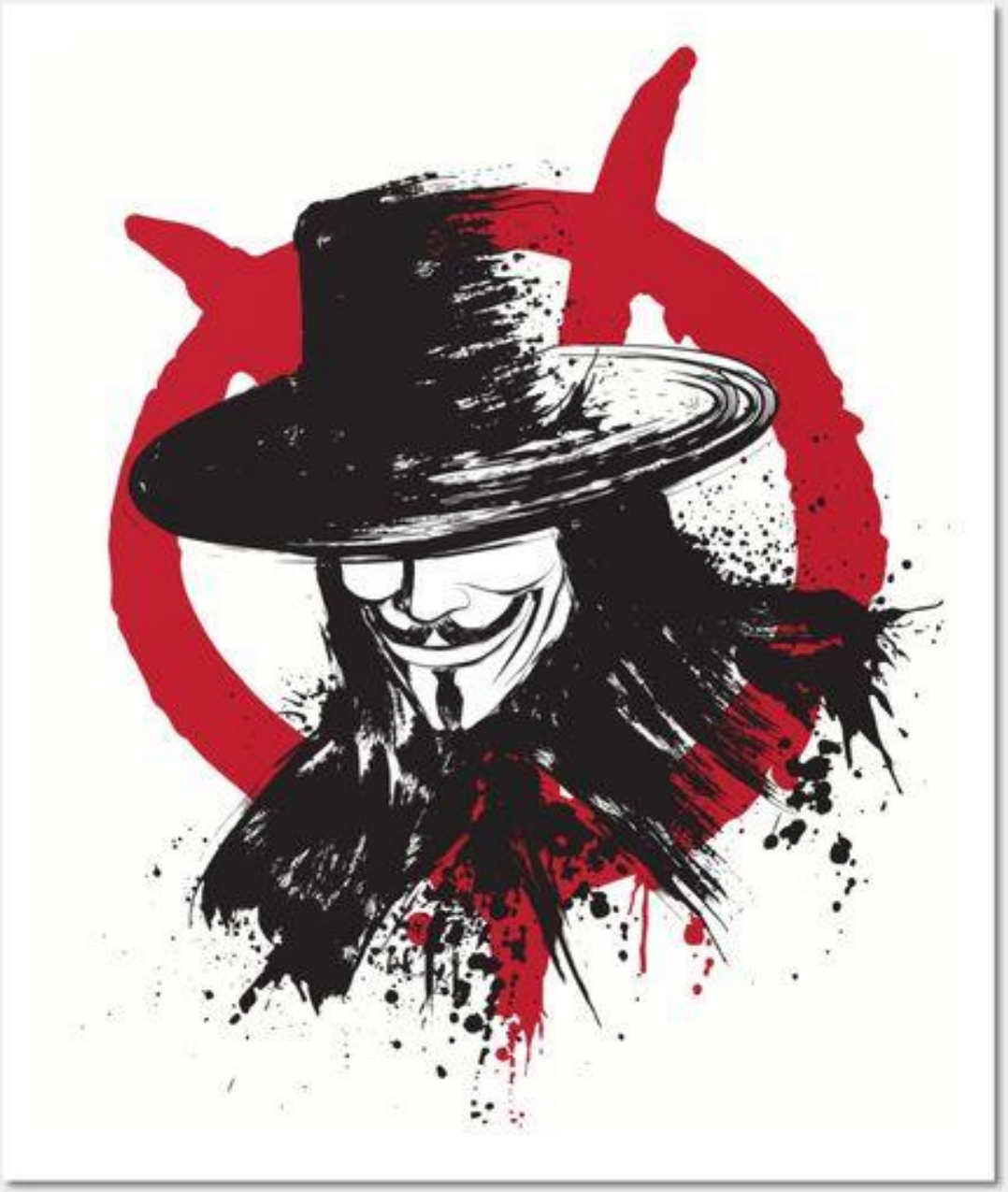
Web Application Penetration Testing



#2 Session Notes

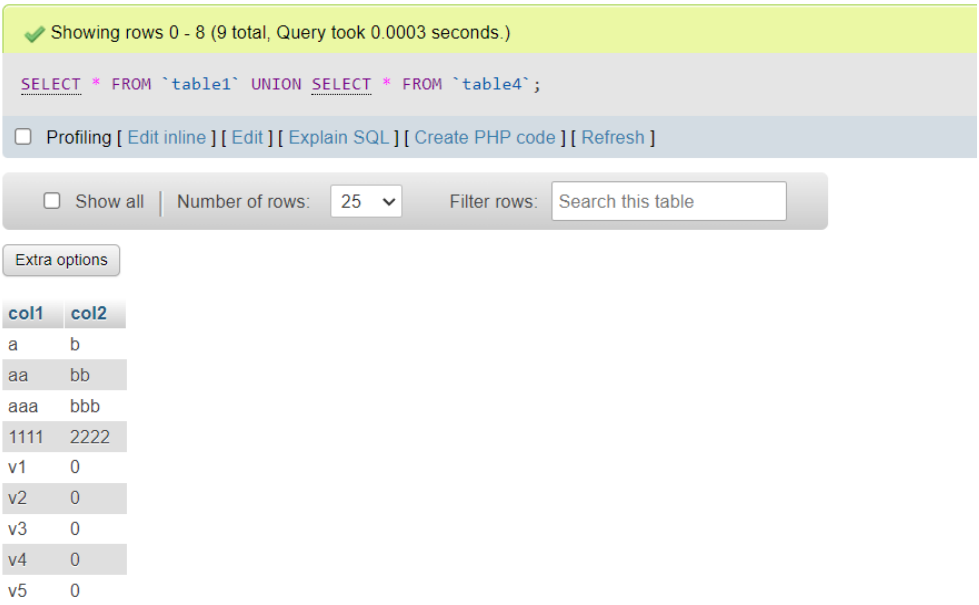
By TheSecDude

اجتماع دو Table در SQL چگونه است ؟ اجتماع دو Table به معنی نشون دادن دادههای آن دو در یک Table است . اگر دستور SELECT رو یادتون باشه از طریق میتونستیم مقادیر همه Column های داخل این Table رو بیرون بیاریم و همچنین میتونستیم مقادیر برخی از Column ها رو با نام بردن این Column ها ببینیم . دستور اجتماع ما بین دو دستور SELECT استفاده می‌شود و کلمه کلیدی آن UNION به معنی اجتماع و اتحاد است . اگر در ریاضیات بیاد دارید علامت U نشان دهنده اجتماع مابین دو مجموعه بود و SELECT هم در SQL به ما مجموعه میده . سینتکس کلی استفاده از UNION به شکل زیر است :

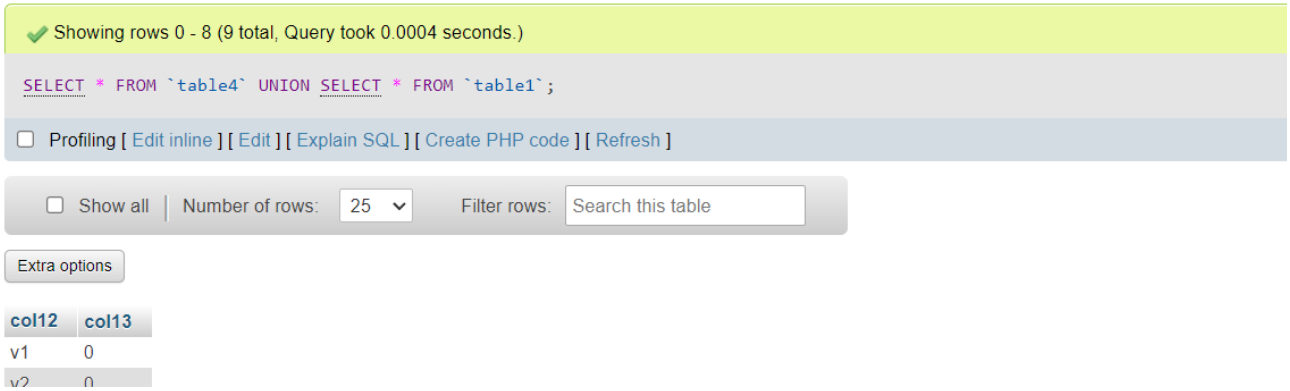
```
SELECT * FROM `TABLE1_NAME` UNION SELECT * FROM `TABLE2_NAME`;
```

دقت کنید که خروجی SELECT اول و خروجی SELECT دوم بایستی دارای تعداد Column های یکسانی باشد وگرنه خطا دریافت خواهید کرد . یعنی اگر میخواهید از * SELECT های دو جدول رو UNION کنید، این دو جدول باید تعداد Column های یکسانی داشته باشد . در مثال زیر ما دو جدول با نامهای table۱ و table۲ داریم که هردو دارای دو ستون با نامهای col۱ و col۲ هستند . اگر بخواهیم آن‌ها را اجتماع کنیم میتوانیم به شکل زیر دستور را وارد نماییم :

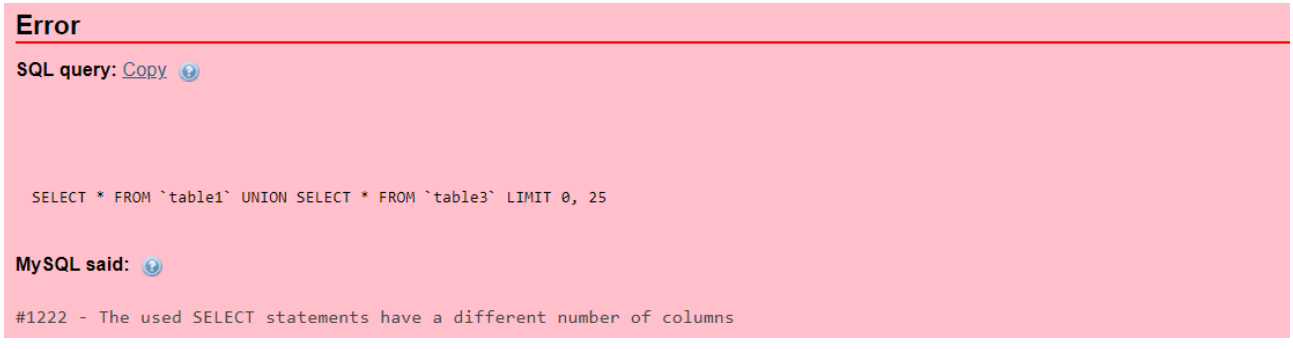
در مثال زیر ما مجموع table۱ و table۴ رو بدست آوردیم . table۱ دو ستون با نامهای col۱ و col۲ دارد و table۴ دو ستون با نامهای col۱۲ و col۱۳ دارد . وقتی اجتماع این دو Table رو میگیریم کاری به نام ستونهای table۴ ندارد و اطلاعات table۴ رو هم در زیر ستونهای table۱ نشون میده . تنها چیزی که مهمه آینه که table۴ به اندازه table۱ ستون داره یعنی دوتا :



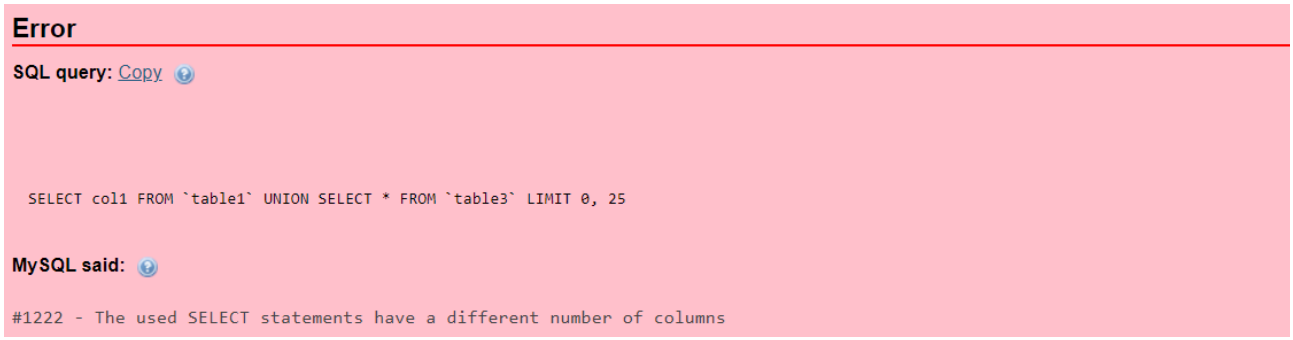
اگر ما SELECT اول رو table۴ بنویسیم و SELECT دوم رو table۱ نتیجه در زیر ستونهایی همنام ستونهای table۴ نشون داده خواهد شد :



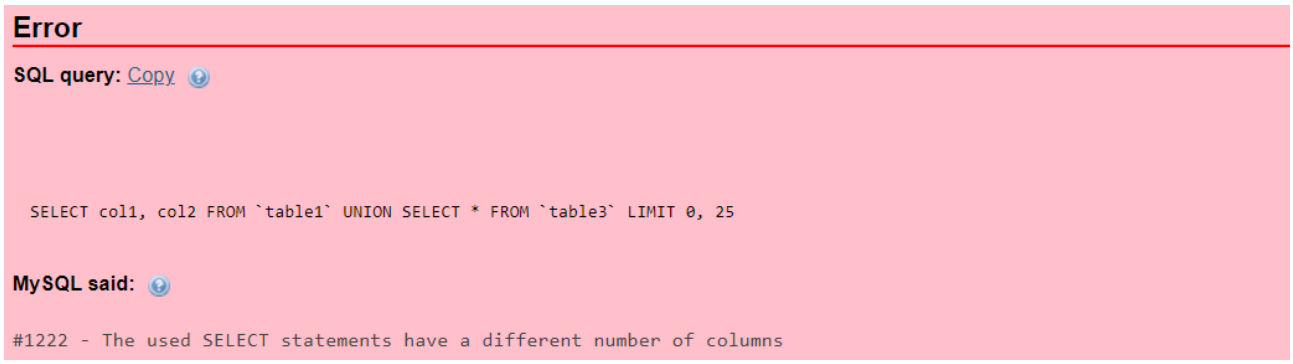
پس تعداد ستونها اهمیت زیادی دارن و می بایست SELECT اول و SELECT دوم نتیجه‌ها رو در زیر تعداد ستون یکسانی برگردونن وگرنه خطا خواهیم گرفت . فرض کنید یک جدول به نام table۱ داریم که دو ستون دارد و یک جدول به نام table۳ داریم که سه ستون دارد . اگر اونها رو با * SELECT اجتماع کنیم خطای زیر را خواهیم گرفت :



ما نمیدونیم که تعداد ستونهای table۳ چند ستون است ولی از ستونهای table۱ اطلاع داریم و نامهای اونها رو میدونیم . پس میایم در SELECT اول به جای * از نام های ستونهای table۱ استفاده میکنیم و در اولین تست نام ستون اول رو مینویسیم :



میبینیم که باز خطا میده، پس نتیجه میگیریم که table۳ بیشتر از یکی ستون دارد و خب SELECT اول رو به دو ستون تغییر میدیم :



و خب قاعدتاً باز هم خطا میگیریم و باید تعداد ستونها رو بیشتر از دوتا وارد کنیم ولی table۱ فقط دو ستون دارد ! میتوانیم به جای نام ستون '' رو استفاده کنیم و جواب میده :

یعنی درسته که در table۱ سه ستون نداریم ولی خب میتونیم از نتیجه بخوایم که سه ستون به ما نشون بده و اگر مقادیر در ستون سوم نبود آن را خالی بزاره، میبینید که مقادیر ستون سوم در table۱ خالیه . این روش رو میتونیم استفاده کنیم تا نامهای کاربری و رمزهای عبور ذخیره شده در جدول کاربران یک سایت رو زمانی که SQL Injection داره بدست بیاریم . البته که در ۹۹ درصد اوقات کاراکتر ' رو نمیشه در آدرس بار مرورگر استفاده کرد چرا که سایت تارگت اون رو فیلتر میکنه و ما باید اون رو Encode کنیم .

ORDER BY چه میکند ؟ دیدیم که دستور SELECT دادههای داخل جداول رو میخونه و برای ما نمایش میده . ممکن است یه زمانی بخواید که دادههایی که از طریق دستور SELECT برگردانده شدهاند رو بر اساس یکی از ستونهای آن مرتب کنید . سینتکس کلی این دستور به شکل زیر است :

```
SELECT * FROM `TABLE_NAME` ORDER BY `COLUMN`;
```

هرجا که SELECT استفاده میشود میتوان از ORDER BY هم استفاده کرد . مثلاً توی UNION هم میتونید استفاده کنید . در مثال زیر ما یک جدول داریم به نام table۱ که دو ستون دارد با نامهای city و number . ما میخوایم نتیجه SELECT ما بر اساس مرتب کردن ستون city به صورت حروف الفبا باشد . برای اینکار کافیست که به شکل زیر Query رو بزنیم :

میبینید که بر اساس حروف الفبا از A تا Z آن‌ها را مرتب کرد . حال اگر بخواهیم بر اساس ستون number این کار رو انجام بدیم کافیست که به شکل زیر عمل کنیم :

```
SELECT * FROM `table1` ORDER BY `number`;
```

اما میتوانیم حالت مرتب سازی رو نیز تعیین کنیم . مثلاً به حالت Ascending یعنی صعودی باشه یا Descending یعنی نزولی باشه . برای این کار کافیست که از کلمات کلیدی DESC برای Descending و ASC برای Ascending استفاده کنیم .

```
SELECT * FROM `table1` ORDER BY `number` ASC;
SELECT * FROM `table1` ORDER BY `number` DESC;
```

یکی از نکاتی که میتواند در آینده در اکسپلویت کردن SQL Injection به ما کمک کنه این هست که میتونیم به ORDER BY به جای نام Column یک شماره بدهیم که اشاره کنه به اون ستون مورد نظر ما . مثلاً اگر SELECT دو ستون رو بر میگردونه میتونیم بگیم که ORDER BY بر اساس ستون شماره ۱ یا شماره ۲ انجام بشه :

Showing rows 0 - 6 (7 total, Query took 0.0003 seconds.)

```
SELECT * FROM `table1` ORDER BY 1;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

city	number
Ablexchester	4
Glopving	5
Qupgas	1
Yraemont	2
Zlugate	7
Zore	3
Zouis	6

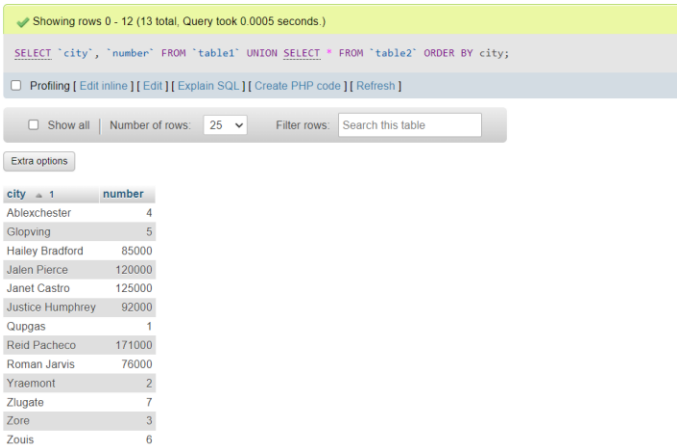
یا هم بگیم بر اساس ستون شماره دو مرتب کن :

```
1 SELECT * FROM `table1` ORDER BY 2;
```

دقت کنید که گفتیم ORDER BY هر جا SELECT باشه میشه استفاده بشه و خب توی UNION دیدی که SELECT رو داریم، پس میتونیم اونجا هم ازش استفاده کنیم :

```
1 SELECT `city`, `number` FROM `table1` UNION SELECT * FROM `table2` ORDER BY city;
```

دقت کنید که این ORDER BY بر روی نتیجه UNION اعمال میشه . بر اساس این دستور نتیجه UNION شامل دو ستون city و number است پس ORDER BY هم باید بر روی یکی از این دو ستون اعمال بشه .



خواندن یک فایل با دستورات SQL چگونه انجام می‌شود ؟ اگر Permission های یک کاربر SQL درست تعریف نشده باشد و اجازه ها به دسترسی به فایل‌های حساس برای او فیلتر نشده باشد، این کاربر میتواند فایلهایی را در سیستم بخواند . برای خواندن یک فایل از دستور LOAD_FILE استفاده می‌شود و سینتکس کلی آن به شکل زیر است :

