.

A seguir é apresentado um exemplo de exploração desta vulnerabilidade. Ele consiste em passar como parâmetro um *pipe,*juntamente com um comando a ser executado, ao invés do nome do arquivo.

Exemplo de exploração do código Perl vulnerável:

http://www.ex.com/script.cgi?arquivo=|cat%20/etc/passwd|

Com isso, o código Perl, ao executar a função *open()*, abrirá o *pipe* e executará o comando *cat /etc/passwd*.Isso fará com que o conteúdo do arquivo seja exibido na tela do navegador do atacante. O resultado para a execução do código Perl ficaria:*open($arq, "|cat /etc/passwd|");*.

Na Figura 3 é ilustrado um segundo exemplo, agora PHP, de código vulnerável a RCE. Ao acessar a página que contém esse código o usuário poderá passar por parâmetro o valor da variável *nLinhas*. A princípio, isso significa que o funciona-mento do comando *tail* poderá, pegando mais ou menos linhas do *arquivo.txt*. Como é realizada uma chamada *system*, com execução de comandos semelhante a um *shell*, é possível executar mais de um comando numa única linha. Para isso existe um separador especial ";", que permite ao usuário introduzir vários comandos na sequência em uma única linha de execução. Isso significa que é possível incluir comandos além do valor da variável *nLinhas*. Acrescentando ";" seria possível executar uma sequência qualquer de

comandos devido à falta de verificação do valor da variável *nLinhas*. A seguir é apresentando um exemplo de inclusão do comando *cat*.

```
1. <?php
2. $nLinhas = $_GET['linhas'];
3. system("tail arquivo.txt -n $nLinhas");
4. ?>
5.
```

Figura 3. Exemplo de código PHP vulnerável a RCE

Exemplo de alteração do valor da variável*nLinhas*:
http://www.example.com/index.php?nLinhas=15

Exemplo de alteração do valor da variável *nLinhas*com a inclusão de um segundo comando a ser executado pela função *system*:

http://www.example.com/index.php?nLinhas=15; cat /etc /passwd

O resultado da chamada *system* ficaria:

system("tail arquivo.txt –n 15 ;cat /etc/passwd");

No momento em que a função *system* for invocada, tanto o comando *tail* quanto o comando *cat* serão executados sequen-cialmente.No exemplo, o comando *cat* irá trazer o conteúdo do arquivo */etc/passwd* para a tela do navegador do atacante. Uma das formas de resolver o problema de falta de verificação da variável *nLinhas* é através da inclusão de uma conversão de tipo de dados, também conhecida como *casting*, com o objetivo de forçar que o valor de *nLinhas* seja um número inteiro, ou seja, evitando que venha a ser uma cadeia de caracteres (*string*) com instruções potencialmente maliciosas e prejudiciais ao sistema Web.

As vulnerabilidades LFI, RFI e RCE são exploradas e diagnosticadas de forma similar em códigos PHP e Perl. Estas duas linguagens são amplamente utilizadas na construção de sistemas Web. Esse foi um dos motivos daescolhadessas linguagens de programação como o alvos iniciais para o desenvolvimento dos primeiros *plug-ins* do Uniscan.

2.2. Metodologias de Detecção

Existem basicamente dos tipos de testes utilizados na detecção de vulnerabilidades em sistemas Web, o estático e o dinâmico. A metodologia estática possui algumas vantagens em relação ao método dinâmico, entre elas: (a) ideal para testes de softwares de prateleira (softwares que não são feitos sob encomendas), pois sem uma customização, os nomes de arquivos e variáveis do sistema Web são conhecidos; (b) verifica a existência de arquivos que não estão ligados (exemplo: mesmo não existindo um *link* para o arquivo X no sistema Web, este será testado se estiver no banco de dados do *scanner*). Como desvantagens podem ser apontadas: (i) as diversas requisições de arquivos que não existem no servidor Web; e (ii) a falta de capacidade para analizar páginas de um software customizado, o que é comum em sistemas online.

Na metodologia dinâmica o *scanner*vasculha o sistema e servidor Web alvo a procura de todos os arquivos existentes. Em outras palavras, antes de iniciar os testes de vulnera-bilidades, o próprio *scanner*criaa lista dos arquivos existentes no servidor Web através de um *crawler* (rastreador). Todas as verificações de vulnerabilidades são realizadas sobre a lista de arquivos descobertos pelo *crawler*. A vantagem do método dinâmico é a possibilidade de descobrir erros em arquivos distintos nos sistemas Web, sendo a melhor opção para var-reduras em serviços e sistemas customizados.A desvantagem é o fato de não conseguir detectar arquivos não ligados e, consequentemente, não encontrados pelo *crawler*. No caso do método estático existe uma base de dados com conjuntos de arquivos pré-definidos, os quais serão testados no sistema Web, havendo ou não ligação com o sistema alvo. Portanto, a combinação dos dois métodos pode tirar proveito das vanta-gens de ambos.

