SÉRIE WEBAPP PARA PENTESTER E APPSEC

COMO CRIAR SCAN OPEN REDIRECT

O MANUAL PASSO A PASSO

de como criar seus próprios scripts para identificar e tratar vulnerabilidades

FERNANDO MENGALI

SUMÁRIO

INTRODUÇÃO

Nesse artigo, desenvolveremos uma ferramenta com a linguagem de programação Perl que identificará páginas de internet que possuem vulnerabilidades de Open Redirect a nível de comando do sistema operacional.

Primeiro, iremos apresentar o processo de **identificação manual da vulnerabilidade de Open Redirect a nível de comando do sistema operacional**, posteriormente você aprenderá como desenvolver um **script em Perl para detectar automaticamente** esse tipo de vulnerabilidade. Esse artigo não apresenta técnicas avançadas para o desenvolvimento do nosso script em Perl para a identificação de vulnerabilidades. Para a elaboração desse artigo, utilizamos conceitos básicos, mas eficiente para identificar vulnerabilidades de Open Redirect, seja para um alvo específico ou vários alvos.

O conteúdo sobre como identificar vulnerabilidades de Open Redirect nesse artigo não são equivalentes as grandes ferramentas de mercado que atendem a metodologia DAST (Dynamic application security testing).

Não ensinamos a desenvolver algoritmos sofisticados que são utilizadas pelas ferramentas de análise dinâmica disponíveis comercialmente, mas compartilhamos informações suficientes para começar a criar suas primeiras ferramentas para identificar vulnerabilidades e continuar aperfeiçoando suas técnicas de desenvolvimento de scripts de identificação de vulnerabilidades.

2.0 PRÉ-REQUISITOS

Será necessário instalar os softwares abaixo para o desenvolvimento do laboratório:

- Sistema operacional **Microsoft Windows** (no artigo utilizei o Windows 10)

- Download do **WAMP 3.1.9**: https://sourceforge.net/projects/wampserver/

- Download **Perl**: https://www.activestate.com/products/activeperl/downloads/

3.0 CRIANDO O LABORATÓRIO/AMBIENTE

Nessa seção instalaremos o WAMP (Apache, MySQL e PHP) no Windows. Até o desenvolvimento desse artigo, foi utilizado o **WAMP 3.1.9**.

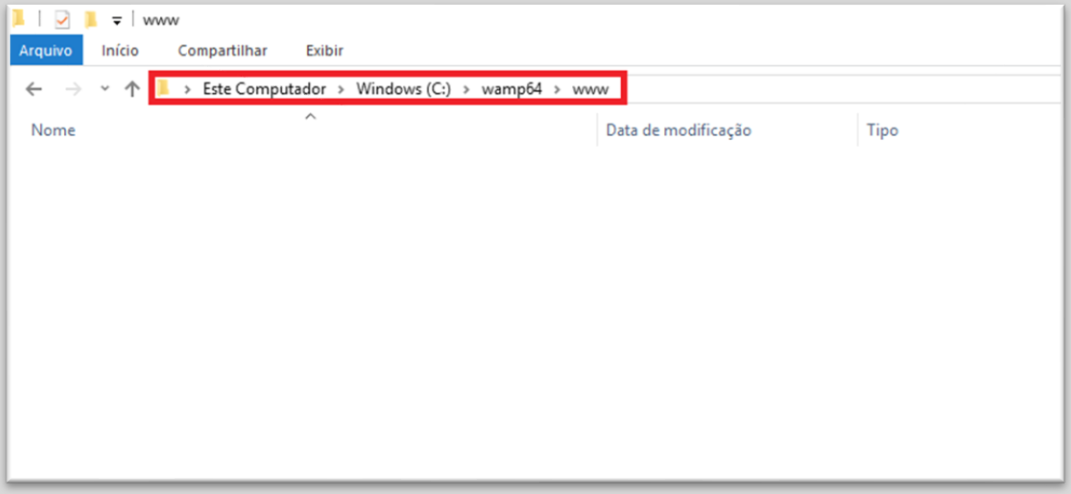
O processo de instalação é muito simples, portanto, não abordaremos.