```
SELECT LOAD_FILE("PATH_TO_FILE");
```

مثلاً کاربر میتواند فایل passwd را از طریق این دستور بخواند :

نوشتن در یک فایل توسط دستورات SQL ؟ علاوه بر اینکه میتوانیم فایل‌هایی را از روی سرور بخوانیم میتوانیم چیزهایی را نیز برروی سرور بنویسیم . این کار هم اگر Permission ها درست حسابی نتظیم شده باشند محدود خواهد شد به دایرکتوری به آدرس /var/lib/mysql که خطری نداره ولی اگر پیکربندی های درست نباشد میتواند موجب شود که یک مهاجم از طریق حفره امنیتی SQL Injection که به او اجازه تزریق دستورات SQL را میدهد بر روی سرور Shell اپلود کند و بتواند دسترسی کامل رو از سرور بگیرد . برای اینکه بتوانیم یک متن رو داخل یک فایل بنویسیم به صورت کلی از سینتکس زیر پیروی میکنیم :

```
SELECT "PAYLOAD" INTO OUTFILE "PATH_TO_FILE";
```

مثلاً در مثال زیر ما کلمه PAYLOAD رو داخل یک فایل در دایرکتوری /var/www/html نوشتیم که درواقع دایرکتوری اصلی سایت روی سرور خواهد بود . میتوان بجای کلمه PAYLOAD یک کد PHP جهت دسترسی کامل به سرور در آن فایل نوشت :

```
SELECT "PAYLOAD" INTO OUTFILE "/var/www/html/shell.php";
```

حال اگر فایل shell.php رو cat کنم میبینید که محتویات داخل آن کلمه PAYLOAD است :

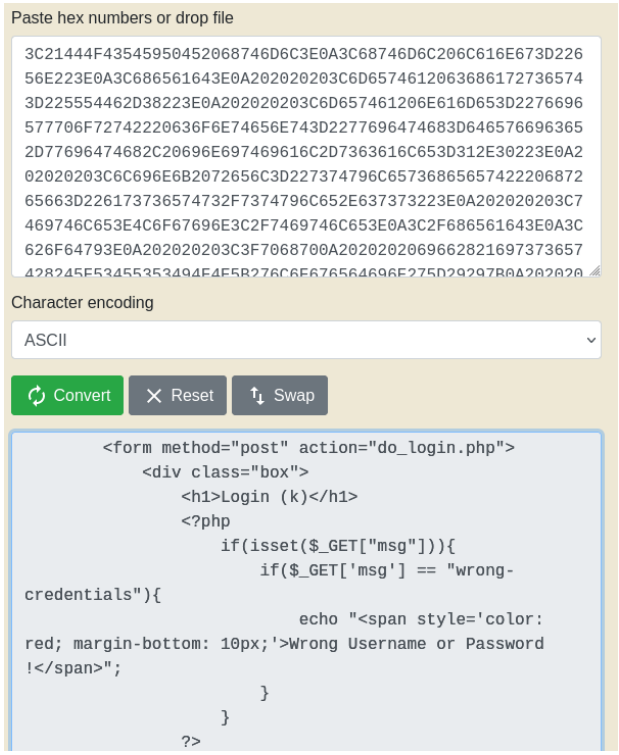
```
username@kali:~$ cat /var/www/html/shell.php
PAYLOAD
```

تابع HEX در SQL چه کاربردی برای ما دارد ؟ این دستور یک مقدار رو میگیره و اون رو به کدهای HEX تبدیل میکنه و سینتکس کلی اون به شکل زیر است :

```
SELECT HEX("VALUE");
```

کجا کاربرد داره ؟ شاید یه وقتی یه روزی یه جایی شما یک SQL Injection پیدا کردید . Permission ها درست پیکربندی نشده بود و شما تونستید که فایلهای حساس رو بخونید . فرض کنید که تونستید یه فایل به نام login.php رو پیدا کنید که اطلاعاتی حساس توش هست، مثلاً User و Pass دیتابیس و ... . اگر بخواید از طریق SQL این فایل رو داخل یک فایل php دیگه LOAD_FILE کنید هر دو فایل داخل هم توسط مفسر PHP اجرا میشن و خب در حین Dump کردن login.php با خطا مواجه میشید . میتونید به جای اینکه اون فایل رو به صورت Plain Text بخونید، محتویات اون رو به صورت کدهای HEX بیرون بکشید و سپس تبدیل کنید به Text و اینطوری با خطا هم مواجه نمیشید . مثلاً :

و اگر نتیجه رو تبدیل کنیم به Text به شکل زیر میشه :



نکته‌ای که لازم میدونم درمورد HEX بگم این هست که واسه BYPASS فایروال ها میتونه استفاده بشه .
یه نکته مهم درمورد BYPASS کردن برخی فیلتر ها بگم که ممکنه برخی برنامه نویسها زرنگ بازی در بیارن و بیان و دستورات SQL رو مثلاً فیلتر کنن . مثلاً ما اجازه نداشته باشیم دستور SELECT رو توی آدرس وارد کنیم و توسط یه برنامه اون رو حذف میکنه . گفتیم که SQL نسبت به حروف بزرگ و کوچیک حساس نیست و ما میتونیم به جای SELECT کلماتی مثل SeLeCT، sELECt, SelecT , ... استفاده کنیم . این موضوع گاهی اوقات میتونه توی BYPASS کردن این مشکل کمک کنه .

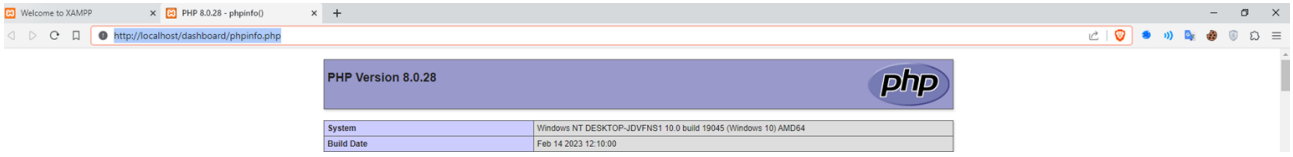
خب بخش MySQL تا همینجا فعلاً کافیه، اگر چیز جدید قرار باشه اضافه بشه (که قرار هست) در آینده خواهیم دید . این توضیحات تا اینجا واسه آشنایی با دستوراتش و کاربردش به نظر مفید بوده . خب بریم سروقت قسمت بعدی یعنی PHP ...

PHP In Web Penetration Testing

خدمتتون که عارض باشم، ما نمیخوایم طراح وب باشیم، ما نمیخوایم BackEnd کار بشیم، همانطور که ما نمیخوایم Font End کار بشیم ولی خب میخوایم طراحی ها و برنامه نویسی های طراحان رو مورد تست نفوذ قرار بدیم . از این رو باید بدونیم که توی Back End یک وبسایت چه میشه ؟ چه چیزی به چه چیزی تبدیل میشه و به عبارتی دیگه و مهم‌ترین نکته این هست که بتونیم Data Flow رو درک کنیم . واسه این کار نیاز به دانش برنامه نویسی داریم، نه در حد عالی (اگه عالی هم باشه چه بهتر) ولی در حدی که اگه خواستیم یه باگ رو اکسپلویت کنیم گیر نکنیم . علت اینکه اینجا میخوایم PHP رو یاد بگیریم این هست که بیشتر سایتهایی که طراحی شده‌اند بر پایه این زبان برنامه نویسی هستند، ممکن است فریمورک اونها متفاوت باشه ولی زبان برنامه نویسی عمدتا PHP است، اگرچه امروزه Python, JavaScript, Ruby و ... هم جایگاه خودشون رو دارن . یه طورایی تصمیم بر این گرفتیم که بریم سمت اون چیزی که رایج تره . یکی دیگه از مزیت‌های یادگیری زبان برنامه نویسی این هست که میتونیم خودمون برای نمونه باگهایی رو تولید کنیم و سعی کنیم اونها رو اکسپلویت کنیم، این کار میتونه توی روند فکری ما برای پیدا کردن باگ در تست نفوذ های بلک باکس بسیار مفید واقع بشه .

PHP چیست ؟ PHP گرفته شده از Hypertext Preprocessor هست و سن نسبتاً بالای هم داره . یه زبانی هست که به صورت گسترده در طراحی وبسایت (قسمت BackEnd) استفاده می‌شود . این زبان میتونه توی کدهای HTML نوشته شود و توسط یک مفسر تفسیر گردد . البته خدمتتون بگم که استانداردهای برنامه نویسی زیاد توی این زبان اجرایی نشده م مثلاً ممکنه شما یه تابع رو ببینی که کلاً با حروف کوچیک نوشته میشه و یه تابع دیگه به صورت CamelCase هست و شاید هم تابع بعدی با _ کلماتش از هم جدا شده . کسی که اینو طراحی کرد قصدش طراحی یه زبان برنامه نویسی با استانداردهای بین‌المللی نبود و صرفاً میخواست یه زبان رو طراحی کنه که هر کسی که خواست باهاش بتونه یه وبلاگ رو طراحی کنه . CMS ها و فریمورک های زیادی با این زبان نوشته شده که معروف ترین CMS موجود WordPress هست و خدایی تعداد باگهایی که داره اصن افتضاحه . یکی دیگه از فریمورک های معروف Laravel است که خدایی خوب ساخته شده و حسابی داره روش کار میشه . خلاصه زبان خوبیه و یادگیریش ضرری نداره .

نصب کردن PHP در ویندوز و لینوکس ؟ برای نصب PHP توی ویندوز کافیه که XAMPP یا WAMP رو نصب کنید و همراه این پکیجا PHP هم نصب میشه و اگر بخواید توی لینوکس PHP رو نصب کنید میتونید از لینک https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-۲۲-۰۴ استفاده کنید که هم PHP و هم Apache و MySQL رو نصب میکنه . نصبش کاری نداره و ساده هست .
وقتی نصب کردید توی ویندوز ما تو جایی که XAMPP رو نصب کردیم آدرس http://localhost/dashboard/phpinfo.php رو توی مرورگر وارد کنید و اگر صفحه زیر رو دیدی یعنی کد های PHP شما داره اجرا میشه :



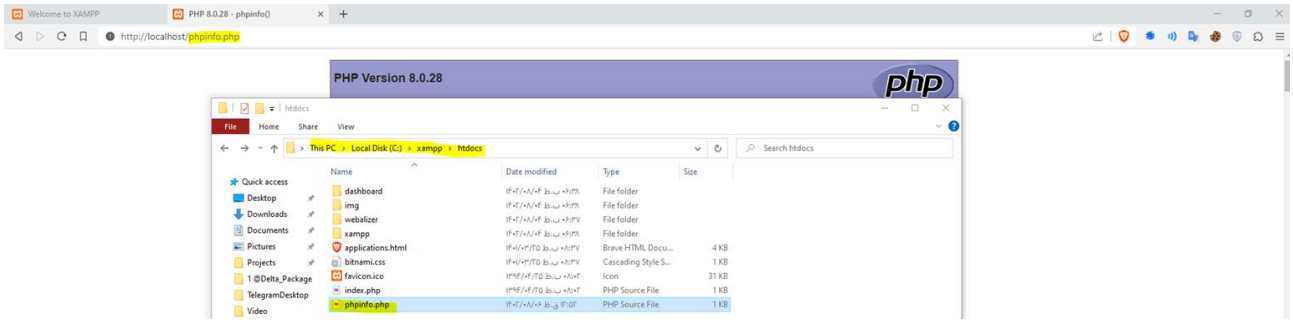
واسه تست توی لینوکس هم کافیه برید توی دایرکتوری /var/www/html و مثلاً example.php رو بسازید و توش کد زیر رو بنویسید :

```
<?php phpinfo(); ?>
```

بعد ذخیره کنید و توی مرورگر آدرس http://localhost/example.php رو وارد کنید و اگر صفحه مشابه تصویر قبلی دیدی یعنی PHP شما داره کار میکنه .
اگر کار نکرد میتونه چندتا مشکل وجود داشته باشه :

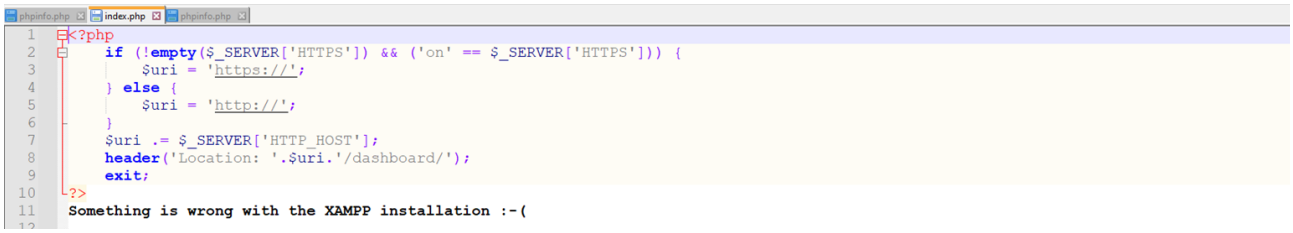
۱. Apache و PHP درست نصب نشدند .
۲. Apache اجرا نیست .
۳. دایرکتوری رو اشتباه انتخاب کردید .
۴. ...

توی ویندوز کد های PHP رو در دایرکتوری htdocs در جایی که XAMPP نصب شده قرار میدیم و اجرا میکنیم . مثلاً در مثال زیر من در این دایرکتوری یک فایل به نام phpinfo.php ساختم و توش کد تصویر قبل رو نوشتم و توی مرورگرم اجرا کردم :



دقت کنیم که وقتی توی مرورگر مینویسیم localhost وب سرور ما یعنی Apache اون رو به دایرکتوری htdocs در جایی که XAMPP نصب شده ارجاع میده . بر اساس پیکربندی های Apache در این دایرکتوری دنبال فایل‌های index.php, index.html, index.html و چنین

چیزهایی میگرده تا به صورت پیش‌فرض اجرا کنه . وقتی شما localhost رو می‌زنید اون فایل index.php داخل این دایرکتوری اجرا میشه که یک کد php داره و کاربر رو به دایرکتوری dashboard توی همینجا منتقل میکنه، محتویات index.php به شکل زیر است :



برای اینکه کدهای PHP رو بنویسیم میتونیم از برنامه‌های مختلفی که وجود دارند استفاده کنیم . از notepad پیش‌فرض ویندوز گرفته تا notepad++ و VSCode و PHPStorm به شما کمک میکنن تا کدهاتون رو بنوسید . من پیشنهادم Notepad++ هست، هم سبکه و هم سادست .

کدهای PHP میتونن با کدهای HTML ادغام بشن و مابین تگ های HTML میتونیم کدهای PHP رو بنویسیم . یکی از رایج ترین کارهایی هست که انجام میشه و البته فایل رو نه با پسوند html. بلکه با پسوند php. باید ذخیره کنیم .

```
<!DOCTYPE html>
<html>
    <head>
        <title>My First PHP Code!</title>
    </head>
    <body>

    //You Can Write You PHP Code here  ...

    </body>
</html>
```

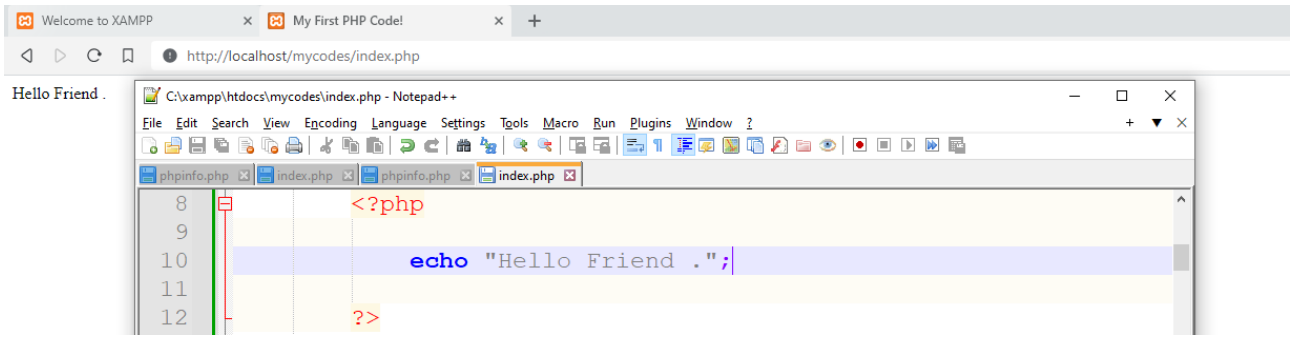
کدهای PHP مابین دو عبارت نوشته می‌شوند وگرنه مفسر PHP اونها رو نمیتونه بخونه . این دو عبارت به عبارت زیرند :

```
    <body>

        <?php

            //You Can Write You PHP Code here  ...

        ?>

    </body>
```

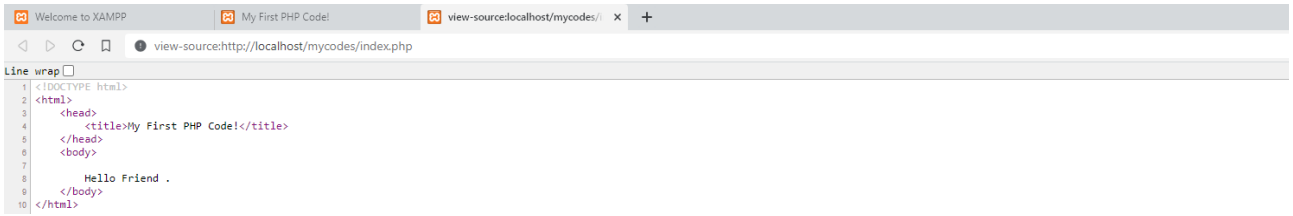
دستور echo و سینتکس آن ؟ echo یکی از ابتدایی ترین کدهای PHP است که باید یاد بگیریم و کار آن نوشتن یک متن بر روی صفحه وب است . سینتکس کلی آن به شکل زیر است : (راستی یادمون نره که در انتهای هر خط PHP باید ; (سمیکولن) رو قرار بدیم وگرنه خطا میده)

```
            echo "Some Text Here ";
```

در مثال زیر از این دستور استفاده کردیم و جمله Hello Friend رو روی صفحه وب نوشتیم :



اگر در صفحه مرورگرمون کلیک راست کنیم و View Page Source رو بزنیم نتیجه زیر رو خواهیم گرفت :



اگه با تصویر قبل مقایسه کنید متوجه میشید که هیچ کد PHP توی این صفحه نشون داده نمیشه و علت هم این هست که اصن نباید نشون داده بشه، کدهای PHP قبل از اینکه برای کاربر ارسال بشن به مفسر PHP داده میشن و مفسر کدها رو اجرا میکنه و نتیجه کدها رو به کاربر میفرسته . مرورگر اصلاً قابلیت اجرای کدهای PHP رو نداره و کدهای PHP باید توسط مفسر آن اجرا شوند .

تعریف متغیر ها در PHP چگونه است ؟ اگه یادتون باشه توی JavaScript درمورد متغیر ها صحبت کردیم و گفتیم که چطوری باید اونها رو تعریف کنیم . از اونجایی که متغیر های جزئی جدایی ناشدنی از زبان‌های برنامه نویسی اند در PHP نیز وجود دارند ولی خب شکل تعریف اونها قاعدتاً با JavaScript و زبان‌های دیگه باید متفاوت باشه . توی PHP وقتی بخوایم یک متغیر تعریف کنیم باید $ رو ابتدای نام متغیر بنویسیم . به مثال‌های زیر نگاه کنید :

```php
<?php
    $variable1_name = "VALUE";
    $variable2_name = 24;
    $variable3_name = 1.2;
    $variable4_name = true;


