

# HTTP

# Hypertext Transfer Protocol

Prof. Plinio Marcos Mendes Carneiro

# HTTP – Introdução

- Hypertext Transfer Protocol;
- Definição simples → protocolo de aplicação que define como fazer solicitações e transferir informações entre *browsers* e servidores;
- Protocolo de transferência base da WWW;
- Requisita meio confiável (TCP 80);
- Início dos anos 90, na versão 0.9, fazia transferência de dados brutos;

# HTTP – Introdução

- Porém, sua versão 1.0 definida pela RFC 1945, permitiu ao protocolo transportar mensagens do tipo MIME (Multipurpose Internet Mail Extensions);
  - Meta-informações sobre os dados transferidos, requisições e respostas;
  - Versão 1.0 não previa suporte adequado para muitas situações: Proxy, Cache, Keep-alive, ...
- Versão 1.1, RFC 2616 em 1999, padrão adotado atualmente.

# Topologia

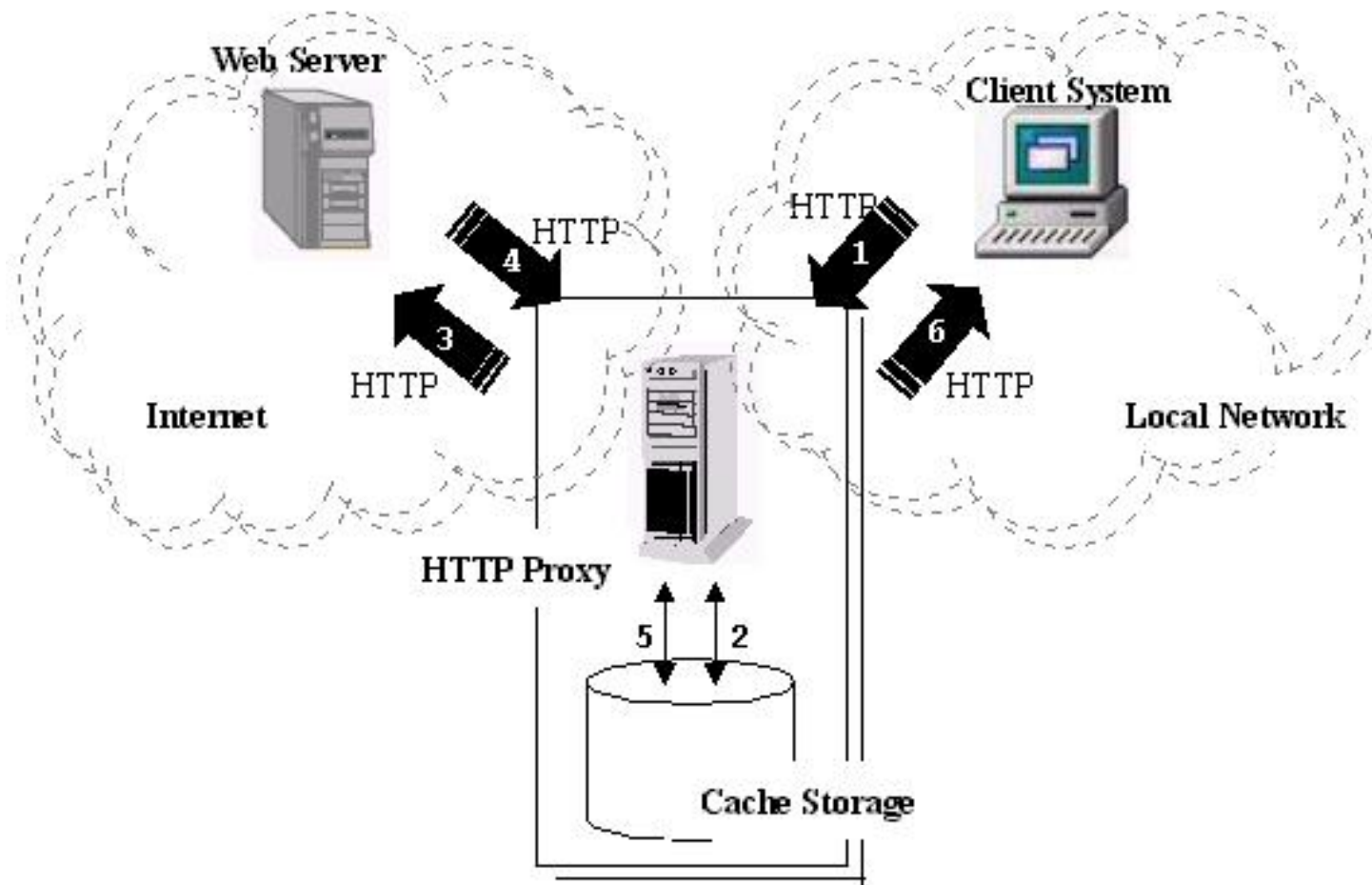
- Sem intermediários no nível de camada de aplicação;