3. Arquitetura do Uniscan

Um *scanner* de vulnerabilidades possui módulos essen-ciais, como o**rastreador** e o**verificador** de vulnerabilidades. O rastreador tenta localizar todos os arquivos e *links* dentro do sitio alvo. Por outro lado, o verificador de vulnerabilidades é responsável por realizar diferentes tipos de testes sobre cada arquivo ou link encontrado. Cada tipo de teste pode ser incorporado à ferramenta como um *plug-in*, tornando-a fle-xível e extensível.

Outra característica importante incorporada ao Uniscan é o suporte a multiprogramação (*multi-threads*), aumentando o desempenho e a capacidade de execução de verificações si-multâneas em sites complexos, que podem conter centenas, ou mesmo milhares, de arquivos e *links* a serem testados.

A Figura 4 ilustra arquitetura projetada e implementadano Uniscan, em sua versão 5.3. Como pode ser observado, ela é composta por cinco grandes módulos com funções dife-renciadas. Os módulos contam com o suporte de *plug-ins*, ou seja, componentes menores e acopláveis ao módulo, trazendo extensibilidade e flexibilidade ao *scanner*.

3.1. Descrição dos Módulos

O primeiro módulo, *setup*, é responsável pela configuração e inicialização do Uniscan. Ele realiza a leitura dos arquivos de configuração e prepara os parâmetros para os outros módulos, como a definição do número de *threads* a serem ativadas no sistema.

O módulo*discovery engine* gera listas de sites, do domínio alvo, utilizando máquinas de busca, como *Google* e *Bing*. Estas listas são utilizadas pelo *scanner* no processo de verificação das vulnerabilidades. Os resultados de buscadores como o *Google* podem ser interessantes pelo fato de, even-tualmente, conseguirem mapear arquivos ou subsistemas ilhados dentro do sitio alvo, que possam não ser localizadas pelo *crawler* do *scanner* devido a quebra ou falta de refe-rências explícitas.

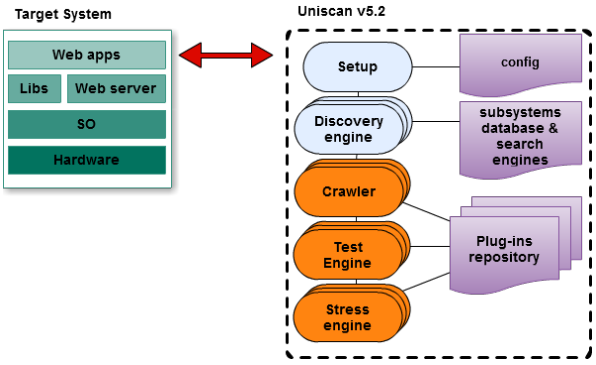

Figura 4. Visão geral da arquitetura do Uniscan

Um dos principais módulosda arquitetura é o *crawler* (rastreador), sendo responsável por identificar todas as páginas do sistema alvo, através de um processo exaustivo de varredura. O *crawler* conta ainda com um repositório de *plug-ins* para a coleta de informações sensíveis, como email de administradores e comentários no HTML,que possam tornar o sistema vulnerável pela potencialidade de auxiliarem no processo de penetração do atacante. A Figura 5 ilustra o fluxo de operação do *crawler*.

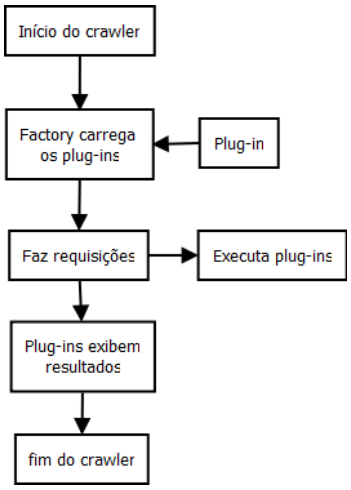

Figura 5. Diagrama do crawler

Omódulo de testes, ou *test engine*, é responsável pelo carregamento, gerenciamento e execução dos *plug-ins* de testes de vulnerabilidades. Cada teste é representado por um *plug-in* armazenado no respositório. A principal tarefa do módulo é carregar e executar todos os *plug-ins* requisitados pelo usuário. A Figura 6descreve o funcionamento, na forma de diagrama, do módulo de testes.

O *stress engine* é responsável por carregar, gerenciar e executar os *plug-ins* de testes de *stress* do sistema e do servidor alvo. O objetivo é verificar a estabilidade e disponibilidade de um sistema Web sob variadas cargas de atividade, visando identificar problemas de projeto, imple-mentação ou configuração dos sistemas.

