به نام خدا

HTTP

یک راهنمای کاربردی برای نوشتن مشتری و سرویس دهنده (Clients & Servers)

HTTP یک پروتکل شبکه ساده و قوی برای وب است. دانستن شیوه کاربرد و کارایی HTTP به شما امکان نوشتن مرورگرهای وب، سرورهای وب، صفحه های دانلود خودکار، چک کننده های لینک و بقیه ابزارهای مفید را میدهد.

این متن آموزشی به سادگی در مورد ارتباط با HTTP توضیح می دهد و جزئیات نوشتن مشتری و سرویس دهنده HTTP (Clients & Servers) را به شما آموزش می دهد. همچنین شما را با کلیات "Socket Programming" آشنا می کند. HTTP برای شروع "Socket Programming" کاملا کافی و ساده است.

هنگامی که شما این Method را می خوانید احتمالاً از CGI استفاده می کنید در غیر این صورت اول باید آن را یاد بگیرید. نیمه اول این متن در مورد پایه ها و کلیات HTTP ۱,۰ و نیمه دوم در مورد نیازمندی های به وجود آمده و امکانات جدید آن بحث می کند.

این متن تمامی جزئیات HTTP ۱,۱ را توضیح نمی دهد بلکه کلیات و چارچوبی از آن را توضیح می دهد تا دریابید به چه طریق نیازها از طریق HTTP بر آورده می شوند و در صورت نیاز کجا می توانید اطلاعات بیشتر به دست بیاورید.

قبل از شروع دو پاراگراف زیر را با دقت مطالعه کنید:

<LECTURE>

نوشتن **HTTP** و یا سایر برنامه های شبکه نیازمند دقت بیشتر نسبت به نوشتن برنامه برای یک سیستم است (به نظر شما چرا؟)!

شما باید طبق قوانین استاندارد شده حرکت کنید (برنامه بنویسید)،در غیر این صورت کسی شما را درک نمی کند و مورد توجه قرار نمی گیرید. ولی از همه چیز مهم تر بار مسئولیتی است که با نوشتن برنامه برای سیستم های دیگر متوجه شما میشود. نوشتن یک برنامه ی بد برای سیسم خودتان موجب تلف شدن منابع خودتان میشود میشود که شامل زیر می شود:

CPU time, bandwidth, memory

نوشتن یک برنامه شبکه بد باعث تلف شدن منابع مردم دیگر و نوشتن یک برنامه واقعاً بد باعث تلف کردن منابع هزاران نفر در یک زمان می شود! برنامه نویسی در هم و برهم و نامرتب شبکه از طریق استانداردها می تواند اصلاح شود. ممکن است مطمئن تر شود ولی کارایی و بهره کمتری نسبت به یک برنامه نویسی مرتب و از روی اصول خواهد داشت. بنابراین با دقت و مودب باشید و به نظرات دیگران احترام بگذارید.

قبل از اینکه بدانید چه می کنید دچار وسوسه نشوید تا برنامه هایی بنویسید که به طور خودکار لینک های وب را دنبال می کنند (**Robots & Spiders**).

این برنامه ها می توانند مفید واقع شوند ولی یک روبات که بد نوشته شده یکی از بدترین نوع برنامه ها بر روی وب است. به طور کورکورانه به تعداد لینک ها می افزاید و به سرعت منابع یک سرور را به یغما می برد. اگر خواستید برنامه ای مثل یک روبات بنویسید، به لینک زیر مراجعه کنید:

http://info.webcrawler.com/mak/projects/robots/robots.html

ممکن است در آنجا برنامه مورد نظر شما موجود باشد و اگر هم می خواهید خودتان بنویسید به سه لینک زیر سر بزنید:

http://info.webcrawler.com/mak/projects/robots/guidelines.html
http://info.webcrawler.com/mak/projects/robots/norobots.html
http://info.webcrawler.com/mak/projects/robots/exclusion.html

</LECTURE>

حال شروع به کار می کنیم.

HTTP چیست؟

HTTP مخفف عبارت "Hypertext Transfer Protocol" یک پروتکل شبکه برای انتقال مجازی تمامی فایل ها و انواع داده ها (With all EXT) که مجموعا منابع نامیده می شوند) بر روی شبکه تار عنکبوتی جهانی (World Wide Web) است. این داده ها میتوانند شامل صفحات HTML، عکس ها، نتایج فرم ها و یا هر چیز دیگر باشد. معمولا **HTTP** بر روی TCP/IP Sockets جای میگیرد (و این متن از شرح بقیه حالت ها صرف نظر می کند).

یک مرورگر وب نوعی مشتری HTTP (HTTP Client) محسوب می شود، زیرا مرورگر درخواست خود را به سرویس دهنده HTTP (HTTP Server) یا وب سرور می فرستد و وب سرور (Web Server) به درخواست او پاسخ می دهد. پورت استاندارد و پیش فرض برای سرویس دهنده HTTP پورت ۸۰ می باشد. بر روی این پورت منتظر درخواست ها می ماند (البته می تواند از هر پورت دیگر نیز استفاده کند).

Resource ها چیستند؟

HTTP علاوه بر انتقال فایلها، برای انتقال منابع نیز به کار می رود، یک Resource توده ای از اطلاعات است که توسط URL مشخص و به سرور منتقل می شود.

بیشترین نوع منابع، فایل ها هستند ولی یک منبع می تواند: نتایج تولید شده توسط یک تولید کننده، خروجی یک اسکریپت CGI، یک سند که قابل دسترسی به زبان های گوناگون است و یا چیزهای دیگر باشد.

