

SÉRIE WEBAPP PARA PENTESTER E APPSEC

USANDO PERL

COMO CRIAR UM SCAN PARA IDENTIFICAR
SERVIDORES WEB VULNERÁVEIS



O MANUAL PASSO A PASSO
de como criar seus próprios scripts para
identificar e tratar vulnerabilidades

SUMÁRIO

INTRODUÇÃO	3
2.0 PRÉ-REQUISITOS.....	4
3.0 CRIANDO O LABORATÓRIO/AMBIENTE	5
4.0 ACESSANDO O LABORATÓRIO.....	5
5.0 CRIAÇÃO DO SCRIPT	6
5.1 BIBLIOTECAS.....	6
5.1.1 IO::SOCKET::INET.....	6
5.2 CRIANDO A ESTRUTURA DO SCRIPT	7
6.0 EXECUTANDO O SCRIPT	11
7.0 IMPLEMENTAÇÕES.....	11
8.0 EXECUTANDO O SCRIPT	15
9.0 CÓDIGO COMPLETO	17
9.1 CÓDIGO COMPLETO DAS FERRAMENTAS	17
10.0 BUSCANDO EXPLOITS.....	19
11.0 APPLICATION SECURITY	21
12.0 SOBRE O AUTOR.....	22

INTRODUÇÃO

Muitas vezes, os profissionais de segurança ofensiva varrem um alvo ou uma rede em busca de vulnerabilidades para comprometer um alvo pelo framework Metasploit ou outros tipos de exploits e isso, sempre representa um sucesso nas atividades do pentester.

Na maioria das vezes, muitos pentester ficam reféns de ferramentas comerciais e não conseguem desenvolver seus próprios scripts, sejam eles em Python, Perl, PHP entre outros, limitando suas estratégias ofensivas e entregando assim, relatórios desprovidos de dados, colocando um alvo (já testado) em risco e futuramente sobre o poder de mãos erradas.

O artigo tem o objetivo de expandir o conhecimento e dar flexibilidade aos pentesters na criação dos seus próprios scripts, não ficando reféns de softwares de terceiros (comerciais e/ou Open-Sources) e futuramente poderão desenvolver funcionalidades mais sofisticadas

Praticar o conteúdo de artigo contribuirá para o aprimoramento das suas habilidades como pentester e profissional de segurança.

Nesse artigo, o intuito é simplificar e tornar compreensível o desenvolvimento de scripts, não apresentando técnicas avançadas para a elaboração de algoritmos e varreduras sofisticadas. Para atender um nível mais avançado é preciso mais implementações, validações e codificações complexas, o nosso objetivo é sermos mais simples e práticos.

Portanto, o script não se compara as funcionalidades de ferramentas comerciais que suportam metodologia de análise dinâmica ou DAST (Dynamic application security testing).

Esse artigo tende a apresentar dados essenciais para você produzir o laboratório e criar um script de identificação de banner de um servidor web Simple Web Server 2.2 rc2, posteriormente você poderá buscar exploits para exploração das vulnerabilidades.

Mas o que você precisa para aprender para criar um script de identificação de vulnerabilidades?

A resposta é simples e a criação do script também, você apenas precisa saber como funciona o processo de identificação da vulnerabilidade, ou seja, a requisição e validação da resposta do servidor... e um pouquinho

de Perl (uma linguagem com alto poder de velocidade para processar dados), isso você aprenderá lendo esse artigo.

2.0 PRÉ-REQUISITOS

Recomendamos a criação de dois ambientes, um ambiente com o servidor web Simple Web Server 2.2 rc2 disponível ou acessível por um usuário. Após criar o ambiente com Windows XP, podemos utilizar uma máquina com a distribuição Kali Linux (pode ser sua máquina):

- **Download do Kali Linux:**
<https://www.kali.org/get-kali/#kali-installer-images/>
- **Download do Windows XP:**
https://archive.org/download/WinXPProSP3x86/en_windows_xp_professional_with_service_pack_3_x86_cd_vl_x14-73974.iso
- **Simple Web Server 2.2 rc2**
<https://www.exploit-db.com/apps/142ba80cfca8f99ac36c92535728844c-sws-2.2-rc2-i686.exe>
- **Download do VMWARE:**
https://customerconnect.vmware.com/en/downloads/info/slug/desktop_end_user_computing/vmware_workstation_pro/15_0

Após fazer download de cada ferramenta, apenas faça o simples processo de instalação e configuração que são necessárias para o funcionamento.