?>
```

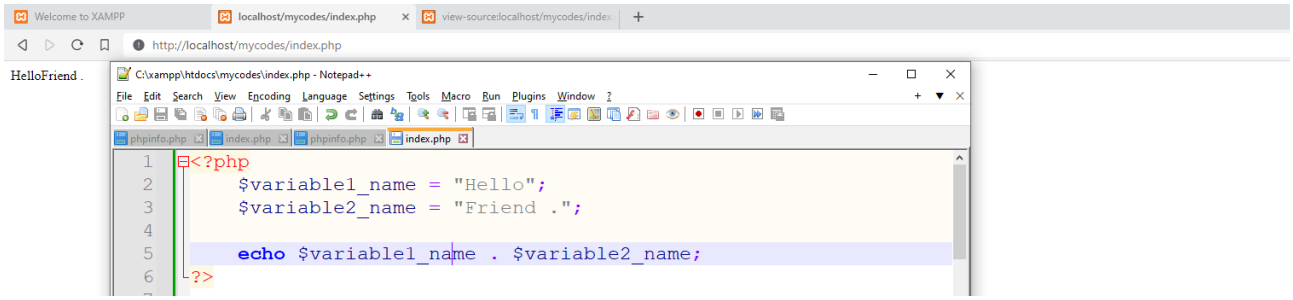
حال اگر بخوایم توسط echo متغیر هایی که تعریف کردیم رو چاپ کنیم کافیست به شکل زیر کد رو بنویسیم :

```php
<?php
    $variable1_name = "VALUE";
    $variable2_name = 24;
    $variable3_name = 1.2;
    $variable4_name = true;

    echo $variable1_name;
    echo $variable2_name;
    echo $variable3_name;
    echo $variable4_name;
?>
```

یه نکته‌ای که درمورد نام دهی متغیر ها یادم رفت تو قسمت جاوااسکریپت بگم این هست که قواعدی داره که بعضیاش به شرح زیرند :

۱. نام متغیر نباید با عدد شروع شود .
۲. در نام متغیر نباید فاصله وجود داشته باشد .
۳. در نام متغیر فقط حروف، _ و اعداد مجاز به استفاده شدن هستند .

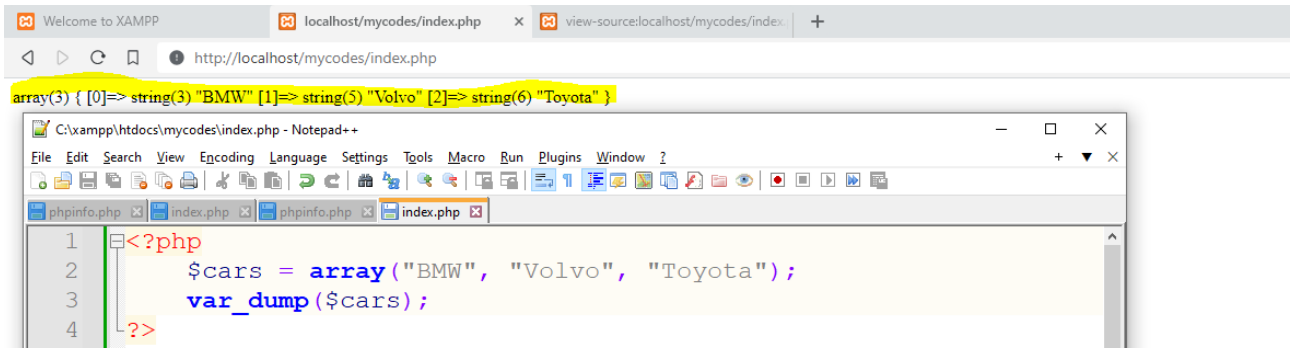
Concatenate کردن دو رشته (String) در PHP ؟ اگه یادتون باشه توی JavaScript اگه میخواستیم دو رشته رو به هم بچسبونیم از + مابین آن‌ها استفاده میکردیم . توی PHP به طرز عجیبی به جای + از . استفاده می‌شود . یعنی علامت نقطه رو مابین دو رشته قرار میدهیم و اونها به هم میچسبند :



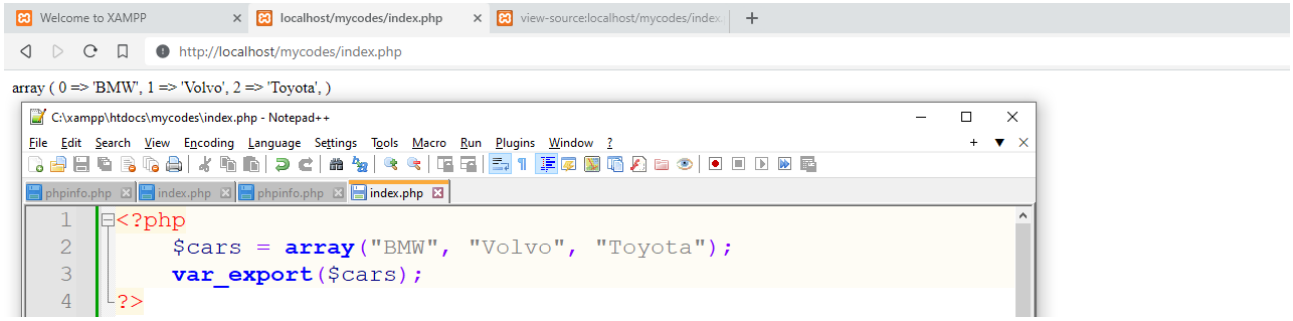
ارایه ها در PHP چگونه تعریف می‌شوند ؟ در PHP برای تعریف ارایه ها باید از تابع array استفاده کنیم . فرض کنید میخواهیم یک ارایه تعریف کنیم و در یک متغیر ذخیره کنیم :

```php
<?php
    $cars = array("BMW", "Volvo", "Toyota")
?>
```

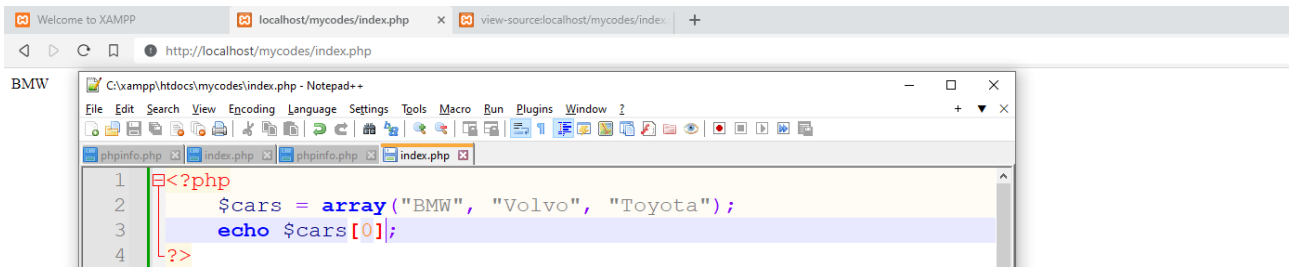
اگر بخواهیم یک ارایه رو بر روی صفحه چاپ کنیم نمیتوانیم از echo استفاده کنیم و باید از تابعی دیگر مثل var_dump استفاده کنیم :



البته از var_export هم میشه برای چاپ کردن ارایه استفاده کرد :



اگر بخواهیم به مقادیر یک ارایه دسترسی داشته باشیم باید مثل JavaScript از [] برای اشاره به index مورد نظرمون در ارایه استفاده کنیم، مثلاً در کد زیر به index شماره صفر از ارایه دسترسی پیدا کردیم :



دستورات شرطی if, elseif و else در PHP ؟ میتونم بگم که تنها تفاوت دستورات شرطی PHP با Javascript واژه elseif است که در JavaScript اون رو به شکل else if استفاده میکنند . یک بلاک if در PHP به شکل زیر تعریف می‌شود :

```php
<?php
    if(condition){
        // Your Code Should Be Here If Condition Is True
    }
?>
```

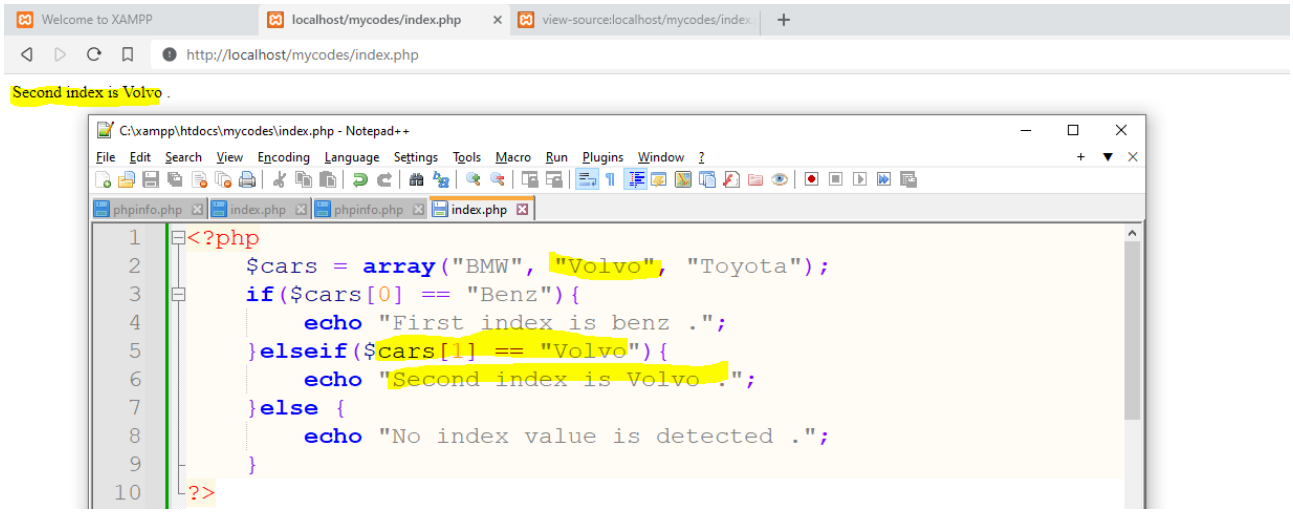
برای تعریف elseif هم به شکل زیر میتوانیم عمل کنیم : دقت کنیم که elseif فقط درصورت غلط بودن if اجرا خواهد شد .

```php
<?php
    if(condition){
        // Your Code Should Be Here If Condition Is True
    }elseif(condition){
        // Your Code Should Be Here If Condition Is True
    }
?>
```

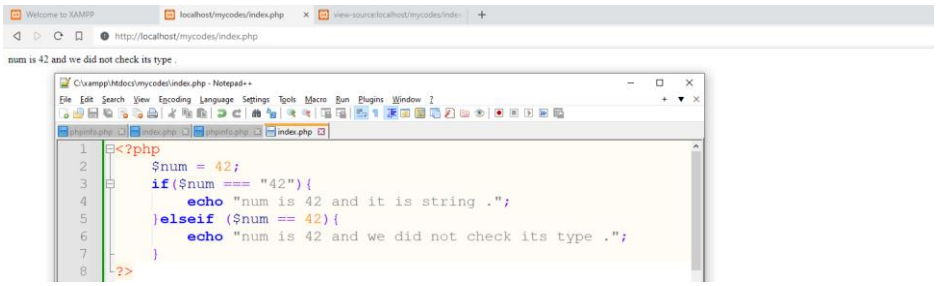
اگر هم بخواهیم else رو تعریف کنیم بسیار ساده است و کافیست که در انتها به شکل زیر عمل کنیم :

```php
<?php
    if(condition){
        // Your Code Will Run If Condition Is True
    }elseif(condition){
        // Your Code Will Run If Condition Is True
    }else {
        // Your Code Will Run If Conditions are not True
    }
?>
```

اگر بخوایم درمورد شرطها مثال بزنیم میتوانیم به مثال زیر اشاره کنیم :



اگر به یاد داشته باشید در JavaScript ما === هم داشتیم که برای مقایسه مقدار و نوع یک متغیر با مقدار و نوع یک متغیر دیگر استفاده میشد . در PHP هم چنین چیزی داریم و دقیقاً همین کار رو انجام میده :



این == و === میتواند منجر به آسیب‌پذیر Type Juggling بشه که در یک سری شرایط مهاجم میتواند از طریق این آسیب‌پذیری صفحاتی را Bypass کند و به صفحاتی دسترسی پیدا کند که نباید . در آینده بررسی میکنیم . البته این باگ به نظرم یه باگ PHP هست و در PHP ورژن ۸ این موضوع برطرف شده است .

در این مسیر ما برخی اوقات منابعی رو معرفی میکنیم که میتوانند در مسیر به ما کمک کنند، یکی از این منابع کمکی سایت medium است که در این سایت مقالات بسیار خوبی درحوزه امنیت و نفوذ پیدا می‌شود . یکی از کسانی که در این سایت مقالات خوبی منتشر میکند خانم Vickie Li هستند که آدرس صفحه medium ایشون https://vickieli.medium.com/ است و همچنین ایشون کتاب خوبی درحوزه باگ بونتی هم دارند که پیشنهاد میشه بخونید البته بعد اینکه دوره تموم شد : Bug Bounty Bootcamp The Guide to Finding and Reporting Web Vulnerabilities by Vickie Li و این کتاب رو میتونید به صورت رایگان از لینک زیر دانلود کنید :

https://github.com/akr3ch/BugBountyBooks/blob/main/Bug%20Bounty%20Bootcamp%20The%20Guide%20to%20Finding%20and%20Reporting%20Web%20Vulnerabilities%20by%20Vickie%20Li.pdf

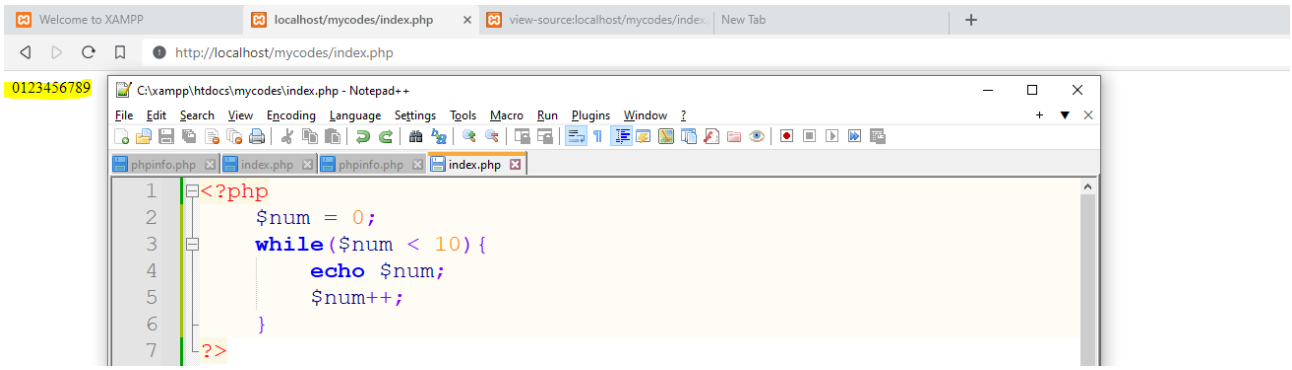
کتاب‌های دیگری هم هستند مثل :

۱. Web Hacking ۱۰۱
۲. Real-World Bug Hunting
۳. Bug Bounty PlayBook

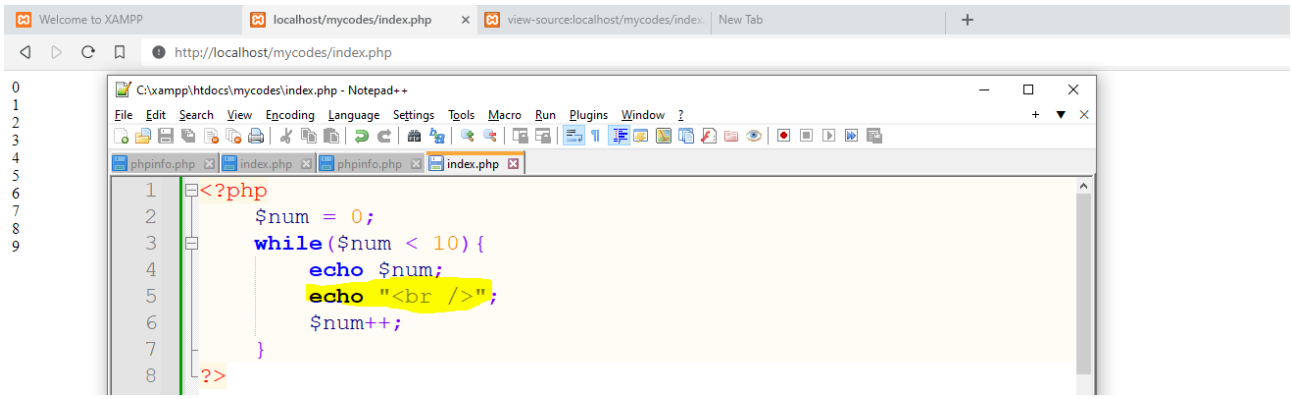
حلقه while در PHP ؟ حلقه while در PHP همان سینتکس JavaScript رو داره و به طور کلی به شکل زیر تعریف می‌شود :

```php
<?php
    while(condition){
        // You code to repeat should be here .
    }
?>
```

اگه بخوام مثال بزنم هم به شکل زیر میتونیم تعریف کنیم :



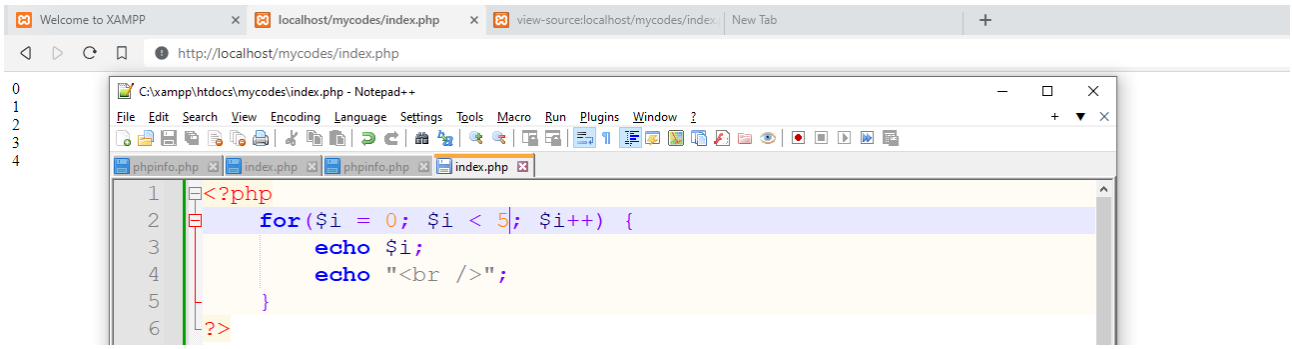
شاید براتون سؤال پیش بیاد چرا همه رو توی یه خط مینویسه ؟ باید بگم که این PHP هست و تا زمانی که نگیم برو خط بعد نمیره . براینکه بگیم برو خط بعد میتونیم از تگ
 توی HTML استفاده کنیم . میتونیم توی حلقه while اونجایی که echo میکنه بعدش بگیم یه تگ
 هم echo کن تا بره خط بعد :



حلقه for در PHP چگونه تعریف می‌شود ؟ سینتکس تعریف حلقه for به مانند تعریف آن در JavaScript است و کافیست که به شکل زیر آن را تعریف کنیم :