در هنگام یادگیری HTTP، برای سادگی بیشتر می توانید منبع را مشابه فایل در نظر بگیرید ولی با حالتی عمومی تر، به عبارت دیگر تقریبا تمامی منابع HTTP، یا فایل و یا خروجی اسکریپت از طرف سرور می باشند.

ساختار چگونگی انجام مذاکرات HTTP:

مشابه اکثر پروتکل های شبکه، HTTP از مدل مشتری-سرویس دهنده (Client-Server) استفاده می کند. مشتری HTTP یک ارتباط (Connection) با سرویس دهنده برقرار می کند و درخواست (Request Message) خود را از طریق آن ارتباط می فرستد.

سپس سرور پیغامی حاوی پاسخ (Response Message) را به مشتری برمی گرداند (Response Message معمولا شامل Resource خواسته شده توسط مشتری است)! بعد از پاسخ به درخواست، سرور ارتباط را قطع می کند. Format پیغام های درخواست و پاسخ مشابه هم میباشند و هر دو از کلمات انگلیسی تشکیل شده اند. هر دو شامل:

- خط آغازین

- یک و یا چند *Header Line*

- یک خط خالی

- و پیام اختیاری (مثل یک فایل و یا نتایج فرم یا خروجی فرم).

در زیر ساختار کلی آنرا می بینید:

Example Code

```
<initial line, different for request vs. response>
Header1: value1
Header2: value2
Header3: value3

<optional message body goes here, like file contents or query data;
it can be many lines long, or even binary data $&*%@!^$@>
```

خطوط Initial و Headers **باید به ترتیب** CRLS **تنظیم شوند** (یعنی از پایان یک خط به متن به آغاز خط بعدی منتقل شوند). **در اینجا** CR **و** LF **کدهای** ASCII **دارای ارزش ۱۳ و ۱۰ هستند** (توجه داشته باشید در سیستم های مختلف شاید این مقادیر تغییر کند).

آغاز کردن خط درخواست:

خط آغازین برای درخواست و عکس العمل متفاوت است. خط درخواست دارای ۳ قسمت است که با Space **از یکدیگر جدا می شوند:**

یک روش درخواست (Get Post Head)، آدرس محلی Resource درخواستی و نسخه HTTP در حال استفاده

مانند:

Example Code

```
GET /path/to/file/index.html HTTP/1.0
```

توجه:

GET متداولترین روش مورد استفاده است که به زبان خودمانی به سرور می گوید: "**Resource را بده به مـن!**" روش هـای دیگـر شامل POST و HEAD هستند که در آینده در مورد آنها توضیح داده می شود. نام روش ها همواره با حروف بزرگ است.

آدرس محلی Resource قسمتی از URL است که بعد از اسم میزبان می آید که request URI نامیـده مـی شـود (یـک URI مشـابه URL است ولی در حالتی کلی تر) نسخه HTTP همواره به فورمت HTTP / x.x در حروف بزرگ درج می شود.

آغاز کردن خط پاسخ (یا خط وضعیت):

خط آغازین پاسخ که *خط وضعیت* هم نامیده می شود، هم دارای ۳ قسمت است که با Space از یکدیگر جدا می شوند. نسخه HTTP در حال استفاده، یک کد مربوط به عکس العمل ارسالی سرور و یک اصطلاح انگلیسی که به شرح کد وضعیت می پردازد. مانند:

Example Code

```
HTTP/1.0 200 OK
Or
HTTP/1.0 404 Not Found
```

توجه:

نسخه HTTP همانند فورمت خط درخواست درج می شود "**HTTP/x.x**".

کد وضعیت داده شده برای خواندن کامپیوتر و به اصطلاح انگلیسی برای خواندن انسان درج می شود و می تواند تغییر کند. کد وضعیت شامل سه عدد صحیح است که در کنار هم آمده اند:

۱xx: تنها نشان دهنده یک پیغام خبری است.

۲xx: نشان دهنده پایان یافتن موفقیت آمیز یک عمل است.

۳xx: مشتری را به یک URL دیگر هدایت می کند.

متداول ترین کدهای وضعیت:

OK ۲۰۰ که به معنای موفقیت عمل است و در پی آن **resource** درخواستی (فایل و یا خروجی یک اسکریپت) در بدنه پیغام فرستاده میشود.

Not Found resource ۴۰۴ درخواستی موجود نمی باشد.

301 Moved Permanently
302 Moved Temporarily
303 See Other (HTTP 1.1 only)

Resource به URL دیگری منتقل شده است (که آدرس جدید در خط: **Location** در **header** پیغام عکس العمل این روش بارها توسط اسکریپت های **CGI** برای تغییر مکان مرورگر استفاده می شود).

Server Error ۵۰۰: یک خطای غیر منتظره در سرور: در اکثر مواقع یک اسکریپت در قسمت سرور دارای خطای منطقی است و یا نمی تواند به درستی اجرا شود.

در لینک زیر لیست کامل این کدها را مشاهده می کنید:

http://g0tr00t.mson.org/docs/misc/HTTP.html#httpspec

خطوط Header:

خطوط Header مهیا کننده اطلاعات لازم برای، درخواست و یا پاسخ و به طور کلی در رابطه با شئ که در بدنه نامه ارسال می شود، هستند.

این خطوط به صورت انگلیسی درج می شوند و هر Header در یک خط جدا تعریف می شود "Header-Name:Value" که به صورت CRLF پایان می یابد. این همان فورمتی است که برای ارسال نامه و یا اخبار استفاده می شود (در RFC-۸۲۲ قسمت سوم توضیح داده شده است).