Vamos considerar que você concluiu a instalação do WAMP e depois de instalado, vamos prosseguir com as configurações.

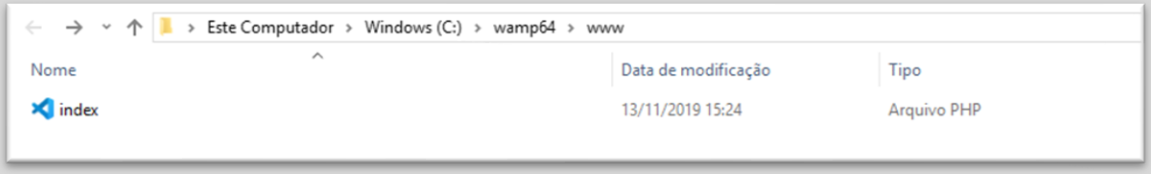
Se desejar acessar somente a seção sobre o desenvolvimento do scanning em Perl, acesse a **seção 8**.

4.0 CRIANDO A PÁGINA PHP VULNERÁVEL

Acesse o diretório **www** para criarmos a página em PHP. Se você utilizou a sugestão do Windows para a instalação, o caminho será "**C:\Windows\wamp64\www"**. Veja abaixo:



4.1.1 Quando você acessar o conteúdo do diretório "**www**", visualizará alguns arquivos. Particularmente, eu removi todos os arquivos, deixando o diretório "**www**" vazio.
A remoção dos arquivos do diretório "**www**" é sua escolha, eu acho melhor para trabalhar.



4.1.2 Nessa etapa iremos criar um arquivo com a extensão "**PHP"** com o nome de "**index"** no diretório "**www"**.

Depois de criar a página index, iremos adicionar o conteúdo ou código PHP vulnerável na página **index.php**.
Se você não codifica em PHP, não se preocupe, abaixo apresentamos o código e depois descrevemos o funcionamento de cada linha.

5.0 A PÁGINA PHP COM O CÓDIGO VULNERÁVEL

Vamos criar uma página com um link para "***logout***" ou finalização da sessão do usuário.

O usuário precisa clicar no link "***logout***" para ser redirecionado para página de autenticação.

No mundo real, existem aplicações que possuem o mesmo recurso de "***logout***", sendo extremamente vulnerável e levando a exploração de outras vulnerabilidades que envolvem sessão do usuário.
E como dito, trata-se de um sistema limitado e inseguro, mas muitas empresas adotam para seus clientes.

Abaixo apresentamos o código PHP vulnerável completo:

```
<?php

if (isset($_GET['id']) && isset($_GET["url"])){

    $id = $_GET['id'];
    $url = $_GET['url'];

    if ($id == 1) {

        header("Location: ".$url);

    }

}
?>

<a
href="http://localhost/logout.php?id=1&url=https://google.com">Logout</a>

<form action="index.php" method="GET">

Read document: <input type="text" placeholder="Type hash..."
name="id"><br>
<input type="submit">
</form>
```

5.0.1 Você poderá copiar esse código e adicionar para a sua página **index.php**.
Não esqueça, sua página **index.php** deverá estar em "**C:\Windows\wamp64\www".**

Essa etapa é bem simples, você não precisa ter conhecimentos de PHP para entender o código.

Se você quiser entender o código, continue lendo essa seção, pois descreverei cada linha na próxima página.
Inicialmente, nosso código receberá um parâmetro GET:

```
if (isset($_GET['id']) && isset($_GET["url"])){

    $id = $_GET['id'];
    $url = $_GET['url'];
```

5.0.2 Temos o if que valida a existência de dados ou pârametros enviados para o método GET. Se houver algum dado trafegando via GET ele entrará no commando bloco IF e será armazenado na variável **id** e na variável **url**, mas não existe nenhuma função para sanitização.

É importante observer que à ausência da sanitização dos dados inputados via GET é totalmente proposital, pois o intuito é entendermos como funciona a exploração de Open Redirect via dados de entrada GET. Mais adiante vamos explicar o processo de sanitização.

