# Einführung in SSRF

Mögliche Angriffe und was man (trotz PHP) dagegen tun kann

*Malte Wessels*

# Über mich

- Doktorand @ TU Braunschweig

- Beisitzer @ Datenanfragen.de e.V.
- CTF @ CyberTaskForce Zero

IAS | INSTITUTE FOR APPLICATION SECURITY

TESTABLE

## TorchServe: Kombination aus drei Lücken

Eine weitere Lücke ermöglicht eine sogenannte Server-Side-Request-Forgery (SSRF), die üblicherweise Zugriff auf eigentlich in Netzwerk abgeschirmte Ressourcen zulässt (CVE-2023-43654, CVSS **9.8**, **kritisch**). Hierdurch können bösartige Akteure aus dem Netz Schadcode einschleusen und von beliebigen Domains Konfigurationen hochladen.

*https://www.heise.de/news/KI-Tool-Kritische-Sicherheitsluecken-in-TorchServe-9325638.html*

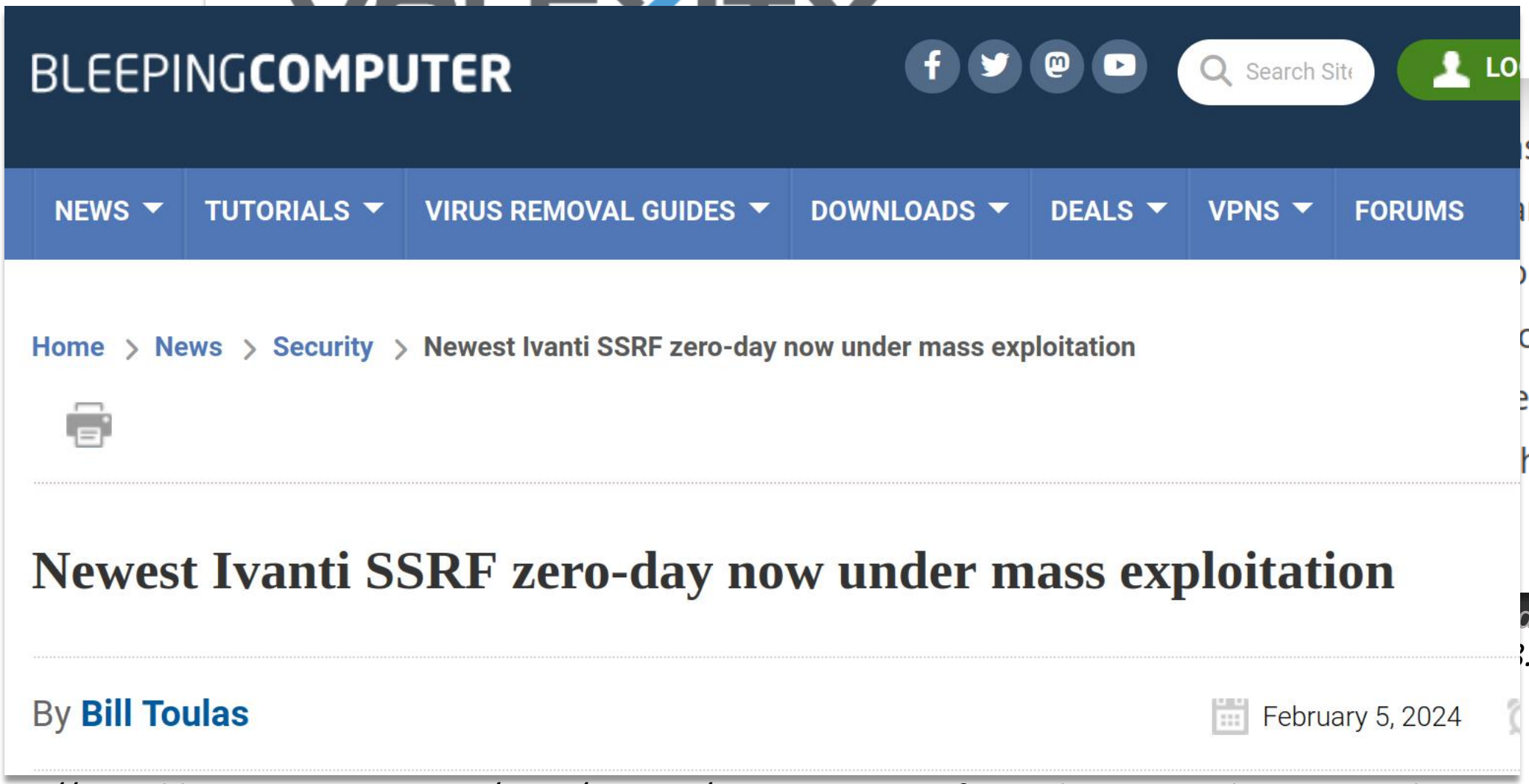
VOLEXITY

PRODUCTS SERVICES

Through its analysis of system memory, Volexity determined the attacker was exploiting a zero-day server-side request forgery (SSRF) vulnerability in Microsoft Exchange (CVE-2021-26855). The attacker was using the vulnerability to steal the full contents of several user mailboxes. This vulnerability is remotely exploitable and does not require authentication of any kind, nor does it require any special knowledge or access to a target environment. The attacker only needs to know the server running Exchange and the account from which they want to extract e-mail.

*https://www.volexity.com/blog/2021/03/02/active-exploitation-of-microsoft-exchange-zero-day-vulnerabilities/*

*https://www.heise.de/news/KI-Tool-Kritische-Sicherheitsluecken-in-TorchServe-9325638.html*




https://www.bleepingcomputer.com/news/security/newest-ivanti-ssrf-zero-day-now-under-mass-exploitation/

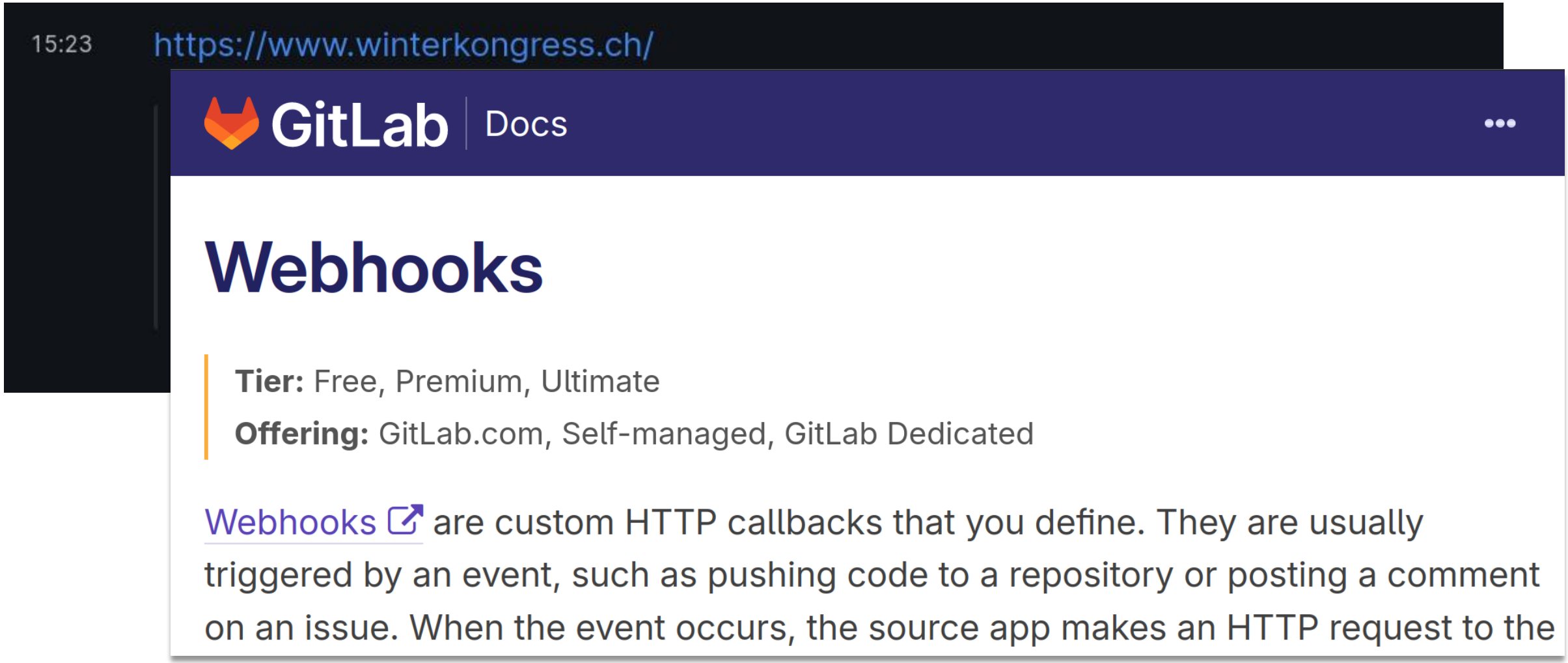
# SSRF: Server-Side Request Forgery

# SSRF: Server-Side Request Forgery
# SSR : Server-Side Request

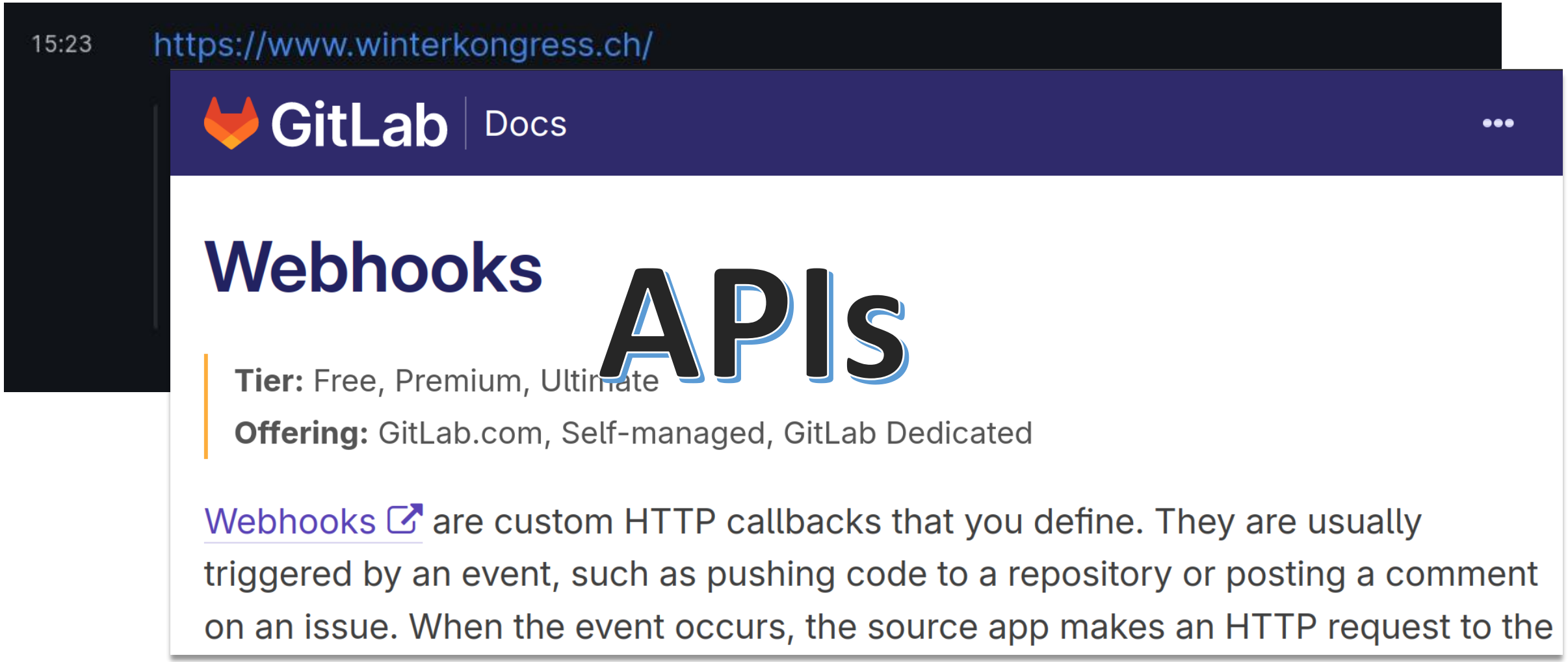
# SSRs in echt


15:23
https://www.winterkongress.ch/
Winterkongress 2024 - Winterkongress 2024
Winterkongress der Digitalen Gesellschaft vom 1./2. März 2024