جزئیات RFC ۸۲۲ در مورد خطوط Header:

همان طور که در بالا گفته شد باید به صورت CRLF پایان یابند. نام Header نسبت به حروف کوچک و بزرگ حساس نمی باشد (Not Case Sensitive). هر تعداد فاصله و یا TAB می تواند بین ":" و ارزش قرار بگیرد. خطوط Header با فاصله و یا TAB آغاز می شوند و بعضی مواقع برای آسان خواندن در چند خط تجزیه می شوند.

پس بنابر توضیحات فوق دو Header زیر با هم هیچ تفاوتی ندارند:

Example Code

```
Header1: some-long-value-1a, some-long-value-1b
HEADER1: some-long-value-1a,
                some-long-value-1b
```

بهتر است بدانید که HTTP ۱,۰ دارای ۱۶ Header تعریف شده است که البته هیچ کدام ضروری نیستند و HTTP ۱,۱ دارای ۴۶ Header تعریف شده است که از میان این Header ها، "Host:" لازم و ضروری است.

برای رعایت کردن آداب شبکه از این Header ها استفاده کنید:

Form Header: که آدرس صندوق پستی شخصی که درخواست را فرستاده مشخص می کند و یا برنامه اجرا شده خود این کار را میکند (این Header باید توسط خود کاربر تعیین شود).

User-Agent Header: مشخص می کند چه برنامه ای درخواست را فرستاده است که بصورت "Program-Name/x.xx" نوشته میشود.

x.xx شماره نسخه برنامه است. به عنوان مثال برای NetScape ۳,۰، قسمت Header به صورت زیر فرستاده می شود:

"User-Agent: Mozilla/3.0 Gold"

این Header به مسئولین سرورها برای رفع مشکل کمک می کند، آنها همچنین مشخصاتی در مورد کاربر آشکار می کنند. وقتی شما تصمیم می گیرید چه Header هایی در برنامه خود قرار بدهید باید از امنیت اطلاعاتی که ارسال می کنید مطمئن شوید..

اگر شما در حال نوشتن یک سرویس دهنده هستید، در مورد قرار دادن این Header ها در جواب خود تصمیم بگیرید:

Server Header: که همانند User-Agent در مورد درخواست است که نام و شماره نسخه برنامه پاسخ دهنده به درخواست را مشخص می کند و به صورت "Program-Name/x.xx" درج می شود برای مثال نسخه آزمایشی سرورهای Apache، قسمت زیر را بر می گرداند:

"Server: Apache/1.2b3-dev"

Last Modified Header: که تاریخ آخرین تغییرات را برای Resource درخواستی ارسال می کند کـه از سـاعت گرینـویچ اسـتفاده میشود:

"Last-Modified: Fri, 31, Dec, 2001 23:59:56 GMT"

بدنه پیغام (Body):

یک پیام HTTP ممکن است بعد از خطوط Header شامل یک بدنه پیغام که داده ها در آن ثبت شده باشد. در پاسخ به درخواست resource درخوستی در همین قسمت پیغام درج به مشتری فرستاده می شود (متداولترین حالت استفاده از بدنه پیغام) و یا متن توضیح داده شده در مورد یک پیغام خطا باشد.

در یک درخواست اطلاعات وارد شده توسط کاربر و یا فایل فرستاده شده توسط اودر این قسمت ثبت و به سرور فرستاده می شود.

اگر پیغامی شامل بدنه باشد همیشه باید خطوط header زیر که در مورد بدنه توضیح می دهند در آن پیغام وجود داشته باشد:

Content-Type: این Header نوع داده ثبت شده در بدنه را مشخص می کند. مثل: Image/gif یا text/html

Content-Length: این Header تعداد بایت های درج شده در متن پیغام را مشخص می کند.

الگوی انجام یک مبادله HTTP:

برای فراخواندن یک فایل در URL مثلا برای http://www.somehost.com/path/file.html، در ابتدا یک ارتباط با میزبان www.somehost.com بر روی پورت ۸۰ برقرار کنید (پورت پیش فرض وب سرور ۸۰ است و به همین دلیل در URL مشخص نمی شود. در غیر این صورت به فورمت http://www.somehost.com:port نوشته می شود).

Example Code

```
GET /path/file.html HTTP/1.0
From: someuser@crouz.com
User-Agent: HTTPTool/1.0
[blank line here]
```

عکس العمل سرور در مقابل درخواست شما چیزی مشابه خطوط زیر خواهد بود:

Example Code

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
<h1>Happy New Millennium!</h1>
(more file contents)
 .
 .
 .

<!-- ZoneLabs Popup Blocking Insertion -->
<script language='javascript'>postamble();</script>
</body>
</html>
```

بعد از ارسال پاسخ، سرور ارتباط را قطع می کند. برای آشنایی بیشتر در زیر به وسیله **telnet** این ارتباط را برقرار می کنیم (تجربه عملی با **HTTP** به وسیله **telnet**):

به وسیله **telnet** شما می توانید یک ارتباط با سرور **http** برقرار کنید و به صورت دستی درخواست خود را تایپ و ارسال نمائید. به این ترتیب پاسخ سرور در صفحه **telnet** شما ظاهر می شود. این خیلی اوقات در مواقع رفع اشکال و آگاهی از طریق پاسخ به یک درخواست خاص، بسیار مفید است.

خوب در command Prompt خود خط زیر را تایپ کنید:

Example Code