Vejo o exemplo de acesso a nossa página PHP via a url:
http://localhost/logout.php?**id**=1&url=https://google.com

Após recebermos os valores via método GET, validarmos se o id é igual a 1, se for verdadeiro a condição, faremos o redirecionamento através da função header, tendo o seguinte resultado:

```
    if ($id == 1) {

        header("Location: ".$url);

    }
```

5.0.3 O bloco condicional validará se o número para redirecionamento é 1, depois redirecionará para a página logout.php.

Vamos criar um arquivo chamado logout.php e armazená-lo no servidor! No mesmo diretório que que a página index.php encontra-se.

Esse será o resultado do código:

```
<h1>Logout success!<h1>
```

5.0.4 O conteúdo da página de logout.php é muito simples, apenas para termos conhecimento que o logout foi feito com sucesso.

6.0 SITE VULNERÁVEL

Nessa seção acessaremos a página **index.php** vulnerável: http://localhost/index.php

Veja o resultado no browser:

6.0.1 O conteúdo do arquivo será exibido no browser, pois o id está associado ao nome do arquivo 1.txt.

7.0 TEORIA: COMO DETECTAR OPEN REDIRECT

Para identificar o nosso alvo vulnerável, utilizaremos uma técnica simples, verificaremos se ao clicarmos em "***logout***" a página "index.php" fará um redirecionamento, respondendo com o valor 301.

Quando um atacante deseja encontrar uma vulnerabilidade de Open Redirect, simplesmente altera o valor do parâmetro url pelo endereço de um domínio qualquer e se houver um redirecionamento, será comprovado a vulnerabilidade

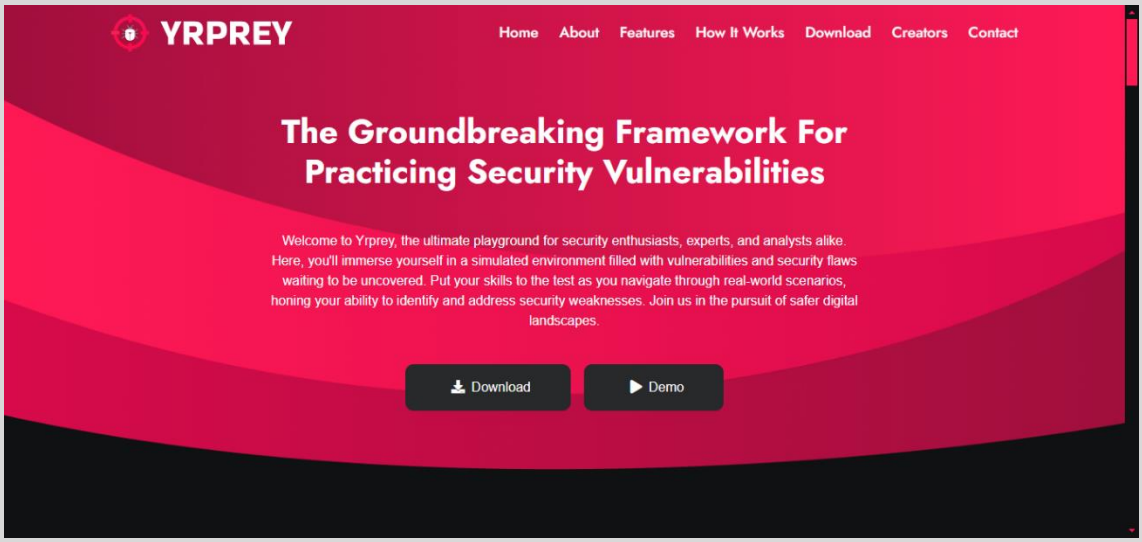
8.0 PRÁTICA: DETECTANDO A VULNERABILIDADE DE RCE

Agora vamos descobrir se a página está vulnerável a Open Redirect. Para essa etapa, será adicionado o endereço https://google.com no parâmetro url para fazermos o redirecionaremo.

8.1 FRAMEWORK PARA TESTAR OPEN REDIRECT

Caso você não queira criar um ambiente pronto, existem frameworks vulneráveis para interessados em aprender e aprimorar técnicas de invasão e mitigação de vulnerabilidades.