# SSRs in echt


15:23
https://www.winterkongress.ch/
GitLab
Docs
Webhooks
Tier: Free, Premium, Ultimate
Offering: GitLab.com, Self-managed, GitLab Dedicated
Webhooks are custom HTTP callbacks that you define. They are usually triggered by an event, such as pushing code to a repository or posting a comment on an issue. When the event occurs, the source app makes an HTTP request to the

# SSRs in echt

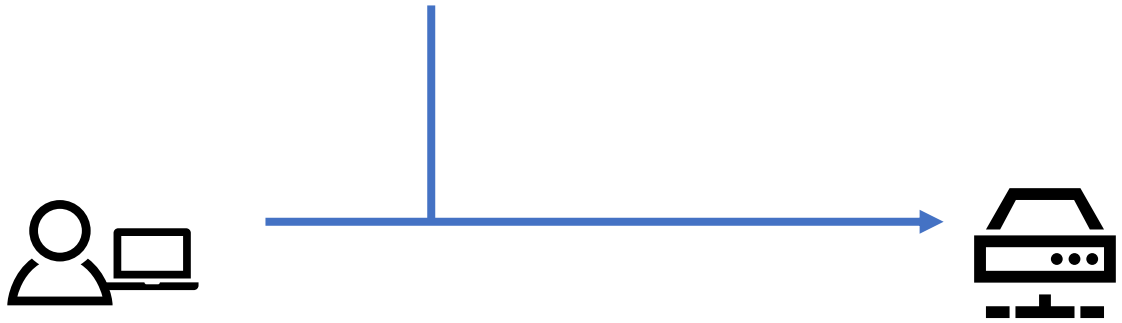

15:23
https://www.winterkongress.ch/
GitLab
Docs
Webhooks
APIs
Tier: Free, Premium, Ultimate
Offering: GitLab.com, Self-managed, GitLab Dedicated
Webhooks are custom HTTP callbacks that you define. They are usually triggered by an event, such as pushing code to a repository or posting a comment on an issue. When the event occurs, the source app makes an HTTP request to the

# Beispiel

- Wir zeigen Link auf unserer Seite (`example.org`) an
- Dafür eine Link-Preview
- Damit es einfach bleibt: nur den Titel
- https://winterkongress.ch → „Winterkongress 2024“

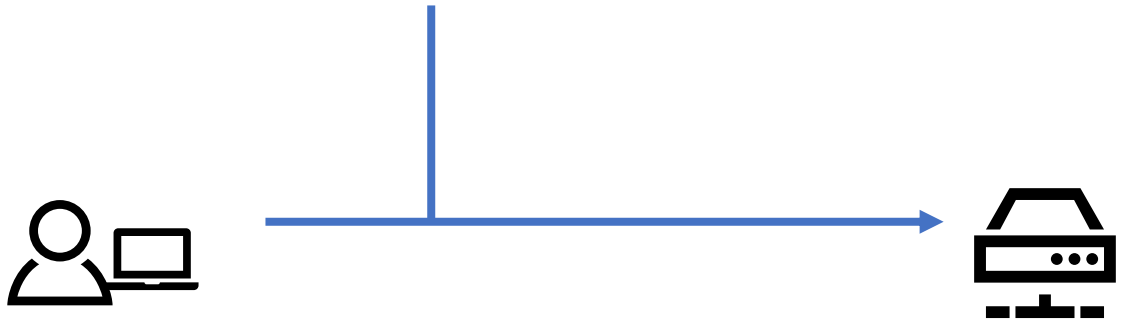
```php
<?php

$url = $_GET["url"];
```

https://example.org/?url=https://winterkongress.ch

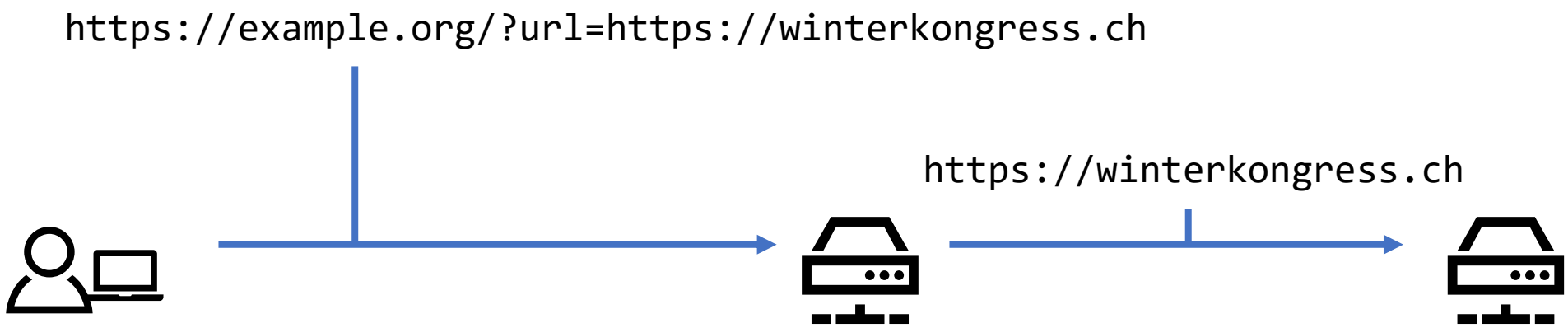


```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
```

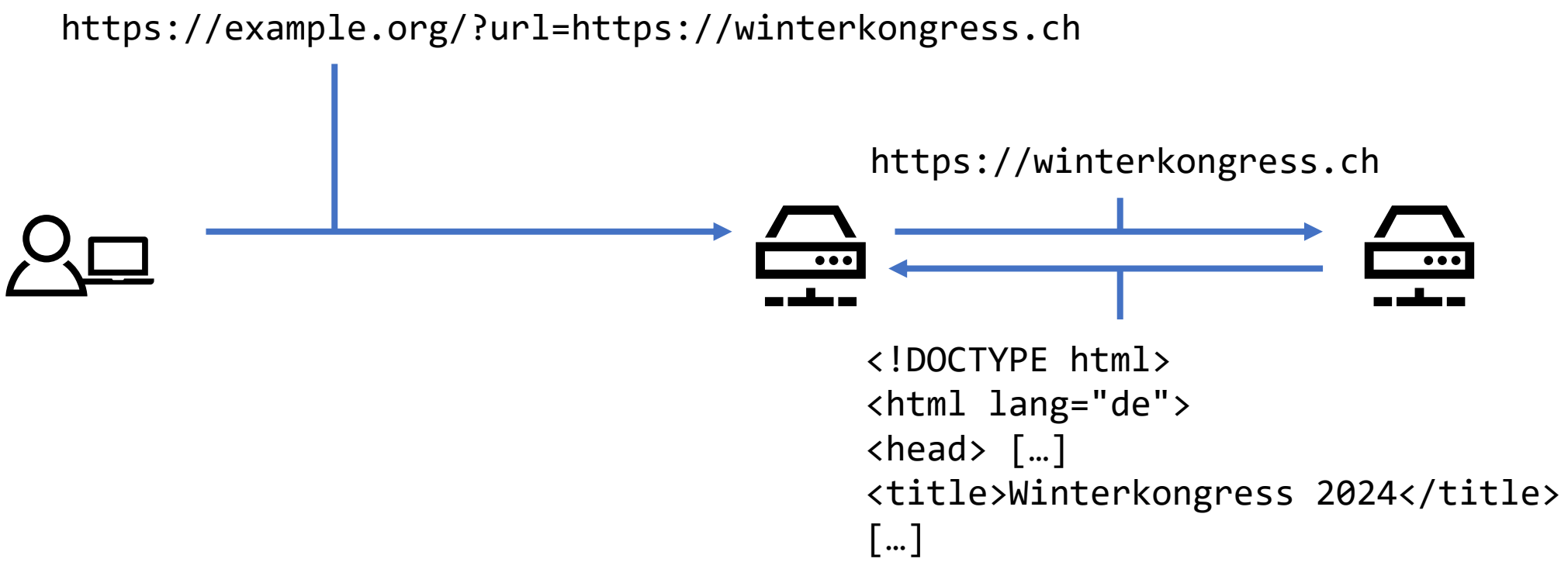
https://example.org/?url=https://winterkongress.ch



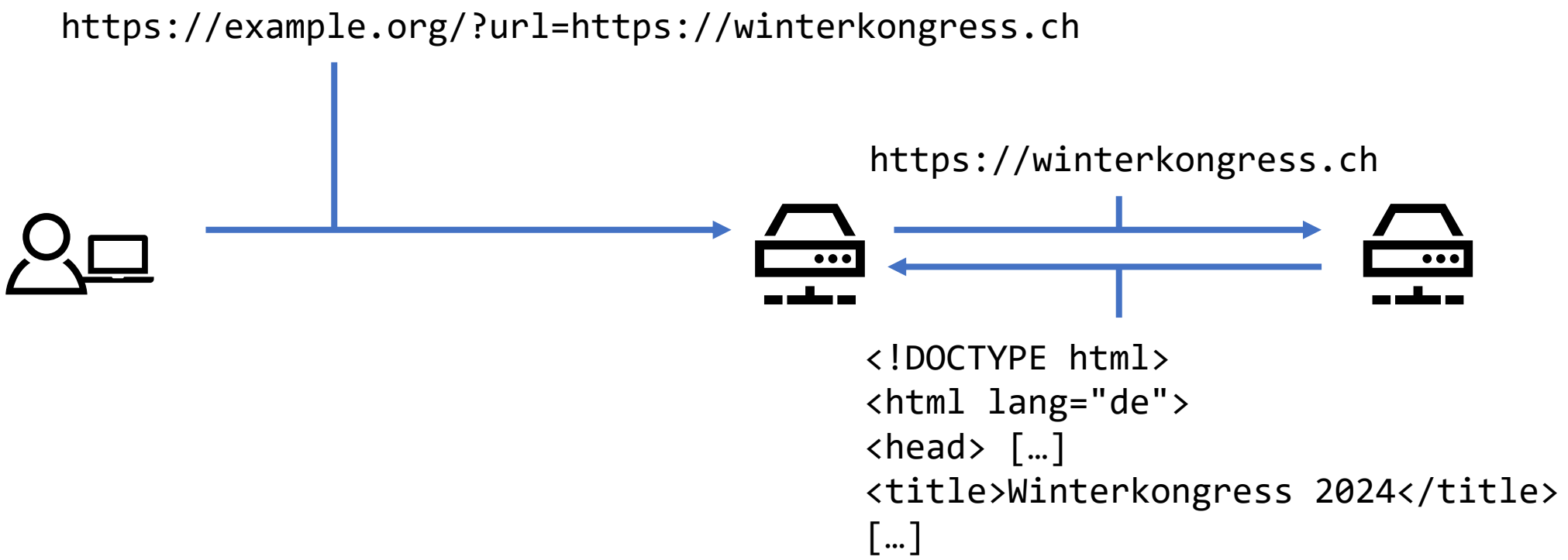
```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
```