```
telnet www.somehost.com 80
```

سپس درخواست خود را خط به خط مشابه زیر وارد کنید:

```
GET /path/file.html HTTP/1.0
[headers here, if any]
[blank line here]
```

وقتی درخواست شما به پایان رسید به وسیله یک خاط خالی به سرور اعلام می کنید و سرور پاسخ را ارسال می کند که شامل خط وضعیت، Header ها و بدنه پیغام است.

روش های دیگر HTTP مانند Head و Post:

در کنار GET، دو روشی که بیشترین کاربرد را دارند، Head و Post هستند.

روش Head:

این درخواست دقیقا مشابه درخواست GET است تنها با این تفاوت که از سرور تنها Header های پیغام عکس العمل را درخواست و هیچ گونه resource دریافت نمی کند (به عنوان مثال بدنه پیغام ارسال نمی شود) و این در مواقعی که می خواهید بدون دانلود یک resource از مشخصات آن آگاه شوید، سودمند است.

در پاسخ به این درخواست، سرور باید هیچ گاه بدنه پیغام را ارسال کند تنها خط وضعیت و Header ها.
این روش را هم می توانید به وسیله telnet همانند روش GET تجربه کنید تنها با این تفوات که خط آغازین یک چیزی مشابه زیر است:

HEAD /path/file.html HTTP/1.0

روش POST:

روش Post برای ارسال داده به سرور هنگامی استفاده می شود که داده ها قبل از ارسال نیاز به پردازش و عبور از بعضی مراحل مشخص شده دارند مانند یک اسکریپت CGI (CGI Script).

تفاوت های درخواست POST و GET:

* در بدنه پیغام درخواست، مجموعه ای از داده ها قرار دارد و معمولا چند header اضافه برای تشریح داده ها مانند: Content-Type و Content-Length همراه درخواست ارسال میشود.

* درخواست URI، به یک resource اشاره نمی کند بلکه آدرس برنامه ای که عملیات را بر روی داده های ارسالی انجام می دهد را مشخص می کند.

* عکس العمل HTTP به طور معمول یک فایل ثابت نیست بلکه خروجی برنامه است.

بیشترین کاربرد POST، تاکنون، ارسال اطلاعات یک فرم HTML به اسکریپت CGI بوده است. در این حالت Content-Type معمولا به صورت:

application/x-www-form-URLEncoded

و Content-Length طول URL رمزگذاری شده داده فرم را مشخص می کند (بعد از مبحث به شرح چگونگی رمزگذاری می پردازیم).

اسکریپت CGI بدنه پیغام درخواست را بهصورت ورودی (STDIN) دریافت می کند سپس رمز آنرا می گشاید. در زیر نوعی ثبت اطلاعات فرم توسط POST آمده است:

Example Code

```
POST /path/script.cgi HTTP/1.0
From: frog@jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies
```

شما می توانید از روش POST نه تنها برای انتقال داده های فرم بلکه برای هر نوع داده ای استفاده کنید فقط باید توجه داشته باشید که هم فرستنده و هم گیرنده هر دو با فورمت ارسالی داده ها آشنا باشند.

روش GET هم می تواند برای ارسال اطلاعات فرم استفاده شود. در این حالت داده های فرم رمزگذاری و به URI افزوده می شوند (در مورد این روش هم توضیح داده می شود).

اگر شما قصد نوشتن سرور HTTO دارید که از اسکریپت CGI پشتیبانی می کند شما باید NCSA's CGI definition را مطالعه کنید و همچنین Environment Variables.

URL-Enconding:

داده های یک فرم html معمولا به صورت URL-encoded در روش های GET یا POST ثبت می شوند و در زیر چگونگی رمزگذاری تشریح شده است:

اول- تمامی کاراکترهایی که غیر از حروف و عدد هستند را "%xx" تبدیل کنید (xx، معادل ارزش کاراکتر در جدول اسکی است). به این کاراکترها؛ کاراکترهای ناامن "UnSafe" نیز گفته می شود (مثل & ^ % $ # @ = +).

دوم- تمام فاصله ها را به "+" تبدیل کنید.

سوم- نام ها و ارزشهای متغیرها را به وسیله = و & به صورت رشته تبدیل کنید. مثل:

Example Code

```
name1=value1&name2=value2&name3=value3
```

چهارم- این رشته در POST در قسمت "بدنه پیام" و در GET قسمت "Query String" می باشد.

برای مثال اگر در یک فرم نام یک field به صورت "Name" تعیین شده باشد و کاربر اسم "Lucy" را ثبت کند و field دیگر با اسم "Neighbors" و ارزش "Fred & Ethel" باشد URL رمزگذاری شده به صورت زیر می باشد:

Example Code

```
name=Lucy&neighbors=Fred+%26+Ethel
```

که دارای طول ۳۴ می باشد.

از لحاظ فنی عمل "URL-encoding" تنها مرحله اول بالا است که در RFC-۲۳۹۶ قسمت ۲,۴ (سابقا در RFC-۱۷۳۸ قسمت ۲,۲) توضیح داده شده است و تمام این مراحل برای تبادل اطلاعات ثبت شده فرم به یک رشته بلند است.

استفاده از GET برای ثبت داده های فرم:

از GET نیز می توان برای ارسال مجموعه ای از داده های کوچک به سرور استفاده کرد. کلید این کار تنها در فهمیدن دقیق URI "یک درخواست" است.

URI حتما نباید اسم فایل باشد بلکه یک رشته بلند است که منبع اطلاعات را بر روی سرور مشخص می کند که این ممکن است اسم فایل یا برای مثال یک آدرس مشخص به یک منبع اطلاعاتی مشخص باشد. نتایج این ثبت داده حتما نباید یک فایل باشد بلکه می تواند نتایج یک جستجو، که موتور جستجو و داده منتقل شده از فرم به همراه هم آن را تولید کرده اند، باشد.

بنابراین برای ارسال اطلاعات به یک اسکریپت CGI توسط GET با گذاشتن علامت پرسش "?" بعد از URL رشته رمزگذاری را وارد کنید:

Example Code