Os recursos e as funcionalidades disponibilizadas pelos módulos da arquitetura do Uniscan são implementados através de *plug-ins*. Sendo assim, a arquitetura e o repositório de *plug-ins* trás benefícios como:

(a) a não necessidade de uma nova versão do *scanner* a cada erro corrigido em um dos *plug-ins*;

(b) não é necessário lançar uma nova versão do *scanner* para adicionar novos testes de vulnerabilidades;

(c) o usuário pode desenvolver seus *plug-ins* sem a neces-sidade de modificar o núcleo da ferramenta;

(d) maior diversidade e flexibilidade, pois podem ser criadas e testadas diferentes variantes de um teste para uma vulnerabilidade em específico;

(e) maior extensibilidade e longevidade, pois novas fun-cionalidades e testes, inicialmente não previstos, podem ser facilmente agregados à ferramenta; e

(f) maior adaptabilidade, pois a ferramenta pode ser facil-mente personalizada para diferentes cenários e casos de teste.

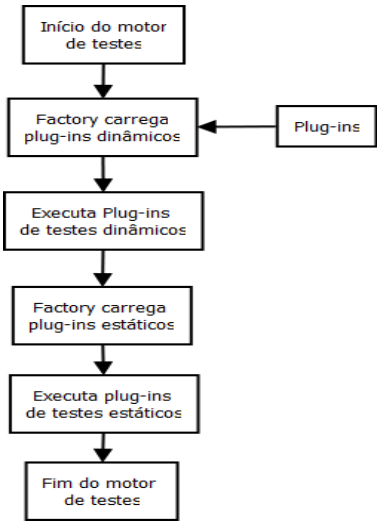

Figura 6. Diagrama do motor detestes

3.2. Implementação

O Uniscan foi desenvolvido na linguagemde programação *Perl*, que é uma linguagem presente em vários sistemas ope-racionais. Como consequência direta, a ferramenta é um *scanner* multiplataforma. Os módulos e plug-ins são estrutu-rados em módulos Perl com a utilização do recurso de *factory* da linguagem.

O módulo *Factory.pm* é responsável por carregar cada *plug-in* do *scanner* e retornar um objeto deste para a área onde ele foi chamado. Devido a essa função, ele é um dos núcleos importantes da implementação da ferramenta.

O módulo *Crawler.pm* realiza a navegação e a coleta de dados no sistema alvo. Para cada requisição de entrada, todos os *plug-ins* habilitados são executados, recebendo como parâ-metros a URL e o seu respectivo conteúdo. Ao final da exe-cução, o método *showResults* de cada *plug-in* é invocado para exibir o resultado da correspondente análise.

O módulo *Scan.pm* é o motor de testes do *scanner*. A sua entrada é a lista de URLs encontradas pelo *crawler*.A função do módulo é executar todos os testes (*plug-ins*) requisitados pelo usuário, incluindo testes dinâmicos ou estáticos.

A Figura 7 apresenta o diagrama da implementação do Uniscan. Ela ilustra a ordem e as principais co-relações dos módulos e repositórios de plug-ins.