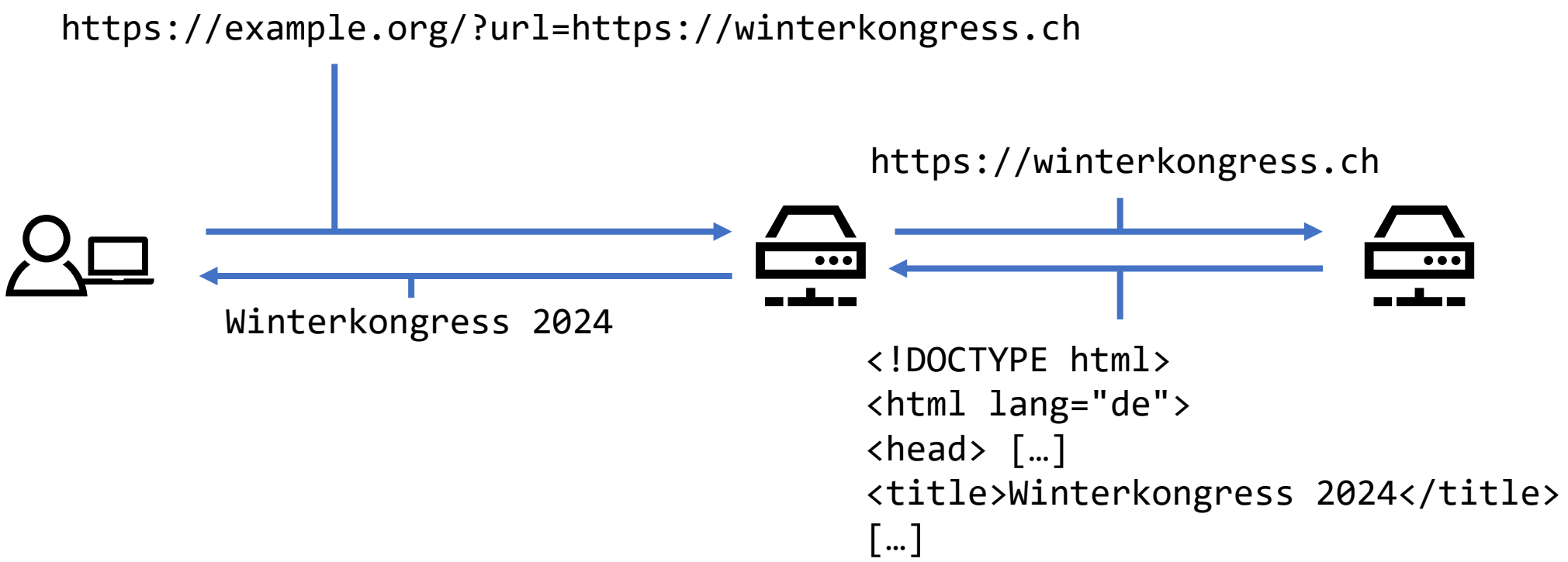
```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
```



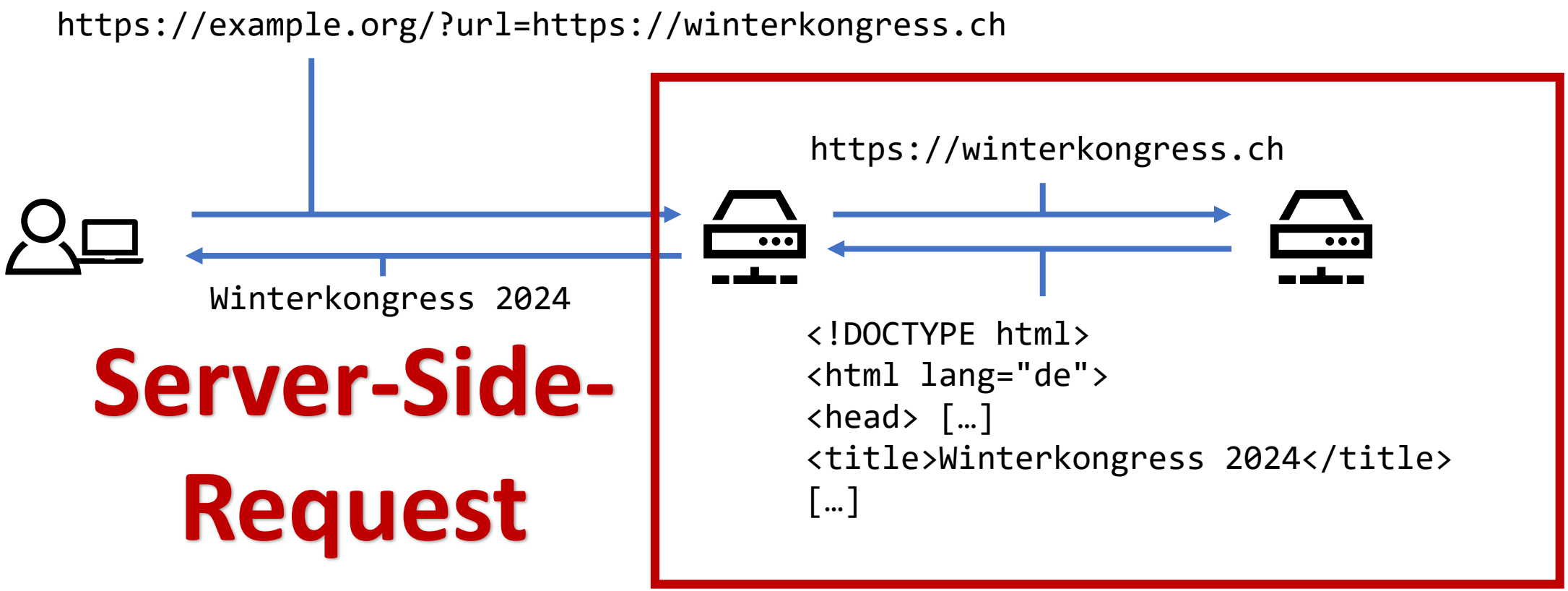
```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
preg_match("/\<title\>(.+)\<\/title\>/", $content, $matches);
```



```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
preg_match("/\<title\>(.+)\<\/title\>/", $content, $matches);
echo $matches[1];
```



```php
<?php

$url = $_GET["url"];
$content = file_get_contents($url);
preg_match("/\<title\>(.+)\<\/title\>/", $content, $matches);
echo $matches[1];
```



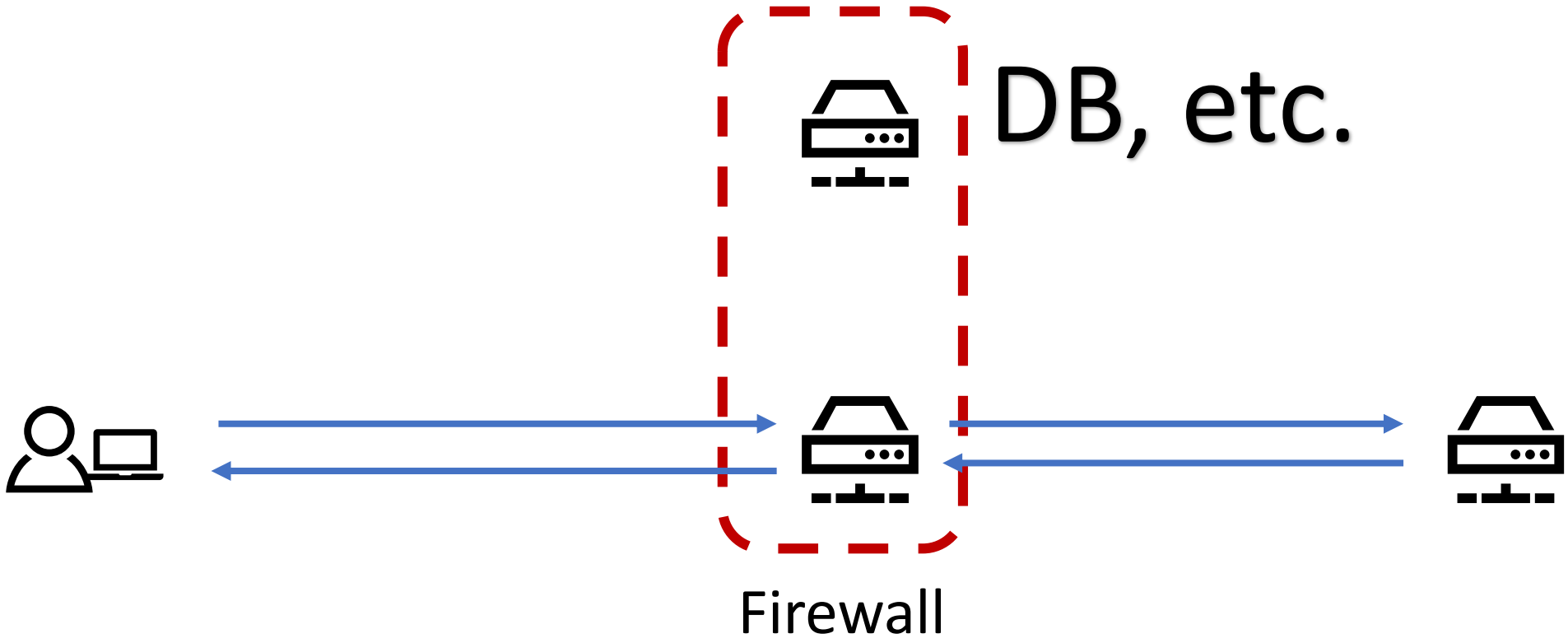
# Was kann da so schief gehen?

