SSRF

What is it and how do we abuse it?

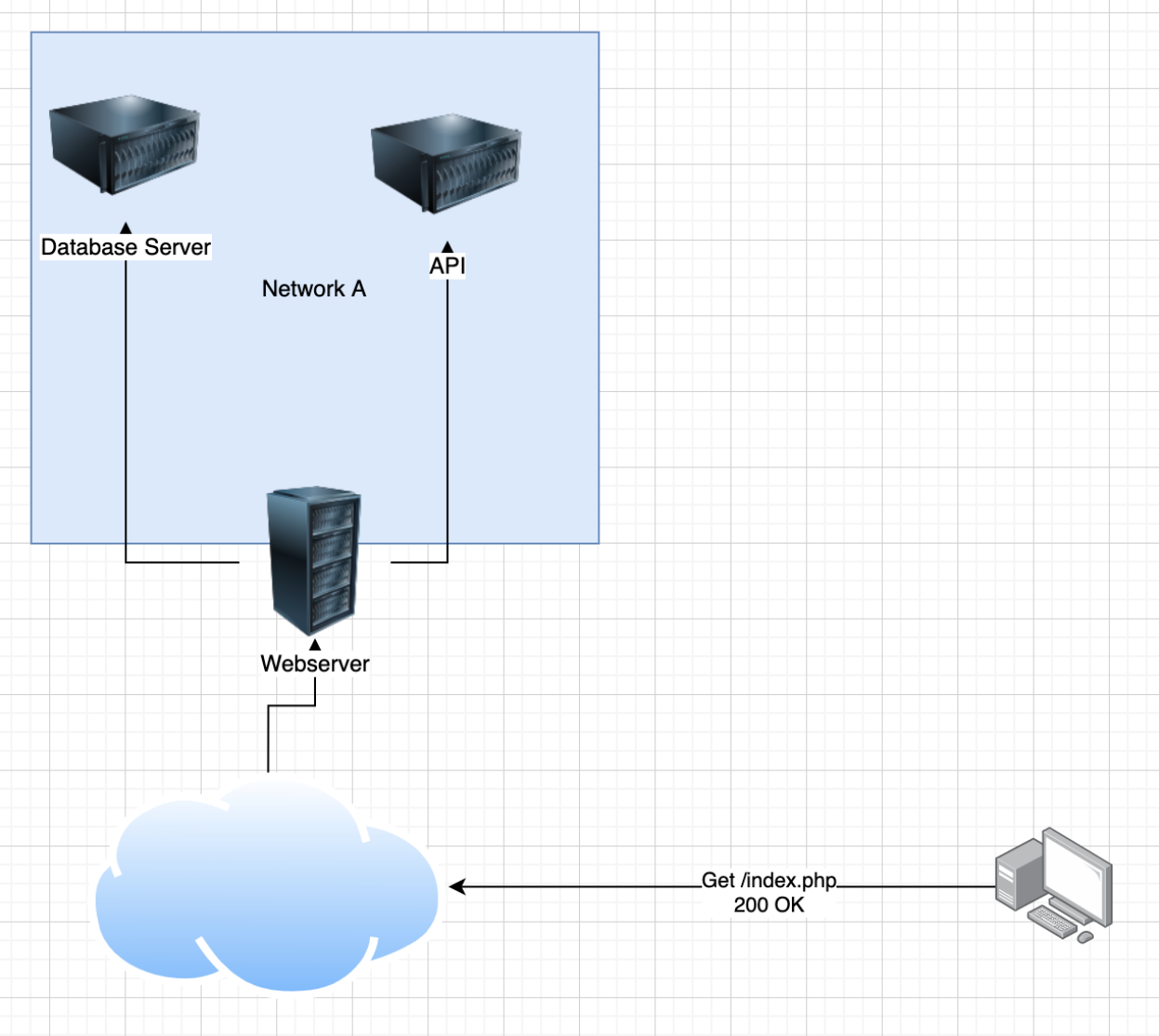
SSRF - Server Side Request Forgery

Server Side Request Forgery can occur in different forms. It occurs when the server makes an HTTP request to an URL which we can control. If we can make the server execute HTTP requests to an arbritrary webserver, we have an SSRF request on our hands.