3.0 CRIANDO O LABORATÓRIO/AMBIENTE

Nessa seção instalaremos um servidor web Simple Web Server 2.2 rc2 vulnerável numa máquina Windows XP.

Como o processo de instalação é muito simples, não abordaremos o processo de instalação.

Vamos considerar que você concluiu a instalação do Windows XP e do Web.

Se desejar acessar somente a seção sobre o desenvolvimento do script em Perl, acesse a **seção 5**.

4.0 ACESSANDO O LABORATÓRIO

Vamos acessar a máquina com Windows XP e iniciar o Simple Web Server 2.2 rc2.



4.0.1 Inicie no servidor Simple Web Server 2.2 rc2.

Com o servidor operando normalmente, vamos iniciar o processo de construção do nosso script em Perl.

Para realizar o teste é obrigatório instalar uma distribuição Kali Linux, se desejar reproduzir o laboratório.

5.0 CRIAÇÃO DO SCRIPT

Nessa seção vamos criar o primeiro script que fará uma verificação de banner de serviço do web.

Se você não conhece a linguagem Perl, não se preocupe, porque vamos apresentar cada parte do script e como funciona.

5.1 BIBLIOTECAS

Quando criamos um script em Perl, precisamos pensar nas bibliotecas para que o script consiga executar suas funções.

Nosso script precisará utilizar algumas bibliotecas, por exemplo para manipular conexões associadas ao serviço de web:

5.1.1 IO::SOCKET::INET

Uma biblioteca padrão e possui funcionalidades que permitem manipular de dados através das comunicações de rede.

Para ambientarmos melhor sobre Sockets, dizemos que é uma biblioteca responsável pela comunicação e troca de dados entre cliente e servidor, sendo rede internas ou externas.

A biblioteca IO::Socket::INET, segue um modelo muito parecido com o chamado raw socket, isto é, torna-se possível manipular o socket de comunicação a um nível muito baixo, a ponto de definir protocolos, times, flags através de simples parâmetros de configurações para conexões.

Para mais informações, sugiro buscar a documentação oficial sobre o Socket na metacpan:

- <https://metacpan.org/pod/IO::Socket::INET>

Para iniciarmos nosso script, definiremos a biblioteca exigida para fazermos comunicações com o servidor remoto, essa é a estrutura:

```
use IO::Socket::INET;
```

Depois de entendermos de forma resumida a utilidade da biblioteca e como ela será fundamental em nossa verificação, vamos estruturar a apresentação das informações de como usar o script.

5.2 CRIANDO A ESTRUTURA DO SCRIPT

A estrutura do script completa será mencionada nas próximas linhas do artigo.

Para começarmos, vamos criar um banner informativo para as pessoas que foram utilizarem o script no laboratório:

```
sub banner {  
  
    print "#####\n";  
    print "#                               #\n";  
    print "#      Scan in Perl to Simple Web Server 2.2 rc2   #\n";  
    print "#                               #\n";  
    print "#      Modo de uso: scanWeb.pl <ip> <port>          #\n";  
    print "#                               #\n";  
    print "#      Created by: Fernando Mengali                 #\n";  
    print "#                               #\n";  
    print "#####\n";  
  
}
```

Essa estrutura de código temos um banner que exibirá a mensagem ou modo de uso do scanWeb.pl.