# Was kann da so schief gehen?

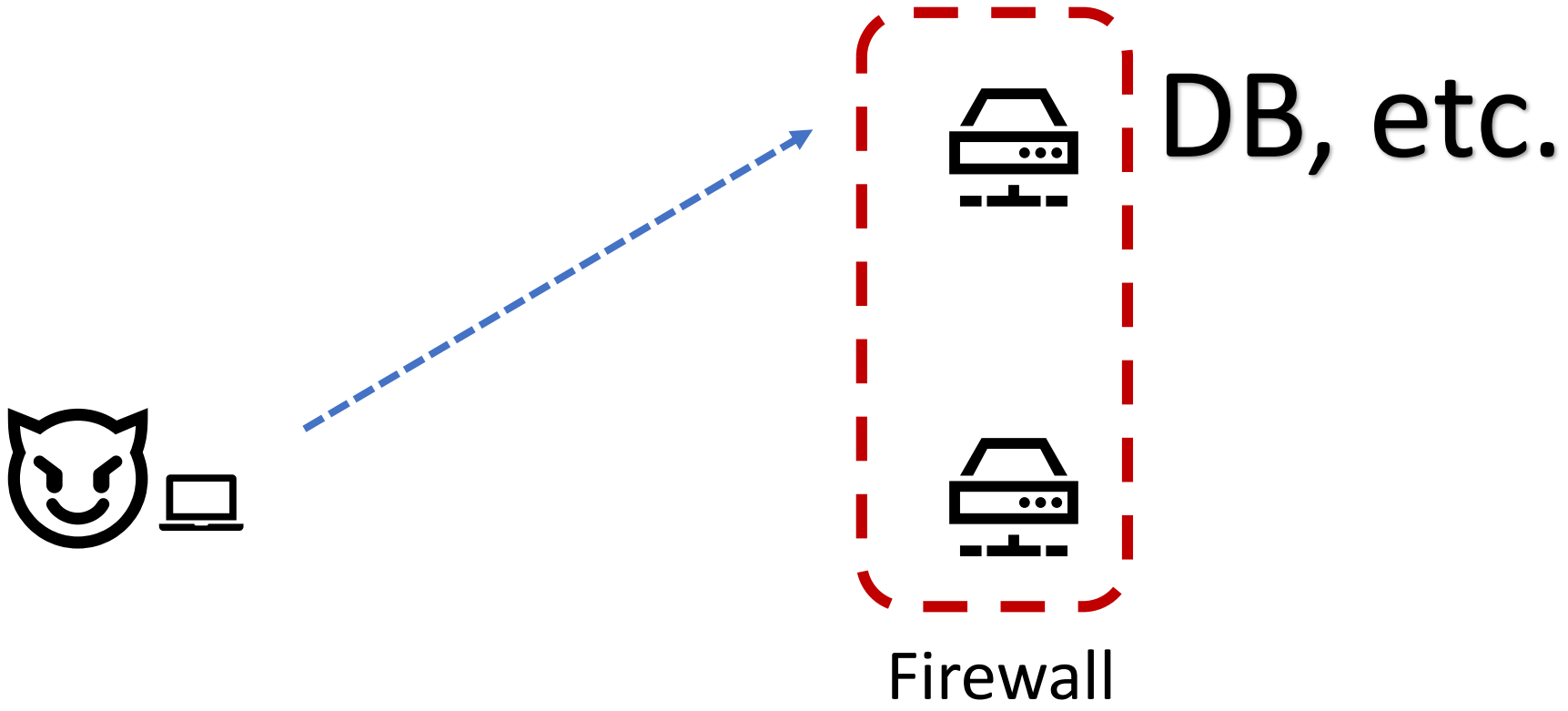
# Server-Side-Request-Forgery (SSRF)

> It allows an attacker to coerce the application to send a crafted request to an unexpected destination
>
> *https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/*

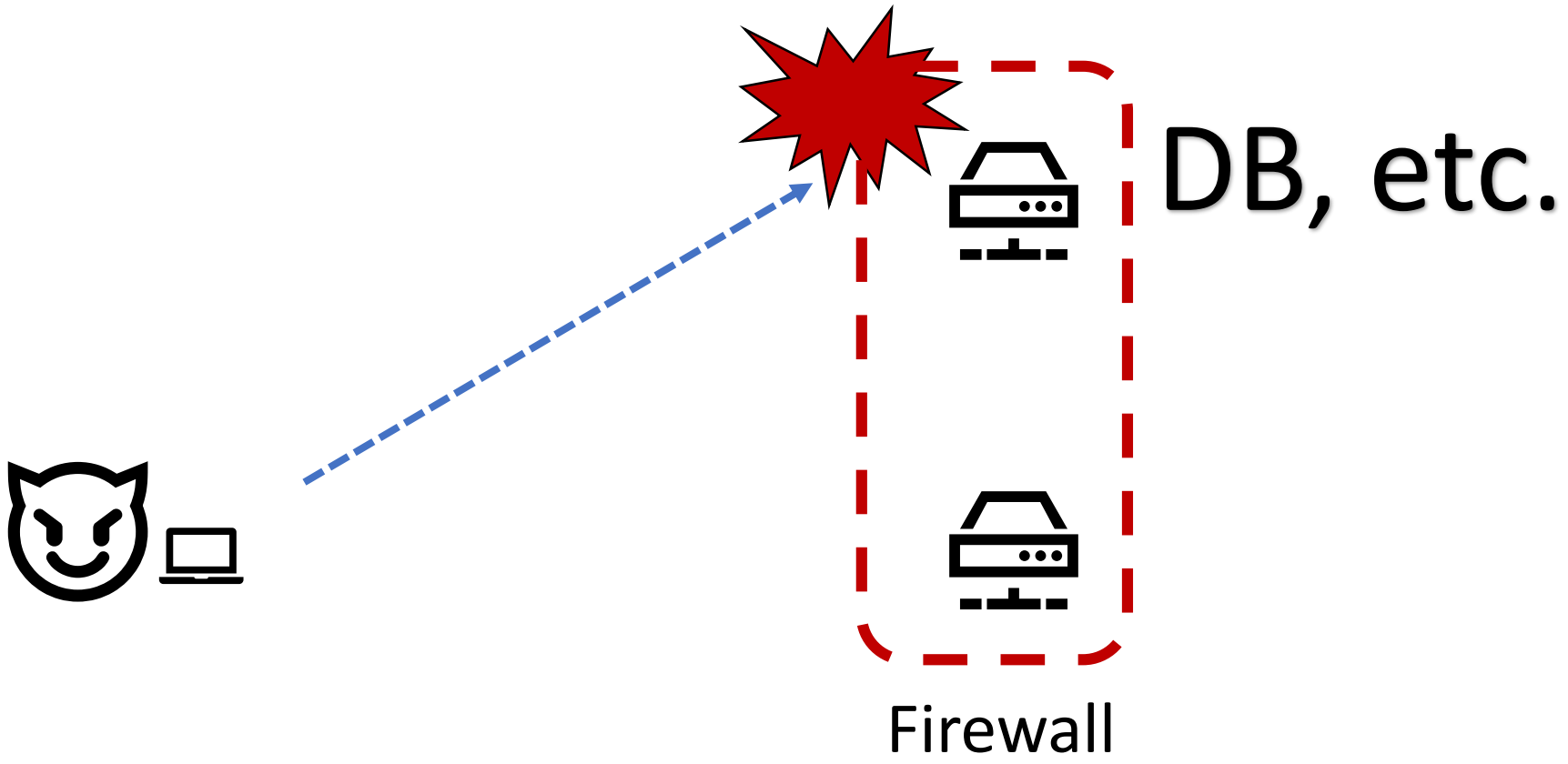
# Hinter der Firewall



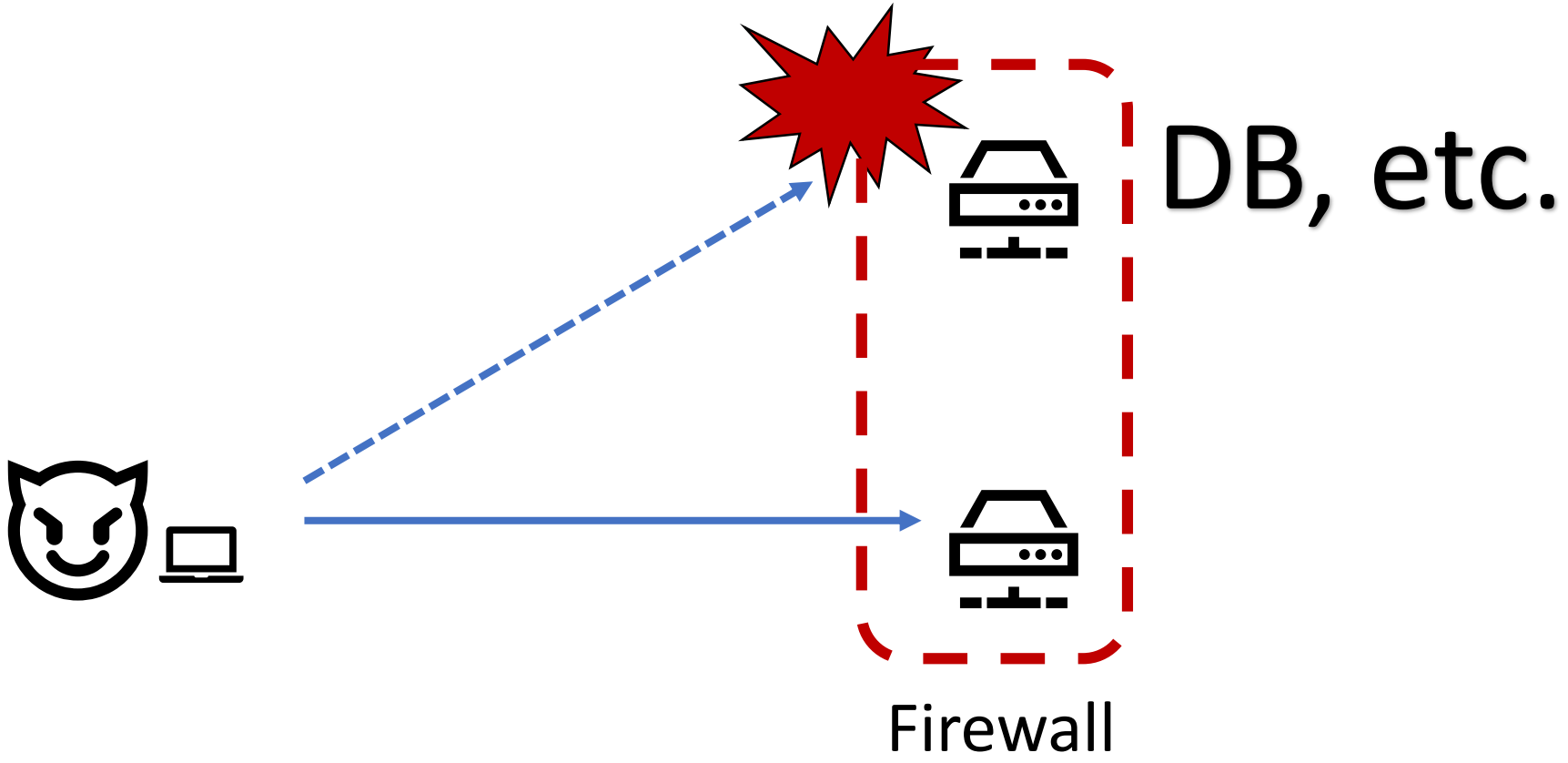
# Nicht alleine im Netzwerk?