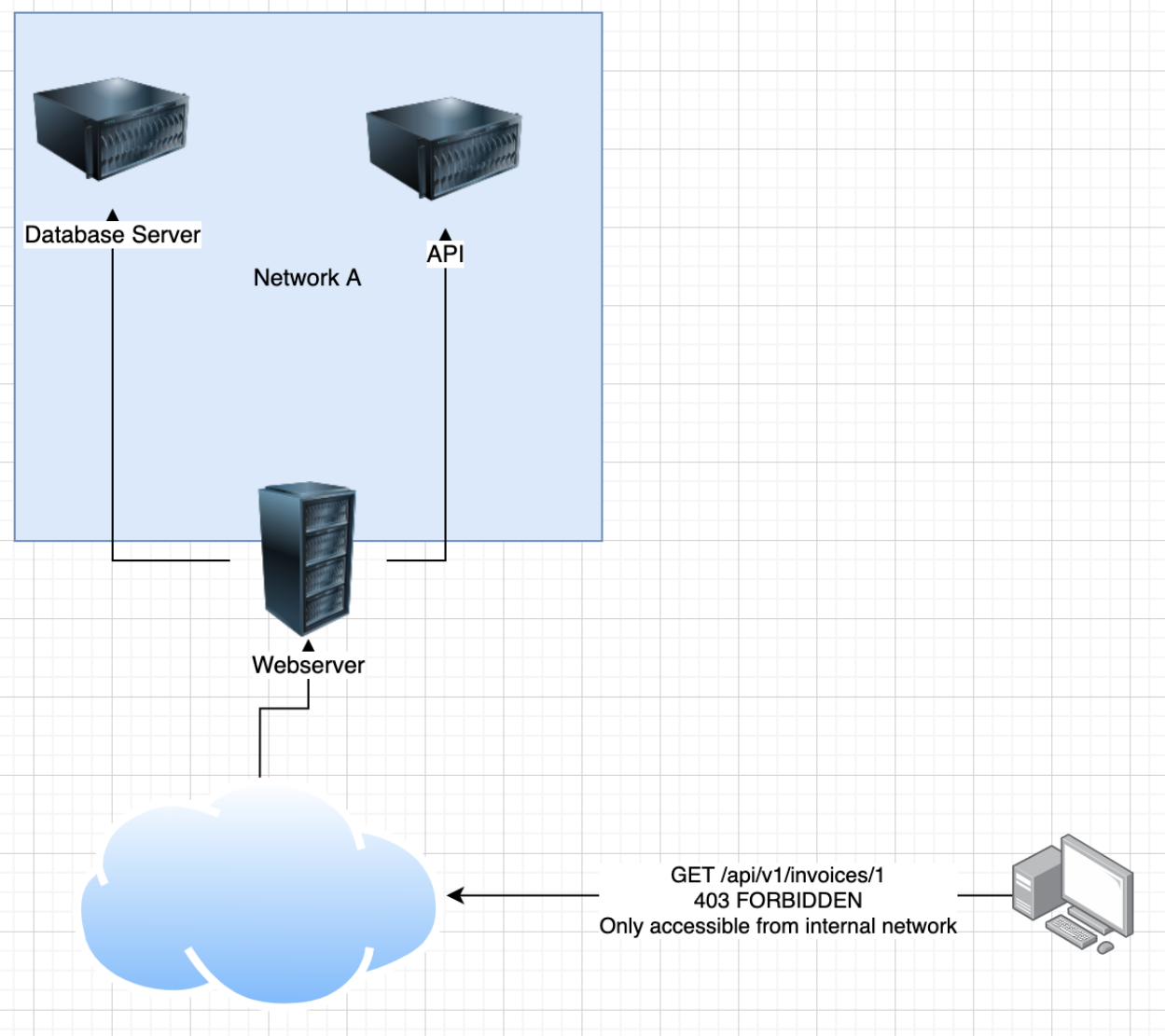
Usually we will abuse this vulnerability as bug bounty hunters by making the server execute a request to itself on a port which only accepts requests from the internal network or to other servers who also only accept requests from the internal network.