```
GET /path/script.cgi?field1=value1&field2=value2 HTTP/1.0
```

این بود طریقه ارسال داده های یک فرم به وسیله GET که برای انتقال داده های کوچک استفاده می شود (برای داده های بزرگ از POST استفاده کنید).

HTTP Proxies:

یک HTTP Proxy در واقع برنامه ای است که بین مشتری و سرویس دهنده قرار می گیرد در خواست ها از مشتری به Proxy فرستاده می شود و Proxy درخواست را به سرویس دهنده می فرستد. پاسخ هم از همین راه و از طریق Proxy به مشتری می رسد. به این ترتیب proxy هم نقش سرویس دهنده و هم نقش مشتری را اعمال می کند. Proxy ها معمولا در Firewall ها مورد استفاده قرار می گیرند.

وقتی یک مشتری از proxy استفاده می کند تمام درخواست های خود را به جای سرور به آن proxy ارسال می کند. درخواستی که به proxy فرستاده می شود با درخواستی که به سرور ارسال می شود متفاوت است. در خط اول از URL کامل resource درخواستی (به جای path) استفاده می شود:

Example Code

```
GET http://www.somehost.com/path/file.html HTTP/1.0
```

از این طریق proxy می داند که درخواست را باید به کدام سرور ارسال کند.

پایان قسمت اول:

خوب به پایان نیمه اول این آموزش رسیدیم. در این قسمت شما با کلیات و ساختمان HTTP ۱,۰ آشنا شدید. ادامه این آموزش به نحوه ارتقا برنامه های شما برای استفاده از HTTP ۱,۱ می پردازد و نیازمندی های جدید آن را بررسی میکند. در پایان این بخش به یاد داشته باشید که شعار برنامه نویسی برای شبکه در یک جمله خلاصه می شود: " برای چیزی که درخواست می کنید و در نحوه درخواست خود سخت گیری و در برابر درخواست دیگران مدارا کنید" . به عبارت دیگر مشتری ها و سرور های دیگر شاید به نوعی دیگر درخواست خود را به شما ارسال کنند ولی شما به نحوی باید برنامه را بنویسید تا به درخواست های غیر معمول احتمالی هم پاسخ دهد:

* خطوط Header باید به صورت CRLF پایان یابد ولی شخصی ممکن است به صورت LF به خط پایان دهد. بنابراین هم به CRLF و LF پاسخ گویید.
* سه قسمت خط آغازین باید به وسیله یک SPACE از یکدیگر جدا شوند ولی ممکن است از چند SPACE استفاده شود. پس هر تعدادی فضای خالی را مابین این قسمت ها قبول کنید.

خوب قبل از شروع قسمت دوم به مثال GET URL که در perl نوشته شده، توجه کنید. این برنامه یک resource را که آدرس آن داده شده به وسیله روش GET دریافت و آنرا ذخیره می کند.

Download geturl10.pl or view it

HTTP ۱,۱:

همانند دیگر Protocolها، HTTP هم حرکتی رو به رشد دارد. بعد از HTTP نسخه اول، HTTP ۱,۱ به وجود آمد که دارای برتری هایی نسبت به نسخه اولیه است:

* عکس العمل سریع تر به وسیله انجام چندین معامله از طریق یک ارتباط مداوم.
* عکس العمل سریع تر و حفظ کردن پهنای باند به وسیله اضافه کردن پشتیبانی Cache
* عکس العمل سریع تر برای صفحات تولید شده پویا به وسیله پشتیبانی "Chunked Encoding" که اجازه ارسال پاسخ قبل از مشخص شدن طول آن را می دهد.
* موثر در استفاده از IP Addresses، به وسیله سرویس دادن به چندین دامنه توسط یک IP address.

HTTP ۱،۱ یک سری امکانات بیشتر نسبت به نسخه قبلی آن برای استفاده مشتری و سرویس دهنده نیاز دارد، در دو قسمت آینده در مودر جزئیات نوشتن مشتری و سرویس دهنده برای HTTP ۱,۱ توضیح داده می شود. اگر شما فقط مشتری می نویسید فقط قسمت اول کفایت می کند ولی چنانچه در حال نوشتن سرور هستید باید هر دو قسمت را بخوانید.

در دو قسمت بعدی به شرح مشتری و سرویس دهنده برای HTTP ۱,۱ می پردازیم.

HTTP ۱,۱ Clients: برنامه های مشتری برای سازگاری با HTTP ۱,۱ باید:

* برای هر درخواست Header، Host را هم درج کنند.
* پاسخ هایی که به صورت Chunked ارسال می شود را قبول کنند.
* باید ارتباط مداوم را پشتیبانی کنند یا با هر درخواست Header، "Connection: Close" را ارسال کنند.
* با کد "۱۰۰ Continue" آشنا باشند.

Host Header:

به وسیله HTTP ۱,۱ این امکان فراهم شده است که یک سرور یا یک IP به صورت multi-homed باشد. به عبارت دیگر یک مکان باری چندین دامنه وب. برای مثال www.host۱.com و www.host۲.com هر دو بر روی یک سرور قرار بگیرند.
قرار داشتن چندین دامنه بر روی یک سرور همانند با اشتراک گذاری تلفن بر چند نفر است، شخصی که تماس می گیرد می داند با چه کسی کار دارد ولی شخصی که تلفن را پاسخ می دهد، نمی داند!

بنابراین هر درخواست HTTP باید نام میزبانی را (و همچنین پورت) که درخواست برای او فرستاده می شود، مشخص کند. این کار با قراردادن Header، Host: صورت می گیرد.

یک درخواست کامل HTTP ممکن است به صورت زیر باشد:

Example Code