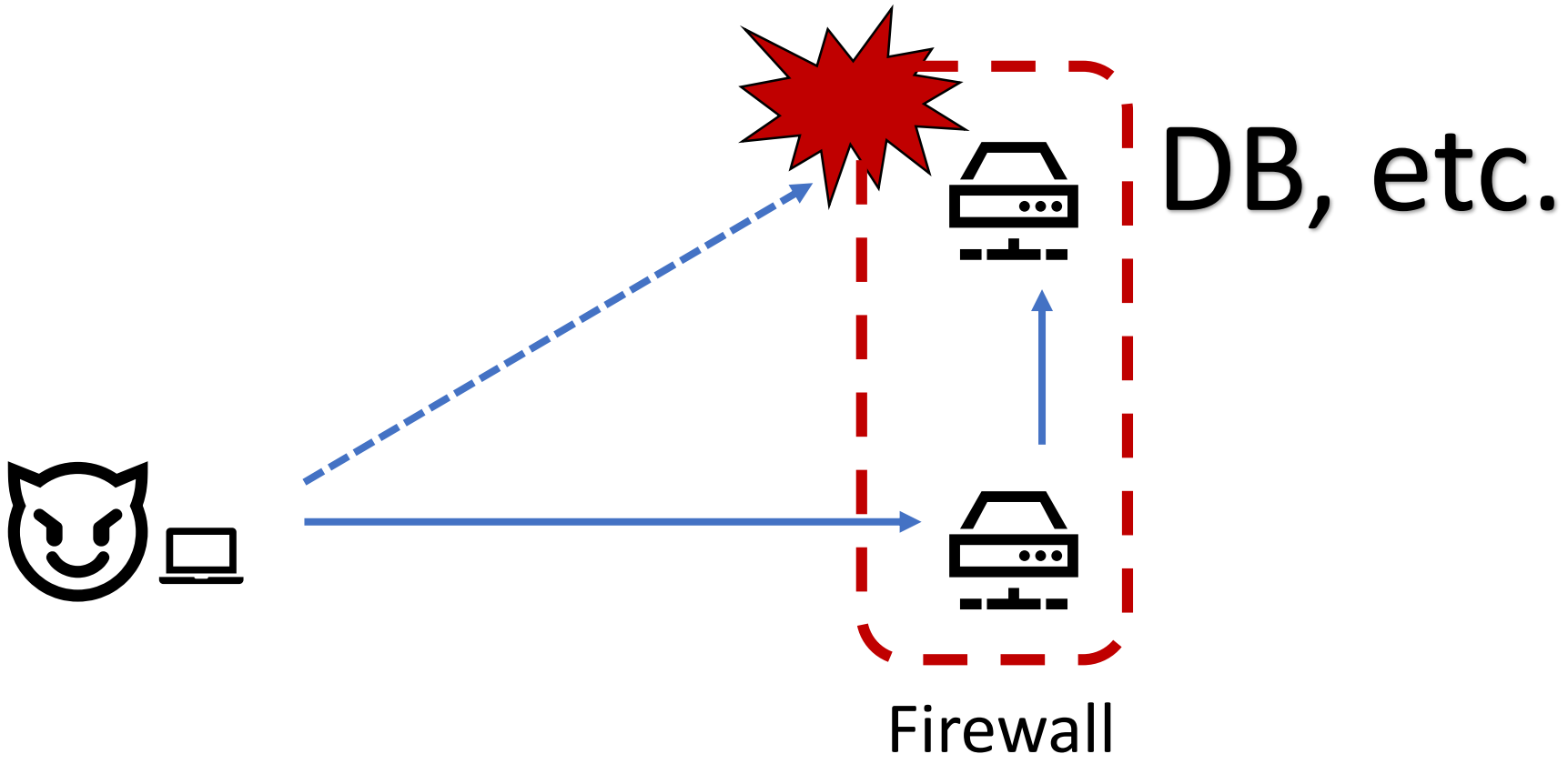
# Nicht alleine im Netzwerk?



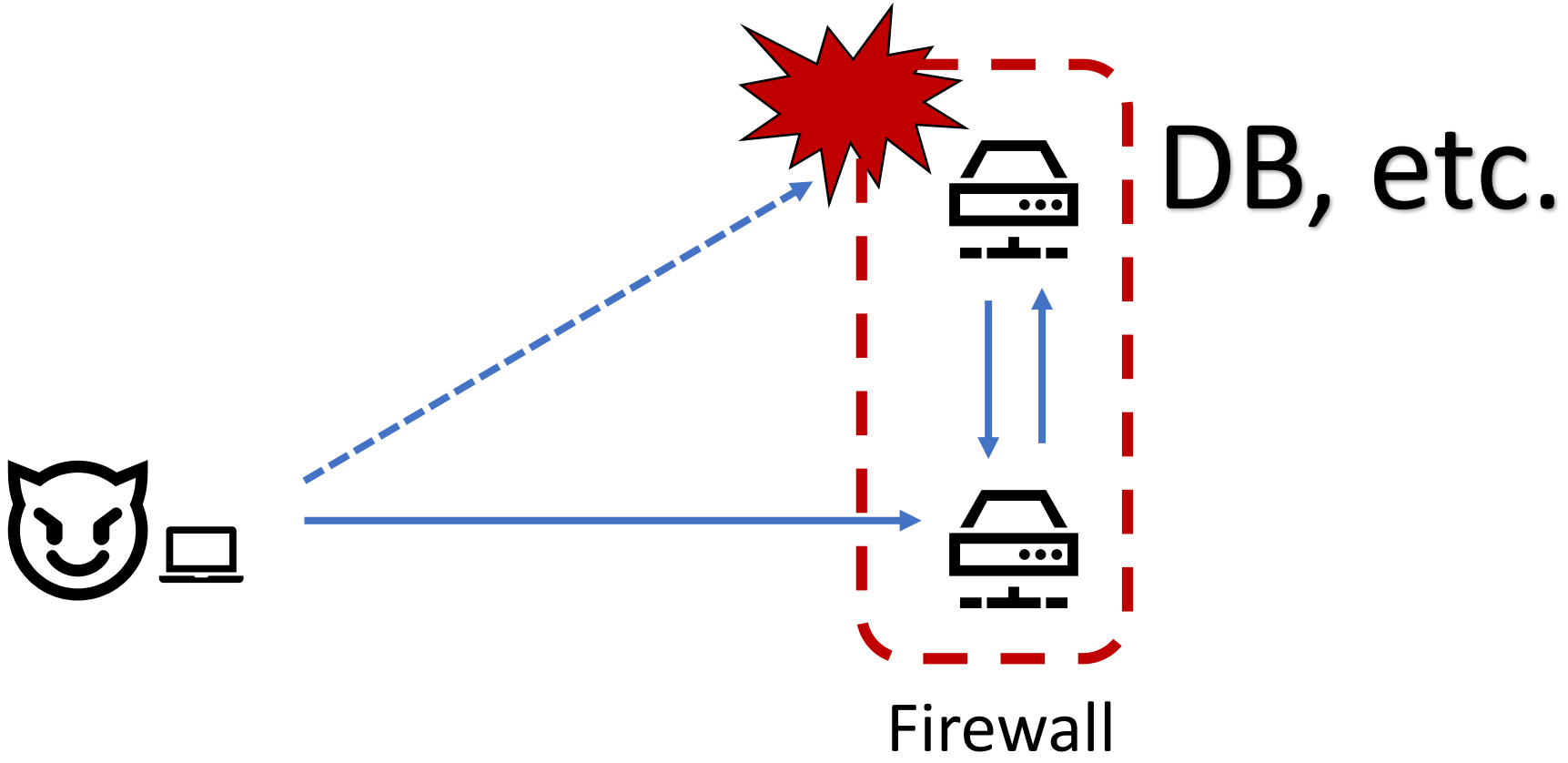
# Nicht alleine im Netzwerk?



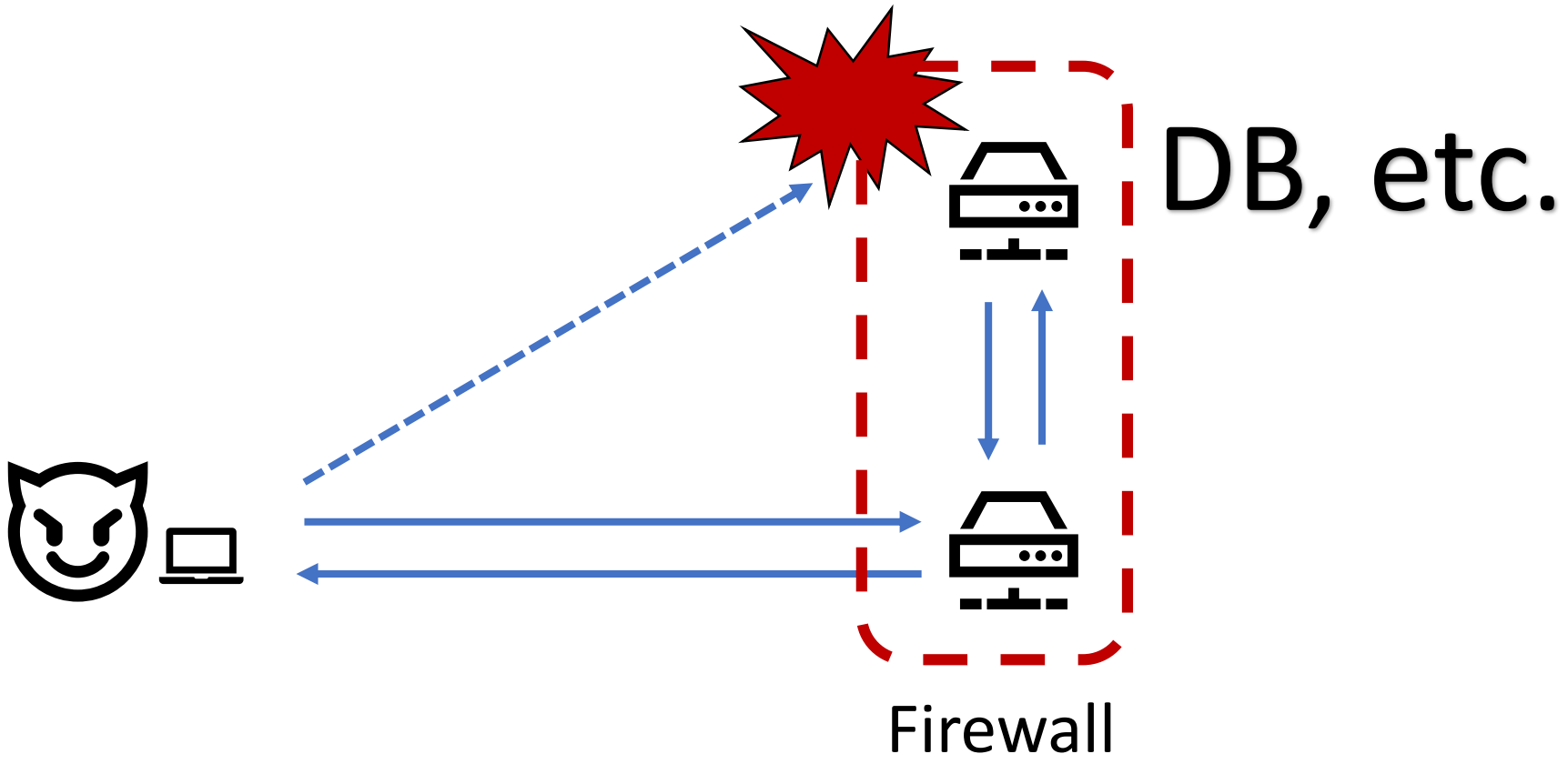
# Nicht alleine im Netzwerk?



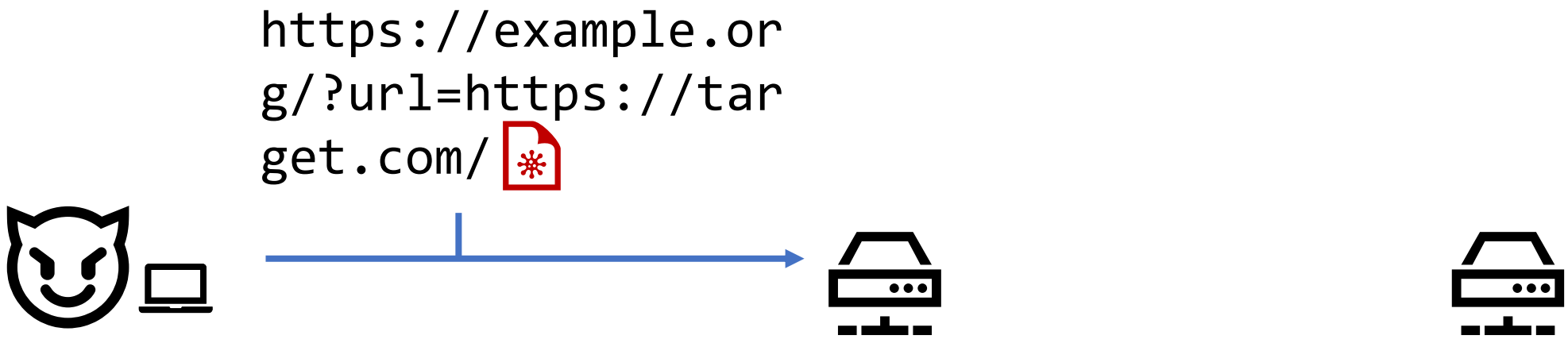
# Nicht alleine im Netzwerk?