Another way to abuse this vulnerability would be to make a request to a third party or external server that only accepts requests coming from our target.

Let's give you guys an example to make things more clear:

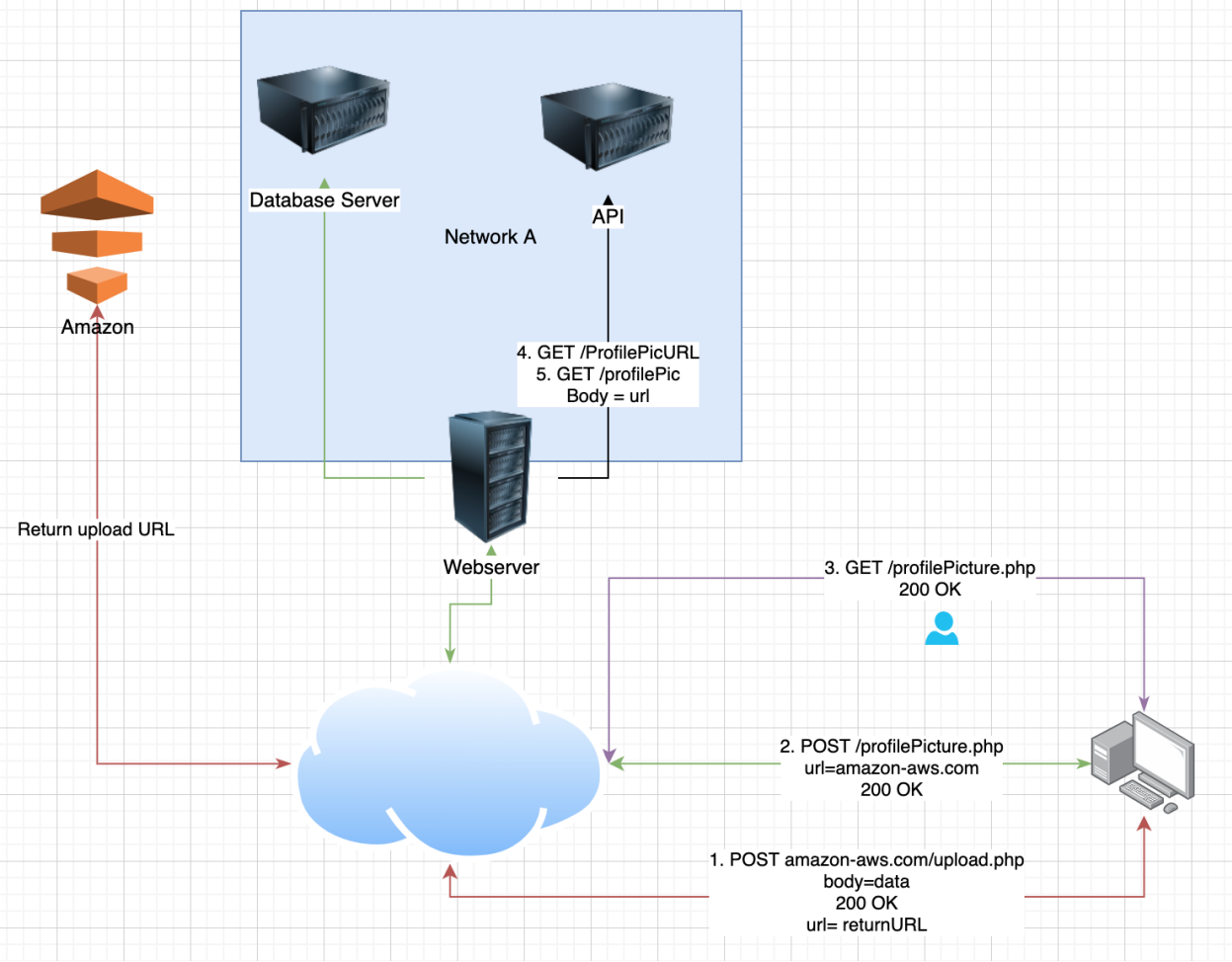


If the attacker tries to make a request to a webserver, it will return a 200 OK



However, if the attacker tries to request data straight from the API server, the get a 403 forbidden because it's only accesible from the internal network. This is expected behavior.

To illustrate how SSRF requests work, we will first discuss an imaginary software system for saving a profile picture.



1. You upload the profile picture to amazon AWS
2. You save the URL of the profile picture in the database
3. You get the profile picture
4. This triggers a request to the API which will first get the URL of the profile picture on amazon AWS
5. The server will now attempt to get the resource at the URL provided

We can abuse this in several ways if the developer was not careful when implementing this feature.

If we Intercept the request at 2. which saves the URL for the profile picture and instead of the amazon AWS URL, we enter a url to the database server, we should never get a response because the developers should make it so this feature only resolves requests to AWS but if the developers forgot, we have an SSRF on our hands.

Attack strategy

Basic SSRF strategy

We basically want to test any URL that gets resolved by the server and that we can control for SSRF vulnerabilities. First for the basic attacks and if those don't work, we can try to test for blind SSRF vulnerabilities but these will be much harder to exploit and often require RCE (=Remote Code Execution).

Some other harder to locate SSRF attack surface can be found at:

- Partial URLs in the body instead of a full URL
- URLs within data files such as XML files or CSV files (import functionality)
- The referer header can sometimes contain SSRF defects

SSRF against the server itself