Um exemplo é o framework yrprey, totalmente gratuito e com várias vulnerabilidades para ser explorado. Você pode testá-lo online pelo endereço: http://yrprey.com.

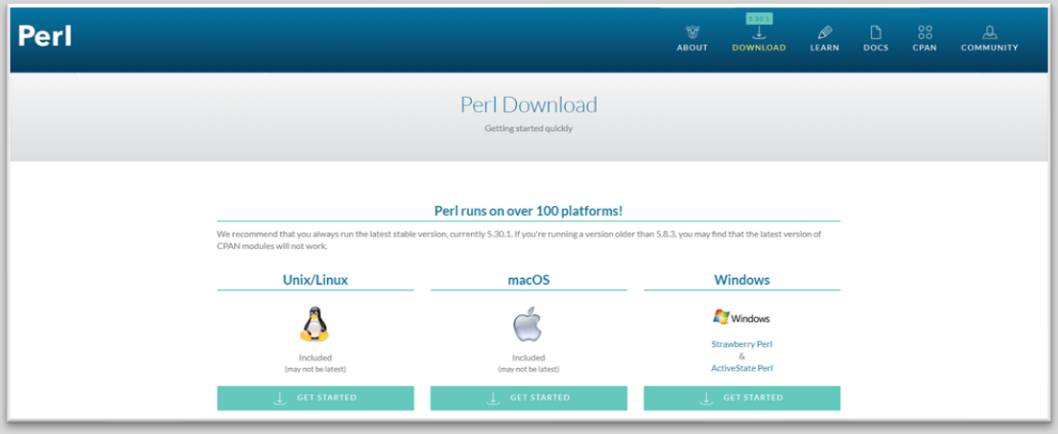


8.1.1 interface do yrprey.com até a elaboração desse artigo.

9.0 CONSTRUÇÃO DO SCANNING

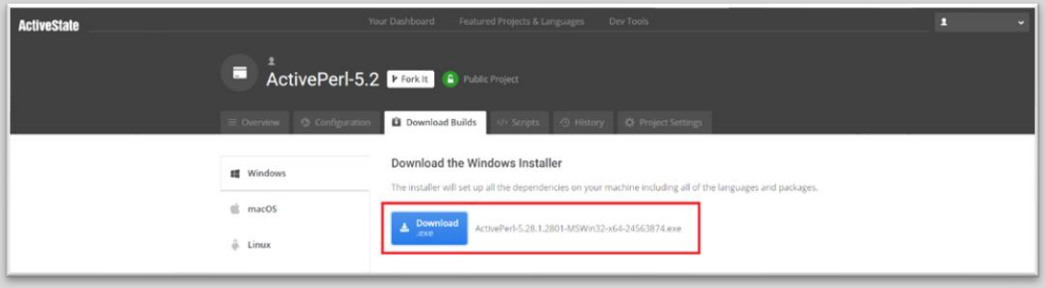
A linguagem de desenvolvimento escolhida para o desenvolvimento do script será o Perl. Você precisará de conhecimentos de programação em Perl, pois a ferramenta terá erros propositais, ou seja, apenas desenvolvedores, analistas de seguranças e interessados com aptidões de desenvolvimento entenderão melhor o código.

9.1 BAIXANDO O PERL PARA WINDOWS



9.1.1 Acesse a URL https://www.perl.org/get.html e escolha a plataforma que você está utilizando.
Você será redirecionado e solicitado a autenticar ou criar uma conta para baixar o Perl.