```php
<?php
    for($step; condition; $nexStep) {
        // You code should be here ...
    }
?>
```

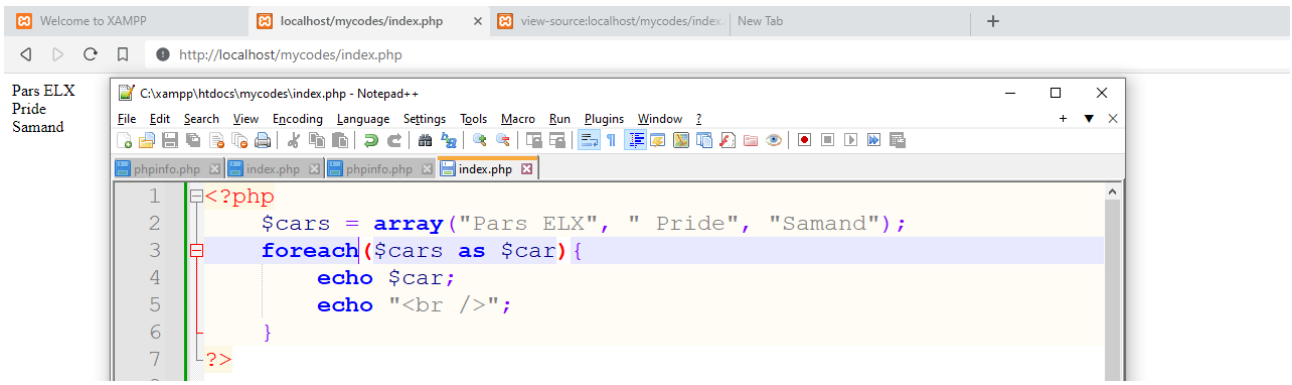
مثال زیر هم یک تعریف از حلقه for است :



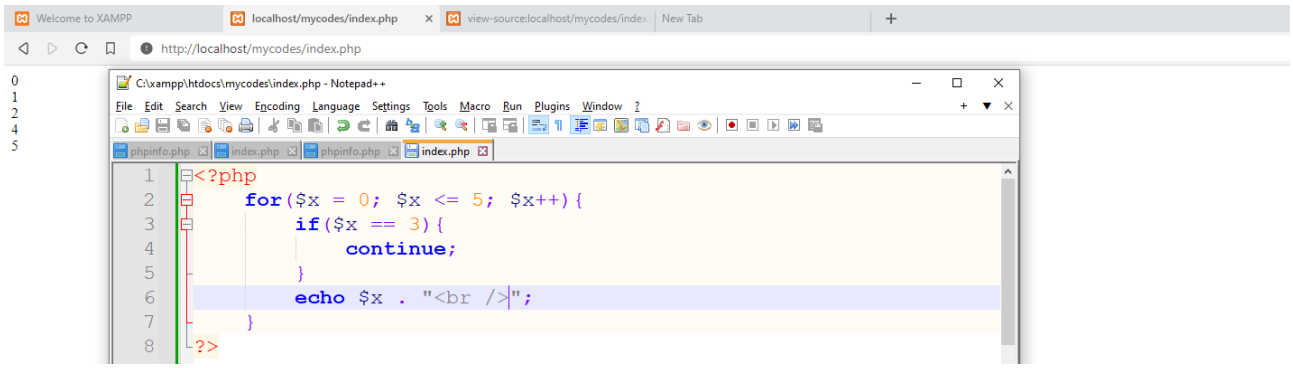
حلقه foreach در PHP چیست ؟ حلقه foreach همان حلقه for...of در JavaScript است و جهت پیمایش بین index های یک ارایه استفاده می‌شود . این حلقه از شما یک ارایه میگیرد و سپس یک به یک index های آن را سپری میکند و هرکاری که خواستید میتونید روی ایندکس ها انجام دهید . سینتکس کلی این حلقه به شکل زیر است :

```php
<?php
    $array = array("Pars ELX", " Pride", "Samand")
    foreach($array as $value){
        // Do something on $value here .
    }
?>
```

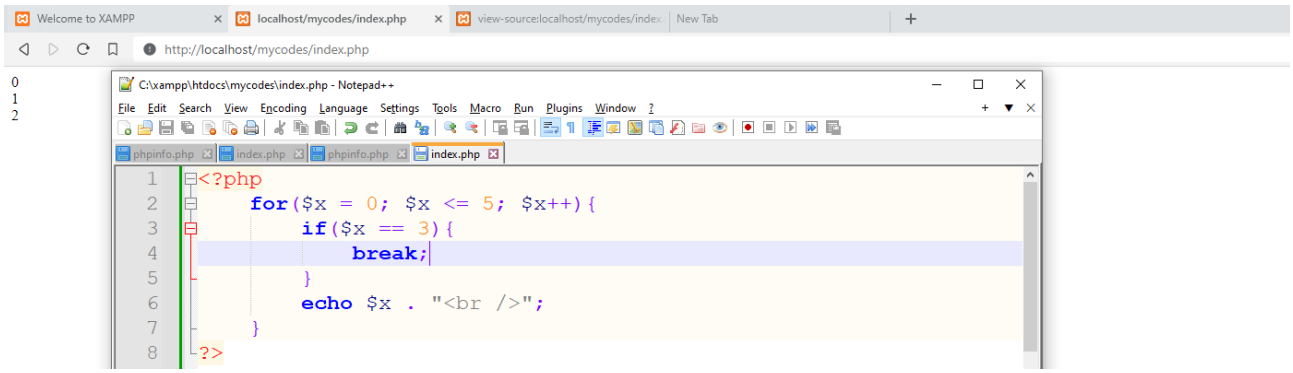
اگر هم بخوایم یه مثال بزنیم میتونیم یک ارایه تعریف کنیم و چند عضو به آن بدهیم و سپس foreach رو روی اون اجرا کنید و بگیم که value ها رو echo کنه و بعد هر value یک تگ
 بزنه :



continue و break هم در PHP وجود دارند و همانکاری رو میکنند که در JavaScript میکردند . continue به حلقه می‌گفت این مورد رو رها کن و برو سروقت مورد بعدی و break به حلقه می‌گفت که از حلقه خارج شو و دیگه تکرار نکن :



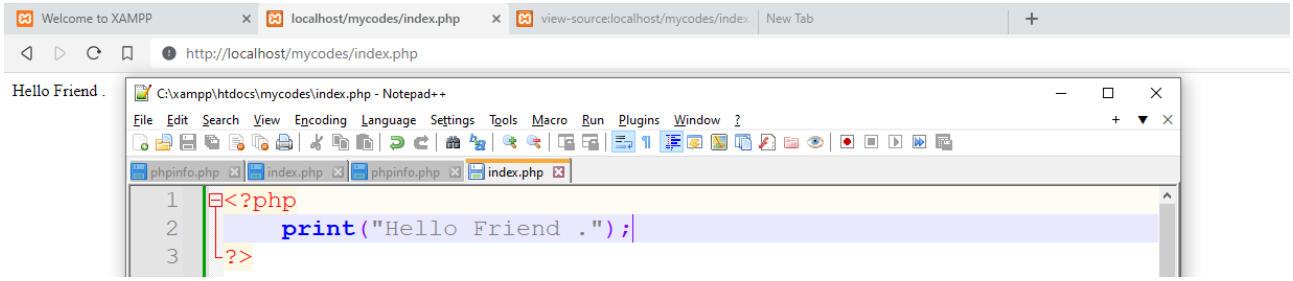
میبینید که در مثال بالا گفته اگه $x برابر ۳ بود از آن گذر کن و به سراغ ۴ برو و میبینیم که ۳ چاپ نشده است .



و

درمثال بالا هم وقتی به ۳ میرسد حلقه رو میشکند و تمام می‌شود .

دستور print هم به مانند دستور echo کار چاپ یک مقدار بر روی صفحه وب رو انجام میده که البته تفاوتهایی داره با echo که اصلاً برای ما اهمیتی نداره .



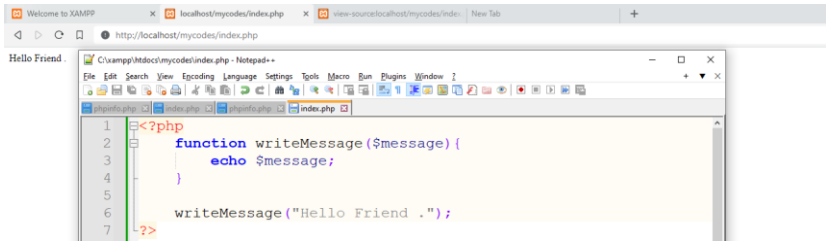
تعریف توابع در PHP چگونه است ؟ دقیقاً به مانند تعریف توابع در JavaScript در PHP نیز برای تعریف از کلمه کلیدی function استفاده می‌شود و بعد از این کلمه نام function قرار میگیرد و سپس در یک بلاک کد های مربوط به تابع نوشته می‌شود . سینتکس کلی تعریف تابع در PHP به شکل زیر است :

```php
<?php
    function function_name($par1, $par2, $par3, ...){
        //Do something with  $par1, $par2, $par4
    }
?>
```

در مثال زیر یک تابع به نام writeMessage تعریف شده است که یک ورودی میگیرد و این ورودی نامش $message است و در بلاک تابع این ورودی را با دستور echo چاپ میکند :