```
GET /path/file.html HTTP/1.1
Host: www.host1.com:80
[blank line here]
```

شماره پورت ":۸۰" در صورتی که سرور بر روی پورت پیشفرض خود باشد، ضروری نیست!

Host: تنها **Header** ضروری در درخواست های **HTTP ۱,۱** است. همچنین مهم ترین نیازمندی برای **HTTP ۱,۱** محسوب می شود. زیرا بدون این خاصیت هر میزبان باید یک **Unique IP** داشته باشد و به زودی با کمبود **IP Addresses** برای دامنه های جدید مواجه می شدیم...

Chunked Transfer-Encoding:

اگر یک سرور بخواهد پاسخ را قبل از اینکه طول کل آن را بداند به مشتری ارسال کند باید از **CTE** استفاده کند که پاسخ کامل را در بسته های کوچکتر به مشتری به صورت سری ارسال می کند. به این صورت توسط مشتری شناسایی می شوند که در **Transfer-Encoding Header** عبارت **The Chunked** درج شده است. تمامی مشتری های **HTTP ۱,۱** باید توانایی دریافت این نوع پاسخ را داشته باشند.

بدنه پیغام **Chunked** شده به این صورت است که اول اطلاعات **Chunk** و سپس یک خط که فقط دارای "۰" و یک سری خطوط اختیاری **footers** که شبیه **headers** هستند و سپس یک خط خالی.

هر **Chunk** دارای دو قسمت است:

* یک خط که حاوی کل اندازه داده های (**Chunk** در هگزا) و یک سری پارامترهای دیگر (که البته هنوز استاندارد نشدند. شما می توانید از آن صرف نظر کنید) است و با **CRFL** پایان می گیرد.

* و خود داده که با **CRFL** به پایان می رسد. بنابراین یک پاسخ **Chunked** مثل پایین است:

Example Code

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Transfer-Encoding: chunked

1a; ignore-stuff-here
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
some-footer: some-value
another-footer: another-value
[blank line here]
```

به خط خالی پس از footer توجه کنید. طول این داده ۴۲ بایت است (a+۱۰۱ در هگز) و خود داده:

abcdefghijklmnopqrstuvwxyz1234567890abcdef

است.

خطوط footers همانند headers ها هستند تنها با این فرق که در انتهای پیغام درج می شوند. Chunk ها می توانند حاوی هر نوع داده باینازی و خیلی بزرگتر از آنچه در اینجا آورده شد، باشند! پارامتر Size به ندرت استفاده می شود ولی بهتراست در صرف نظر کردن از آن دقت کنید. Footer ها هم به ندرت استفاده می شوند ولی برای اعمالی مثل امضای دیجیتالی و مسائل امنیتی مناسب هستند.

برای مقایسه عکس العمل ها این همان پاسخ ولی به صورت Chunk نشده است:

Example Code

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Content-Length: 42
some-footer: some-value
another-footer: another-value

abcdefghijklmnopqrstuvwxyz1234567890abcdef
```

ارتباط مداوم و "Header "Connection: Close":

قبل از پیدایش HTTP ۱,۱، ارتباط بعد از هر درخواست و پاسخ قطع می شد و هر resource نیاز به برقراری ارتباط مجدد و مجزای خودش را داشت.

برقرار کردن و خارج شدن از ارتباط به طور مکرر باعث به هدر رفتن CPU Time، پهنای باند و حافظه می شود. به طور معمول فایل ها و منابع زیادی بر روی یک سرور وجود دارد و به وسیله یک ارتباط مداوم که نیازی به قطع و وصل ندارد، می توان فایل های زیادی را از سرور دریافت کرد.

ارتباطات مداوم در HTTP ۱,۱ به صورت پیش فرض تعریف شده اند بنابراین نیاز به چیز اضافه ای برای استفاده از این قابلیت ندارد. فقط یک ارتباط برقرار کنید و درخواست های خود را بفرستید (Pipelining) و پاسخ را همانگونه که درخواست کردید دریافت کنید. به خاطر داشته باشید در این حالت برای تشخیص پاسخ ها و جداسازی آنها به طول آن توجه کنید.

اگر مشتری از Header، "Connection: Close" استفاده کند، ارتباط بعد از فرستادن پاسخ قطع می شود. از این header در هنگامی که مشتری شما از ارتباط مداوم پشتیبانی نمی کند یا هنگامی که می دانید این آخرین درخواست شما از سرور است، استفاده کنید.

پاسخ "۱۰۰" Continue:

هنگامی که یک مشتری HTTP ۱,۱ درخواست خود را به سمت سرور می فرستد، سرور ممکن است پاسخ "۱۰۰" Continue را ارسال کند. این به آن معناست که سرور قسمت اول درخواست را دریافت کرده و منتظر بقیه درخواست است. تمامی مشتری های HTTP ۱,۱ باید با این پاسخ آشنا باشند و آن را handle کنند. پاسخ "۱۰۰" Continue همانند دیگر پاسخ های HTTP است (به

عبارت دیگر دارای خط وضعیت، Header های اختیاری و یک خط خالی است) و برخلاف پاسخ های دیگر در ادامه یک پاسخ کامل کننده فرستاده می شود به عنوان مثال دو پاسخ فرستاده شده توسط سرور:

Example Code

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/plain
Content-Length: 42
some-footer: some-value
another-footer: another-value

abcdefghijklmnoprstuvwxyz1234567890abcdef
```