Basic SSRF's against the server itself can arise when we want to attack a service that runs on the loopback ip adress but only accepts requests from within the server. For example if we have a webserver running on port 80 and an admin panel running on port 81, we might only be able to access the admin panel from the server itself.

- [ ] ADD TEST HERE
- [ ] ADD VM CHAPTER HERE

Example:

```
POST /profilePicURL.php
url=127.0.0.1:81/getUsers.php
-----------------------
GET /profilePic.php
200 OK
TestName1 TestName2 TestName1@test.test
TestName3 TestName4 TestName5@test.test
```

Additional practice can be found over at https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-localhost

SSRF against other backend systems

These Basic SSRF attacks can occur when we have a service running on a port that only resolves requests from the internal network such as an admin panel on port 3387 that can only be access from internal networks.

- [ ] ADD TEST HERE
- [ ] ADD VM CHAPTER HERE

Example:

```
POST /profilePicURL.php
url=192.168.32.123:3387/getUsers.php
-----------------------
GET /profilePic.php
200 OK
TestName1 TestName2 TestName1@test.test
TestName3 TestName4 TestName5@test.test
```

Additional practice can be found over at https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-backend-system

Blind SSRF

We need to have our burp collaborator open for this and use the URL provided or use the public burp collaborator URL but portswigger offers no availability warranty for this so beware.

Simply replace the attack vectors you would use for normal SSRFs with the burp collaborator URL. When you see a HTTP request come in, you might have an SSRF vulnerability but it will be hard to exploit since you might need to find an RCE for this. Exploiting blind SSRF is out of the scope of this document since uncle rat usually shy's away from this.

If you only receive a DNS request and no HTTP request, that means the HTTP requests are filtered when outgoing which makes it nearly impossible to exploit SSRF.