```php
<?php
    function writeMessage($message){
        echo $message;
    }
?>
```

خب تابعی فراخوانی نشود کد اضافه است و به همین خاطر باید تابعی که مینویسیم بالاخره یه جایی استفاده کنیم . برای اینکه یک تابع رو فراخونی کنیم دقیقاً مثل JavaScript کافیست که نام آن را بنویسیم و بعد از نام () قرار دهیم و این یعنی تابع رو فراخونی کن و اگر تابع ورودی داشت آن را در () ارسال میکنیم :



خب تا همینجا واسه مقدمات PHP کافیه و اگر بخوایم بیشتر یاد بگیریم در طول جلسات بعدی با تعریف پروژه یاد خواهیم گرفت . بعد از PHP ما باید آگاهی نسبی نسبت به پروتکل HTTP پیدا کنیم چرا که وب بر اساس این پروتکل کار میکنه . بریم که HTTP رو داشته باشیم :

[۰۱:۲۱:۵۲]

خب حالا HTTP چیه ؟ HTTP مخفف Hyper Text Transfer Protocol است . یک پروتکل جهت انتقال Hyper Text یا ابرمتن ها و یا به صورت کلی متنها و توسط Tim Berners-Lee ساخته و اولین نسخه آن یعنی ۰.۹ در سال ۱۹۹۱ منتشر شد . این پروتکل در OSI و TCP/IP در لایه Application قرار میگیرد و به جز نسخه ۳ بقیه نسخه ها از TCP به عنوان پروتکل لایه Transport استفاده میکند . پروتکل HTTP یک Communication Protocol محسوب می‌شود چرا که بر طبق قوانینی که تعریف میکنه اجازه میدهد که دو یا چند موجودیت (Entities واقعاً نتونستم معنی دیگه ای پیدا کنم واسش) اطلاعاتی را به هم انتقال دهند . این پروتکل مثل هر پروتکل دیگری دارای مولفه هاییست که ما میبایست آن‌ها را بشناسیم و با عمل‌کرد آن‌ها آشنا باشیم . در کل HTTP پروتکل پیچیده‌ای نیست و کارهای پیچیده توسط پروتکل های پایین‌تر اعمال میشن . HTTP رو یک پروتکل Stateless میدونن

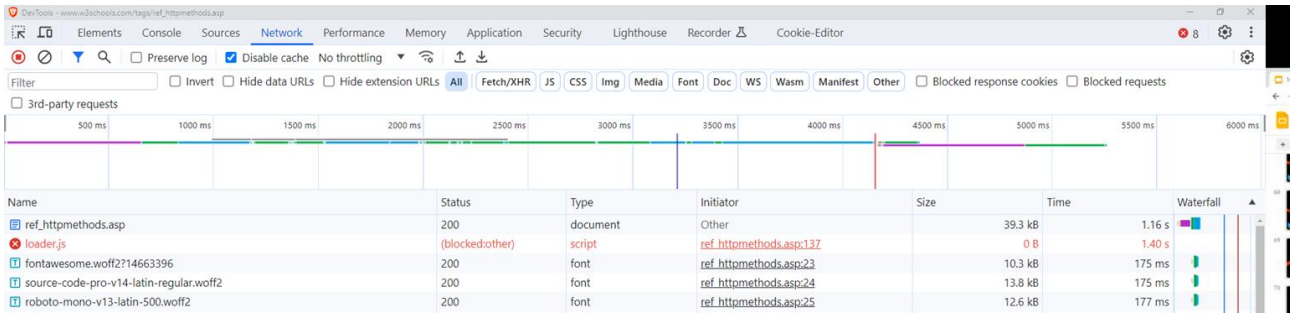
RFC که این پروتکل رو توضیح میده رو میتونید از لینک https://datatracker.ietf.org/doc/html/rfc۱۹۴۵ و https://datatracker.ietf.org/doc/html/rfc۹۱۱۰ ببینید و بخونید تا از نحوه عمل‌کرد آن به خوبی مطلع شوید گرچه قرار است که آن را توضیح دهیم . پیشنهاد بر این است که تا میتونید این پروتکل رو بهتر بفهمید و هرچه بیشتر بفهمید بیشتر به خودتون لطف کردید . تا آخر دوره تمام

مباحث یه طورایی به HTTP ختم میشه و قراره کلی ازش بفهمیم، تا زمانی که این پروتکل رو خوب نشناسیم نمیتونیم یک هانتر خوب در حوزه Web Application باشیم .
یه کتاب هم هست که این پروتکل رو به نحوی احسن توضیح میده و پیشنهاد میشه حتماً حتماً اون رو مطالعه کنید و اسمش HTTP: The Definitive Guide است و میتونید از لینک زیر اون رو دانلود کنید :
https://github.com/oxidation۹۹/MyBooks-۱/blob/master/HTTP٪۲۰The٪۲۰٪۲۰Definitive٪۲۰Guide.pdf

HTTP یک پروتکل Stateless است و یک پروتکل Stateless پروتکلیست که کلاینت یک Request رو به سرور ارسال میکنه و سرور یک Response رو بر اساس Request کلاینت برای او می‌فرستد . این نوع پروتکلها اطلاعاتی درمورد Session و وضعیت طرفهای ارتباط در ارسال چند Request نگهداری نمیکند . یعنی اگر ما یک Request ارسال کردیم، ارتباط قطع شد و بار دیگر ما یک Request دیگر ارسال کردیم، Server ما رو نمیشناسه و نمیدونه که ما همونی هستیم که Request اول رو ارسال کردیم . البته شاید بگید که ما وقتی لاگین میکنیم توی یک وبسایت، توی tab های مختلف Login هستیم و اگر چند ساعت دیگه هم برگردیم توی سایت باز لاگین هستیم ! اگه HTTP یک پروتکل Stateless هست و در هر Request ما رو نمیشناسه پس چطوری میدونه که ما لاگین هستیم ؟ خب جواب این هست که HTTP ذاتاً Stateless است و با ترفند هایی مثل Session و Cookie که Session ID رو ذخیره میکنه اون رو شبه Stateful کردند .
پروتکل هایی مثل HTTP, UDP, DNS از نوع Stateless پروتکلها هستند .

ورژن های متعددی از HTTP تا کنون ارائه شده است که عبارت اند از : ۰.۹, ۱.۰, ۱.۱, ۲ و ۳ . ورژن ۰.۹ و ۱.۰ منسوخ شده‌اند و شاید نتوان سایتی را پیدا کرد که بر اساس این دو کار کنند و حتی در برخی فایروال ها ارسال درخواست به یک سرور HTTP ورژن ۰.۹ و ۱.۰ و یا از طریق HTTP ورژن ۰.۹ و ۱.۰ را بسته اند و امکانش وجود نداره ولی عمده وبسایت های امروزی بر روی ۱.۱ , ۲ متمرکز هستند و ورژن ۳ هم در حال رونق گرفتن است . دونستن تفاوت ورژن های مختلف مهم است و در آینده به بررسی این تفاوت‌ها میپردازیم .

HTTP یک پروتکل Request-Reponse است و سرور HTTP بر روی یک پورت تعیین شده (معمولاً ۸۰ برای HTTP و ۴۴۳ برای HTTPS) منتظر است که از طرف یک کلاینت به او یک Request ارسال شود، این Request کلاینت را دریافت کند و آن را تجزیه و تحلیل نماید و سپس یک Response متناسب با Request کاربر برای او ارسال کند . این Request کاربر و همچنین Response سرور دارای یک Header است که در این Header مجموعه‌ای از مولفه ها وجود دارد . یکی از این مولفه ها Method است که از الزامی ترین مولفههای HTTP محسوب می‌شود . متد های متنوعی وجود دارد که برخی از آن‌ها عبارت اند از : GET, POST, HEAD, OPTIONS, TRACE و ... . شما در هر صفحه وب که از طریق مرورگر باز میکنید کلید F۱۲ رو بزنید صفحه‌ای به نام DevTools را خواهید دید و اگر از منوی افقی بالای این صفحه تب Network رو باز کنید تمام Request هایی که از طرف شما به سایت مورد نظرتون رفته رو میبینید و علاوه بر Request ها میتوانید Response های هر Request رو هم ببینید .



حال اگر بر روی هر کدام از این Request ها کلیک کنید، جزئیات آن و Response آن را خواهید دید :

General نشون دهنده اطلاعات کلی Request و Response با هم است :

▼ General

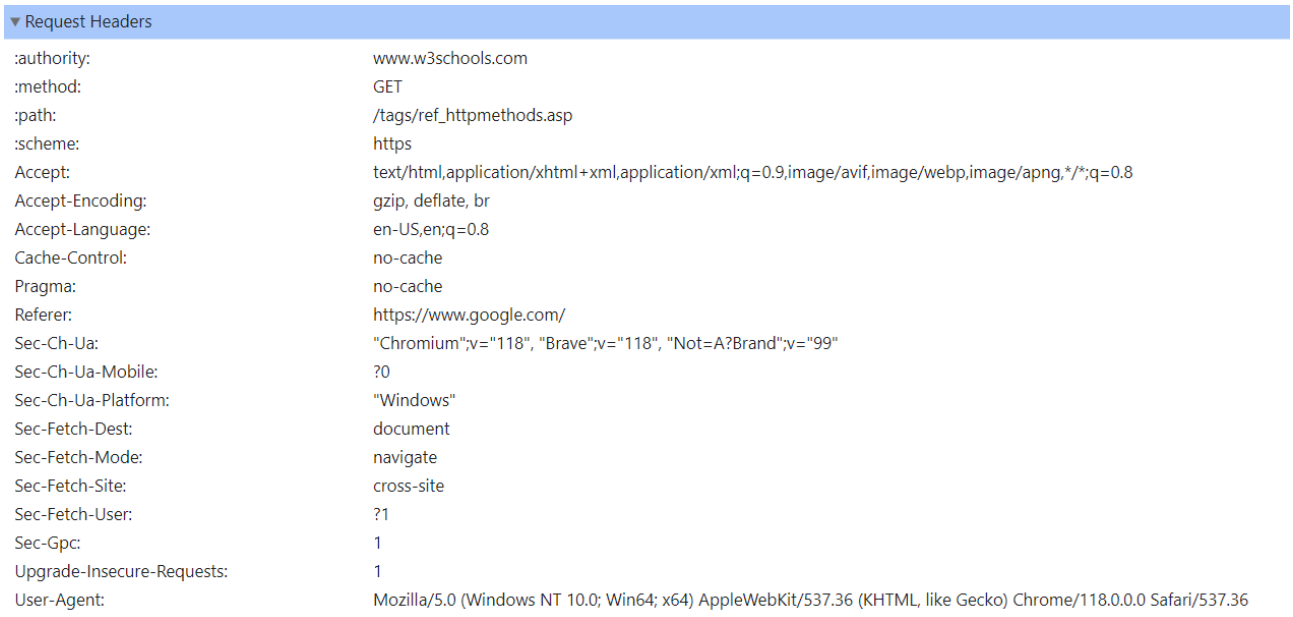
Request URL:	https://www.w3schools.com/tags/ref_httpmethods.asp
Request Method:	GET
Status Code:	200 OK
Remote Address:	127.0.0.1:10808
Referrer Policy:	origin

Response Header شامل تمام مؤلفه های Response است که از طرف سرور HTTP دریافت کرده‌ایم :

▼ Response Headers

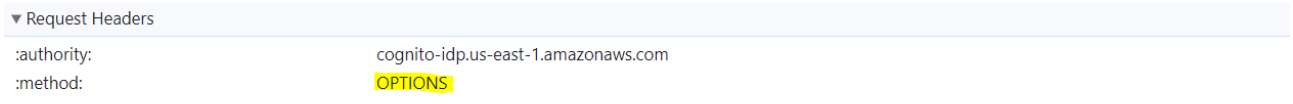
Accept-Ranges:	bytes
Age:	5380
Cache-Control:	Public,public
Content-Encoding:	gzip
Content-Length:	38960
Content-Security-Policy:	frame-ancestors 'self' https://mycourses.w3schools.com;
Content-Type:	text/html
Date:	Sun, 29 Oct 2023 09:30:11 GMT
Expires:	Sun, 29 Oct 2023 13:30:11 GMT
Last-Modified:	Sun, 29 Oct 2023 08:00:32 GMT
Server:	ECS (frb/6793)
Vary:	Accept-Encoding
X-Cache:	HIT
X-Content-Security-Policy:	frame-ancestors 'self' https://mycourses.w3schools.com;
X-Powered-By:	ASP.NET

و Request Header شامل مولفه های Request ما است که از طرف ما به سمت سرور HTTP ارسال شده است :



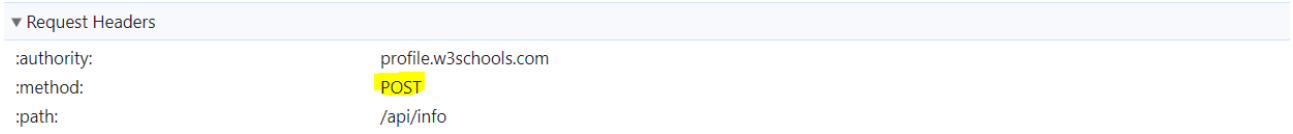
Request Headers	
:authority:	www.w3schools.com
:method:	GET
:path:	/tags/ref_httpmethods.asp
:scheme:	https
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Accept-Encoding:	gzip, deflate, br
Accept-Language:	en-US,en;q=0.8
Cache-Control:	no-cache
Pragma:	no-cache
Referer:	https://www.google.com/
Sec-Ch-Ua:	"Chromium";v="118", "Brave";v="118", "Not=A?Brand";v="99"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"Windows"
Sec-Fetch-Dest:	document
Sec-Fetch-Mode:	navigate
Sec-Fetch-Site:	cross-site
Sec-Fetch-User:	?1
Sec-Gpc:	1
Upgrade-Insecure-Requests:	1
User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.0.0 Safari/537.36

گفتیم که method یکی از مهم‌ترین و الزامی ترین مولفه هاست که شما میتوانید آن را در Request Headers در تصویر بالا ببینید که مقدار آن در این مورد GET است . در نمونه زیر هم یک Request با متد OPTIONS میبینیم :



Request Headers	
:authority:	cognito-idp.us-east-1.amazonaws.com
:method:	OPTIONS
:path:	/

و در نمونه زیر هم یک Request با متد POST :

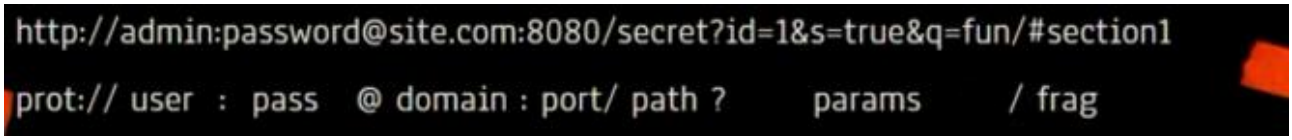


Request Headers	
:authority:	profile.w3schools.com
:method:	POST
:path:	/api/info

برخی نمونه‌های متد های مختلف HTTP رو دیدیم . جالبه که بدونیم نزدیک به ۴۹ متد مختلف وجود داره و اگر خیلی بخوایم بگیم که کار میکنیم و برامون اهمیت دارد ۱۰ مورد از انهاست، اون هم اگر یه وقتی API کار کنید . معروف ترینها GET, POST, OPTIONS, HEAD, PUT, DELETE هستند .

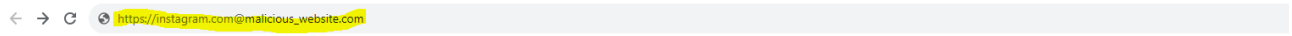
اگر بخوایم پکت HTTP بسازیم کافیه که یک کانکشن TCP برقرار کنیم و شروع کنیم به نوشتن پکت ها و اونها رو ارسال کنیم به سمت Server, از نرم افزارهایی که اجازه میده به سادگی پکت های HTTP رو بسازیم میتونیم به Netcat, OpenSSL, Telnet, BurpSuite و ... اشاره کنیم .
البته دقت کنید که Netcat و Telnet به خاطر اینکه به صورت Clear Text عمل میکنند نمیتونن پکت های HTTPS رو ارسال کنند و یا دریافت کنند و برای اینکار کافیه که از OpenSSL و یا BurpSuite استفاده کنیم . در ادامه تست میکنیم :)

ساختار URL چگونه است ؟ یکی از مهم‌ترین قسمت‌هایی که بایستی بفهمیم و بتونیم تشخیص دهیم ساختار URL است . در تصویر زیر این ساختار رو داریم :

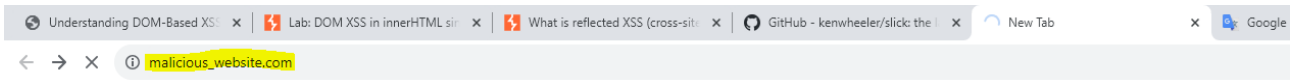


اون قسمت Protocol رو Wrapper یا Scheme هم میگن که Scheme واژه بهتریه چرا که تنها پروتکل نیست که میتونه اونجا قرار بگیره . ftp, http و چندتا دیگه پروتکلن ولی file و expect , ogg و ... میتونن اونجا باشن پروتکل نیستن و قاعدتاً Wrapper هستند .

قسمت دوم یعنی user:pass مربوط میشه به Simple Authentication که یک قابلیت هست که خیلی وقت پیشا مرورگرا واسه ورود به وبسایتها با Credentials اتخاذ کرده بودند . هنوز فعاله ولی سایتای خیلی کمی هستند که از این روش جهت لاگین استفاده کنند . روش امنی نیست و خب Credentials داره توی آدرس بار نشون داده میشه و این اصن چیز خوبی نیست . البته وجود این مورد در مرورگرها میتونه منجر به حمله فیشینگ بشه . اگه ما آدرس زیر رو وارد مرورگر کنیم چه میشه ؟

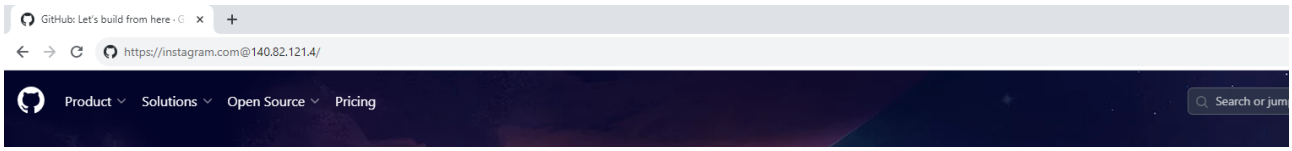


ما اومدیم و به جای Credentials آدرس یک وبسایت Valid رو نوشتیم و علامت @ رو گذاشتیم و در انتها آدرس یک وبسایت مخرب . مرورگر در این موارد instagram.com رو به عنوان Credentials به malicious_website.com ارسال میکند . یعنی کاربر به malicious_website.com ارجاع داده می‌شود در حالی که در ابتدا برای گول زدن کاربر instagram.com رو نوشته‌اند .

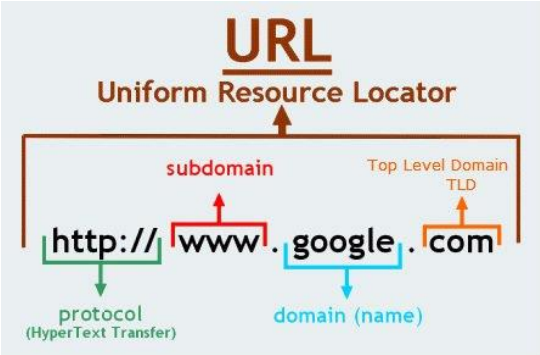


این کار میتونه منجر به حمله Phishing بشه . حال این مورد یه خورده ضایع بود و میتونن کارای مختلفی بکنن . فرض کنید که github.com یک وبسایت مخرب است . آدرس IP ان ۱۴۰.۸۲.۱۲۱.۴ است و اگر این رو وارد Address Bar مرورگر کنید وارد گوگل می‌شود . اگر این آدرس IP رو تبدیل به Decimal کنیم مقدار **۲۳۵۴۲۱۵۱۷۲** به ما داده می‌شود . همین عدد رو اگر در مرورگر وارد کنیم وارد Github خواهیم شد :

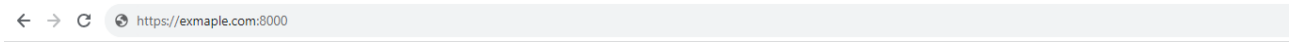
یک مهاجم با استفاده از Credentials شما رو به صفحه Github.com ارسال میکند در حالی شما فکر میکنید مثلاً دارید وارد اینستاگرام می‌شوید :



قسمت سوم از مرورگر domain می‌باشد و اون قسمت com. یا هر چیزی دیگه رو TLD میگن یا Top-Level Domain . در تصویر زیر مشخص است :

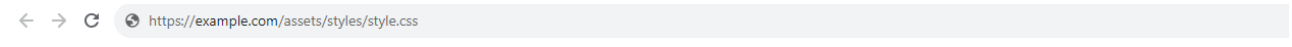


پس از Domain یک علامت : (کولن) وجود دارد که پس از آن آدرس پورت قرار میگیرد . شاید بگید که ما که آدرس پورت رو توی مرورگر نمیبینیم و درست هم میگید . مرورگر پورت استاندارد پروتکل HTTP رو ۸۰ و پورت استاندارد HTTPS رو ۴۴۳ در نظر میگیرد و اگر شما پورت رو نزنید به صورت پیش‌فرض به پروتکل نگاه میکند و اگر HTTP بود، پورت ۸۰ در نظر میگیرد و اگر HTTPS بود پورت ۴۴۳ . فرض کنید که شما یک وبسایت HTTPS دارید ولی این وب سرور بر خلاف همیشه روی پورت ۴۴۳ کار نمیکنه و شما پورت رو به ۸۰۰۰ تغییر داده‌اید . برای اتصال به این وبسایت با پروتکل HTTPS باید در انتهای Domain علامت : و شماره پورت رو وارد کنید :

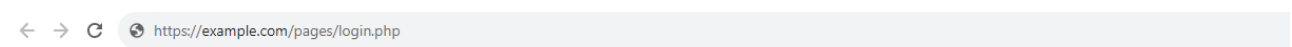


پورتهایی که معمولاً برای وب استفاده می‌شوند ۸۰، ۴۴۳، ۸۰۰۰، ۸۰۸۰، ۸۰۸۱ می‌باشد .

قسمت چهارم از URL رو Path میگن . تا قبل از Path ما با اینترنت سر و کار داشتیم تا به وب سرور مدنظر خودمون متصل بشیم و حال که اتصال پیدا کردیم وقت آن است که به فایلهای Storage وب سرور دسترسی پیدا کنیم . Path به یک فایل داخل Storage وب سرور اشاره میکنه .



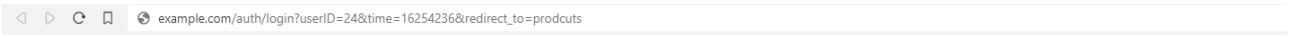
آدرس بالا میگه که فایل style.css رو که در دایرکتوری styles که در دایرکتوری assets است و دایرکتوری assets که در / وجود داره رو به من بده . یا مثلاً صفحه لاگین در یک فایل به نام login.php است و این فایل در دایرکتوری pages در / قرار دارد و برای دسترسی به اون میتونیم از آدرس زیر استفاده کنیم :



البته بگم که امروزه با اومدن Framework های جدید اصطلاحی به نام URL Nicing یا URL Rewriting برای زیبایی و مباحث SEO و امنیت بیشتر بوجود اومده که Path دقیقاً به فایل روی Storage وب سرور اشاره نمیکنه و به طور کلی به یک تابع در Source Code پشت وبسایت اشاره میکنه که اون تابع به یک فایل بر روی Storage دسترسی داره و میتونه محتویات فایل رو برای ما بفرسته .

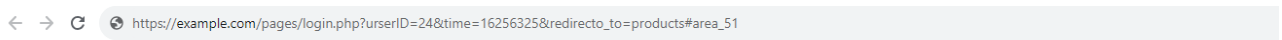
در آدرس بالا معلوم نیست که فایل صفحه login نامش چیست و در کجا قرار داره ولی آدرس بالا یک تابع (یا بهتره بگیم یک Controller، مباحث مربوط به فریمورک ها و MVC است اگه نمیدونین برید یاد بگیرید، با سپاس) رو فراخونی میکنه و اون هست که فایل صفحه login رو میخونه و محتویاتش رو برای ما ارسال میکنه .

قسمت پنجم Parameter هاست که اهمیت ویژه ای برای ما که میخوایم Pentest کنیم دارن، بعد از path یک علامت ? می اد و پارامتر یا پارامتر ها به صورت ParameterName=ParameterValue جلوی آن قرار میگیرند . دقت کنید که مابین هر پارامتر با پارامتر دیگه علامت & قرار میگیره و اونها رو از هم جدا میکنه . این موارد زمانی که میخوایم بررسی کنیم یک وبسایت چطوری URL رو Parse میکنه اهمیت پیدا میکنه و گاهی اوقات ازشون باگ میزنه بیرون و گاهی اوقات هم میتونیم ازشون برای Bypass برخی موارد امنیتی استفاده کنیم .

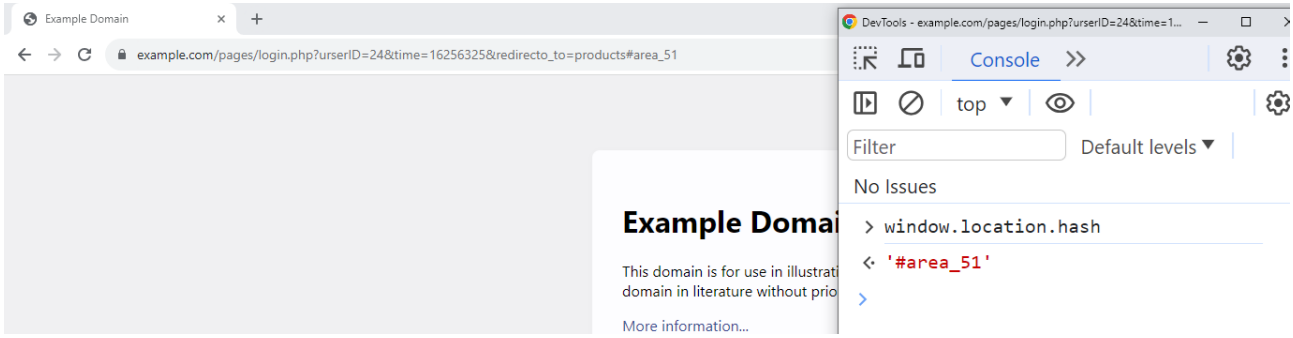


در مثال تصویر بالا سه پارامتر وجود داره که عبارت اند از userID, time, redirect_to که هر سه مقادیری را دارند . دقت کنید که این پارامتر ها یا توسط JavaScript داخل صفحه استفاده می‌شوند و یا توسط Controller که آدرس Path به آن اشاره میکند و یا توسط فایلی که آدرس Path به آن اشاره میکند . برخی اوقات باگهای Prototype Pullotion , XSS و ... رو میشه از همین‌ها استخراج کرد .

قسمت آخر از URL رو بهش میگن Fragment، این قسمت اصلاً توسط وب سرور دیده نمیشه و توسط پکتها شناسایی نمیشه و کلاً کاربر سمت کاربرد داره . Fragment از لحاظ لغوی یعنی قطعه قطعه و چنین چیزهایی . فرض کنید که یک صفحه بزرگ از متن های چرت و پرت داریم و خب هر کدام از پاراگرافهای توسط یک هدر مشخص شده است . میخواهیم یک فهرست بسازیم و با کلیک بر روی هر کدام از متن های فهرست، صفحه کاربر به هدر پاراگراف مورد نظر ما Scroll شود . برای این کار به هر کدام از هدر ها یک id اختصاص میدیم، مثلاً id یکی از آن‌ها area_۵۱ است . اگر توی آدرس بار مرورگر آدرسی با Fragment برابر area_۵۱ بنویسید کاربر به سرعت به هدر مورد نظر ما Scroll میشه :



میتونیم توی JavaScript از طریق کد زیر به Fragment دسترسی داشته باشیم :



دقت کنید که هرچیزی که برنامه نویس از URL که توسط کاربر وارد شده دریافت میکند و جایی در کدهای جاوااسکریپت خود استفاده میکند احتمال بروز آسیب پذیریهایی مثل XSS رو زیاد میکنه . پس ما باید یاد بگیریم که چطوری توسط جاوااسکریپت میتونن URL رو Parse می کنند و هرجایی که اینکار رو میکنن دقیقتر بررسی کنیم .

[۰۱:۴۹:۰۸]

ساختار پکت HTTP چگونه است ؟ پکت HTTP از پخشهای زیر ساخته می‌شود و ساختار آن به شکل تصویر زیر است :

```
[METHOD] [/PATH+PARAMS] [PROTCOL/VERSION]
[HEADERS]
[CRLF][CRLF]
```

در تصویر بالا چند بخش رو میبینید . البته چیزی که بالا نشون داده یک HTTP Response است . در ابتدای پکت یک کلمه داریم که بیان کننده Method است . این کلمه میتونه GET, POST, PUT, DELETE, OPTIONS, HEAD و ... باشه . بعد از Method یک فاصله داریم و سپس URL رو نوشته، این URL تمام قسمتهای URL رو شامل نمیشه و فقط از Parameter ها به بعد است . بعد این قسمت یک فاصله داریم و کلمه HTTP/[VERSION] که ورژن HTTP رو مشخص میکنه . بعد از Protocol Version میریم خط بعد . بعد از خط اول Headers قرار میگیره که شامل مجموعه‌ای از مولفه های HTTP و مقادیر انهاست . بعد از اینکه آخرین مولفه هدر رو نوشتیم میریم خط بعدی و یک CRLF دیگه میزنیم که یعنی تموم شد . تصویر زیر یه کم واضح تره :

```
GET /secret?id=1&s=true&q=fun HTTP/1.1
[HEADERS]
[CRLF][CRLF]
```

حالا با telnet و netcat و openssl تمرین میکنیم متوجه میشیم . قبل از اینکه تمرین کنیم بزار بگم CRLF چیه ؟ تصویر زیر یک HTTP Request به سایت google.com است :

```
GET / HTTP/1.1\r\n
Host: google.com\r\n
Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"\r\n
Sec-Ch-Ua-Mobile: ?0\r\n
Sec-Ch-Ua-Platform: "Windows"\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/118.0.5993.88 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7\r\n
X-Client-Data: COKWywE=\r\n
Sec-Fetch-Site: none\r\n
Sec-Fetch-Mode: navigate\r\n
Sec-Fetch-User: ?1\r\n
Sec-Fetch-Dest: document\r\n
Accept-Encoding: gzip, deflate, br\r\n
Accept-Language: en-US,en;q=0.9\r\n
Connection: close\r\n
\r\n
```

در انتهای هر خط \r و \n رو میبینید . CR یعنی \r که مخفف Carriage Return است و LF یعنی \n که مخفف LineFeed است . در پکتهای HTTP وجود این دو کاراکتر یعنی خط بعد . اگر خط ۱۶ رو نگاه کنید میبینید که \r\n رو داره و همینطور در خط ۱۷ هم \r\n رو داره و میشه دوتا \r\n که یعنی پکت تمومه . وقتی میخوایم یک پکت رو بسازیم کافیه کلید Enter رو بزنیم و خودش CRLF تشخیص میده . نیازی به نگرانی نیست . البته لازم به ذکر است که در سیستم عاملهای قدیمی، در ویندوز هردوی این کاراکتر ها باید قرار میگرفتند تا خط بعدی رو ایجاد کنند، در لینوکس فقط نیاز به \n بود و در مک نیاز به \r . این ناهماهنگی موجب میشد که اگه یه فایلی از لینوکس به ویندوز انتقال پیدا کنه موجب بشه که لاینهای بعدی تشخیص داده نشه و همه خطوط پشت هم باشند ولی در سیستم عامل های جدید این مورد رفع شده و نیازی به نگرانی نیست . راستی URL Encode این دو کاراکتر رو هم بهتره بدونیم چون به کارمون میاد، \r مقدار ٪۰d و \n مقدار ٪۰a رو میگیره بعد از Encode شدن و مقدار HEX این دو کاراکتر به ترتیب x۰d و x۰a است .

تصویر زیر هم یک نمونه از HTTP Response است :