- Na internet certamente há intermediários no nível de rede.

# Topologia

- Com HTTP Proxy intermediando;



# Topologia

- Há certamente topologias mais complexas, envolvendo por exemplo: serviços de balanceamento de carga, *proxy* reverso, *firewalls* de camada de aplicação e sistemas identificadores de intruso – IDS.
- Para compreensão do funcionamento do serviço HTTP o paradigma cliente-servidor é certamente o mais indicado.

# Versionamento

- Exemplos
  - HTTP/0.9
  - HTTP/1.0
  - HTTP/1.1
- HTTP/1.11 > HTTP/1.2, ou seja é diferente de 1,2 > 1,11

# URL - Uniform Resource Locator

- Geral → protocolo://host/caminho/recurso
- HTTP → http://host.dominio:porta/caminho\_no\_host/recurso
- Ex. <http://www.w3.org/Addressing/URL/uri-spec.html>
  - Protocolo → http
  - Host → www
  - Domínio → w3.org
  - Porta → Foi omitida, então "80"
  - Caminho → Addressing/URL/
  - Recurso → uri-spec.html



# Formato de data e hora

- São especificados três formatos de data e hora:
  - Sun, 06 Nov 1994 08:49:37 GMT
  - Sunday, 06-Nov-94 08:49:37 GMT
  - Sun Nov 6 08:49:37 1994
- O 1º formato é preferido por manter tamanho fixo sempre e utilizar 4 dígitos no ano.
- Um dispositivo HTTP/1.1 deve usar por padrão o 1º formato mas deve saber lidar com os outros para manter compatibilidade.

# HTTP – Funcionamento Básico

- Protocolo de requisições e respostas;
- Requisição → Método + URI + Versão do Protocolo + Cabeçalho da Mensagem + Mensagem

GET /Status\_Router.live.asp HTTP/1.1

Host: 192.168.1.1

User-Agent: Mozilla/5.0 Gecko/20091102 Firefox/3.5.5

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

Accept-Encoding: gzip,deflate

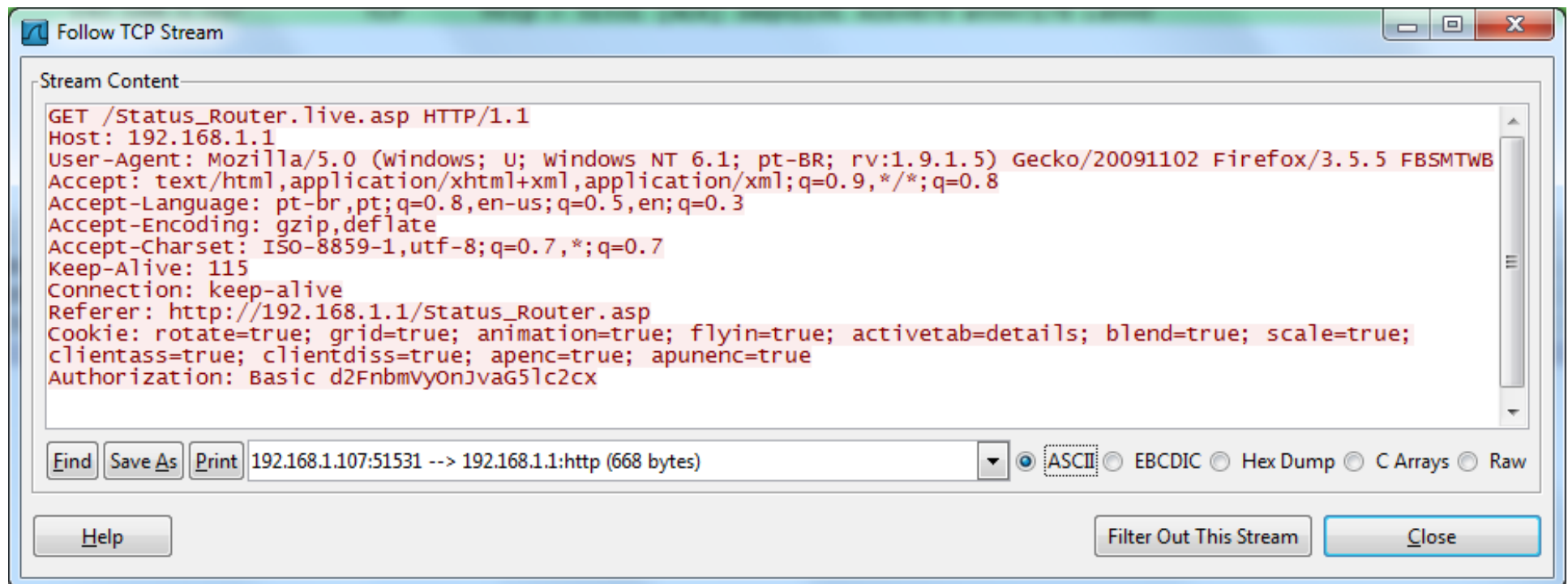
Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 115