Por enquanto, a função “**banner**”, não está sendo chamada ou exibida, para exibirmos a mensagem, precisamos utilizar a função “**banner()**” em alguma parte do código, exemplo:

```

sub banner {

    print "#####\n";
    print "#                               #\n";
    print "#      Scan in Perl to Simple Web Server 2.2 rc2      #\n";
    print "#                               #\n";
    print "#      Modo de uso: scanWeb.pl <ip> <port>              #\n";
    print "#                               #\n";
    print "#      Created by: Fernando Mengali                       #\n";
    print "#                               #\n";
    print "#####\n";

}

banner();

```

A chamada da função banner para exibir o banner, ocorre quando “**banner()**” é executada com sucesso.

Para deixarmos o script mais interessante, vamos validar se o usuário passou ou digitou para o script scanWeb.pl o endereço de IP e porta do alvo, caso não mencionou o endereço IP ou alvo, exibimos o banner com o modo de uso.

```

if (!$ARGV[0] || !$ARGV[1]) {

    banner();
    exit;

}

```

Agora, quando o usuário não digitar o endereço IP e a porta de conexão para o servidor web a mensagem do banner será exibida dessa forma:

```
—(root@kali)-[/home/kali/Desktop]
└─# perl scanWeb.pl
#####
#                                     #
#       Scan in Perl to Simple Web Server 2.2 rc2       #
#                                     #
#       Modo de uso: scanWeb.pl <ip> <port>            #
#                                     #
#       Created by: Fernando Mengali                    #
#                                     #
#####

└─(root@kali)-[/home/kali/Desktop]
└─#
```

Agora, vamos abordar sobre a Entrada de dados ou como o script pode ser informado em qual endereço IP e porta conectar para identificar o banner e definir se está vulnerável:

```
if (!ARGV[0] || !ARGV[1]) {

    my $alvo = $ARGV["0"];
    my $porta = $ARGV["1"];

}
```

Quando apontamos o script no terminal de comando, precisamos passar o endereço do alvo que será analisado.

Após termos os dados do nosso alvo, faremos uma conexão para porta de destino do servidor, a porta 80.

Para fazermos uma conexão, precisamos utilizar IO::Socket::INET para o endereço informado e a porta de destino.

```
$conexao = IO::Socket::INET -> new (Proto => "tcp", PeerAddr =>
$alvo, PeerPort => $porta, Timeout => "7");

print $conexao "GET / HTTP/1.1\r\nHost: $alvo\r\n\r\n";
```

A variável \$conexao, conecta na porta 80 utilizando como base o protocolo TCP e aguarda a resposta 7 segundos. Para completar o processo de conexão, será enviado uma simples conexão para o servidor de destino com o intuito de obtermos o banner do servidor.

Se a conexão for bem-sucedida, o lado cliente receberá informações sobre o banner do servidor alvo e posteriormente, seguimos com o tratamento dos dados.

```
my $banner;
read($conexao, $banner, 1024); # Lê até 1024 bytes da resposta
```

Na variável \$banner armazenamos a resposta do servidor, agora temos os dados do servidor alvo e agora podemos comparar com algum banner de serviço vulnerável e com exploit disponível em repositórios na internet.

```
if ($banner =~ "PMSoftware-SWS/2.2") {

    print "[+] => ".$alvo." VULNERABLE \n";
}
else {

    print "[-] => ".$alvo." NOT VULNERABLE \n";
}
```

Nossa linha de condição, verifica se o banner de resposta possui alguma informação relacionada com a string (“texto entre aspas”), se sim, apresentamos na tela a informação de alvo vulnerável.

6.0 EXECUTANDO O SCRIPT

Depois de salvar o nosso primeiro script com o nome de scriptHTTP.pl, vamos executá-lo, passando os argumentos para capturar o banner do servidor web. Você precisa utilizar o comando:

```
perl scanWeb.pl 192.168.176.131 80
```

Vejamos o resultado do script em nosso terminal do Kali Linux:

```
—(root@kali)-[/home/kali/Desktop]
└─# perl scanWeb.pl 192.168.176.131 80

[+] => 192.168.176.131 VULNERABLE

└─(root@kali)-[/home/kali/Desktop]
└─#
```

7.0 IMPLEMENTAÇÕES

Algumas empresas possuem diversos sistemas, portais, sites internos e externos. Podemos adicionar todas as urls em um arquivo texto e alimentar o script **scriptHTTP.pl**.