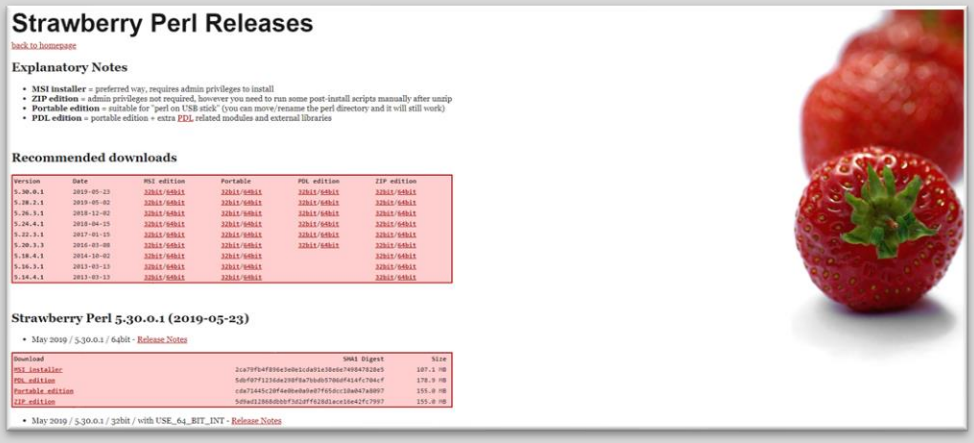
Depois de autenticado, você poderá baixar o Perl:



9.1.2 Clique no botão "**Download**".

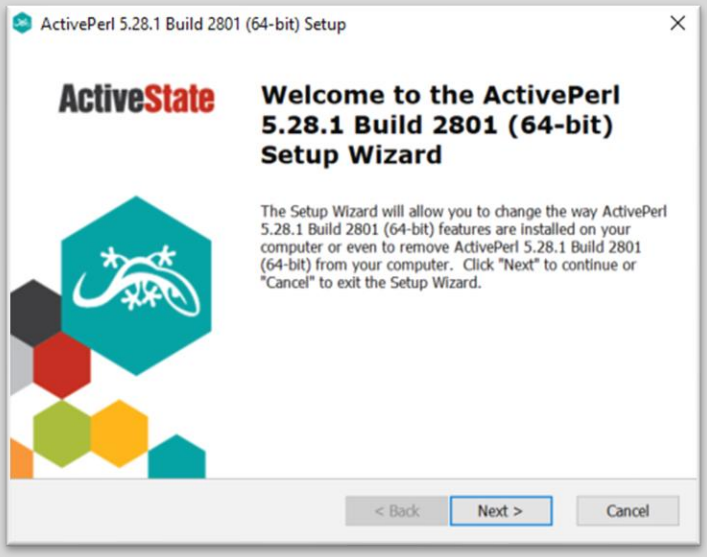
9.2 OPÇÃO 2: STRAWBERRY PERL PARA WINDOWS

Outra opção é utilizar **Strawberry Perl**:
http://strawberryperl.com/releases.html



9.2.1 Veja a página acima.

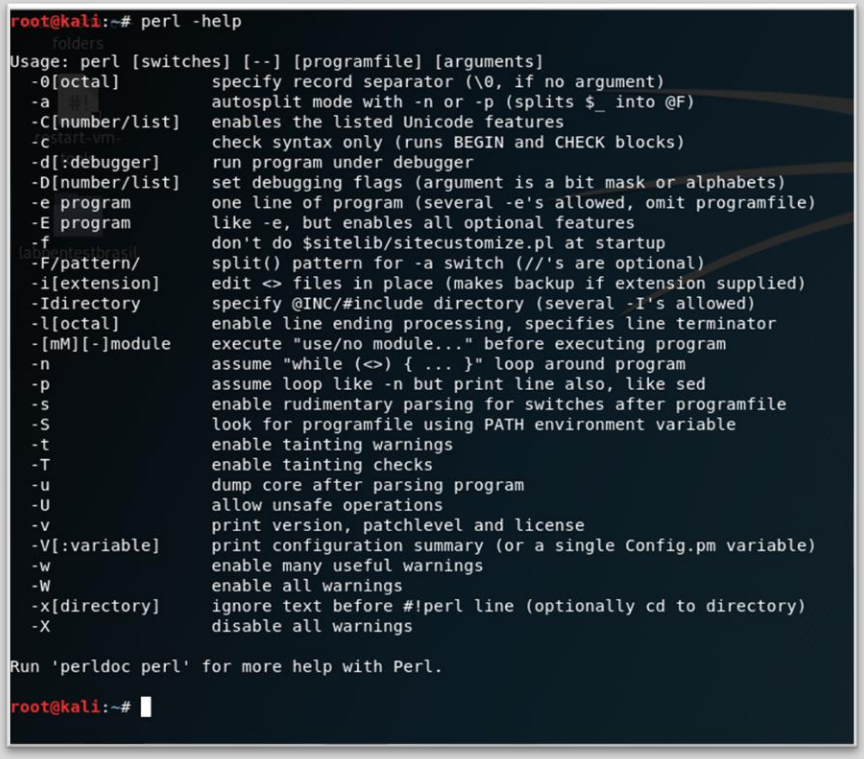
Depois de baixar o Perl para Windows, siga os procedimentos comum de instalação no Windows:



9.2.2 O restante você já sabe.

10.0 PERL NO LINUX

Se você está utilizando uma distribuição Linux, de preferência o Kali Linux, por padrão o Perl já está instalado.
Caso você deseje verificar se o Perl está instalado, digite os comandos **perl -help** no terminal do Kali Linux:



11.0 CODIFICANDO A FERRAMENTA DE AUTOMAÇÃO

Nessa etapa iremos utilizar uma requisição para nossa página http://localhost/index.php?id=1&url=https://google.com.
Utilizaremos um comando do type, junto a leitura do arquivo hosts do Windows e trataremos a resposta.

11.1 CLASSES DE REQUISIÇÕES

Nosso código precisará de duas classes para fazer requisições para a página com vulnerabilidade.

São elas:

- LWP::UserAgent

LWP::UserAgent

É uma classe responsável por atuar como um agente, durante uma requisição ou solicitação da web. Quando uma requisição é realizada será criado um objeto LWP::UserAgent com valores padrões.

11.2 COMEÇANDO COM A CODIFICAÇÃO

Utilizando os três módulos descritos acima, poderemos adicioná-los no início do script e depois criar a estrutura de requisição em Perl para a página vulnerável.

```
#!/usr/bin/perl

use LWP::UserAgent;
```

11.2.1 Não esqueça de usar **#!/usr/bin/perl**, se estiver usando linux.

11.3 ENTRADA DE DADOS

A entrada de dados será utilizada para informar qual URL será verificada. Caso não queira informar a URL utilizando uma entrada de dados, poderá deixar o endereço de forma estática na variável, mas vamos criar algo dinâmico:

```
my $url = <stdin>;
```

Na linha acima criarmos uma variável **$url** e depois adicionamos a entrada padrão **<stdin>**.
STDIN, poderá ser substituída por **<>**.

Agora, podemos desenvolver a arquitetura da requisição:

```
5
6   our $ua = LWP::UserAgent->new();
7
8   $ua->requests_redirectable(undef);
9
10  $ua->default_header("User-Agent" => "Mozilla/5.0");
11
12  my $response = $ua->get($url);
13
14  my $resp = $response->code();
```

11.3.4 A estrutura da requisição.

"Nessa etapa, exige conhecimentos de programação em Perl ou similar".
Na **linha 6** utilizamos o módulo LWP::UserAgent e definimos requests_redirectable como indefinido para podermos tratarmos a resposta.
Na **linha 14** a variável $resultado armazena a resposta.
Como nossa requisição acessa uma página de redirecionamento, estamos

capturando o valor numérico da resposta.

Se o valor número contém ***302,*** a página é vulnerável!

```
if ($resp eq "302") {

  print "\n\n Vulnerable! \n\n";

}
else {

  print "\n\n Not vulnerable! \n\n";

}
```