Connection: keep-alive

# HTTP – Funcionamento Básico

- Requisição → Método + URI + Versão do Protocolo + Cabeçalho da Mensagem + Mensagem



The screenshot shows a window titled "Follow TCP Stream" with a text area containing an HTTP request. The request is a GET method for the URI "/Status\_Router.live.asp" using HTTP/1.1. The host is 192.168.1.1. The user agent is Mozilla/5.0 (Windows NT 6.1; pt-BR; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 FBSMTWB. The request includes several headers: Accept, Accept-Language, Accept-Encoding, Accept-Charset, Keep-Alive, Connection, Referer, Cookie, and Authorization. The status bar at the bottom indicates the connection is from 192.168.1.107:51531 to 192.168.1.1:80 (http) and that 668 bytes were received. The encoding is set to ASCII.

```
GET /Status_Router.live.asp HTTP/1.1
Host: 192.168.1.1
User-Agent: Mozilla/5.0 (windows; U; windows NT 6.1; pt-BR; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 FBSMTWB
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
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://192.168.1.1/Status_Router.asp
Cookie: rotate=true; grid=true; animation=true; flyin=true; activetab=details; blend=true; scale=true;
clientass=true; clientdiss=true; apenc=true; apunenc=true
Authorization: Basic d2FnbmVyaG5lc2cx
```

Find Save As Print 192.168.1.107:51531 --> 192.168.1.1:http (668 bytes) [v] [x] ASCII [x] EBCDIC [x] Hex Dump [x] C Arrays [x] Raw

Help Filter Out This Stream Close

# HTTP – Funcionamento Básico

- Resposta → Versão do Protocolo + Código de Resposta (com descrição) + Cabeçalho da Mensagem + Mensagem

HTTP/1.0 200 Ok

Content-Type: text/html

Server: httpd

Date: Fri, 21 May 2010 17:02:56 GMT

Connection: close

Cache-Control: no-store, no-cache, must-revalidate

Cache-Control: post-check=0, pre-check=0

Pragma: no-cache

Cache-Control: no-cache

Expires: 0

{router\_time::Fri, 21 May 2010 17:02:56}

{mem\_info::'total:','used:','free:','shared:','buffers:','cached:','Mem:','30310400','16400384','13910016','0','1720320','5292032','Swap:','0','0','0','MemTotal:','29600','kB','MemFree:','13584','kB','MemShared:','0','kB','Buffers:','1680','kB','Cached:','5168','kB','SwapCached:','0','kB','Active:','1345','kB','Inactive:','699','kB','HighTotal:','0','kB','HighFree:','0','kB','LowTotal:','29600','kB','LowFree:','13584','kB','SwapTotal:','0','kB','SwapFree:','0','kB','Dirty:','0','kB','Writeback:','0','kB','Mapped:','285','kB','Slab:','85','kB','CommitLimit:','14800','kB','Committed\_AS:','3128','kB','PageTables:','1712','kB','VmallocTotal:','1048404','kB','VmallocUsed:','17248','kB','VmallocChunk:','1031148','kB'}