No momento que o script solicita o endereço da url, podemos substituir por uma função que solicite o nome do arquivo com as urls internas ou externas a serem testadas.

Esse processo facilita a execução de verificações e economiza tempo do analista de segurança.

Não precisa executar o script várias vezes, uma única execução é o suficiente para analisar vários endereços.

Abaixo, estou apresentando um modelo de arquivo texto com as urls a serem testadas como exemplo:

```
1 192.168.176.10
2 192.168.176.20
3 192.168.176.30
4 192.168.176.40
5 192.168.176.50
6 192.168.176.55
7 192.168.176.65
8 192.168.176.80
9 192.168.176.111
10 192.168.176.120
11 192.168.176.125
12 192.168.176.131
13 192.168.176.140
14 192.168.176.155
```

7.0.1 Lista de endereços de rede interna que iremos testar.

Vamos mudar os requisitos para exibirmos o banner com informações sobre o modo de uso do script. No script anterior era necessário informamos o endereço IP e a porta de conexão do servidor web. Nessa alteração e implementação, podemos informar ao profissional de segurança que é obrigatório informar um nome de arquivo contendo uma lista “endereços IPs” para ser analisados. Um exemplo de conteúdo de lista de “endereços IPs” é apresentado acima, na seção 7.0.1.

Abaixo temos um exemplo da exibição do banner, caso o profissional não informe o nome do arquivo com a lista de IPs:

```

sub banner {

    print "#####\n";
    print "#                               #\n";
    print "#       Scan in Perl to Simple Web Server 2.2 rc2   #\n";
    print "#                               #\n";
    print "#       Modo de uso: scanWeb.pl <ip> <port>           #\n";
    print "#                               #\n";
    print "#       Created by: Fernando Mengali                   #\n";
    print "#                               #\n";
    print "#####\n";

}

```

Para o banner ser exibido o profissional não passará um parâmetro que é o nome da lista de “*endereços IPs*” após o nome do script “*scanWeb.pl*”.

```

if (!$ARGV[0]) {

    banner();
    exit

}

```

Depois da função “*sub banner*”, temos o nosso bloco condicional que verifica se o nome do arquivo “*.txt*” foi informado pelo profissional pentester. Se o nome do arquivo “*.txt*” não foi digitado, entra no bloco condicional, chama a função “*banner*”, exibido a mensagem do modo de uso do scanWeb.pl e depois encerra o script.

Abaixo, vamos apresentar o processo de “*automatização*” de varredura, ou seja, como podemos verificar uma lista de endereços IPs para serem verificados de forma automatizada, sem digitar cada endereço IP e verificar manualmente os resultados.

```

our $lista = $ARGV["0"];

open( LISTADDR, "< $lista") or die ("Não foi possível abrir o
arquivo: $!")

our @array = <LISTADDR>;

my $lastitems = $#array;
my $porta = "80";

for (my $i=0; $i <= $lastitems; $i++) {

    $alvo = $array[$i];

```

Acima, fizemos as seguintes mudanças, depois do profissional pentester informar o nome da lista de “**endereços IPs**”, usamos a variável \$ARGV[0] para receber o nome da lista com os IPs que deverão ser analisados pelo script.

A linha 7 o arquivo é aberto e na linha 11 é contabilizado a quantidade de linha totais e armazenado na variável \$lastitems;

Na linha 12 temos a porta padrão do servidor web, isto é, 80.

Na linha 14 fazemos as interações necessárias para capturarmos o alvo e fazermos a conexão na porta 80.

O restante do script, segue a mesma estrutura de verificação da seção 5.2, a única diferença é processo de execução automatizado do script.