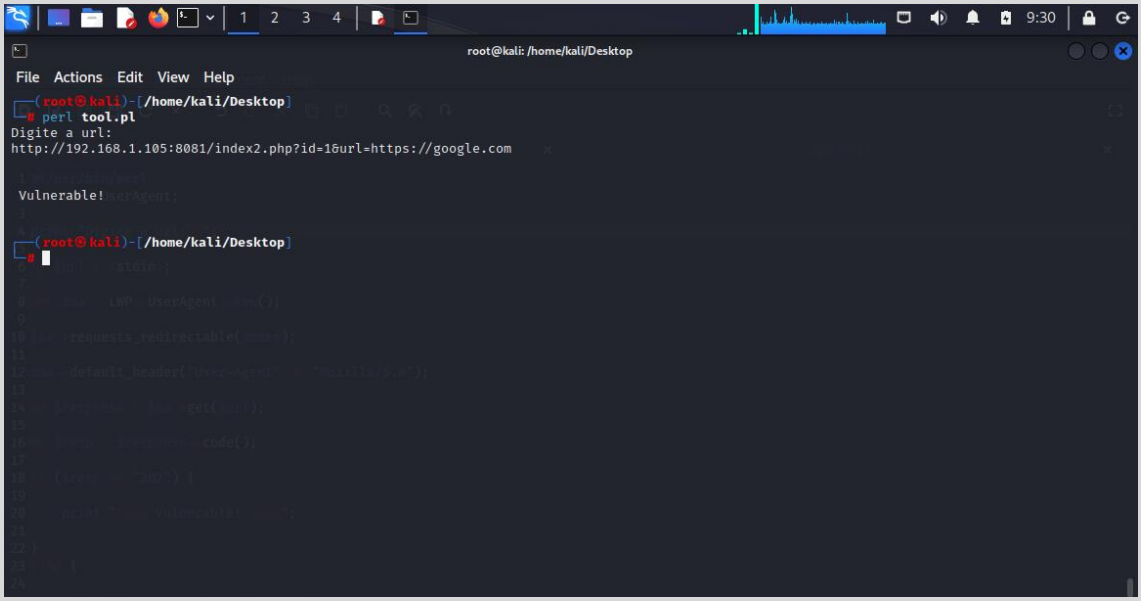
11.3.5 Podemos adicionar outros erros para serem comparados com o conteúdo de uma página.

Embora a ferramenta tenha uma feature adaptada para a identificação de vulnerabilidades de Open Redirect, pode não funcionar em determinados alvos, por causa de validação de headers como UserAgent, Cookies Custom ou outros tipos de Headers Customs, mas o conteúdo apresentado nesse documento é extremamente útil e importante para você aprender como criar seus primeiros scripts e aperfeiçoá-los.

Apenas precisa ser manipulado a tratamento do response.

11.5 EXECUTANDO O SCRIPT

Esse é o resultado do script.



12.0 IMPLEMENTAÇÕES

Algumas empresas possuem diversos sistemas, portais, sites internos e externos. Podemos adicionar todas as urls em um arquivo texto e alimentar o script **tool.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 http://192.168.1.105:8081/index2.php?id=1&url=https://google.com
2 http://192.168.1.106:8081/index2.php?id=1&url=https://google.com
3 http://192.168.1.107:8081/index2.php?id=1&url=https://google.com
4 http://192.168.1.108:8081/index2.php?id=1&url=https://google.com
5 http://192.168.1.109:8081/index2.php?id=1&url=https://google.com
6 http://192.168.1.110:8081/index2.php?id=1&url=https://google.com
7 http://192.168.1.111:8081/index2.php?id=1&url=https://google.com
8 http://192.168.1.112:8081/index2.php?id=1&url=https://google.com
9 http://192.168.1.114:8081/index2.php?id=1&url=https://google.com
10 http://192.168.1.135:8081/index2.php?id=1&url=https://google.com
11 http://192.168.1.125:8081/index2.php?id=1&url=https://google.com
12 http://192.168.1.165:8081/index2.php?id=1&url=https://google.com
13 http://192.168.1.155:8081/index2.php?id=1&url=https://google.com
14 http://192.168.1.185:8081/index2.php?id=1&url=https://google.com
```

12.0.1 Lista de urls de sistemas, portais, sites internos e externos.

```
9
10  print "Digite a lista de urls:\n";
11
12  my $url = <stdin>;
13
14  open ( URL, "< $url") or die ("Can't open file $!");
15
16  @vector = <URL>;
17
18  $last = $#vector;
19
20  for ($i=0 $i<= $last; $i++) {
21
22    print "Verificando => ".$vector[$i]."\n";
23
24    $url = $vector[$i];
25
26  my $response = $ua->get($url);
27
28  my $resp = $response->code();
29
```

12.0.2 Na **linha 14** adicione o código responsável por ler o arquivo texto com urls.
A **linha 16** armazena todo o conteúdo do arquivo no array **@vector**.
Na **linha 18** acessamos o último elemento do array.
Na **linha 20** desenvolvemos um for para acessar cada linha ou url armazenado no arquivo.
Na **linha 24**, temos o armazenamento de cada endereço na variável **$url**.
Na **linha 26**, fazemos a requisição para os sites alvos.