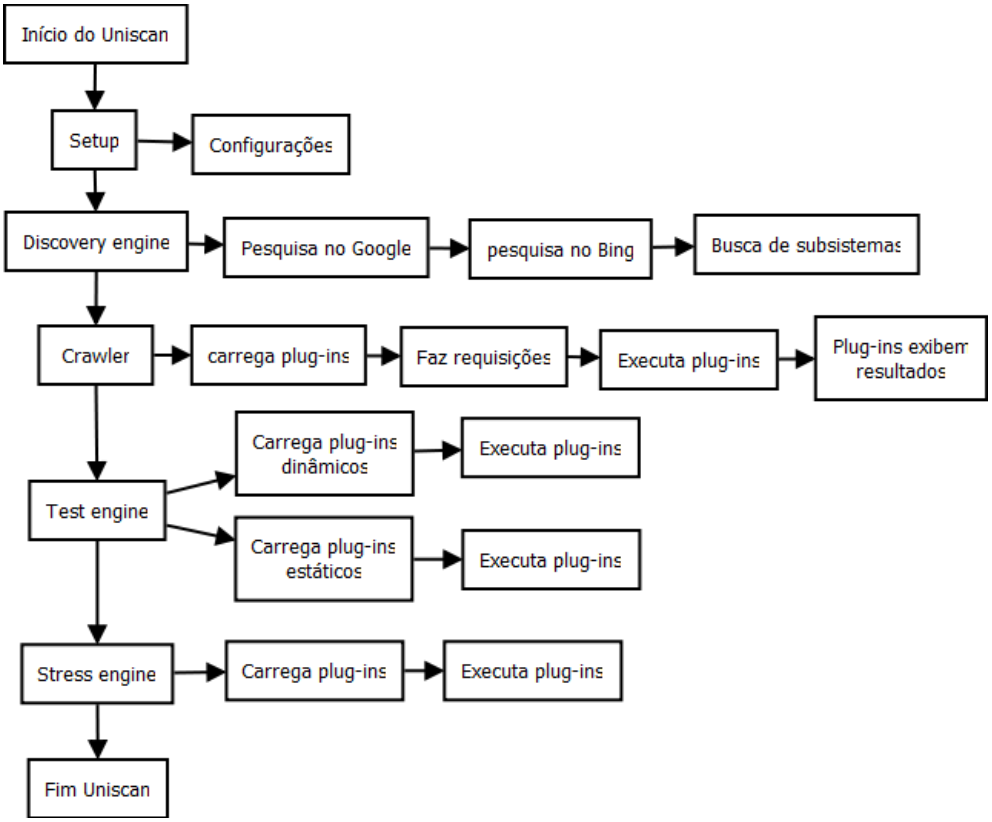

Figura 7. Diagrama geral da implementação do Uniscan

4. Experimentos Realizados

Nesta seção são brevemente apresentadas as principais ferramentas similares utilizadas nos testes. Adicionalmente, os cenários de testes são descritos, bem como os resultados obtidos com os experimentos realizados. Foram definidos e utilizados dois tipos de cenários, os controlados e os reais. O primeiro grupo teve como função uma validação e melhor avaliação e comparação inicial das ferramentas. Por outro lado, o segundo grupo visou verificar a efetividade das ferramentas em cenários reais, ou seja, sistemas Web (de empresas e outras organizações) em linha de produção.

4.1. Ferramentas Similares

Foram escolhidas e utilizadas seis ferramentas similares para análise de características e funcionalidades, bem como comparação de resultados em cenários de testescontrolados e reais. As ferramentas escolhidas foram: Acunetix[23], Nikto[22], Powerfuzzer[21], Shadow Security Scanner (Shadow)[20], WebCruiser[19] e Websecurity[18].As razões da escolha podem ser definidas como: (a) acesso livre e gratuito a uma cópia do *scanner*para análise e avaliação; (b) similaridade de funcionalidades e objetivos; e (c) percepção de utilização no mercado. O primeiro item foi um dos fatores determinanentes na escolha das ferramentas. Adicionalmente, foram utilizadas também outras três ferramentas em dois casos específicos de teste.

No conjunto das ferramentas avaliadas, o Acunetix é consi-derado uma dos melhores *scanners* de vulnerabilidades do mercado[11]. Com isso, é possível concluir que a escolha das ferramentas é adequada aos objetivos pretendidos, como a comparação de funcionalidades e capacidade de detecção dos tipos de vulnerabilidades alvo do Uniscan.

A Tabela 1 apresenta os resultados da comparação inicial de funções e recursos dos *scanners* de vulnerabilidades. O "X" indica a capacidade de detecção, para o caso das vulnera-bilidades RFI, LFI e RCE, e a utilização ou não dos modelos de teste estático e dinâmico.

Tabela 1. Funções e recursos dos *scanners*

Scanner	RFI	LFI	RCE	Dinâmico	Estático
Acunetix 6.5 free				X	
WebCruiser				X	
Shadow				X	X
Websecurify 0.7				X	
Nikto v2.1.3				X	X
PowerFuzzer	X	X		X	
Uniscan	X	X	X	X	X

Para a avaliação das ferramentas, sendo algumas delas proprietárias e de código fechado, foi preparado um cenário empírico, constituído de um servidor Web Apache e três arquivos PHP com as formas mais simples de cada uma das vulnerabilidades. No caso dos testes dinâmico e estático, a análise foi realizada através da atividade decada ferramenta no servidor Web, observando os registros doservidor Web Apache.

Como pode ser observado na Tabela 1, além do Uniscan, apenas o *scanner* PowerFuzzer detectou as vulnerabilidades RFI e LFI. No entanto, apenas o Uniscan foi capaz de

detectar a presença das três vulnerabilidades alvo. Por fim, apenas três ferramentas fazem uso simultâneo das metodologias de testes estáticose dinâmicos, o Shadow, o Nikto e o Uniscan.

4.2. CenÁRios Controlados

Os cenários controlados possuem um número de vulnera-bilidades conhecidas, ou seja, os resultados possíveis e espe-rados dos *scanners* são previamente conhecidos. O cenário controlado 1contem três vulnerabilidades: RCE, RFI e LFI. O cenário controlado 2 é um sistema de *blog*que contém duas vulnerabilidades, uma *XSS* e outra*Blind SQL Injection*. Os dois cenários foram preparados na plataforma *Linux Ubuntu Server 10.04.3,* com servidor Web Apache 2.2.15, PHP versão *5.3.3-1ubuntu9.3* e *Perl* versão *5.10.1*.