```
HTTP/2 403 Forbidden\r\n
Content-Type: text/html; charset=UTF-8\r\n
Referrer-Policy: no-referrer\r\n
Content-Length: 1579\r\n
Date: Mon, 30 Oct 2023 04:30:02 GMT\r\n
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000\r\n
\r\n
<!DOCTYPE html>\n
<html lang=en>\n
  <meta charset=utf-8>\n
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width
  <title>
    Error 403 (Forbidden)!!1
  </title>\n
  <style>\n
    *{
      margin:0;
```

تفاوتش با HTTP Request اخیر این هست که در انتهای پکت بعد از دوتا r/n/ قسمتی دیگه هم وجود داره به نام BODY که محتویات یک فایل رو برای ما انتقال داده . قسمت BODY به صورت Optional هست و یک پکت HTTP فارغ از Request یا Response بودن میتونه داشته باشه و میتونه هم نداشته باشه .

رفتار Web Server پس از دریافت HTTP Packet چگونه است ؟ HTTP براساس مدل Request-Response کار میکنه و این یعنی یک سرور همیشه منتظر دریافت Request است و یک کلاینت (معمولاً وب بروزر و یا یک اپلیکیشن) یک Request رو ارسال میکنه به سرور که هدفش هم دسترسی به یک Resource روی سرور است و سرور اون رو پردازش میکنه و Response مناسب اون Request رو برای کلاینت ارسال میکنه .
حال میخوایم بدونیم چطوری وب سرور Request رو پردازش میکنه ؟ این مبحث خیلی گسترده هست و توی کتاب HTTP: The Definitive Guide توضیح داده شده و فعلاً اینجا ما فقط در حد خلاصه صحبت میکنیم .

۱. کلاینت Request رو ارسال میکنه : کلاینت یک Connection رو با سرور آغاز میکنه و یک Request رو به سمت اون میفرسته . این Request شامل اطلاعاتی مثل HTTP Method, Request Resource (URL), Headers و ممکنه Body رو داشته باشه در POST Request ها .
۲. سرور Request رو دریافت میکنه : همونطور که قبلاً دیدیم Request های HTTP یک فرم و مولفه های مختلفی دارند . در اولین خط از HTTP Request ما Request Line رو داریم . توی این خط مقادیر با استفاده از "فاصله" از هم جدا شده‌اند . Request Line رو بر اساس فاصله از هم جدا میکنه و سه مولفه تشکیل میشه . مولفه اول Request Method و مولفه دوم Path و مولفه سوم HTTP Version . دقت داشته باشید که در Path میتونه Query ها یا همون Parameter ها هم قرار داشته باشه و همچنین Path میتونه یک Absolute URI هم باشه که Scheme داره و ... و در انتهای خط به یک CRLF برخورد میکنه که یعنی خط بعد.

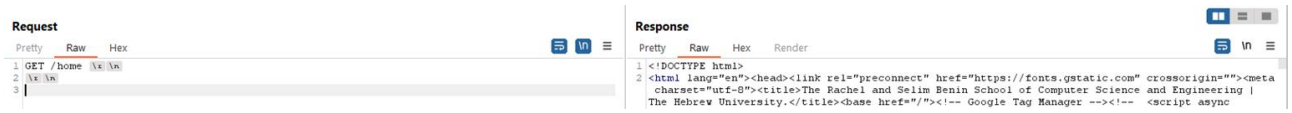
بعد از این خط، HTTP Server میدونه که Header ها قرار دارند . هر مولفه Headers در یک خط قرار داره که با یک CRLF از مولفه دیگه جدا شده . هر مولفه چیزی رو واسه Web Server مشخص میکنه که بعداً بررسی میکنیم .
و در انتهای Headers همیشه دو CRLF وجود داره و بعد از این دو CRLF میتونه Body باشه و میتونه هم نباشه و وب سرور بررسی میکنه که آیا Body هست و یا نیست . در Post Request ها مثلاً API ها Body وجود داره و مقادیری رو داخل خودش به Web Server میده .

۳. بعد از اینکه Web Server درخواست HTTP کلاینت رو بررسی و پردازش کرد برای این درخواست یک جواب مناسب رو میسازه و اون رو به سمت کلاینت ارسال میکنه .

دقت کنید که وب سرور فقط همین کار رو میکنه و کارهایی مثل رمزنگاری، مشخص کردن پورت مقصد و مبدا، مشخص کردن IP Address ها و Routing و رساندن درخواست و یا جواب به مقصد توسط پروتکل های TCP/IP پایین‌تر از HTTP انجام میشه که درواقع پیچیدگی بسیار زیادتری نسبت به کاری داره که HTTP انجام میده .

گفتیم که HTTP ورژن های مختلفی داره که عبارت اند از ۰.۹, ۱.۰, ۱.۱, ۲ و ۳ . و گفتیم که ۰.۹ و ۱.۰ دیگه منسوخ شدن و عموم وبسایت ها بر روی ۱.۱ و ۲ هستند و وبسایت های دارای HTTP Version ۳ هم درحال افزایش است . خب HTTP با منتشر کردن هر ورژنی تغییراتی رو داخل ساختار HTTP اعمال کرد و ما باید این تفاوت‌ها رو بدونیم . من یک به یک این ورژن ها رو توضیح میدم و خودتون تفاوتها رو ببینین .

HTTP/۰.۹ ؟ نام دیگر The One-Line Protocol هست . این ورژن به شدت ساده بود و تقریباً میتونم بگم که هیچ چیزی نداشت به جز یک خط شامل دو مولفه . در این ورژن فقط و فقط یک Method و اون هم GET وجود داشت که بعد از کلمه GET عبارت Path می اومد و درخواست ساخته میشد و به سایت مورد نظر ارسال میشد . در جواب این درخواست هیچ چیزی به جز متحویات Resource درخواست شده به کلاینت ارسال نمیشد و فقط و فقط هم فایل HTML امکان انتقال از طریق این ورژن از HTTP رو داشته چیزی دیگه ای نمیتونسته انتقال پیدا کنه . این ورژن رو وقتی با ورژن حال حاضر قیاس میکنی کامل شدن یک پروتکل در طول زمان رو متوجه میشی، از هیچ رسیده به HTTP Version ۳ .



HTTP ۱.۰ ؟ ورژن ۰.۹ بسیار محدود بود و تقریباً چیزی نداشت . ورژن ۱.۰ که اومد امکاناتی به این پروتکل اضافه شد . ورژن ۰.۹ اومد و بعد نیازهای جدیدی همراه اون پدید اومد، کمبود ها رو حس کردن و گفتن که توی ورژن بعدی اون رو اعمال کنید و ورژن بعدی بهتر از قبلی و همینطور تا امروز . توی ورژن ۱.۰ میتونیم بگیم که چیز های زیر اضافه شد و بهبود پیدا کرد :

۱. اطلاعات ورژن رو باید در هر Request ارسال کنی، یعنی توی هر Request عبارت HTTP/۱.۰ رو باید بنویسی .
۲. متدهای GET, POST, HEAD پشتیبانی میشدند .
۳. ارتباط سریعاً بعد از ارسال پاسخ به کلاینت قطع میشد .
۴. Status Code در ابتدای هر Response باید ارسال بشه . این Status Code موجب شد که Browser ها بتونن موفقیت آمیز بودن و یا شکست خوردن هر Request رو متوجه بشن و نسبت به اون رفتاری خاص رو نشون بدن مثلاً اگر Status Code نشون داد که یک Request با شکست مواجه شده برن و Local Cache ها رو اپدیت کنن و بعد دوباره Request رو ارسال کنن .
۵. مفهوم HTTP Headers برای Request ها و Response ها تعریف شد و موجب شد که Metadata هایی رو انتقال بدن که پروتکل رو انعطاف پذیر تر میکرد .
۶. یکی از مولفه های Header ها Content-Type بود که موجب شد Document هایی جز HTML فایلها هم قابل انتقال بشن .
۷. Host Header اضافه شد ولی به صورت Optional بود .

```
vagrant@kali:~$ telnet www.google.com 80
Trying 216.239.38.120...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.0
host: google.com

HTTP/1.0 403 Forbidden
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1579
Date: Mon, 30 Oct 2023 14:26:37 GMT

<!DOCTYPE html>
```

نمونه زیر هم ببینید که ما میتونیم به جز Content-Type: text/html چیزای دیگه هم داشته باشیم :

همچنین اون ۲۰۰ که جلوش OK رو نوشته هم Status Code هست که کد ۲۰۰ یعنی Successful بودن . کدای مختلفی داریم مثلاً ۴۰۴ به معنای عدم وجود منبع مورد درخواست، ۴۰۳ یعنی Forbidden و اجازه دسترسی نداشتن (یعنی وجود داره ولی به تو نمیدن)، ۵۰۰ یعنی Server Internal Error و خطای داخل سیستم و کدای دیگه ...

اون قضیه اینکه Connection بعد از ارسال پاسخ به کلاینت بسته میشه هم جالبه و در مثال زیر نمونش رو داریم :

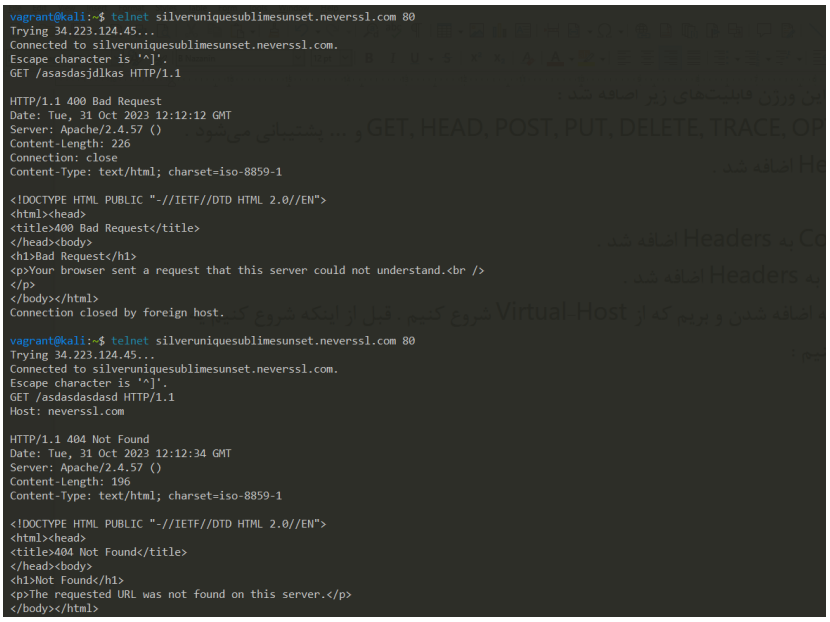
```
vagrant@kali:~$ telnet silveruniquesublimesunset.neverssl.com 80
Trying 34.223.124.45...
Connected to silveruniquesublimesunset.neverssl.com.
Escape character is '^]'.
GET /testpagenotfound HTTP/1.0
Host: neverssl.com

HTTP/1.1 404 Not Found
Date: Tue, 31 Oct 2023 12:07:06 GMT
Server: Apache/2.4.57 ()
Content-Length: 196
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
Connection closed by foreign host.

vagrant@kali:~$
```

میبینید که عبارت Connection closed by foreign host رو آخرش برای ما نوشت که یعنی ارتباط قطع شد .

۱. HTTP۱ ؟ این ورژن از HTTP در حال حاضر وجود داره و زیاد استفاده میشه . تقریباً بر روی عموم وب سرور ها کار میکنه و شما میتوینند از طریق این ورژن درخواست ارسال کنید و وب سرور از طریق این ورژن به شما پاسخ بده . خب قاعدتاً وقتی یه ورژن جدید میاد قابلیت‌های جدیدی هم افزوده میشه . به این ورژن قابلیت‌های زیر اضافه شد :

۱. مجموعه Method های GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS و ... پشتیبانی می‌شود .
۲. مولفه Host اجباری شد و جهت دسترسی به Virtual-Host هم استفاده می‌شود .
۳. مولفه Connection: keep-alive به Headers اضافه شد .
۴. مولفه Upgrade: websocket به Headers اضافه شد .

خب اینکه توی HTTP/۱.۱ مولفه Host اجباری شد رو میتونید توی تصویر زیر ببینید . در مرتبه اول من Host رو تعریف نکردم و خطای ۴۰۰ یعنی Bad Request رو بهم داد که یعنی Request من مشکل داره و مرتبه دوم Host رو تعریف کردم و به من ۴۰۴ داد، چون Path ورودی من وجود نداشت و این یعنی پاسخ داده و Request من مشکل نداشته :



در HTTP/۱.۱ برخلاف HTTP/۱.۰ بعد از پاسخ دادن به یک Request ارتباط توسط وب سرور Close نمیشود و وب سرور منتظر ما میماند تا درخواست بعدی خود را برای او ارسال کنیم . در تصویر زیر این مورد رو میبینید .

البته این قطع شدن و یا نشدن ارتباط توسط وب سرور در قالب مولفه Connection در Headers قابل تعریف است . این مولفه به وب سرور میگه که آیا TCP Connection یا حالا UDP رو قطع کنه یا نکنه . دو مقدار میتونه بگیره که به صورت Default مقدارش keep-alive است و میتونید مقدار close رو هم بهش بدید تا بعد از ارسال پاسخ ارتباط رو قطع کنه :

```
vagrant@kali:~$ telnet silveruniquesublimesunset.neverssl.com 80
Trying 34.223.124.45...
Connected to silveruniquesublimesunset.neverssl.com.
Escape character is '^]'.
GET /notfoundpage HTTP/1.1
Host: neverssl.com
Connection: close

HTTP/1.1 404 Not Found
Date: Tue, 31 Oct 2023 12:23:51 GMT
Server: Apache/2.4.57 ()
Content-Length: 196
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
Connection closed by foreign host.
```

و نمونه زیر هم keep-alive هست :

```
vagrant@kali:~$ telnet silveruniquesublimesunset.neverssl.com 80
Trying 34.223.124.45...
Connected to silveruniquesublimesunset.neverssl.com.
Escape character is '^]'.
GET /notfoundpage HTTP/1.1
Host: neverssl.com
Connection: keep-alive

HTTP/1.1 404 Not Found
Date: Tue, 31 Oct 2023 12:24:24 GMT
Server: Apache/2.4.57 ()
Content-Length: 196
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
```

ممکنه که روی یک سرور چندین Virtual-Host وجود داشته باشه . این یعنی اینکه یک وب سرور با یک آدرس IP میتونه چند هاست مجازی در خود داشته باشه . اگر اینطور نبود که با این میزان از وب سرور هایی که در دنیا وجود دارد تمام IP Address گرفته میشد . برای همین اومدن گفتن که شما میتونید روی یک سرور، یک وب سرور رو اجرا کنید و روی اون وب سرور چند Virtual-Host تعریف کنید و هر کاربری که خواست مثلاً با Virtual-Host با آدرس example.local ارتباط بگیره باید توی درخواستش بزنه که مولفه Host برابر است با example.local و اینکار توسط وارد کردن example.local توی Address Bar مرورگر انجام میشه و وب سرور پس از دریافت Request این مورد رو مدیریت میکنه، توی Virtual-Host های خودش اونی که ادرسش example.local هست رو نشون میده، دقت کنید که درسته شما دارید توی Address Bar می‌زنید example.local ولی این بیشتر اینکه کاربردی باشه جهت راحتی بیشتر کاربر ایجاد شده . یعنی Request شما به IP Address متناظر با example.local ارسال میشه و اصلاً هیچ کدام از Request ها به example.local ارسال نمیشه و همه چیز در پشت پرده با IP Address رخ میده . مورد زیر یک مثال از این موضوع است :

Request

Pretty Raw Hex

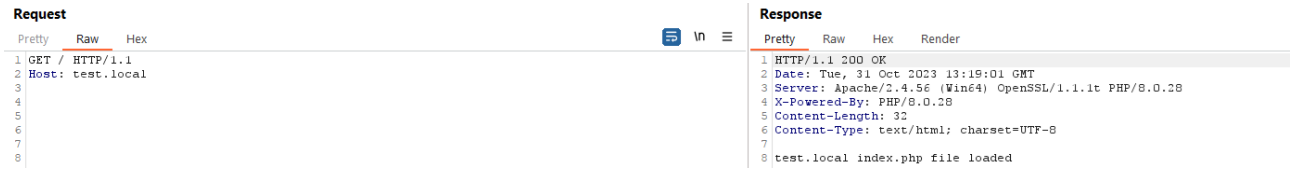
```
1 GET / HTTP/1.1
2 Host: example.local
3 Content-Length: 12
4
5
6
7
8
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Tue, 31 Oct 2023 13:19:34 GMT
3 Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.0.28
4 X-Powered-By: PHP/8.0.28
5 Content-Length: 35
6 Content-Type: text/html; charset=UTF-8
7
8 example.local index.php file loaded
```

و مثلاً درمورد زیر خواستیم به هاست مجازی test.local روی همون IP آدرس دسترسی پیدا کنید و چنین کردیم :



خب تا اینجا ما از telnet و BurpSuite جهت ایجاد ارتباط و دیدن HTTP Request ها و HTTP Response ها استفاده کردیم . گفتیم که Telnet فقط بر روی پورت ۸۰ میتواند داده ارسال کند، چرا که پورت ۴۴۳ دادههای رمزنگاری شده ارسال و دریافت میکند و این درحالیست که Telnet به صورت Plain Text عمل مینماید و همچنین گفتیم که BurpSuite میتونه هم بر روی پورت ۸۰ و هم روی پورت ۴۴۳ داده ارسال کنه . یکی دیگر از ابزارهایی که استفاده می‌شود openssl است که اون هم با پورت ۴۴۳ کار میکنه . سینتکس ایجاد یک ارتباط رمزنگاری شده توسط openssl به شکل زیر است :

```
vagrant@kali:/etc$ openssl s_client -connect [WEB_SERVER_DOMAIN_OR_IP_ADDRESS]:[PORT]
```

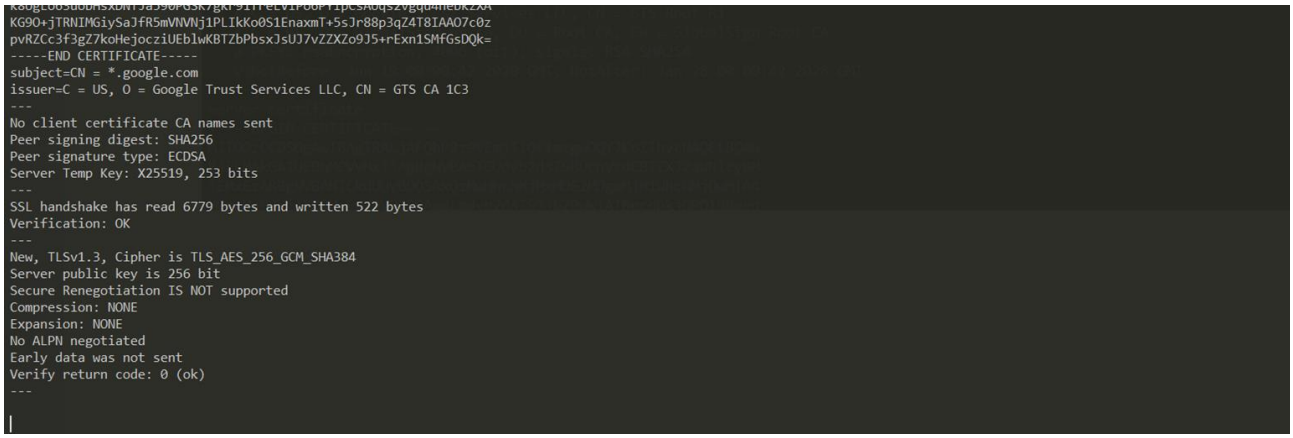
مثلاً میخوام به google.com از پورت ۴۴۳ که مربوط به HTTPS است اتصال پیدا کنم، پس مینویسم :

```
vagrant@kali:/etc$ openssl s_client -connect google.com:443
```

اما خروجی این دستور چه چیزی رو نشون میده ؟

