What is an HTTP Host header attack?

Attacks that involve injecting a payload directly into the Host header.

Web applications often generate an absolute URL (e.g., included in an email) using the value of Host header:

<a href="https://\_SERVER['HOST']/support">Contact support</a>

Supply an arbitrary Host header

Modify the Host header (to different domains, Burp Colaborator), check if it still reaches the target or gets reflected.

Check for flawed validation

Some websites omit the port from the Host header, meaning that only the domain name is validated => inject a payload via the port

Host: vulnerable-website.com:**bad-stuff-here**

Host: vulnerable-website.com:**443@collaborator-domain.net**

Try adding payload to the host:

Host: **bad-stuff-here**vulnerable-website.com

Host: **bad-stuff-here**.vulnerable-website.com

Host: vulnerable-website.com#**bad-stuff-here**

Host: **bad-stuff-here**@vulnerable-website.com

Inject duplicate Host headers

Host: vulnerable-website.com

Host: bad-stuff-here

Try changing their order, adding spaces,…

GET /example HTTP/1.1

**Host: bad-stuff-here**

Host: vulnerable-website.com

Supply an absolute URL

GET **https://vulnerable-website.com/** HTTP/1.1

Host: bad-stuff-here

Try using http instead of https

Inject host override headers

**X-Forwarded-Host** header contains the original value of the Host header from the client's initial request. Use this header to inject payload

GET /example HTTP/1.1

Host: vulnerable-website.com

**X-Forwarded-Host: bad-stuff-here**

Other headers for the same purpose:

* X-Host
* X-Forwarded-Server
* X-HTTP-Host-Override
* Forwarded

Use Param Miner to check for not-used, but supported headers.

Password reset poisoning

Attacker manipulates a vulnerable website into generating a password reset link pointing to a domain under their control. Attacker can then steal the secret tokens for resetting the users' passwords and, ultimately, compromise their accounts.

Try using **Dangling markup attack** to send the temporary password/token in victim’s E-mail to our exploit server.

### 

### Web cache poisoning via the Host header

Use the above-mentioned techniques to check if host header is reflected in the response markup.

### **Exploiting classic server-side vulnerabilities**

SQLi, SSRF,.. => scan Host header and the whole request

### **Accessing restricted functionality**

It is common for websites to restrict access to certain functionality to internal users only.

### Find **/robots.txt** (information disclosure)

Try **Host: localhost**

### **Accessing internal websites with virtual host brute-forcing**

Companies sometimes make the mistake of hosting publicly accessible websites and private, internal sites on the same server. Attacker can typically access any virtual host on any server that they have access to (**from outside**), provided they know the hostnames.

### **Routing-based SSRF**

Supply the domain of Collaborator server in the Host header. If we subsequently receive a DNS lookup from the target server, this indicates that we may be able to route requests to arbitrary domains. We can exploit this behavior to access internal-only systems.

Use Burp Intruder to scan internal **Host: 192.168.0.§0§** (disable “Update Host header to match target”)

### **Connection state attacks**

The first request may be strictly validated. The following may be not

-> create a group of one good (**Connection: keep-alive**) and one bad request

-> **Send group (single connection)**

### **SSRF via a malformed request line**

When a reverse proxy takes 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