{uptime:: 17:02:56 up 1 day, 3:28, load average: 0.00, 0.00, 0.00}

{ip\_contrack::98}

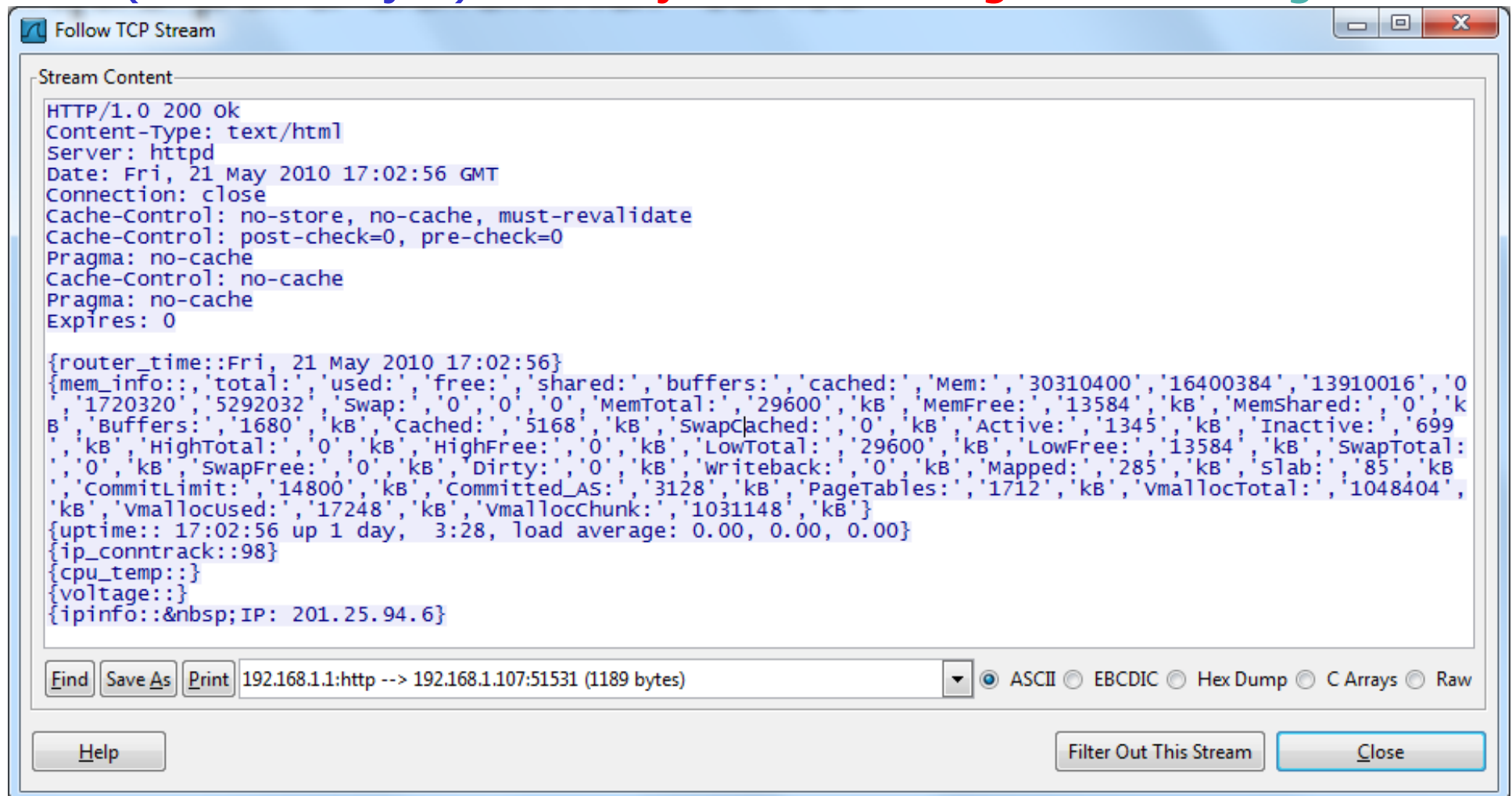
{cpu\_temp::}

{voltage::}

{ipinfo::&nbsp;IP: 201.25.94.6}

# HTTP – Funcionamento Básico

- Resposta → Versão do Protocolo + Código de Resposta (com descrição) + Cabeçaho da Mensagem + Mensagem