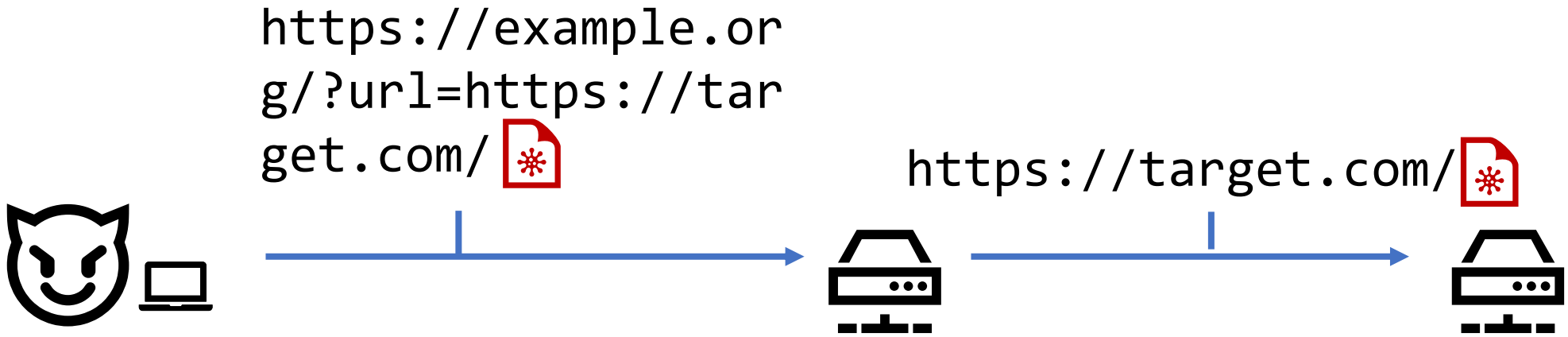
# Nicht alleine im Netzwerk?



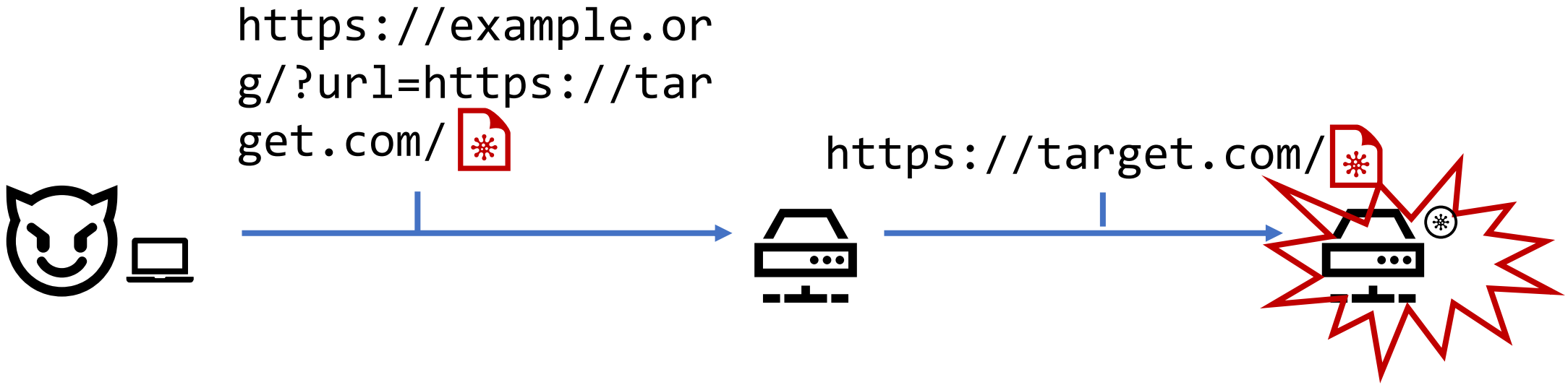
# Nicht alleine im Internet?



# Nicht alleine im Internet?



# Nicht alleine im Internet?

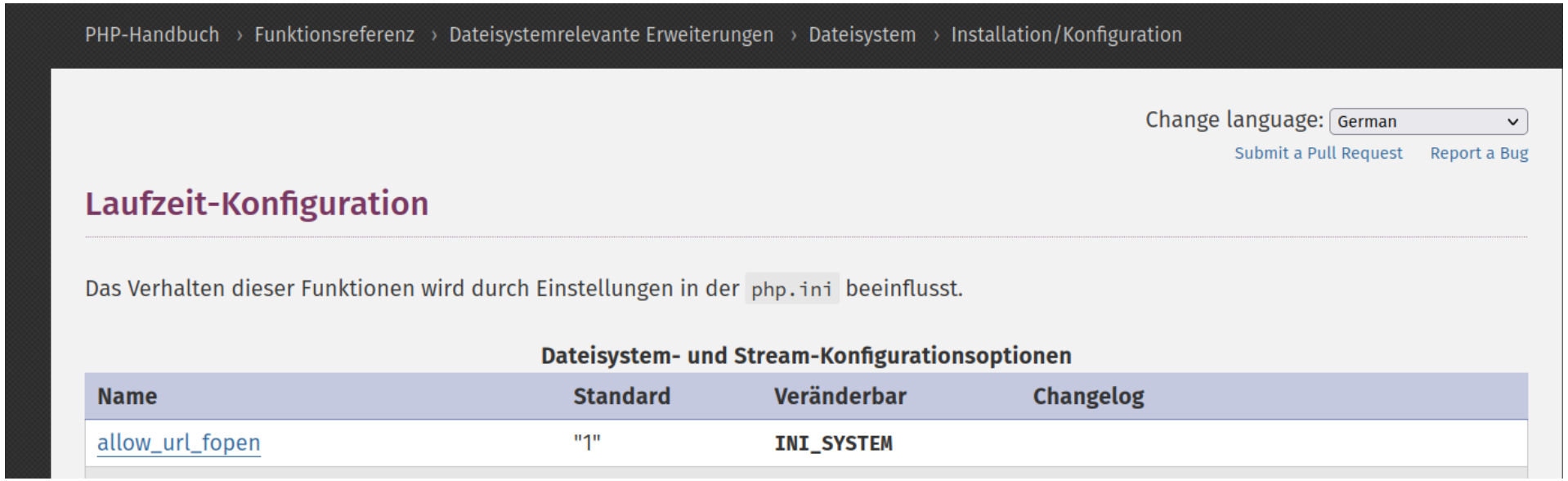


# Was machen wir jetzt dagegen?

- Netzwerkrequests finden und absichern

# Was machen wir jetzt dagegen?

- Netzwerkrequests finden und absichern




*https://www.php.net/manual/en/filesystem.configuration.php#ini.allow-url-fopen*

# Unerwartete Funktionalität

- Es gibt viele Funktionen die URLs als „Datei“ akzeptieren

# Unerwartete Funktionalität

- Es gibt viele Funktionen die URLs als „Datei“ akzeptieren
- `getimagesize(), file_get_contents(), …`

# Unerwartete Funktionalität

- Es gibt viele Funktionen die URLs als „Datei“ akzeptieren
- `getimagesize(), file_get_contents(), …`
- `get_headers()` macht GET request

# Unerwartete Funktionalität

- Es gibt viele Funktionen die URLs als „Datei“ akzeptieren
- `getimagesize(), file_get_contents(), …`
- `get_headers()` macht GET request
- opendir() öffnet auch remote Verzeichnisse

# Unerwartete Funktionalität

- Es gibt viele Funktionen die URLs als „Datei“ akzeptieren
- `getimagesize(), file_get_contents(), …`
- `get_headers()` macht GET request
- opendir() öffnet auch remote Verzeichnisse
- Früher: Requests durch XML parser
  - „External Entity“

# Verteidigung auf Stringebene?

# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`

# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`
  - `127.0.0.1`

# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`
  - `127.0.0.1`
- `str_contains($input, "127.0.0.1")`

# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`
  - `127.0.0.1`
- `str_contains($input, "127.0.0.1")`
  - `http://0x7F000001`, etc.

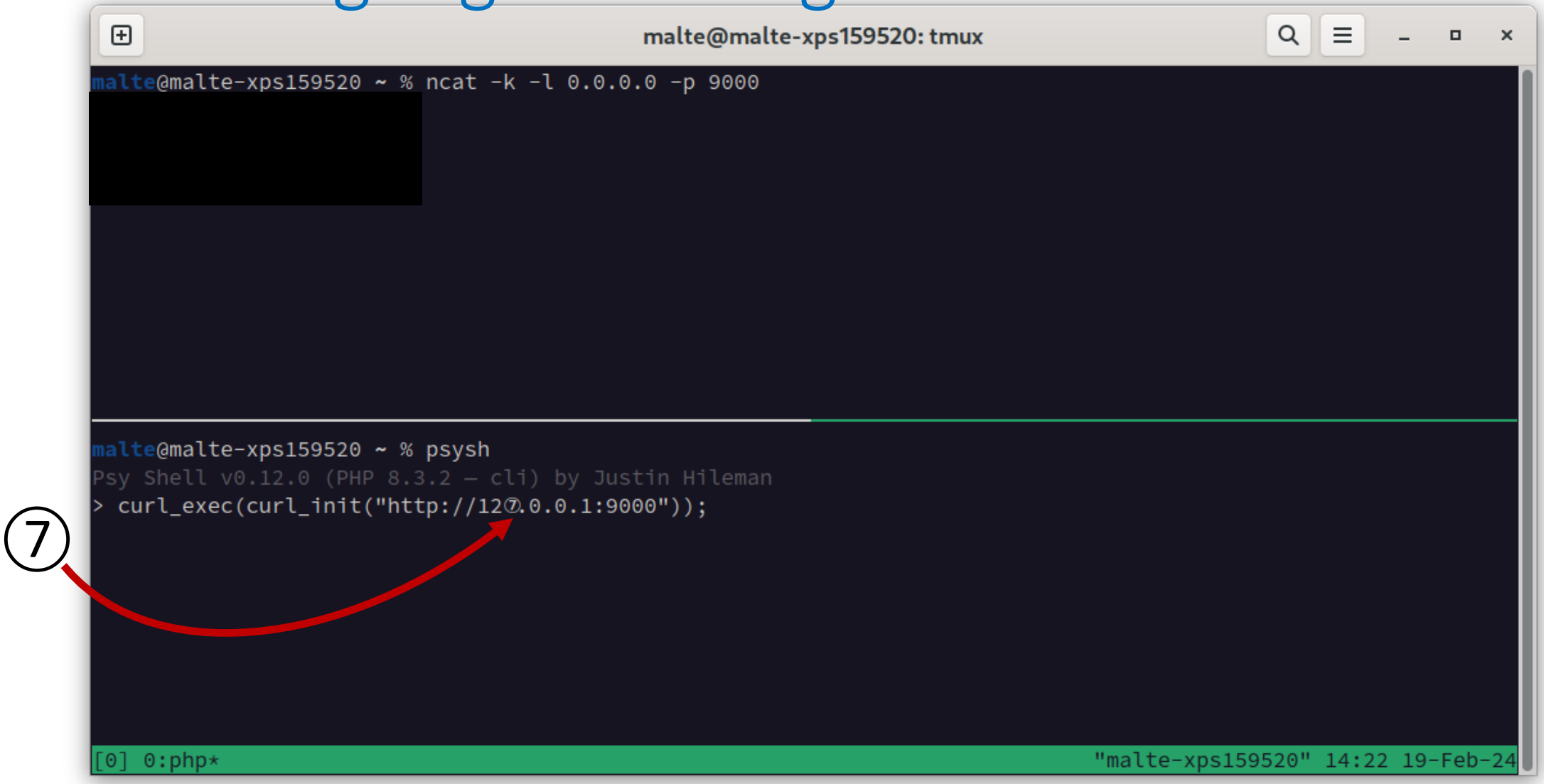
# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`
  - `127.0.0.1`
- `str_contains($input, "127.0.0.1")`
  - `http://0x7F000001`, etc.
- Unicode