```
CONNECTED(00000003)

depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1
verify return:1
depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
verify return:1
depth=0 CN = *.google.com
verify return:1
---
Certificate chain
 0 s:CN = *.google.com
   i:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
   a:PKEY: id-ecPublicKey, 256 (bit); sigalg: RSA-SHA256
   v:NotBefore: Oct 16 08:02:35 2023 GMT; NotAfter: Jan  8 08:02:34 2024 GMT
 1 s:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3
   i:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   a:PKEY: rsaEncryption, 2048 (bit); sigalg: RSA-SHA256
   v:NotBefore: Aug 13 00:00:42 2020 GMT; NotAfter: Sep 30 00:00:42 2027 GMT
 2 s:C = US, O = Google Trust Services LLC, CN = GTS Root R1
   i:C = BE, O = GlobalSign nv-sa, OU = Root CA, CN = GlobalSign Root CA
   a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
   v:NotBefore: Jun 19 00:00:42 2020 GMT; NotAfter: Jan 28 00:00:42 2028 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIOOzCCDSOgAwIBAgIRALjAFQbP8z9vEmjIlQf1mzgwDQYJKoZIhvcNAQELBQAw
RjELMAkGA1UEBhMCVVMxIjAgBgNVBAoTGUdvb2dsZSBUcnVzdCBTZXJ2aWNlcyBM
TEMxEzARBgNVBAMTCkdUUyBDQSAxQzMwHhcNMjMxMDE2MDgwMjM1WhcNMjQwMTA4
MDgwMjM0WjAXMRUwEwYDVQQDDAwqLmdvb2dsZS5jb20wWTATBgcqhkjOPQIBBggq
```

تا تصویر پایین :



ایجاد TCP Connection و TLS Handshake رو نشون میده . یعنی کلاینت با سرور توافق کردن بر سر یک کلید که از طریق اون کلید دادههایشان رو رمزنگاری کنن و در مقصدها آن‌ها را رمزگشایی . در انتها منتظر است که من درخواست مورد نظر خودمو وارد کنم تا رمزنگاری شود و بعد به سرور ارسال شود و توسط سرور دریافت شود و رمزگشایی شود و پردازش شود و پاسخ متناسب با آن توسط سرور تهیه شود و رمزنگاری شود و به من (کلاینت) ارسال شود و توسط کلید توافق شده رمزگشایی شود و به من نشان داده شود . مثلاً درخواست زیر :

گفتیم که توی HTTP/۱.۰ ارتباط کلاینت و سرور بعد از دریافت پاسخ یک Request بسته می‌شود و کاربر برای اینکه Request دوم رو ارسال کنه باید دوباره Connection رو ایجاد کنه . در HTTP/۱.۱ یک مولفه به نام Connection توی Headers اضافه شد که کاربر میتونست تعیین کنه که آیا سرور Connection رو نگه داره یا نه . اگر مقدار Connection برابر keep-alive بود یعنی ارتباط رو حفظ کن و میخوام درخواست دیگه ای ارسال کنم و اگر مقدار اون close بود یعنی ارتباط رو ببند و درخواستی ندارم . حال این keep-alive تا چه زمانی ارتباط رو حفظ میکنه و نمیبنده ؟ اگر کلاً نبنده به عنوان یک misconfiguration بد تلقی میشه و یک مهاجم میتونه صدها Connection ایجاد و منابع سرور رو آشغال کنه و حمله DOS انجام بده . ولی این ارتباط برای همیشه باقی نمیمونه و توی تنظیمات Web Server میتونید تعیین کنید که ارتباط تا چند ثانیه برقرار باشه و کی Close بشه . توی XAMPP پیکربندی این مورد توی فایل httpd-default.conf وجود داره و این فایل توی xampp\apache\conf\extra هست . مورد زیر تعیین کننده زمان برقرار ماندن درخواست است که به صورت پیش‌فرض ۳۰۰ ثانیه تعیین شده :

```
#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 300
```

مورد زیر تعیین کننده این است که آیا پس از پاسخ به یک Request ارتباط رو برقرار نگه داره یا نه که on یعنی اره و off یعنی نه

```
#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive on
```

مورد زیر هم تعیین کننده این است که، حداکثر تعداد Request ها در یک Connection چقدر باشد :

```
#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100
```

مورد زیر هم تعیین میکنه که پس از پاسخ به اولین Request چند ثانیه منتظر Request بعدی بمونه :

```
#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 100
```

توی لینوکس هم شما اگه HTTP Server Apache رو نصب کرده باشید، بر اساس اینکه با چه پیکربندی نصب کردید در مسیر های زیر میتونید فایل پیکربندی رو پیدا کنید :

- `/etc/apache2/httpd.conf`
- `/etc/apache2/apache2.conf`
- `/etc/httpd/httpd.conf`
- `/etc/httpd/conf/httpd.conf`

بعد میتونید با استفاده از یک ادیتور متن اون رو باز کنید و پیکربندی های ربوط به Timeout ارتباط ها رو به شکل زیر ببینید و درصورت نیاز اونها رو تغییر بدید :

```
#
# Timeout: The number of seconds before receives and sends time out.
#
Timeout 300

#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive On

#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100

#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 5
```

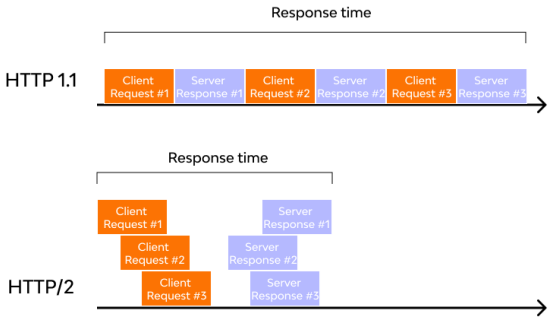
مواردی که گفتیم در وب سرور Apache بودند و همین موارد با تفاوتهایی در وب سرور های دیگه مثل nginx هم وجود داره و قابل پیکربندی است . توی پیکربندی های nginx ما دو مولفه به نامهای client_body_timeout و client_header_timeout داریم که کار مولفه های بالای Apache رو میکنند .

اما پیکربندی های بالا قابل سواستفاده شدن هستند . مثلاً یک مهاجم میاد و Timeout وب سرور رو محاسبه میکنه و بعد یک اسکریپت مینویسه که یک ثانیه مونده به پایان زمان Timeout یک درخواست دیگر ارسال کند و با ایجاد چنین ارتباطاتی موجب آشغال شدن منابع سیستم میشه . یک آسیب‌پذیری به نام SlowLoris وجود داره که به همین شکل عمل میکنه و ممکنه در آینده طریقه اکسپلویت و Prevention این آسیب‌پذیری رو یاد بگیریم .

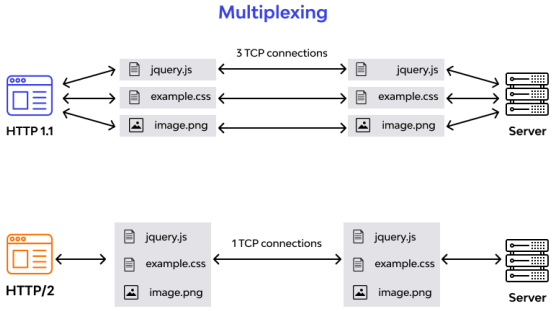
HTTP/۲ ؟ این نسخه از HTTP سال ۲۰۱۵ با نام HTTP/۲.۰ ارائه شد . این ورژن با اهدافی ارائه شد که برخی از اونا در زیر گفته شده :

۱. این نسخه باید با HTTP/۱.۱ سازگاری بالایی داشته باشد .
۲. کاهش تأخیر و بهبود سرعت بارگذاری صفحات با روشهایی مثل فشرده‌سازی HTTP Headers, استفاده از روش Server Push, الویت بندی کردن Request ها، Multiplex کردن درخواست ها بر روی یک TCP Connection
۳. ...

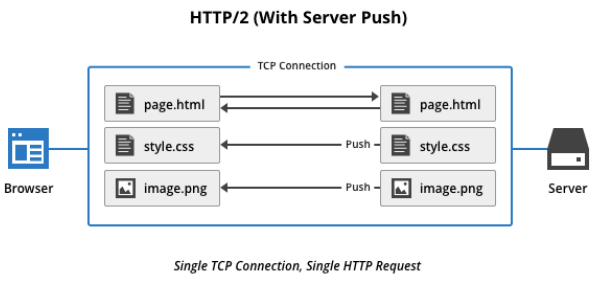
اما تفاوت این نسخه با HTTP/۱.۱ چی هست ؟ قاعدتاً یک نسخه جدید باید تفاوتهایی با نسخه قبلی داشته باشه و کاستی های نسخه قبلی رو ترمیم کنه . HTT/۲ بیشتر سینتکس HTTP/۱.۱ رو مثل Methods, Status Codes, Header Fields, ... رو تغییر نداد اما تغییراتی در نحوه فریم کردن دادهها و انتقال اونها ما بین کلاینت و سرور ایجاد کرد . یعنی اومد بیشتر روی Back-End پروتکل کار کردند . یکی از این تغییرات این بود که در HTTP/۱.۱ در یک Request سرور یک Response رو برای کاربر ارسال میکرد و Request جدید توسط کلاینت در صورتی ارسال میشد که Response درخواست قبلی دریافت شده بود وگرنه منتظر Response درخواست قبلی میموند، حال اگر تهیه این Response برای Server طول میکشید خب دندش نرم، همینه که هست، واستا تا برات بفرستم . توی HTTP/۲ اومدن و این موضوع رو Optimize کردن اونم به این شکل که کلاینت تمام درخواست هایش را پشت سر هم به Web Server ارسال میکنه و وب سرور هم هرکدام از Response ها رو زودتر آماده کرد سریعاً به کلاینت میفرسته و در نهایت کلاینت تما Response رو ها کم کم به هم میچسباند و تمام وب اپلیکیشن لود می‌شود . دیدی که گاهی قسمتی از وبسایت رو باید منتظر بمونید تا لود بشه اونم در حالی که قسمتهای دیگه لود شده و علتش همینه که Server هنوز Response اون Request شما براتون نفرستاده . این Optimization موجب شد که سرعت به طرز وحشتناکی افزایش پیدا کنه .



ویژگی دیگر HTTP/۲ رو Multiplexing میگن . کلمه Multiplexing رو میتونیم مرکب ترجمه کنیم . HTTP/۲ رو به این علت Multiplex میگن که برای چند Request از یک TCP Connection استفاده میکنه حال اینو با چند Request که هر کدام از یک TCP Connection جدا استفاده میکنند مقایسه کنید . همین ایجاد یک TCP Connection حداقل انتقال ۳ پکت داده رو نیاز داره . این ویژگی به طرز جالب انگیزی سرعت وب رو افزایش داده . ترکیب این ویژگی با ویژگی قبلی دهشتناک می شه . دیگه نمیدونم چی بگم . یه تصویر از این ویژگی ببینم حال کنیم :

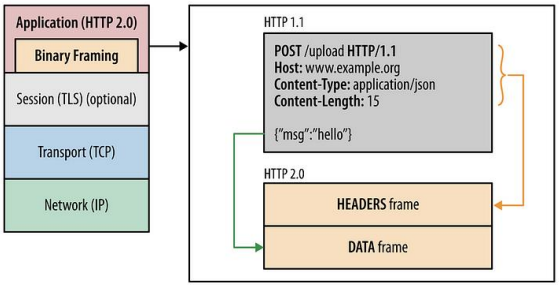


و ویژگی دیگه که به HTTP/۲ افزوده شد رو Server Push میگن . این یعنی چی ؟ یعنی اینکه فرض کنید که یک کلاینت از یک سرور درخواست فایل page.html رو میکنه و این فایل نیاز به style.css و script.js داره و سرور میگه آقا/خانم کلاینت شما نیازی نیست که واسه این دوتا فایل که page.html بهشون نیاز داره Request بزنید من خودم بزرگ شدم و میدونم که باید اینا رو هم براتون بفرستم و این فرستادن این فایلا میگن Push کردن و چون سرور Push میکنه بهش میگن Server Push . این ویژگی از تعداد Request های کلاینت به سرور به اندازه زیادی کاست و خب قاعدتاً منابع کمتری از سرور مصرف میشه و سرور میتونه به Request های دیگه پاسخ بده .



Single TCP Connection, Single HTTP Request

ویژگی آخری که میخوام از HTTP/۲ بگم این هست که در این ورژن ساختار پکتها از حالت Text به Binary تغییر پیدا کرد .



لینک های زیر توضیحاتی درمورد HTTP/۲ و مقایسه اون با HTTP/۱.۱ ارائه کرده‌اند و میتونید بخونید تا بیشتر بدونید :

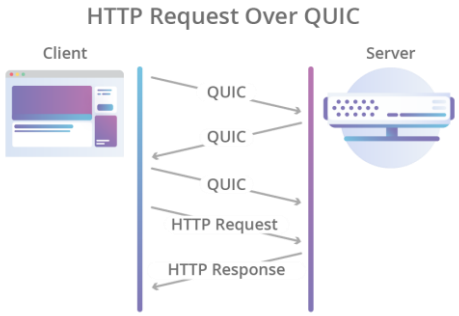
https://www.wallarm.com/what/what-is-http-۲-and-how-is-it-different-from-http-۱

https://blog.codavel.com/http۲-multiplexing

https://httpwg.org/specs/rfc۹۱۱۳.html

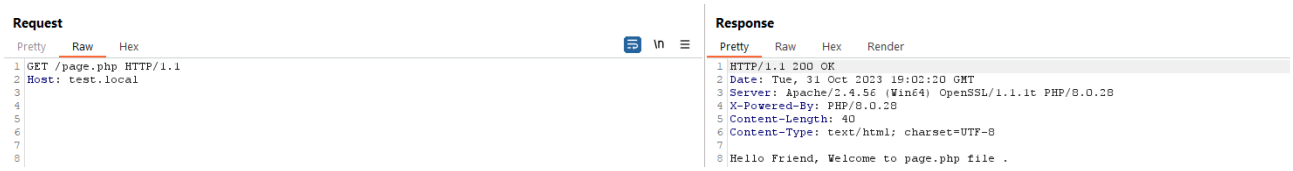
https://medium.com/walmartglobaltech/introduction-to-http-۲-d۳e۳b۴f۴d۶۶۲#:~:text=HTTP%۲F۲%۲۰is%۲۰binary%۲C%۲۰instead,within%۲۰a%۲۰single%۲۰TCP%۲۰connection.

HTTP/۳ یا H۳ ؟ جدید ترین نسخه HTTP است که در حال حاضر منتشر شده و دامنه استفاده از آن در حال گسترش است . در این نسخه هم به مانند HTTP/۲ پکت ها ساختاری Binary دارد . این نسخه نسبت به نسخه های قبلی تفاوت‌های اساسی دارد . H۳ برخلاف دیگر ورژن ها بر بستر یک پروتکل به نام QUIC کار می‌کند . پروتکل QUIC پروتکلی نسبتاً جدید است که در لایه Transport از UDP بهره میگیرد . پس میتونیم نتیجه بگیریم که پروتکل HTTP/۳ بر خلاف ورژن های قبلی که از TCP به عنوان پروتکل لایه Transport استفاده میکردند از پروتکل UDP به عنوان پروتکل لایه Transport استفاده میکند . تفاوت دیگری که این ورژن با ورژن های قبلی دارد این است که این ورژن ذاتاً رمزنگاری شده است . برخلاف HTTP های ورژن قبلی که Plain Text بودند و برای رمزنگاری باید ازSSL/TLS به صورت جداگانه استفاده میکردند و این استفاده چندین مرحله طول میکشید H۳ در خودش ذاتاً SSL/TLS رو دارد و نیازی به استفاده جداگانه نیست . اگر بخوایم یه خورده جزئی تر نگاه کنیم بهتره Data Flow رو به صورتی کلی بررسی کنیم . در TCP سه مرحله طی میشد تا TCP Handshake انجام شود و ۹ الی ۱۲ مرحله هم جهت TLS Handshake و رمزنگاری دادهها باید طی میشد ولی در H۳ که از QUIC استفاده میکند این مراحل کلاً به ۳ مرحله که توسط QUIC انجام می‌شود خلاصه شده است و QUIC به صورت پیش‌فرض از TLS/۱.۳ که قویترین پروتکل رمزنگاری است استفاده میکند . حالا بشینید حساب کنید که چقدر سریعتر شده چیزمیزا .

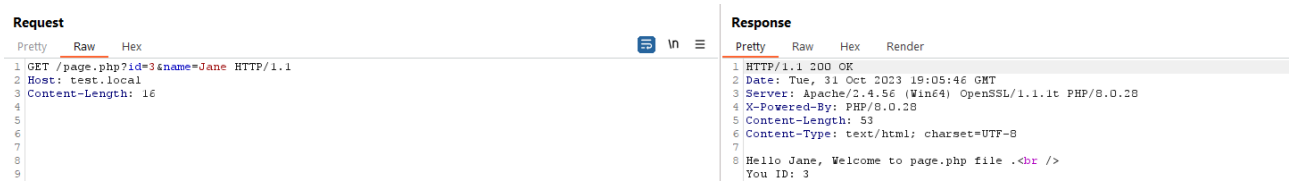


Method ها در پکت های HTTP چیستند ؟ متد ها تعیین کننده ساختار پکت، طریقه ارسال Data و عمل‌کرد پکت هستند . یعنی HTTP Server اولین چیزی که نگاه میکنه متد پکت دریافت شده است و بعد نسبت به اینکه اون متد چی هست رفتاری رو اتخاذ میکنه . حدود ۴۹ متد مختلف در HTTP وجود داره که برخی از اونها عبارت اند از : GET, POST, PUT, DELETE, HEAD, OPTIONS و ... . متدهایی مثل GET و POST جهت ارسال داده استفاده می‌شوند و از معروف ترین متدهای HTTP هستند . در ادامه بعضی از اونها رو با هم بررسی میکنیم .

متد GET ؟ متد GET درخواست یک منبع رو از سرور میکند . این متد میتواند با خود Data داشته باشد و میتواند هم نداشته باشد و فقط نکته مهم این است که اگر Data داشته باشد نباید این Data حساس و امنیتی باشد یعنی نباید Data مهم، حساس و امنیتی رو از طریق درخواست GET ارسال کرد به چند علت : اول اینکه در معرض دید است، دوم اینکه اگر صفحه Bookmark شود داده هم Bookmark میشه و قابل دسترسیست و سوم اینکه گاهی اوقات این دادهها رمزنگاری نمیشوند و با sniff اون مهاجم میتونه به انتها دسترسی پیدا کنه . مثال زیر یک نمونه درخواست GET بدون هیچگونه Data است :



و در تصویر زیر نمونه انتقال داده از طریق متد GET رو میبینیم :



دادهها در متد GET از طریق Parameter های URL انتقال پیدا میکنند و در سمت سرور از طریق زبانهای برنامه نویسی میتوانیم آن‌ها را دریافت و پردازش کنیم . در مثال بالا id با مقدار ۳ و name با مقدار Jane از طریق URL به سمت وب سرور انتقال پیدا کرد . این متد در Form های

HTML هم قابل استفاده است و همچنین دادههایی که از طریق این متد توسط کلاینت دریافت می‌شوند میتوانند توسط مرورگر Cache شده و بار دیگر مرورگر آن‌ها را از طریق Cache ها بخواند . نکته دیگر درمورد این متد این است که به صورت پیش‌فرض هر Request که کلاینت به سمت سرور ارسال میکنه از نوع GET است مگر اینکه برنامه نویس نوع دیگری را تعیین کرده باشد مثلاً در تگ form ها یک Attribute به نام method داریم که تعیین میکند دادههای این فرم از طریق چه متدی به سمت سرور ارسال شود .