Tabela 2. Resultados dos testes nos cenários controlados 1 e 2

Cenário Controlado	1			2	
Scanner	RFI	LFI	RCE	XSS	Blind SQL-i
Uniscan	1	26	6	0	1
Acunetix	1	2	0	3	6
Nikto	0	0	0	0	0
WebCruiser	0	0	0	0	0
Shadow	0	2	0	0	0
Websecurify	0	0	0	0	0
PowerFuzzer	1	1	0	0	0

Os números da Tabela 2 representam o número de combi-nações de testes detectados por cada ferramenta sobre as vulnerabilidades detectadas. Como exemplo,pode ser citada a vulnerabilidade LFI. O Uniscan realiza 43 testes distintos (detectou no sistema alvo apenas 26 formas de exploração) para esta vulnerabilidade, enquanto que o Acunetix realiza 5 (detectou duas formas de explorar o sistema alvo).

Como é possível observar, para os resultados do cenário controlado 1, o Uniscan foi o único *scanner* que conseguiu detectar as três vulnerabilidades presentes. O Acunetix e o PowerFuzzer foram capazes de detectar as vulnerabilidade RFI e LFI.

Outro aspecto a observar é o fato de os *scanners* Shadow e PowerFuzzer terem detectado as vulnerabilidades LFI e RFI, o que não ocorreu nos testes realizados para a Tabela 1. Isso comprova, mais uma vez, a variabilidade nos resultados para uma mesma ferramenta, executada em cenários diferentes, ou entre os diferentes scanners, conforme comprovado por algu-mas pesquisas[11, 12, 13].

O segundo cenário continha apenas duas vulnerabilidades, sendo uma XSS e outra Blind SQL-i. A ferramenta Acunetix obteve os melhores resultados, conseguindo explorar de dife-rentes formas as duas vulnerabilidades. O Uniscan detectou apenas a vulnerabilide Blind SQL-i devido ao fato de ainda não possuir um interpretador *Javascript*, o que reduz o escopo de pesquisa dentro do sistema alvo. Um dos trabalhos futuros previstos para a ferramenta é a inclusão de interpretadores para linguagens de scripting como *Javascript* e *Actionscript*.

A Tabela 3 apresenta os resultados dos cenários contro-lados 3, 4 e 5. O cenário 3 representa um Plugin Annonces versão 1.2.0.0[29] para Workpress versão 3.3.2. Este plugin contém vulnerabilidades LFI conhecidas[32,33]. O cenário 4 representa o sistema Supernews versão 2.6[30]. Este sistema Web contém uma vulnerabilidade de SQL Inject que pode ser explorada de diferentes formas[34]. O cenário 5 representa o sistema Membis versão 2.0.1[31]. Este sistema contém vulne-rabilidades Cross Site Scripting (XSS) e SQL Injection[35]. Todos os três sistemas dos cenários 3, 4 e 5 foram instalados e avaliados sobre uma plataforma Ubuntu Server 11.04 com servidor Web Apache 2.2.17.

Tabela 3. Resultados dos testes nos cenários controlados 3, 4 e 5

Cenário Controlado	3	4	5	
Scanner	LFI	SQL-i	XSS	Blind SQL-i
Uniscan	1	3	37	1
Acunetix	0	3	4	4
OpenVAS	0	0	0	0
Arachni	0	1	1	1
W3af	0	0	0	0

Nos cenários 3, 4 e 5, além do Uniscan e do Acunetix, foram testados três *scanners* de vulnerabilidades adicionais, sendo eles o OpenVAS[26], Arachni[27] e W3af[28]. Como pode ser observado, mais uma vez, existem variações entre as ferramentas. Entretanto, considerando os testes empíricos e estatísticos realizados e apresentados neste artigo, as ferra-mentas Acunetix e Uniscan possuem um desempenho superior as demais na detecção e diagnóstico de vulnerabilidades em sistemas Web.

4.3. CenÁRios Reais

A utilização de cenários reais serve para comparar os resultados de todos os *scanners* sobre cenários do dia-a-dia, suportados por diferentes sistemas operacionais, como Windows Server, GNU/Linux e FreeBSD. A linguagem de programação também varia de um cenário para outro, incluindo PHP, ASP e Perl. Visando manter a privacidade dos sites testados, os nomes e endereços serão omitidos.