The screenshot shows a window titled "Follow TCP Stream" with a "Stream Content" tab. The content displays an HTTP response from a server. The response headers include "HTTP/1.0 200 ok", "Content-Type: text/html", "Server: httpd", "Date: Fri, 21 May 2010 17:02:56 GMT", "Connection: close", and several "Cache-Control" and "Pragma" directives. The body of the response is a JSON object containing system information from a router, such as memory usage, uptime, and IP address.

```
HTTP/1.0 200 ok
Content-Type: text/html
Server: httpd
Date: Fri, 21 May 2010 17:02:56 GMT
Connection: close
Cache-Control: no-store, no-cache, must-revalidate
Cache-Control: post-check=0, pre-check=0
Pragma: no-cache
Cache-Control: no-cache
Pragma: no-cache
Expires: 0

{router_time::Fri, 21 May 2010 17:02:56}
{mem_info::total::used::free::shared::buffers::cached::Mem::30310400,16400384,13910016,0,1720320,5292032,Swap::0,0,0,MemTotal::29600,kB,MemFree::13584,kB,MemShared::0,kB,Buffers::1680,kB,Cached::5168,kB,SwapCached::0,kB,Active::1345,kB,Inactive::699,kB,HighTotal::0,kB,HighFree::0,kB,LowTotal::29600,kB,LowFree::13584,kB,SwapTotal::0,kB,SwapFree::0,kB,Dirty::0,kB,writeback::0,kB,Mapped::285,kB,slab::85,kB,CommitLimit::14800,kB,Committed_AS::3128,kB,PageTables::1712,kB,vmallocTotal::1048404,kB,vmallocUsed::17248,kB,vmallocChunk::1031148,kB}
{uptime::17:02:56 up 1 day, 3:28, load average: 0.00, 0.00, 0.00}
{ip_contrack::98}
{cpu_temp::}
{voltage::}
{ipinfo::&nbsp;IP: 201.25.94.6}
```

Find Save As Print 192.168.1.1:http --> 192.168.1.107:51531 (1189 bytes) [dropdown] ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

# Request Line e Status Line

- Request Line, formada por (cliente):
  - Request Method (Método);
  - URI (Uniform Resource Identifier): identificação do recurso ao qual o Request Method deve ser aplicado;
  - Versão do protocolo (atualmente HTTP/1.1).
  - GET /exploits/1056 HTTP/1.1
- Status Line, formada por (servidor):
  - Versão do protocolo;
  - Código de erro ou de sucesso;
  - Texto descritivo do erro ou sucesso.
  - HTTP/1.1 200 OK

# Cabeçalho da Mensagem

- Um ou mais campos no formato:
  - nome\_do\_campo:valor\_do\_campo
- Alguns campos são específicos da mensagem enviada pelo cliente, já outros do lado do servidor.

GET /exploits/1056 HTTP/1.1

Host: [www.milw0rm.org](http://www.milw0rm.org)

User-Agent: Mozilla/5.0 Gecko/20091102 Firefox/3.5.5 FBSMTWB

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

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 115

Connection: keep-alive

Referer: <http://www.milw0rm.org/search.php>

# Cabeçalho da Mensagem - Requisição

- Os campos de uma requisição são classificados nos seguintes grupos:
  - **General Header** (dizem respeito à mensagem sendo transmitida);
  - **Request Header** (fornecem informações adicionais sobre a requisição, a respeito do próprio cliente, ao servidor);
  - **Entity Header** (definem metainformações a respeito do corpo da mensagem ou do recurso identificado pela mensagem).



# Cabeçalho da Mensagem - Resposta

- Os campos de uma resposta são classificados nos seguintes grupos:
  - **General Header** (dizem respeito à mensagem sendo transmitida);
  - **Response Header** (fornecem informações adicionais a respeito da resposta e do Servidor);
  - **Entity Header** (definem metainformações a respeito do corpo da mensagem ou do recurso identificado pela mensagem).