به عنوان مثال اگر مشتری HTTP ۱,۱ پاسخ ۱۰۰ Continue دریافت کرد تنها باید از آن صرف نظر کند و پیغام بعدی را مورد بررسی قرار دهد.

HTTP ۱,۱ Servers:

برنامه های سرور برای سازگای با HTTP ۱,۱:

* نیاز به Host Header از مشتری دارند.
* URL های مستقل را هم قبول می کنند.
* درخواست های Chunk شده را هم قبول می کنند.
* از ارتباط مداوم پشتیبانی یا با هر در پاسخ "Connection: Close" را ارسال می کنند.
* در هر پاسخ Date Header را هم قرار می دهند.
* طریق پاسخ به درخواست های دارای :If-Modified-Since یا :If-Unmodified-Since را می داند.
* حداقل از دو روش GET و HEAD پشتیبانی می کنند.
* تمامی درخواست های HTTP ۱,۰ را هم پشتیبانی می کنند.

نیاز به Host Header:

به دلیل ضرورت و اهمیت استفاده از این Header سرورهای HTTP ۱,۱ مجاز به پاسخ گویی به درخواست های فاقد این Header نیستند و اگر سروری درخواستی بدون این Header دریافت کند پاسخ "۴۰۰ Bad Request" را ارسال می کند. مانند زیر:

Example Code

```
HTTP/1.1 400 Bad Request
Content-Type: text/html
Content-Length: 111

<html><body>
<h2>No Host: header received</h2>
```

```
HTTP 1.1 requests must include the Host: header.
</body></html>
```

البته این تنها در مورد مشتری های HTTP ۱,۱ صدق می کند و اگر مشتری از نسخه قبلی استفاده کند، سرور از URL به جای Host: استفاده می کند (قسمت بعد را ببینید – بنابراین اگر مشتری از HTTP ۱,۰ استفاده کند، درخواست بدون این Header توسط سرور پاسخ داده می شود).

قبول کردن URL مستقل:

استفاده از Host Header یک راه حل موقت برای مشکل میزبان ها است. در نسخه بعدی HTTP از URL کامل به جای pathname استفاده می شود. مثال:

Example Code

```
GET http://www.somehost.com/path/file.html HTTP/1.2
```

برای فعالسازی این روش، سرورهای HTTP ۱,۱ باید این نوع درخواست ها را قبول کنند و به مشتری های HTTP ۱,۱ که نه به این روش و نه استفاده از Host، فرستاده می شوند، پاسخ خطا بفرستد و Chunked Transfer-Encoding همانند مشتری های HTTP ۱,۱ سرورها هم باید از این نوع داده پشتیبانی کنند. برای اطلاعات بیشتر به قسمت مشتری مراجعه کنید.

سرورها نمی توانند پیغام های Chunk شده را تولید کنند بلکه تنها باید آن را دریافت کنند!

ارتباط مداوم و استفاده از Header "Connection: Close":

اگر یک مشتری HTTP ۱,۱ چندین درخواست را به وسیله ارتباط به سرور ارسال کند، سرور باید پاسخ هر درخواست را مطابق با آنچه خواسته شده و به صورت مجزا به مشتری ارسال کند و همه اینها حکایت از این دارد که سرور باید ارتباط مداوم را پشتیبانی کند. درخواستی که دارای "Connection: Close" باشد آخرین درخواست در ارتباط خودش است و سرور بعد از ارسال پاسخ ارتباط را قطع می کند همچنین سرور یک زمان مشخصی در انتظار درخواست می ماند و اگر درخواستی ارسال نشد ارتباط را قطع می کند (می تواند هر فاصله زمانی باشد، ولی به طور معمولا ۱۰ ثانیه مناسب است).

اگر شما نمی خواهید که ارتباط مداوم را پشتیبانی کنید در پاسخ خود از Header "Connection: Close" استفاده کنید. از این Header برای قطع ارتباط استفاده می شود و یک مشتری HTTP ۱,۱ به خوبی آن را درک می کند.

استفاده از عکس العمل "۱۰۰" Continue:

همان طور که در قسمت مشتری توضیح داده شد، این پاسخ برای ارتباطات با سرعت پائین سودمند است. وقتی یک سرور HTTP ۱,۱ اولین قسمت از درخواست مشتری (HTTP ۱,۱ یا بالاتر) دریافت کرد یا باید پیغام خطا دهد یا کد "۱۰۰ Continue" را ارسال کند. اگر از این کد استفاده کرد، علاوه بر این باید یک پاسخ نهایی هم (که شامل آنچه درخواست شده است) ارسال کند.
کد "۱۰۰ Continue" هیچ گونه header لازم ندارد ولی یک خط خالی بعد از پیغام ضروری است. مثل:

Example Code

```
HTTP/1.1 100 Continue
[blank line here]
[another HTTP response will go here]
```

به یاد داشته باشید که این کد را به مشتری HTTP ۱,۰ ارسال نکنید.