Como cenários reais, foram escolhidos aleatoriamente 29 sites, servindo de alvo para os *scanners* realizarem suas análises. Os sites estão distribuídos na Internet e localizados em diferentes países. Cada*scanner* foi executado para cada um dos 29 sites.Do conjunto de sitios analizados:(a) apenas 24.13% não apresentaram nenhuma vulnerabilidade; (b) 20.68% apresentaramas vulnerabilidades LFI, RFI ou RCE; (c) 75.86% continham vulnerabilidades XSS; (d) 24.13% apresentaramproblemas de SQL-i; e (e) 41.37% continham a vulnerabilidade Blind SQL-i.

A seguir são apresentados dois exemplos, dentro dos 29 sitios avaliados, de sistemasalvo utilizados nos testes. As tabelas apresentam os resultados obtidos por cada um dos *scanners*.

O site A, cujos resultados são apresentados na Tabela 4, é um site sobre a cultura e o comércio chinês. O site utiliza a linguagem de programação Perl e é suportado por um

servidor Web Apache 1.3.34. Como é possível observar, o Uniscan conseguiu detectar mais vulnerabilidades do que os demais *scanners*.Isto se deve ao fato do Uniscan ter mais e diferentes combinações de testes de vulnerabilidade do que as outras ferramentas. Além disso, apenas o Uniscan foi capaz de dete-ctar a vulnerabilidade RCE.

Tabela 4. Vulnerabilidades detectadas no site A

Scanner	RFI	LFI	RCE	XSS	SQL-i	Blind SQL-i
Uniscan	0	72	151	906	0	23
Acunetix	0	0	0	176	0	9
Nikto	0	0	0	0	0	0
WebCruiser	0	0	0	0	0	0
Shadow	0	2	0	0	0	0
Websecurify	0	0	0	20	0	0
PowerFuzzer	0	0	0	0	0	0

Tabela 5. Vulnerabilidades encontradas no site B

Scanner	RFI	LFI	RCE	XSS	SQL-i	Blind SQL-i
Uniscan	0	0	0	5018	821	1532
Acunetix	0	0	0	19	0	351
Nikto	0	0	0	0	0	0
WebCruiser	0	0	0	0	15	0
Shadow	0	0	0	0	0	0
Websecurify	0	0	0	3	22	0
PowerFuzzer	0	0	0	0	0	0

O cenário B, cujos resultados são apresentados na Tabela 5,representa um site de uma academia de artes. O sistema utiliza a linguagem de programação ASP e é executando sobre um servidor Web Microsoft-IIS 6.0. Os *scanners* Acunetix, Uniscan e Websecurify foram os únicos a detectar a vulnerabilidade XSS. Os *scanners* Uniscan, WebCruiser e Websecurify detectaram a presença da vulnerabilidade SQL *injection*. Por fim, o Acunetix e o Uniscan detectaram a vul-nerabilidade Blind SQL *injection*. No somatório geral, indi-vidualmente, o Uniscan detectou o maior número de vulne-rabilidades e o maior número de combinações para exploração das vulnerabilidades.

A seguir, na Tabela 6, são apresentados os resultados totalizadospara cada ferramenta. Os números representam o total de combinações, de exploração da vulnerabilidade, dete-ctadas para o conjunto dos 29 sites. Pode-se notar que o Uniscan foi o único *scanner* que conseguiu detectar as seis vulnerabilidades, além de, na média, encontrar um maior número de combinações de exploração das vulnerabilidades existentes nos sistemasalvo.

Tabela 6. Resultados acumulados dos sites analisados

Scanner	RFI	LFI	RCE	XSS	SQL-i	Blind SQL-i
Uniscan	1	79	257	8051	1055	2363
Acunetix	1	0	0	923	9	389
Nikto	0	0	0	0	0	0
WebCruiser	0	0	0	20	60	4
Shadow	0	2	0	1	0	0
Websecurify	0	0	0	48	34	0
PowerFuzzer	0	0	0	0	0	0

É possível observar na Tabela 6 que apenas o Uniscan foi capaz de detectar a vulnerabilidade RCE. Outro ponto explicitado na tabela é o fato de dois *scanners* não terem detectado nenhuma vulnerabilidade nos sites analizados.

A partir dos resultados apresentados, pode-se afirmar que o *scanner*desenvolvido contribui para a detecção e o diagnós-tico de vulnerabilidades em sistemas Web. Em alguns casos, a ferramenta desenvolvida apresentou resultados melhores que os *scanners* similares.

Um segundo, mas não menos importante, resultado posi-tivo do projeto foi o reconhecimento das potencialidades da ferramenta, levando a inclusão do Uniscan, desde a sua versão 3.2,na distribuição *Linux BackTrack*.Esta distribuição GNU/Linux é uma das que contempla a maior compilação de aplicativos voltados para a segurança e auditoria de sistemas. O número de utilizadores do Uniscan, no contexto do *Linux BackTrack*, é desconhecido. Entretando, sabe-se que o número de usuários da distribuição ultrapassa a marca de um milhão.