O restante do script, não passa por alterações, exceto informar ao script onde os colchetes associados as interações do “**for**” terminam no script. Veja o resultado:

```

$conexao = IO::Socket::INET -> new(PeerAddr => $alvo, PeerPort =>
$porta, Proto => "tcp", Timeout => "7");

print $conexao "GET / HTTP/1.1\r\nHost: $alvo\r\n\r\n";

my $banner;
read($conexao, $banner, 1024); # Lê até 1024 bytes da resposta

if ($banner =~ "PMSoftware-SWS/2.2") {

    print "[+] => ".$alvo." VULNERABLE \n";

}
else {

    print "[-] => ".$alvo." NOT VULNERABLE \n";

}
}

```

8.0 EXECUTANDO O SCRIPT

```

—(root@kali)-[~/Desktop]
└─# perl scanWeb.pl list.txt

[-] => 192.168.176.10 NOT VULNERABLE

[-] => 192.168.176.20 NOT VULNERABLE

[-] => 192.168.176.30 NOT VULNERABLE

[-] => 192.168.176.40 NOT VULNERABLE

[-] => 192.168.176.50 NOT VULNERABLE

[-] => 192.168.176.55 NOT VULNERABLE

```

```
[ - ] => 192.168.176.65 NOT VULNERABLE

[ - ] => 192.168.176.80 NOT VULNERABLE

[ - ] => 192.168.176.111 NOT VULNERABLE

[ - ] => 192.168.176.120 NOT VULNERABLE

[ - ] => 192.168.176.1110 NOT VULNERABLE

[ + ] => 192.168.176.131 VULNERABLE

[ - ] => 192.168.176.140 NOT VULNERABLE

[ - ] => 192.168.176.151 NOT VULNERABLE
```

```
└─(root@kali)-[/home/kali/Desktop]
└─#
```

8.0.1 Esse é o resultado do script no terminal do Kali Linux.

Uma informação muito importante, não recomendamos ou indicamos adaptar o script para varreduras numa estrutura de verificação multithreadings.

Para tal, será necessário adicionar novas formas e métodos de validações e tratamentos dos resultados, tornando o assunto um pouco mais complexo. Nesse artigo, nosso intuito é entregar um conteúdo essencial para uma boa compreensão sobre o funcionamento de scripts que verificam vulnerabilidades no alvo.

9.0 CÓDIGO COMPLETO

Abaixo é apresentado ambos os códigos em Perl para testar no seu ambiente particular.

9.1 CÓDIGO COMPLETO DAS FERRAMENTAS

Nessa seção temos a ferramenta para identificar vulnerabilidades, utilizando recursos simples de um scanning de vulnerabilidade, mas não implementamos o banner de exibição de modo de uso.

```
#!/usr/bin/perl

use IO::Socket::INET;

$alvo = $ARGV["0"];
$porta = $ARGV["1"];

$conexao = IO::Socket::INET -> new (Proto => "tcp", PeerAddr =>
$alvo, PeerPort => $porta, Timeout => "7");

print $conexao "GET / HTTP/1.1\r\nHost: $alvo\r\n\r\n";

my $banner;
read($conexao, $banner, 1024); # Lê até 1024 bytes da resposta

if ($banner =~ "PMSoftware-SWS/2.2") {

    print $alvo." VULNERABLE \n";

}
else {

    print $alvo." NOT VULNERABLE \n";

}
```

9.2 A FERRAMENTA COM IMPLEMENTAÇÕES

Nessa seção utilizamos um recurso de scanning para verificar o banner de vários IPs ou urls no servidor web.