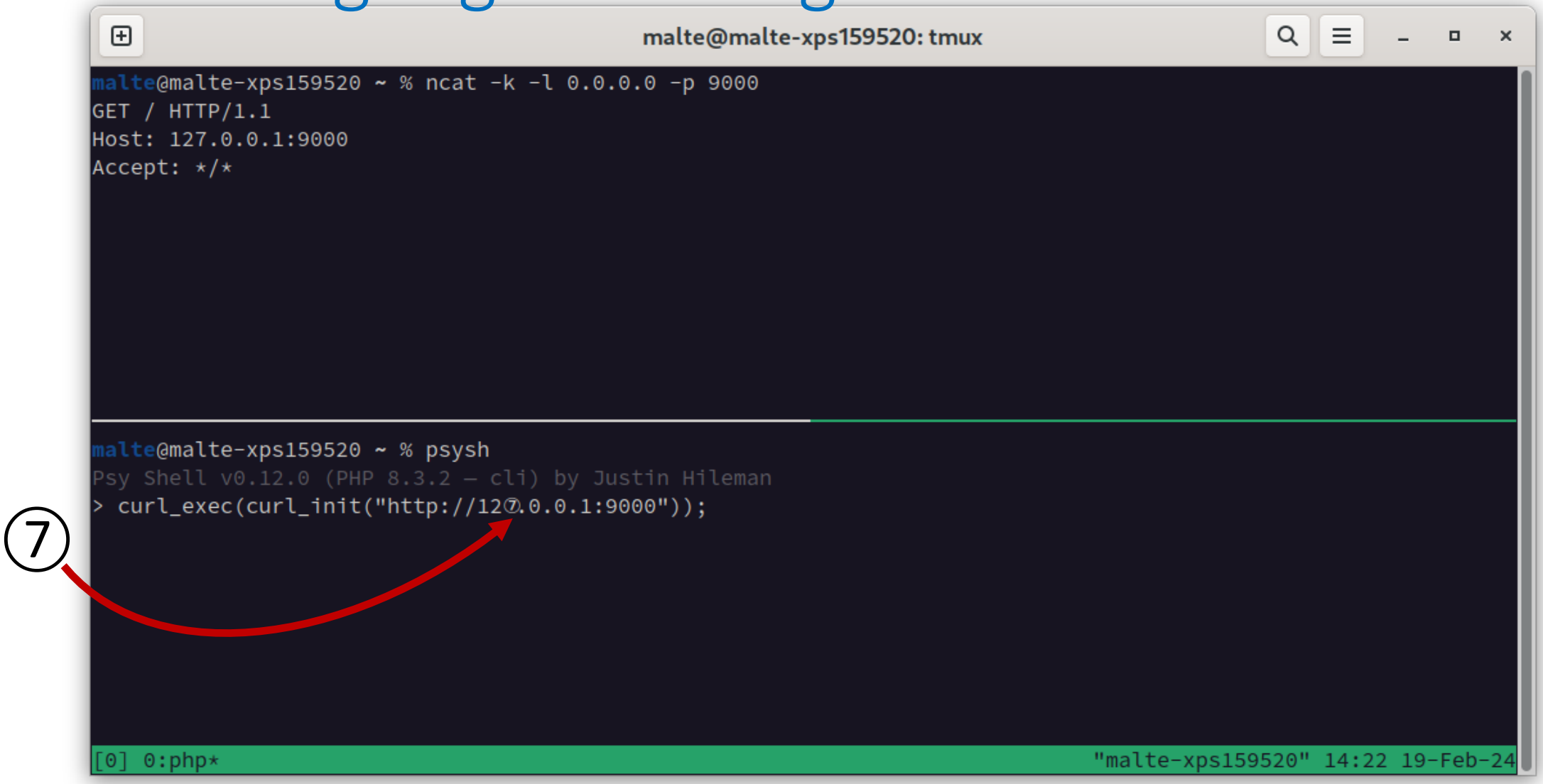
# Verteidigung auf Stringebene?

- `str_contains($input, "localhost")`
  - `127.0.0.1`
- `str_contains($input, "127.0.0.1")`
  - `http://0x7F000001`, etc.
- Unicode

# Verteidigung auf Stringebene?



# Verteidigung auf Stringebene?



# Domain prüfen

# Domain prüfen

- Regex? Z.B. `/http://.*example\.com/`

# Domain prüfen

- Regex? Z.B. `/http://.*example\.com/`
- `http://evil.com?attack=example.com`

# Domain prüfen

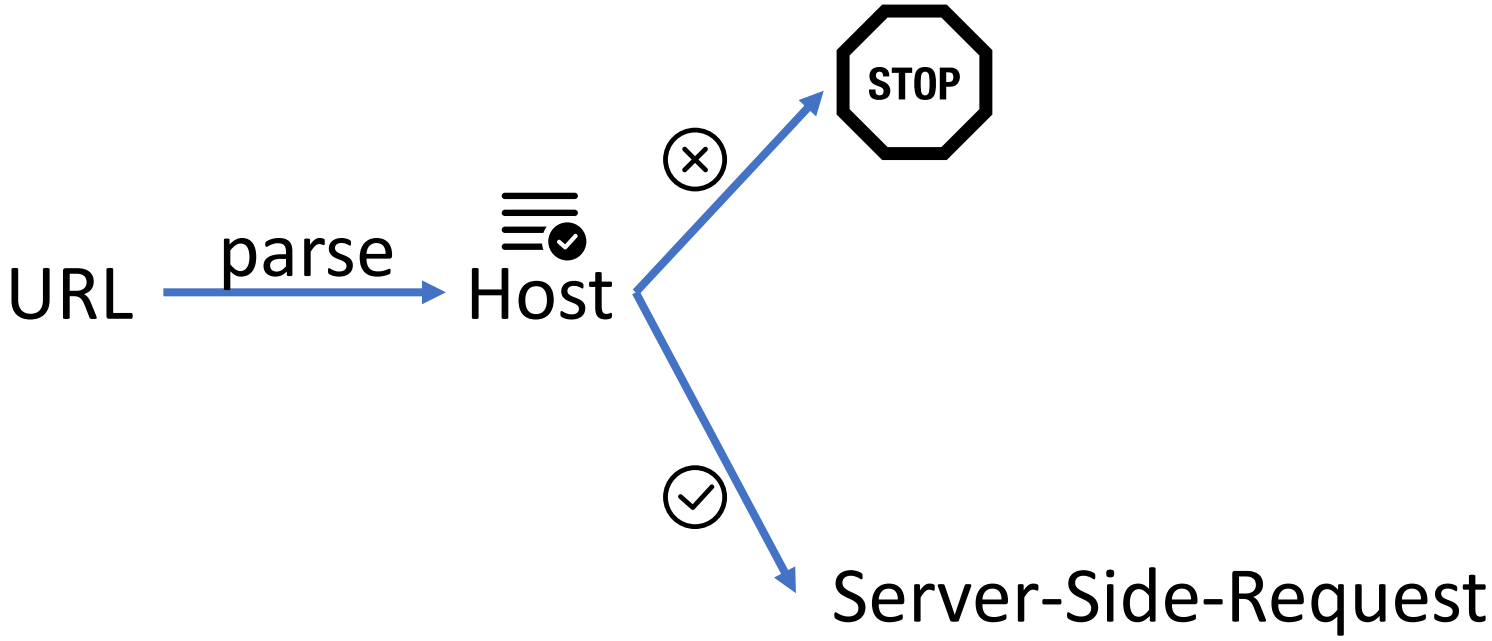
- Regex? Z.B. `/http://.*example\.com/`
- `http://evil.com?attack=example.com`

# Domain prüfen

- Regex? Z.B. `/http://.*example\.com/`
- `http://evil.com?attack=example.com`
- Besser: `parse_url($url, PHP_URL_HOST)`

# Domain prüfen

- Regex? Z.B. `/http://.*example\.com/`
- `http://evil.com?attack=example.com`
- Besser: `parse_url($url, PHP_URL_HOST)`



# Beispiel Allowlist

# Beispiel Allowlist

```
1. $input          = "https://app.com/link/page.html"
```

# Beispiel Allowlist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
```

# Beispiel Allowlist

```
1. $input        = "https://app.com/link/page.html"
2. $host         = parse_url($input, PHP_URL_HOST)

3. $allowedhosts = ["app.com", "good.example"]
```

# Beispiel Allowlist

```
1. $input        = "https://app.com/link/page.html"
2. $host         = parse_url($input, PHP_URL_HOST)

3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed    = in_array($host, $allowedhosts)
```

# Beispiel Allowlist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed    = in_array($host, $allowedhosts)

5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

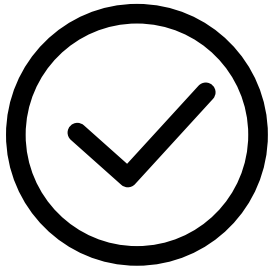
```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)

5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)
                   = true
5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)
                   = true
5. if ($isAllowed) {file_get_contents($input);}
```



# Beispiel Allowlist

```
1. $input          = "https://localhost/attack"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed    = in_array($host, $allowedhosts)

5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

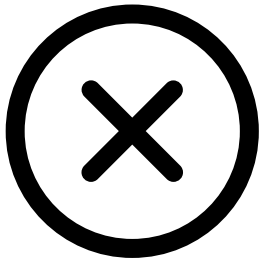
```
1. $input          = "https://localhost/attack"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)

5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

```
1. $input          = "https://localhost/attack"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)
                   = false
5. if ($isAllowed) {file_get_contents($input);}
```

# Beispiel Allowlist

```
1. $input          = "https://localhost/attack"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $allowedhosts = ["app.com", "good.example"]
4. $isAllowed      = in_array($host, $allowedhosts)
                   = false
5. if ($isAllowed) {file_get_contents($input);}
```



# URL Parser Confusion

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`
  - → nur einen Parser nehmen

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# URL Parser Confusion

- Unterschiede zwischen mehreren Parsern können ausgenutzt werden
  - `http://1.1.1.1 &@2.2.2.2# 3.3.3.3/`
  - → nur einen Parser nehmen
- In der Vergangenheit gab es sogar PHP-interne Differenzen
  - `parse_url vs readfile`
  - `parse_url vs curl`

Beispiel von Orange Tsai @ Blackhat‘17:
https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf

# Allowlist: Einen guten Parser

# Beispiel Denylist

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
```

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
```

# Beispiel Denylist

```
1. $input       = "https://app.com/link/page.html"
2. $host        = parse_url($input, PHP_URL_HOST)

3. $deniedhosts = ["localhost", "evil.com"]
```

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)
```

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)

5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

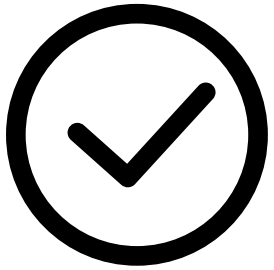
```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)

5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)
                   = false
5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