O Uniscan está disponível como ferramenta livre e gratuita[16]. As últimas estatísticas, de fevereiro de 2012, do próprio SourceForge, informam que o Uniscan já ultrapassou a marca 6.000*downloads*, oriundos de mais de 100 paises[17].

5. Conclusão

Scanners de vulnerabilidades são ferramentas importantes para a detecção e o diagnóstico de vulnerabilidades em sistemas Web. Entretanto, inexistem ferramentas capazes de detectar todos os tipos e combinações de exploração das vulnerabilidades existentes. Consequentemente, existeespaç o para novas pesquisas e desenvolvimentos. Neste sentido, o desenvolvimento do Uniscan procurou suprir algumas lacunas na detecção de vulnerabilidades em sistemas Web, com especial atenção para as vulnerabilidades LFI, RFI e RCE.

Os estudos realizados permitiram identificar características importantes em um *scanner* de vulnerabilidades, incorporadas ao Uniscan, como a utilização simultânea das metodologias dinâmica e estática. A primeirautiliza um rastreador para navegar e coletar dados do sistema alvo de forma autônoma. Por outro lado, a segunda metodologiapermite identificar subsistemas e *links* do sistema alvo que não são rastreáveis por um *crawler*.

A arquitetura do Uniscané outro ponto forte e um dife-rencial em relaçãoa maioria das demais ferramentasanali-zadas. O fato de utilizar *plug-ins* fracamente acomplados torna o *scanner* mais flexível, dinâmico e extensível,eliminando a necessidade dealterações,na ferramenta, para a inclusão de novas funcionalidades ou novos testes de vulnerabilidade. Com uma arquitetura modular, o próprio usuário pode rapida-mente criar os seus próprios *plug-ins*. Esta característica possibilita ainda configurações dinâmicas,

sob-demanda, onde o usuário pode escolher quais testes deseja executar sobre um determinado sistema alvo.

Com base nos experimentos e testes realizados, o Uniscan foi o único *scanner* a detectar todas as seis vulnerabilidades testadas, LFI, RFI, RCE, XSS, SQL-i e Blind SQL-i. Os números apresentados comprovam a sua capacidade de contri-buir na detecção e no diagnóstico de vulnerabilidades em sistemas Web. O fato de ser livre e aberto contribui para a sua utilização e para o desenvolvimento do conhecimento desta importante linha dentro da área de segurança da informação.

Finalmente, a ferramenta apresentada está em constante evolução através do desenvolvimento e da agregação de novas funcionalidades, *plug-ins*, recursos e características que possam ser úteis para contribuir ainda mais na detecção e no diagnóstico de vulnerabilidades em sistemas Web. As contri-buições da comunidade, através do site do projeto[16], são importantes para a continuidade do desenvolvimento e da evo-lução do Uniscan.

REFERÊNCIAS

[1] Fong, E.; Okun, V. "Web Application Scanners: Definitions and Functions". 40th Annual Hawaii International Conference on System Sciences, Jan. 2007.

[2] Nguyen-Tuong, A.; Guarnieri, S.; Greene, D.; Shirley, J.; Evans, D. “Automatically Hardening Web Applications Using Precise Tainting”. Security and Privacy in the Age of Ubiquitous Computing. In IFIP Advances in Information and Communication Technology, Springer Boston, 2005.

[3] Antunes, N.; Vieira, M. "Detecting SQL Injection Vulnerabilities in Web Services". Fourth Latin-American Symposium on Dependable Computing, pp.17-24, 1-4 Sept. 2009.

[4] Walden, J; Doyle, M;Welch, G. A; Whelan, M. Security of open source web applications. In Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, Washington, DC, USA, 545-553, 2009.

[5] Kals, S; Kirda, E; Kruegel, C.; Jovanovic, N. SecuBat: a web vulnerability scanner. In Proceedings of the 15th international conference on World Wide Web. ACM, NY, USA, 247-256, 2006.

[6] Hui-zhong Shi; Bo Chen; Ling Yu; , "Analysis of Web Security Comprehensive Evaluation Tools". Second International Conference on Networks Security Wireless Communications and Trusted Computing, pp.285-289, 24-25 April 2010.

[7] Jensen, M.; Gruschka, N.; Herkenhöner, R. A survey of attacks on web services. Computer Science - Research and Development. Springer Berlin / Heidelberg, vol. 24, issue 4, 2009.

[8] Teodoro, N.; Serrao, C. "Web application security: Improving critical web-based applications quality through in-depth security analysis". International Conference on Information Society (i-Society), pp.457-462, 27-29 June 2011.