Para fazer essa verificação será preciso somente passar ou informar um arquivo de texto com vários ips ou urls.

```
#!/usr/bin/perl

use IO::Socket::INET;

$alvo = $ARGV[0];

open( LISTADDR, "< $ARGV[0]" ) or die( "Nao foi possível abrir o arquivo:
$!" );

our @array = <LISTADDR>;

my $lastitems = $#array;
my $porta = "80";

for ($i = 0; $i <= $lastitems; $i++) {

    $alvo = lastitems[$i];
    $porta = "80";

    $conexao = IO::Socket::INET -> new(PeerAddr => $alvo, PeerPort =>
$porta, Proto => "tcp", Timeout => "7");

    print $conexao "GET / HTTP/1.1\r\nHost: $alvo\r\n\r\n";

    my $banner;
    read($conexao, $banner, 1024); # Lê até 1024 bytes da resposta

    if ($banner =~ "PMSoftware-SWS/2.2") {

        print $alvo." VULNERABLE \n";

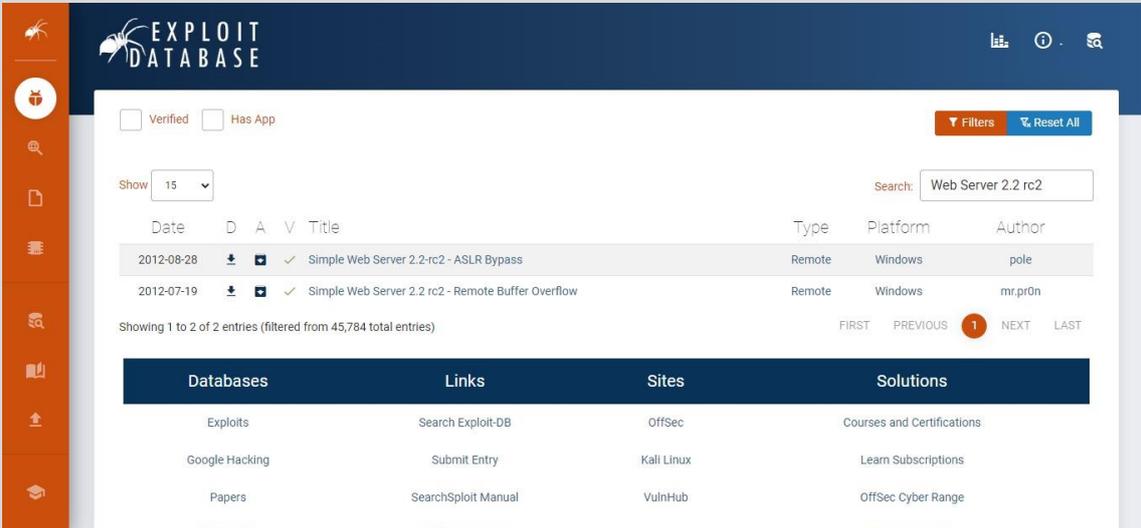
    }
    else {
```

```
print $alvo." NOT VULNERABLE \n";  
  
}  
}
```

10.0 BUSCANDO EXPLOITS

Depois de observarmos o resultado do script, um pesquisador de segurança ou pentester pode buscar por exploits nos repositórios da internet ou utilizar o framework Metasploit para explorar a vulnerabilidade no servidor web.

Vamos pesquisar se a versão do servidor web identificado pelo script HTTP possui algum exploit disponível na internet.



The screenshot shows the Exploit Database website interface. The search bar contains the text "Web Server 2.2 rc2". The search results table displays two entries:

Date	D	A	V	Title	Type	Platform	Author
2012-08-28	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Simple Web Server 2.2-rc2 - ASLR Bypass	Remote	Windows	pole
2012-07-19	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Simple Web Server 2.2 rc2 - Remote Buffer Overflow	Remote	Windows	mr.p0rn

Below the search results, there is a navigation bar with "FIRST", "PREVIOUS", "1" (current page), "NEXT", and "LAST" buttons. At the bottom of the page, there is a navigation menu with four categories: Databases, Links, Sites, and Solutions.

Procurando no exploit-db, encontramos exploits para explorar as vulnerabilidades no can in Perl to Simple Web Server 2.2 rc2.