The Date Header:

Cache کردن یکی از مهمترین امکانات در HTTP ۱,۱ است که بدون قید شدن تاریخ در پاسخ ها امکان پذیر نیست. بنابراین سرور باید با هر عکس العملی تاریخ ارسال پاسخ را هم قید کند. این کار به وسیله Date Header صورت میگیرد که به صورت زیر است:

Example Code

```
Date: Fri, 31 Dec 1999 23:59:59 GMT
```

تمامی عکس العمل ها به استثناء پیغام های ۱۰۰ باید دارای این Header باشند. تمامی زمان ها بر اساس گرینویچ می باشد.

پاسخ گویی به درخواست های دارای If-Modified-Since و If-Unmodified-Since:

برای پرهیز ارسال فایل هایی که نیازی به آنها نیست و همچنین حفظ پهنای باند دو header به نام If-Modified-Since و If-Unmodified-Since در HTTP ۱,۱ تعریف شده اند.

در واقع این دو Header می گویند: "تنها در صورتی این فایل را بفرست که از این تاریخ تغییر کرده باشد". و دیگری معنی مخالف این را می دهد.

مشتری ها الزامی در استفاده از این ندارند ولی سرور HTTP مستلزم متاسفانه در نسخه های جدیدتر تاریخ به یکی از سه فورمت زیر است:

Example Code

```
If-Modified-Since:  Fri, 31 Dec 1999 23:59:59 GMT
If-Modified-Since:  Friday, 31-Dec-99 23:59:59 GMT
If-Modified-Since:  Fri Dec 31 23:59:59 1999
```

همانطور که می دانید تمامی زمان ها در گرینویچ است ولی با زمان های غیر از گرینویچ هم مدارا کنید و به درخواست آنها پاسخ گویید. سرورها باید هر سه نوع این درخواست ها را بپذیرند ولی مشتری ها تنها از نوع اول باید استفاده کنند. اگر زمان ذکر شده اشتباه بود و یا در آینده بود تنها از این Header صرف نظر کنید. اگر بدون این Header درخواست، نتیجه ای ناموفق در پی داشت

(که غیر از ۲۰۰ – از آن صرف نظر کنید و کد غیر ۲۰۰ ارسال کنید و تنها در صورتی header را دخالت دهید که می دانید فایل به هر حال فرستاده می شود).

به همراه یک درخواست GET فرستاده می شود. اگر فایل درخواست شده از تاریخ داده شده تغییر کرده بود، header را نادیده بگیرید و فایل را بفرستید وگرنه یک پیغام "۳۰۴" Not Modified همراه تاریخ و بدون بدنه ارسال کنید. مانند زیر:

Example Code

```
HTTP/1.1 304 Not Modified
Date: Fri, 31 Dec 1999 23:59:59 GMT
[blank line here]
```

If-Unmodified-Since مشابه است ولی می تواند با هر روشی ارسال شود و اگر فایل درخواستی از تاریخ داده شده تغییر نکرده باشد فایل ارسال می شود در غیر این صورت پیغام "۴۱۳" Precondition Failed ارسال می شود.

Example Code

```
HTTP/1.1 412 Precondition Failed
[blank line here]
```

پشتیبانی از دو روش GET و HEAD:

برای سازگاری با HTTP ۱,۱، سرورها باید حداقل دو روش GET و HEAD را پشتیبانی کنند و اگر شما می خواهید که از اسکریپت های CGI هم استفاده کنید باید روش POST هم پشتیبانی کنید. چها رروش دیگر (PUT,DELETE,OPTIONS,TRACE) در HTTP ۱,۱ تعریف شده اند ولی به ندرت مورد استفاده قرار می گیرند. اگر یک مشتری از روشی که در سرور از آن پشتیبانی نشده، استفاده کنید پیغام "۵۰۱" Not Implement دریافت می کند. مثل:

Example Code

```
HTTP/1.1 501 Not Implemented
[blank line here]
```

پشتیبانی کردن از درخواست های HTTP ۱,۰:

خوب همان طور که می دانید هنگامی که سرور با یک درخواست HTTP ۱,۰ مواجه می شود. مراحل زیر مدنظر هستند:

* نباید به دنبال Host Header (Host:) بگردد.

* از کد "۱۰۰" Continue استفاده نکند!

ضوابط و مشخصات HTTP:

RFC هایی در این زمینه هستند که زیاد مهم نیستند ولی اگر می خواهید دقیق تر و رساتر بفهمید PM بزنید تا لیست RFC های مربوط به آن هم برای شما بازگو کنم.

خوب این آموزش اینجا به پایان می رسد. حال برای اینکه چند نمونه مثال از مطالب گفته شده ببینید و به اهمیت این مطلب پی ببرید من اینجا دو تمرین برای کسانیکه که این مطلب رو خوانده اند مطرح می کنم:

اول – حتما با برنامه Evil HTTP Server آشنا هستید، با توجه به مطالبی که آموختید ساختار و طریقه کارکرد این برنامه را دقیق پیش خودتون تجزیه و تحلیل کنید.

دوم – چند نمونه EXPLOIT مرتبط با مطالب گفته شده برای HTTP سرورها و مشتری ها را پیدا کرده و سپس روی آن کار کنید).

پایان مقاله

Written by: **James Marshall**
Translate: **wolf_of_night**
Mail: **sarve_paidar@yahoo.com**

Report any typing mistake in the text

Crouz Security Team

Czar Admins
Rara Avis Member
Oxlip Site
Ubiquitous and
Zealot men