[9] Curphey, M.; Arawo, R. "Web application security assessment tools". IEEE Security & Privacy, vol.4, no.4, pp.32-41, Jul-Aug. 2006.

[10] Arkin, B.; Stender, S.; Mcgraw, G. “Software Penetration Testing”. IEEE Security & Privacy, v. 3, n. 1, p. 84-87, 2005.

[11] Bau, J.; Bursztein, E.; Gupta, D.; Mitchell, J. "State of the Art: Automated Black-Box Web Application Vulnerability Testing". IEEE Symposium on Security and Privacy, pp.332-345, May 2010.

[12] Vieira, M.; Antunes, N.; Madeira, H. "Using web security scanners to detect vulnerabilities in web services". IEEE/IFIP International Conference on Dependable Systems & Networks,pp.566-571, 2009.

[13] Le, H. T.; Loh, P. K. K. "Unified Approach to Vulnerability Analysis of Web Applications". The International e-Conference on Computer Science (IeCCS 2007), 14-23 Dec. 2007.

[14] Fonseca, J.; Vieira, M.; Madeira, H. "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," 13th Pacific Rim International Symposium on Dependable Computing, pp.365-372, 17-19 Dec. 2007.

[15] Simmons, S.; Edwards, D.; Wilde, N.; Just, J.; Satyanarayana, M. "Preventing unauthorized islanding: cyber-threat analysis". IEEE/SMC International Conference on System of Systems Engineering, April 2006.

[16] Uniscan Project at SourceForge. http://sourceforge.net/projects/uniscan/. Último acesso em de fevereiro de 2012.

[17] SourceForge. Uniscan Project Download Statistics: All Files. URL:http://sourceforge.net/projects/uniscan/files/stats/timeline?dates=2011-01-01+to+2012-02-23. Último acesso em fevereiro de 2012.

[18] Websecurify. Web Application Security Scanner and Manual Penetration Testing Tool. URL: http://www.websecurify.com.Último acesso em fevereiro de 2012.

[19] Janus Security.WebCruiser - Web Vulnerability Scanner V2.5.0. URL: http://sec4app.com/.Último acesso em fevereiro de 2012.

[20] SAFETY-LAB. Shadow Security Scanner. URL: http://www.safety-lab.com/.Último acesso em fevereiro de 2012.

[21] Kozlowski, M. Powerfuzzer. URL: http://www.powerfuzzer.com/. Último acesso em fevereiro de 2012.

[22] CIRT, Inc. Nikto2 | CIRT.net. URL: http://cirt.net/nikto2/. Último acesso em fevereiro de 2012.

[23] ACUNETIX. Website Security - Acunetix Web Security Scanner. URL: http://www.acunetix.com. Último acesso em fevereiro de 2012.

[24] ABYSSSEC. PHP Fuzzing In Action. URL: http://www.exploit-db.com/download_pdf/12943. Último acesso em fevereiro de 2012.

[25] ZONE-H. Defacements Statistics 2010: Almost 1,5 million websites defaced, what’s happening? URL: http://www.zone-h.org/news/id/4737/. Último acesso em fevereiro de 2012.

[26] OpenVAS. OpenVAP advanced Open Source vulnerability scanner and manager. URL: http://www.openvas.org/. Último

acesso em junho de 2012.

[27] Arachni. Arachni – web application security scanner framework. URL: http://arachni-scanner.com/. Último acesso em junho de 2012.

[28] W3af. W3af Web Application Attack and Audit Framework.URL: http://w3af.sourceforge.net/. Último acesso em junho de 2012.

[29] WordPress.org. Wordpress Plugin Annonces 1.2.0.0. URL: http://wordpress.org/extend/plugins/annonces/. Último acesso em junho de 2012.

[30] Pontes, F. da S. Supernews (sistema de Notícias baseado em PHP e Mysql). URL: http://phpbrasil.com/script/vT0FaOCy SSH/supernews. Último acesso em junho de 2012.

[31] Membris. Membris 2. URL: http://www.membris.fr/. Último acesso em junho de 2012.

[32] Offensive Security. Wordpress Annonces Plugin 1.2.0.0 Remote File Inclusion. URL: http://www.exploit-db.com/exploits/17863/. Último acesso em junho de 2012.

[33] IBM Internet Security Systems (Ahead of the threat). Annonces Plugin for WordPress uploadPhoto.php remote file include. URL: http://xforce.iss.net/xforce/xfdb/69932. Último acesso em junho de 2012.

[34] Offensive Security. Supernews <= 2.6.1 (noticias.php cat) SQL Injection. URL: http://www.exploit-db.com/exploits/18961/. Último acesso em junho de 2012.

[35] Offensive Security. Membris v2.0.1 Multiple Vulnerabilities. URL: http://www.exploit-db.com/exploits/18970/. Último acesso em junho de