# Campos do cabeçalho

General Header	Entity Header	Request Header	Response Header
Cache-Control Connection Date Pragma Trailer Transfer-Encoding Upgrade Via Warning	Allow Content-Encoding Content-Language Content-Length Content-Location Content-MD5 Content-Range Content-Type Expires Last-Modified	Accept Accept_Charset Accept_Encoding Accept_Language Authorization Expect From Host If-Match If-Modified-Since If-None-Match If-Range If-Unmodified-Since Max-Forwards Proxy-Authorization Range Referer User-Agent	Accept-Ranges Age ETag Location Proxy_Authenticate Retry-After Server Vary WWW-Authenticate

# Corpo da Mensagem

- O corpo da mensagem é o container dos dados que foram referenciados pelo URI de uma requisição ou de uma resposta;
- Pode conter também páginas/textos informando erros ao tratar a solicitação;

# Métodos HTTP

- Indicam a ação a ser executada com o recurso especificado, em outras palavras o método diz o que fazer com o URI solicitado pelo cliente.
- **OPTIONS** → Solicita ao servidor a lista dos recursos aceitos por ele;
- **GET** → É o método mais comum, é utilizado na simples solicitação de um recurso qualquer no servidor seguido de um retorno do servidor.

# Métodos HTTP

- **HEAD** → Semelhante ao GET, porém o servidor responde sem o corpo da mensagem, ou seja, somente com cabeçalhos. Bastante útil, por exemplo, para saber se o cache local está atualizado.
- **PUT** → Envia um recurso, como por exemplo um arquivo.
- **DELETE** → deleta um recurso, geralmente um arquivo.

# Métodos HTTP

- **POST** → Envia dados para serem processados pelo recurso, os dados seguem embutidos no corpo da mensagem e formatados como uma *query string*. Também fornece cabeçalhos adicionais especificando tamanho e formato dos dados. Por isso esse método é reconhecido como mais seguro para transferência de informações do cliente ao servidor, ao contrário do método GET no qual as informações são passadas na URL do recurso.

# Métodos HTTP

- **TRACE** → Permite ao cliente saber como sua requisição está sendo tratada no servidor, podendo usar essa informação para fazer *debug*.
- **CONNECT** → Método utilizado para criar um túnel através de um *proxy*, que pode ser criptografado. Útil quando se quer usar um *Proxy HTTP* para se conectar em uma aplicação não HTTP.

# Métodos HTTP

- Nem todos os métodos podem ser utilizados em um determinado recurso, caso o cliente faça uso de um método inválido ele receberá uma resposta com código de erro:
  - **405** → Método existem mas não é permitida sua utilização naquele recurso;
  - **501** → Método não implementado ou desconhecido.



# Códigos de Resposta

- 1XX → Informacional, apenas informa que a requisição foi recebida e que será dado início ao processo;
- 2XX → Mensagem de sucesso. A requisição foi recebida, entendida e aceita. O processo foi iniciado e concluído com sucesso;
- 3XX → Redirecionamento. Outra ação deve ser tomada para que a requisição seja atendida com sucesso;
- 4XX → Erro no cliente. A requisição contém algum erro, como por exemplo erro de sintaxe;
- 5XX → Erro no servidor. O servidor falhou em atender uma requisição aparentemente válida.

# Virtual Hosting

- O que é? → Um único servidor hospedando vários sites da seguinte forma:
  - Vários FQDNs apontando para um único endereço IP e o servidor WEB determinando o site através do campo "host" da requisição HTTP.
- Outras formas
  - Vários caminhos;
  - Vários IPs no servidor;
  - Várias portas;

# Tipos de Autenticação

- Basic (desde o HTTP/1.0);
  - Muito simples e inseguro (MITM);
  - Userid e senhas enviados codificado em base64;
- Digest Access Authentication - RFC 2069;
  - Mais seguro;
  - Userid enviado em texto claro;
  - Mecanismo de desafio para evitar *replay attacks*;
  - Baseado no algoritmo de *hash* MD5;
  - Vulnerável a MITM+Brute Force;
  - Melhorado na RFC 2617, diminuindo a vulnerabilidade a criptoanalise.