```
9    <form method="POST" action="">
10       <!-- Form Code -->
11   </form>
```

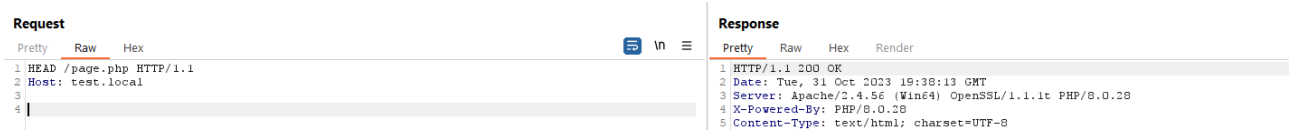
و اگر به شکل زیر باشد به صورت پیش‌فرض با متد GET ارسال می‌شود :

```
9    <form action="">
10       <!-- Form Code -->
11   </form>
```

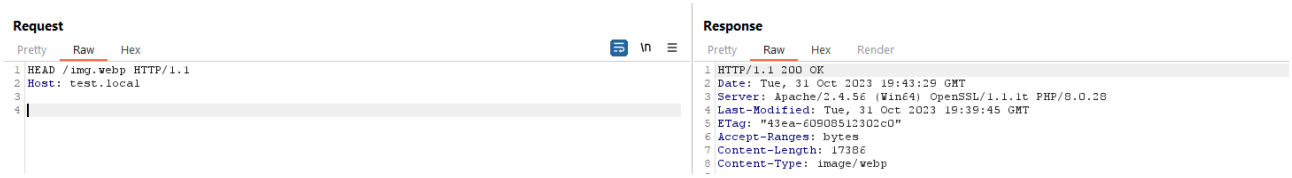
بر اساس استانداردهایی که در RFC تعیین شده است دادههای انتقالی از طریق متد GET تنها میتوانند KB۲ یعنی ۲۰۴۸ کاراکتر باشند و خب این استاندارد است و ممکن است برخی از مرورگرها پیاده‌سازی نکرده باشند .

متد POST ؟ این متد برای انتقال داده از طریق Request استفاده می‌شود و هرگونه داده‌ای رو میشه از طریقش انتقال داد . درمثال زیر میبینید که name و phone با مقادیرشان به وب سرور انتقال داده شده‌اند . این مورد امن است و میتوان دادههای حساس رو از طریقش انتقال داد و دادهها در Body درخواست یعنی بعد از دوتا CRLF انتقال پیدا میکنند و در طول انتقال رمزنگاری می‌شوند . خلاصه اینکه حله حاجی استفاده کن مشکلی نیست . همچنین برخلاف GET که KB۲ داده بیشتر نمیشد از طریقش انتقال داد از طریق پست محدودیتی در حجم داده انتقالی وجود ندارد .

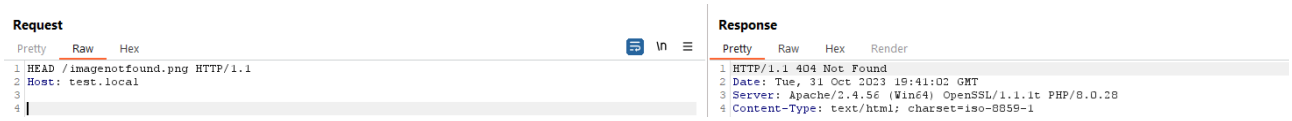
متد HEAD ؟ این متد به صورت پیش‌فرض توسط مرورگر ارسال میشه . در جواب این Request فقط و فقط Headers برگردانده می‌شود و هیچ داده‌ای انتقال نمیدهد و درجواب هم هیچ داده‌ای دریافت نمیشود .



درخواست HEAD خیلی سریعه چون که عملاً داده‌ای از طریق اون انتقال پیدا نمیکنه و گفتیم که به صورت پیش‌فرض توسط مرورگر استفاده میشه و ارسال میشه ولی چرا ؟ فرض کنید که مرورگر میخواهد یک درخواست GET برای یک فایل ارسال و آن را دریافت کند ولی مطمئن نیست که آیا این فایل وجود داره یا نداره و اگه هم GET بزنه ممکنه یک کم طول بکشه واسه همین میاد و یک HEAD میفرسته و اگه جواب ۲۰۰ بود یعنی OK هست و وجود داره ولی اگه ۴۰۴ بود دیگه Request رو ارسال نمیکنه چرا که Not Found است :



یعنی فایل وجود داره و بعد سریعاً GET میزنه و اگر وجود نداشته باشه :



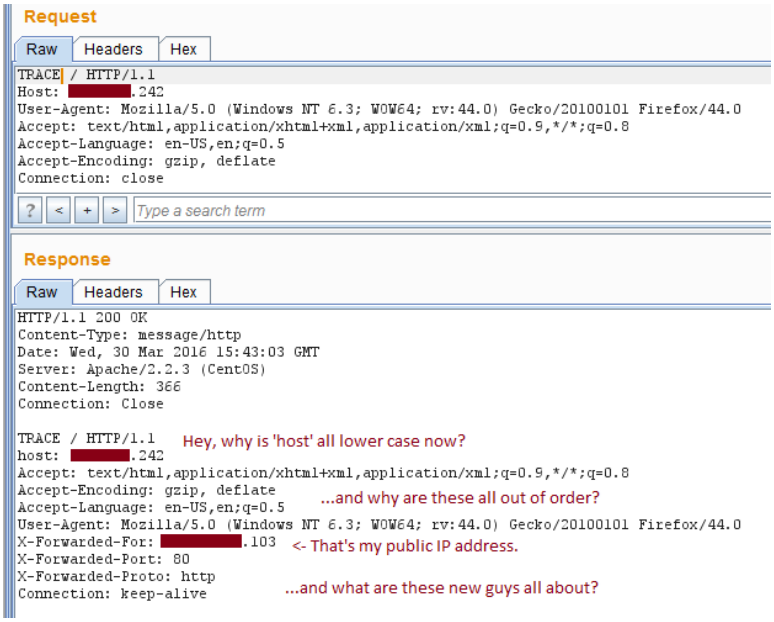
و دیگه زحمت GET زدن رو به خودش نمیده . این درخواست توسط BurpSuite کپچر نمیشه ولی خب اگه با Wireshark ترافیک رو Capture کنید اون رو میبینید .

متد OPTIONS ؟ این متد درصورتی که جواب بدهد یک مولفه هدر به نام Allow رو همراه مولفه های دیگه برمیگردونه که نشون میده اون Path که شما Request رو بهش فرستادید چه Method هایی رو قبول میکنه چون ممکنه روی یک Path چند متد بتونه کار کنه مثلاً GET رو بتونید بزنید و POST رو هم بتونید . این متد ممکنه که به شما Status Code شماره ۲۰۰ رو بده ولی مولفه Allow رو برنگردونه . به مثال زیر توجه کنید که مولفه Allow چه مقداری داره :

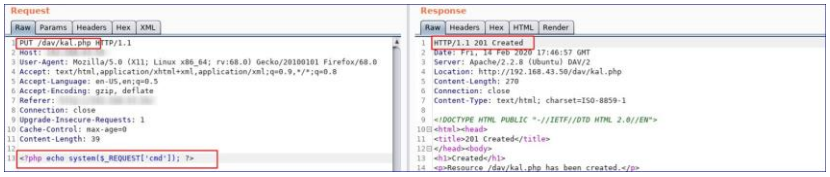
```
vagrant@kali:~$ telnet example.org 80
Trying 93.184.216.34...
Connected to example.org.
Escape character is '^]'.
OPTIONS / HTTP/1.1
Host: example.org

HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Tue, 31 Oct 2023 20:26:21 GMT
Expires: Tue, 07 Nov 2023 20:26:21 GMT
Server: EOS (vny/044F)
Content-Length: 0
```

مقدار ALLOW نشون میده که ما میتونیم بر روی این Path از example.org پورت ۸۰ درخواست های متدهای GET, HEAD, POST رو بفرستیم و وب سرور به ما جواب خواهد داد . البته روی یک وب سرور متدهای GET, HEAD, POST حتماً فعال است وگرنه اون وب سرور به مشکل میخوره ولی یکی از متد های خطرناک که در آینده بررسی میکنیم متد PUT است که فعال بودن اون میتونه به مهاجم امکان اپلود Shell و دسترسی بده .

متد TRACE ؟ این متد برخی اوقات بر روی وب سرور ها فعال است (بسیاری اوقات هم Not Allowed است) و Request ها رو هم با Status Code شماره ۲۰۰ پاسخ میده ولی خب بیشتر اوقات درست جواب نمیده و صرفاً یک OK میفرسته . اما این متد برای Debugging کاربرد داره . فرض کنید که از ارتباط کلاینت تا وب سرور یک سری تجهیزات دیگه ای مثل Reverse Proxy, WAF و ... دارید و هرکدوم از این تجهیزات ممکن که چیزهایی رو روی پکت اعمال کنند مثلاً WAF مولفه X-Forwarded-For رو به هدر اضافه میکنه و .... Request هایی که با متد TRACE ارسال میشن در نهایت درصورتی که درست جواب بدهد اون پکتی که از همه تجهیزات گذر کرده و به وب سرور رسیده رو به شما نشون میده . تصویر زیر مثالی از پاسخ درست یک وب سرور به متد TRACE است :



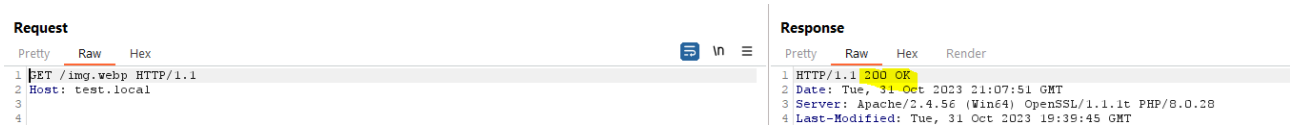
متد PUT ؟ این متد برای اپلود یک فایل بر روی سرور استفاده می‌شود و خب متد خطرناکیه و معمولاً Not Allowed است ولی خب چند مدت پیش یک نفر تونسته بود از طریق همین متد یک شل بر روی سرور یکی از شرکتهای بزرگ ماشین سازی اپلود کنه . اغلب اوقات لازمش این هست که webdav فعال باشه . درایندە مثال این متد رو به خوبی بررسی میکنیم . مثال زیر نمونه ارسال یک Request با متد PUT است :



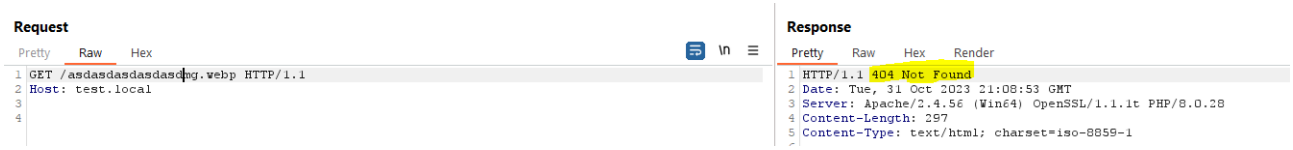
متد DELETE ؟ این متد هم مثل متد PUT یک متد نسبتاً خطر ناک محسوب میشه چون اگه Not Allowed نباشه و یک نفر بتونه ازش استفاده کنه میتونه فایل رو از روی Server پاک کنه و خب به همین علت عموما Not Allowed است . لازمه این متد هم مثل متد PUT فعال بودن WebDav است .

نکته‌ای که باید درمورد متد های PUT و DELETE در نظر بگیرید این است که در سرویس های API این دو متد رو خواهید دید و اونها با این چیزی که اینجا توضیح دادم یکی نیستن و کاربردشون متفاوت است .

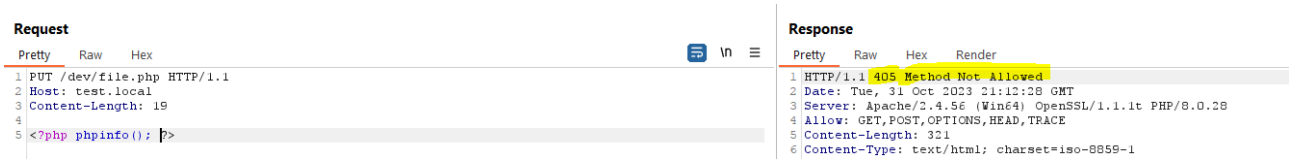
HTTP Status Code ها ؟ وب سرور در جواب هر Request با هر متدی یک Code به نام Status Code را بر میگرداند که در اولین خط از Response اون رو میتونید بعد از HTTP/[Version] ببینید . مثلاً در تصویر زیر یک نمونه رو مشاهده میکنید :



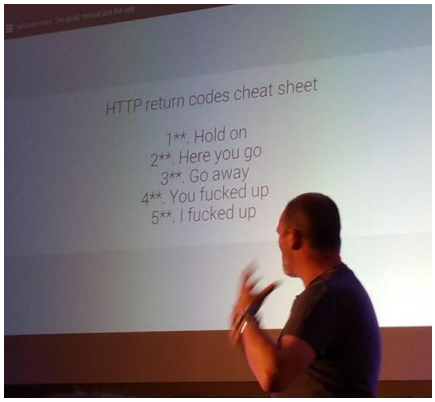
کد ۲۰۰ یعنی همه چی اوکیه و جواب درست توسط وب سرور برای کلاینت ارسال شده است و مثال پایین هم کد ۴۰۴ به معنی پیدا نشدن منبع درخواستی کلاینت رو میبینید :



هر جا ۴۰۴ رو دیدی یعنی چیزی که درخواست کردید وجود نداره و یا Path رو اشتباه نوشتید و ۴۰۵ هم یعنی Method Not Allowed و خب یعنی متدی که توی Request زدید مجاز نیست :



تعداد این Status Code ها زیاده ولی خب یک راه اسون تری هم هست که در تصویر زیر میبینید :



تو عکس میگه که ۵ نوع Status Code داریم که به شرح زیرند :

۱. اونایی که با ۱ شروع میشن و معنیشون این هست که صبر کن مثل ۱۰۱ که یعنی صبر کن دارم Protcol رو Redirect میکنم .
۲. اونایی که با ۲ شروع میشن یعنی همه چی اوکیه مثل ۲۰۰و ۲۰۱ و . . . .

۳. اونایی که با ۳ شروع میشن یعنی برو گمشو (البته چیزیه که تو عکس نوشته) مثل ۳۰۱, ۳۰۲, ... که درواقع کاربر رو به جایی دیگه Redirect میکنن .
۴. اونایی که با ۴ شروع میشن یعنی کاربر یه جایی خراب کرده، مثل ۴۰۰ یعنی Bad Request و ۴۰۳ یعنی Forbidden و ...
۵. اونایی که با ۵ شروع میشن یعنی سرور یه جایی خراب کرده، مثل ۵۰۰ که یعنی Internal Error و ۵۰۲ یعنی Bad Gateway و ...

لیست کامل کدهایی که هست رو میتونید از لینک زیر ببینید :
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
البته نیازی نیست که اینا رو حفظ کنید چون وقتی سرور یک Response رو با یک Status Code بر میگیردونه خودش کنار Code مینویسه که معنیش چیه .

خب فعلاً اینجا HTTP رو تموم میکنیم، البته نه اینکه دیگه بحثی نداریم و درواقع تمام بحثمون جناب‌عالی جناب HTTP هست و باید یادش بگیریم . میریم یه کم HTTP رو داخل PHP ببینیم و کدای PHP رو یه بررسی کنیم .

Superglobals ؟ در PHP متغیر هایی وجود دارد که به صورت Built-In هستند و در هر جایی و در هر Scope از کدهای PHP قابل دسترس هستند همه اونها با _$ شروع می‌شوند و بهشون Superglobals میگن . این متغیر ها به عبارت زیر هستند :

۱. $_GET
۲. $_POST
۳. $_REQUEST
۴. $_SERVER
۵. $_FILES
۶. $_SESSION
۷. $_COOKIE
۸. $_ENV
۹. $GLOBALS

خب حالا اینا چه کاربردی دارن ؟ ما با همه این‌ها سروکار نداریم فعلاً و خب همشون اطلاعاتی رو توی خودشون ذخیره میکنن، در این زمان و در این مکان ما فقط با POST, GET, REQUEST از لیست بالا کار داریم .

$_GET ؟ درمورد متد GET از متدهای HTTP صحبت کردیم و گفتیم که این متد دادههایی رو انتقال میده و یا درخواست یک Resource رو از Server میکنه . دادههایی را که از طریق متد GET انتقال پیدا میکنن به واسه $_GET قابل دسرس هستند . شما زمانی که داده‌ها رو از طریق URL انتقال میدید در کد PHP خودتون، هرجایی که خواستید به اونها دسترسی داشته باشید کافیه که از $_GET استفاده کنید . فرض کنید یک URL به شکل زیر داریم و میخوایم دوتا مقدار با نامهای id و name رو از طریقش انتقال بدیم و خب به شکل زیر این کار رو میکنیم (مقدار id رو ۱۱ و مقدار name رو John قرار داده ام) :

اگر بخوام در کد PHP فایل data.php که این دادهها از طریق GET به او منتقل میشه، $_GET رو از طریق تابع var_dump چاپ کنم به شکل زیر عمل میکنم :

data.php

```php
<?php
    var_dump($_GET);

?>
```

حال اگر بخوایم نتیجه رو ببینیم کافیه که URL تصویر بالاتر رو بارگذاری کنیم :

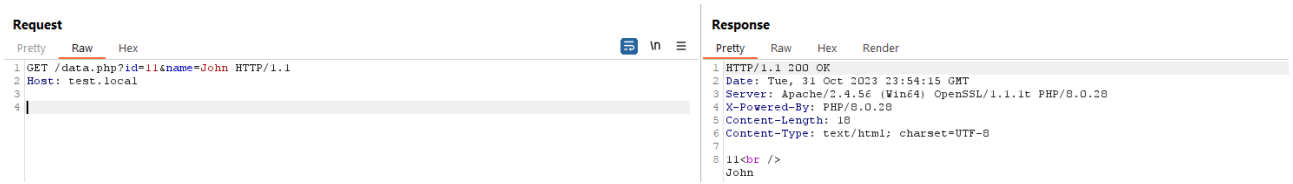
میبینید که یک ارایه برای ما نمایش داده که دو عضو دارد به نامهای id و name و مقادیری در آن‌ها وجود دارد که به ترتیب ۱۱ و John می‌باشد . همان مقادیری که از طریق URL به فایل data.php انتقال دادیم . جالبه ! حالا اگر بخوایم به این دو مقدار دسترسی پیدا کنیم قبلاً گفتم که چطوری به index های یک ارایه دسترسی پیدا کنیم، عمل میکنیم :

data.php

```php
<?php
    echo $_GET['id'];
    echo "<br />";
    echo $_GET['name'];
?>
```

کافیه که در [''] نام پارامتر هایی که از طریق URL فرستادیم رو بنویسیم (اون echo کردن
 هم واسه ایجاد یک خط جدید ما بین دو echo هست) و نتیجه به شکل زیر است :

حال همین مورد رو توی یک Request خام HTTP به شکل زیر میتونیم ارسال کنیم :



$_POST ؟ خب متغیر قبلی دادههای انتقالی از متد GET رو برای ما نشون میداد و این متغیر دادههای انتقالی از متد POST رو برای ما نشون میده به همین سادگی . خب فرض کنید من یک Request با متد POST به شکل زیر به فایل data.php ارسال میکنم :

Request

Pretty Raw Hex \n

```
1 POST /data.php HTTP/1.1
2 Host: test.local
3 Content-Type: application/x-www-form-urlencoded
4 Content-Length: 15
5 Connection: close
6
7 id=11&name=John
8
```

سورس کد فایل data.php به شکل زیر است :

data.php

```
1  <?php
2      echo $_POST['id'];
3      echo "<br />";
4      echo $_POST['name'];
5
6  ?>
```

یعنی بهش گفتم که از Request با متد POST یک مقداری با نام id دریافت میکنی و اون رو echo کن و بعد یک
 رو echo کن که یعنی برو خط بعد و بعد هم از Request که با متد POST دریافت کردی یک مقدار با نام name هم دریافت میکنی و مقدار اون رو echo کن و نتیجه به شکل زیر خواهد بود :

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Date: Wed, 01 Nov 2023 00:01:18 GMT
3 Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.0.28
4 X-Powered-By: PHP/8.0.28
5 Content-Length: 18
6 Connection: close
7 Content-Type: text/html; charset=UTF-8
8
9 11<br />
  John
```

دادهها در Request با متد POST از طریق Body انتقال پیدا میکنند و امن تر هستند .

$_REQUEST ؟ اگر خواستیم بگی که مهم نیست متد REQUEST ما POST یا GET هست ولی تو از هر کدوم که تونستی مقادیرش رو بخون از این متغیر استفاده میکنیم . یه حالت کلی و جمع $_GET و $_POST هست . استفاده ازش هم به شکل همین دوتاست و فرقی نداره . مثال نمیزنم اگه دوست داشتی خودت مثال بزن .