The screenshot shows the Exploit Database interface. At the top, it displays the title 'Simple Web Server 2.2 rc2 - Remote Buffer Overflow'. Below the title, there are several key-value pairs: EDB-ID: 19937, CVE: (blank), Author: MR.PR0N, Type: REMOTE, Platform: WINDOWS, and Date: 2012-07-19. There are also indicators for 'EDB Verified: ✓', 'Exploit: [download icon] / [code icon]', and 'Vulnerable App: [app icon]'. At the bottom, a code block contains the following Perl script:

```

use IO::Socket;
#####
# Exploit Title: SimpleWebServer 2.2-rc2 - Remote Buffer Overflow Exploit
# Date: 19/07/2012
# Author: mr.pr0n (@_pr0n_)

```

Exploit disponível para explorar a falha no can in Perl to Simple Web Server 2.2 rc2:
<https://www.exploit-db.com/exploits/19937>

The screenshot shows the Packet Storm interface. At the top, it displays the title 'Intrasmv Simple Web Server 1.0 Denial of Service (DoS)'. Below the title, there are several key-value pairs: Posted: 2024-01-09, Format: text/plain, Size: 2.93 KB, Source(s): Fernando Mengali, Tag(s): Exploit, Web, Denial of Service, Actions: Share | Download, and SHA-256: d7370c79e707e5078287413c0fbc8380ee94a508c9537f6674c99c31529ba05. Below the key-value pairs, there is a 'Description' section with the text 'Intrasmv Simple Web Server version 1.0 suffers from a denial of service vulnerability.' and a 'Content' section with the following Perl script:

```

#!/usr/bin/perl

use IO::Socket;

# Exploit Title: Intrasmv Simple Web Server 1.0 - Denial of Service (DoS)
# Discovery by: Fernando Mengali
# Discovery Date: 09 January 2024
# Vendor Homepage: http://www.leighb.com/intrasmv.htm

```

Um exploit que publique no Packetstormy, disponível em:
<https://packetstorm.news/files/id/176419/>

Nessa seção, o objetivo foi identificar o banner do servidor web com um simples script em Perl e depois apresentar como funciona as buscas por exploits em repositórios da internet.

Como o intuito é didático e o objetivo é apresentarmos o processo de codificação do script em Perl, não vamos seguir com a exploração da vulnerabilidade no serviço web.

Futuramente com o aparecimento de vulnerabilidades do dia zero ou (Zero Day/0day) você pode utilizar os conhecimentos para desenvolver

scripts e verificar se existe algum servidor em sua rede que esteja vulnerável, contribuindo de forma ágil para correção das falhas de segurança e evitando um vazamento de dados, antes que os hackers façam.

11.0 APPLICATION SECURITY

No contexto de Segurança de Aplicações precisamos adotar algumas medidas de segurança, a fim de proteger de futuros ataques, como por exemplo:

1. Atualização dos patches de segurança;
2. Instalação de dispositivos de rede, como IPs, WAF, Firewall etc
3. Instalação de sistemas a nível de sistema operacional, visando a integridade de proteção de sistemas operacionais.
4. Revisão de políticas de segurança, por exemplo, políticas de acesso etc.
5. Pentest regularmente ao sistema alvo
6. Análise de vulnerabilidade contínuo.

Nesse cenário, a recomendação mais relevante é a remoção das informações de banners, principalmente as informações sobre o versionamento do serviço.

E se houver atualizações, atualize. Caso não haja atualizações, busque serviços equivalentes que atendam suas demandas diárias.

E para complementar, siga os 7 itens acima para garantir maior proteção do seu ambiente.

As informações contidas nessa seção, são recomendações padrões, mas uma análise e um estudo profundo do ambiente deve ser realizado para melhores recomendações mais assertivas e precisas.

12.0 SOBRE O AUTOR

Paper criado por Fernando Mengali no dia 21 de março de 2025.

LinkedIn: <https://www.linkedin.com/in/fernando-mengali-273504142/>

Minha página web com vários Papers para aprendizagem e estudos:

<https://papers.fitoxs.com/>