```
29
30    if ($resp eq "302") {
31
32        print "\n\n Vulnerable! \n\n";
33
34    }
35    else {
36
37        print "\n\n Not vulnerable! \n\n";
38
39    }
40    }
```

12.0.3 Na linha 30 criamos um IF, que verifica se a resposta da página possui algum erro de banco de dados. Se houve algum erro, ele apresenta a imagem "**Vulnerável**" ou senão, "**Não vulnerável**".

12.1 EXECUTANDO O SCRIPT

```
 Not vulnerable!

verificando ⇒ http://192.168.1.105:8081/index2.php?id=1&url=https://google.com

 Vulnerable!

verificando ⇒ http://192.168.1.106:8081/index2.php?id=1&url=https://google.com

 Not vulnerable!

verificando ⇒ http://192.168.1.200:8081/index2.php?id=1&url=https://google.com

 Not vulnerable!

verificando ⇒ http://192.168.1.201:8081/index2.php?id=1&url=https://google.com

 Not vulnerable!

verificando ⇒ http://192.168.1.202:8081/index2.php?id=1&url=https://google.com

 Not vulnerable!
```

12.1.2 Resultado do script verificando cada url armazenada no arquivo texto.

Você poderá adicionar novos recursos no script, como Thread, crawlers de páginas etc.

13.0 CÓDIGO COMPLETO

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

13.1 CÓDIGO COMPLETO DAS FERRAMENTAS

Nessa seção temos a ferramenta para identificar vulnerabilidades, utilizando recursos simples de um scanning de vulnerabilidade.

```
#!/usr/bin/perl
use LWP::UserAgent;

  our $ua = LWP::UserAgent->new();

  $ua->requests_redirectable(undef);

  $ua->default_header("User-Agent" => "Mozilla/5.0");

      print "Digite a url:\n";

      my $url = <stdin>;

  print "Verificando => ".$url."\n";

  my $response = $ua->get($url);

  my $resp = $response->code();

      if ($resp eq "302") {

        print "\n\n Vulnerable! \n\n";

      }
      else {

        print "\n\n Not vulnerable! \n\n";

     }
```

13.2 A FERRAMENTA COM IMPLEMENTAÇÕES

Nessa seção utilizamos um recurso no scanning para verificar vários IPs ou urls com vulnerabilidade de RCE.

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

```
#!/usr/bin/perl
use LWP::UserAgent;

  our $ua = LWP::UserAgent->new();

  $ua->requests_redirectable(undef);
  $ua->default_header("User-Agent" => "Mozilla/5.0");

       print "Digite a lista de urls:\n";

       my $url = <stdin>;

open ( URL, "< $url") or die ("Can't open file $!");

@vector = <URL>;

$last = $#vector;

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

  print "Verificando => ".$vector[$i]."\n";

  $url = $vector[$i];

  my $response = $ua->get($url);
  my $resp = $response->code();

       if ($resp eq "302") {

         print "\n\n Vulnerable! \n\n";

       }
       else {

         print "\n\n Not vulnerable! \n\n";

     }
}
```

14.0 CORRIGINDO VULNERABILIDADE

Irei demonstrar alguns tipos de proteção contra Open Redirect numa aplicação web que possui a tecnologia PHP.

1º Validar dados de entrada

Ao fazer "***logout***", podemos criar uma página interna no servidor que validará os dados de sessão do usuário e fará o redirecionamento para a página inicial do site.

A página inicial do site, poderá ser definida pelo path "***index***", "***default***" ou "***home***", dependerá da estrutura da aplicação.

16.0 SOBRE O AUTOR

Paper criado por Fernando Mengali no dia 05 de março de 2025.

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