# Basic – RFC 1945

- Como se calcula a resposta.

```
$ echo -n "Aladdin:open sesame" | openssl enc -base64
```

```
QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

```
$ echo "QWxhZGRpbjpvcGVuIHNlc2FtZQ==" | openssl enc -base64 -d  
Aladdin:open sesame
```

- Note que por se tratar de codificação é possível voltar ao texto original, diferentemente de quando se utiliza um mecanismo de *HASH*.

# Basic – RFC 1945

## Client request (no authentication):

GET /private/index.html HTTP/1.0  
Host: localhost

## Server response:

HTTP/1.0 401 Authorization Required  
Server: HTTPd/1.0  
Date: Sat, 27 Nov 2004 10:18:15 GMT  
WWW-Authenticate: Basic realm="Secure Area"  
Content-Type: text/html  
Content-Length: 311

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
<HEAD>
<TITLE>Error</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY><H1>401 Unauthorized.</H1></BODY>
</HTML>
```

## Client request (user name "Aladdin", password "open sesame"):

GET /private/index.html HTTP/1.0  
Host: localhost  
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

## Server response:

HTTP/1.0 200 OK  
Server: HTTPd/1.0  
Date: Sat, 27 Nov 2004  
10:19:07 GMT  
Content-Type: text/html  
Content-Length: 10476

# Digest Access Authentication – RFC 2059

$$HA1 = MD5(A1) = MD5(\text{username} : \text{realm} : \text{password})$$

$$HA2 = MD5(A2) = MD5(\text{method} : \text{digestURI})$$

$$\text{response} = MD5(HA1 : \text{nonce} : HA2)$$

# Digest Access Authentication – RFC 2617

$$HA1 = MD5(A1) = MD5(\text{username} : \text{realm} : \text{password})$$

Se a diretiva qop tem valor "auth" ou não está especificada, então HA2 é:

$$HA2 = MD5(A2) = MD5(\text{method} : \text{digestURI})$$

Se a diretiva qop tem valor "auth-int", então HA2 é:

$$HA2 = MD5(A2) = MD5(\text{method} : \text{digestURI} : MD5(\text{entityBody}))$$

Se a diretiva qop tem valor "auth" ou "auth-int", a resposta é:

$$\text{response} = MD5(HA1 : \text{nonce} : \text{nonceCount} : \text{clientNonce} : \text{qop} : HA2)$$

Se a diretiva qop não está especificada resposta é:

$$\text{response} = MD5(HA1 : \text{nonce} : HA2)$$

# Digest Access Authentication – RFC 2617

## Client request (no authentication):

```
GET /dir/index.html HTTP/1.0
Host: localhost
```

## Server response:

```
HTTP/1.0 401 Unauthorized
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
WWW-Authenticate: Digest realm="testrealm@host.com",
                   qop="auth,auth-int",
                   nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                   opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html
Content-Length: 311
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
<HEAD>
  <TITLE>Error</TITLE>
  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY><H1>401 Unauthorized.</H1></BODY>
</HTML>
```



# Digest Access Authentication – RFC 2617

**Client request (user name "Mufasa", password "Circle Of Life"):**

```
GET /dir/index.html HTTP/1.0
Host: localhost
Authorization: Digest username="Mufasa",
    realm="testrealm@host.com",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    uri="/dir/index.html",
    qop=auth,
    nc=00000001,
    cnonce="0a4f113b",
    response="6629fae49393a05397450978507c4ef1",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

**Server response:**

```
HTTP/1.0 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:27:03 GMT
Content-Type: text/html
Content-Length: 7984
```

# Tipos de Autenticação

## ■ Outros

- Algoritmos de chave pública, baseado em certificados digitais;
- Kerberos, mecanismo de autenticação robusto, seguro e bastante complexo;

# Tópicos para se comentar...

- Cookies, Sequestro de sessões, ...
- Carga (implementação, versão, tipo de sites, cache, hardware, so).
- Kernel-mode, User-mode, VMs, Thread, Instancias...
- Tráfego legitimo alto, DDoS, Worms, Robots...
- Load Balanced, Trafic Shaping, Gerenciamento de Banda, Firewalls 7-Layer, Web Cache Server...
- <http://news.netcraft.com/>