```
1. $input          = "https://app.com/link/page.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "app.com"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)
                   = false
5. if (!$isDenied) {file_get_contents($input);}
```



# Beispiel Denylist

```
1. $input          = "https://localhost/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)

5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

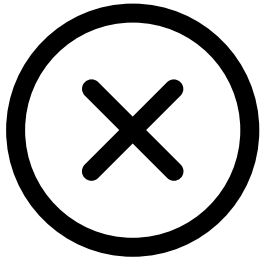
```
1. $input          = "https://localhost/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)

5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

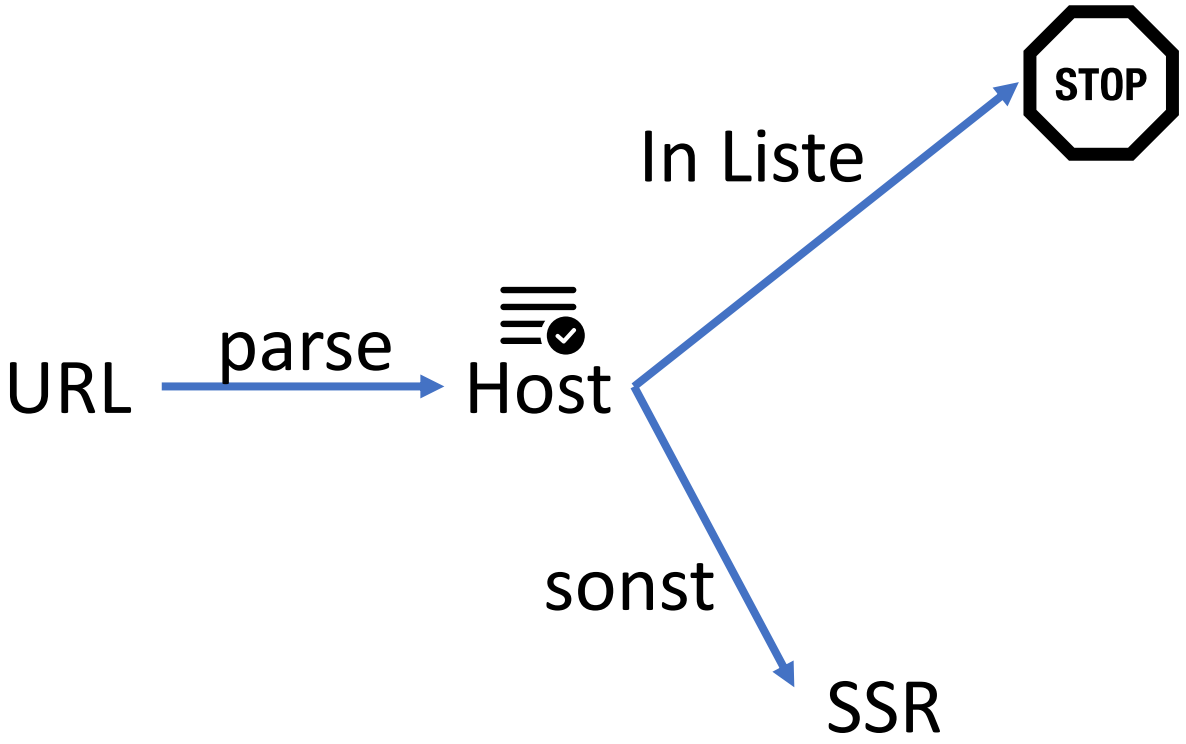
```
1. $input          = "https://localhost/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)
                   = true
5. if (!$isDenied) {file_get_contents($input);}
```

# Beispiel Denylist

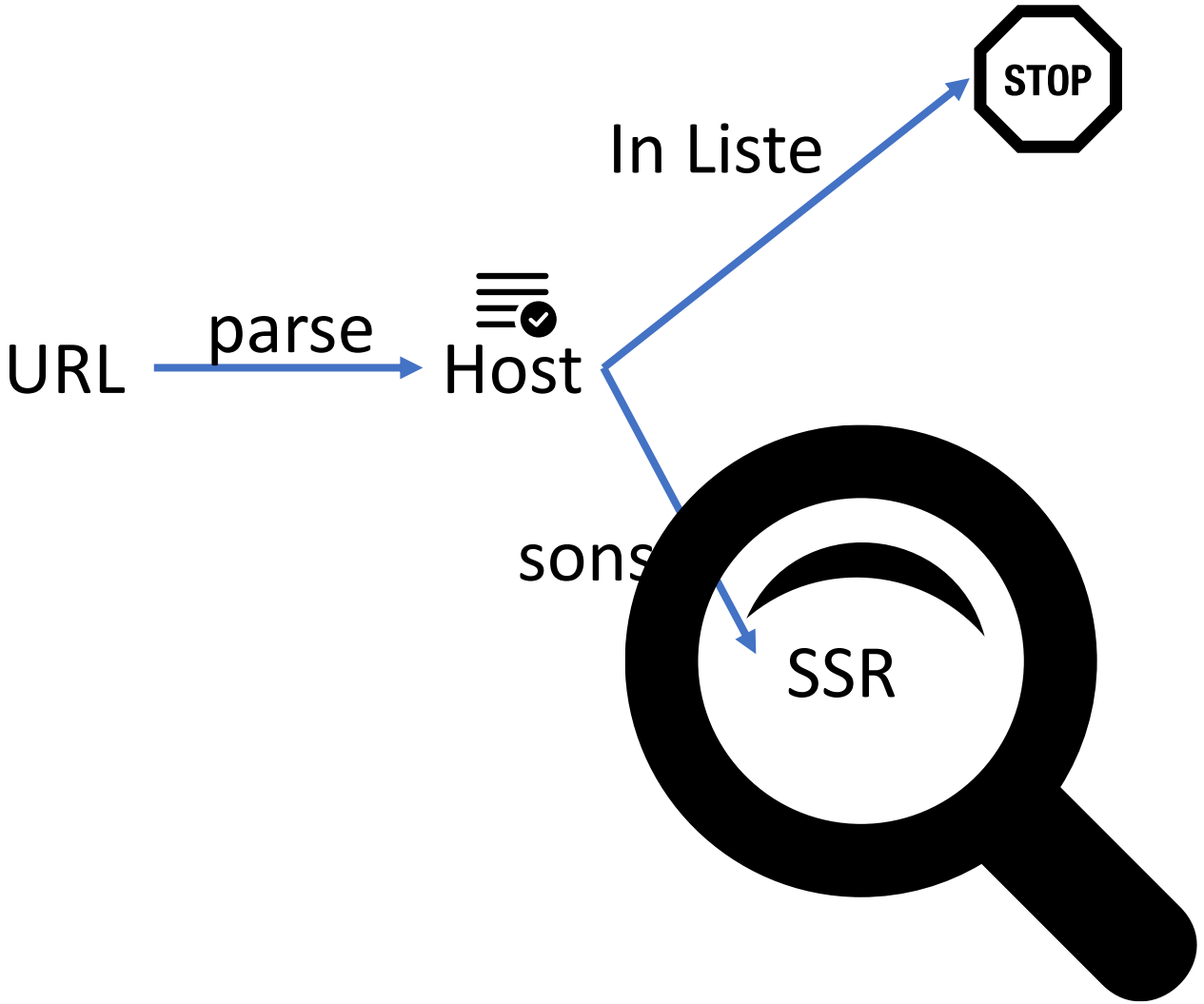
```
1. $input          = "https://localhost/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "localhost"
3. $deniedhosts    = ["localhost", "evil.com"]
4. $isDenied       = in_array($host, $allowedhosts)
                   = true
5. if (!$isDenied) {file_get_contents($input);}
```



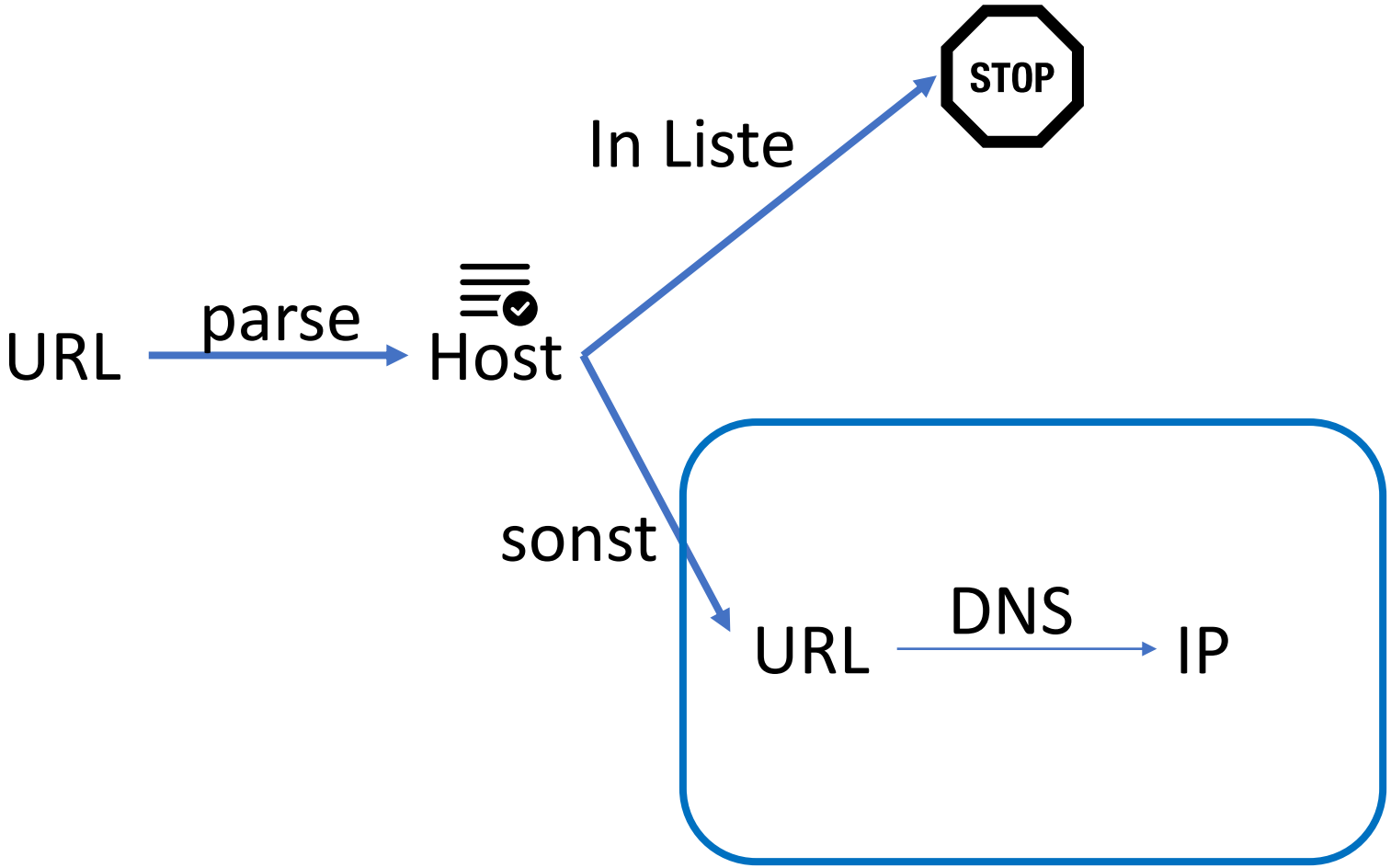
# Host Denylist



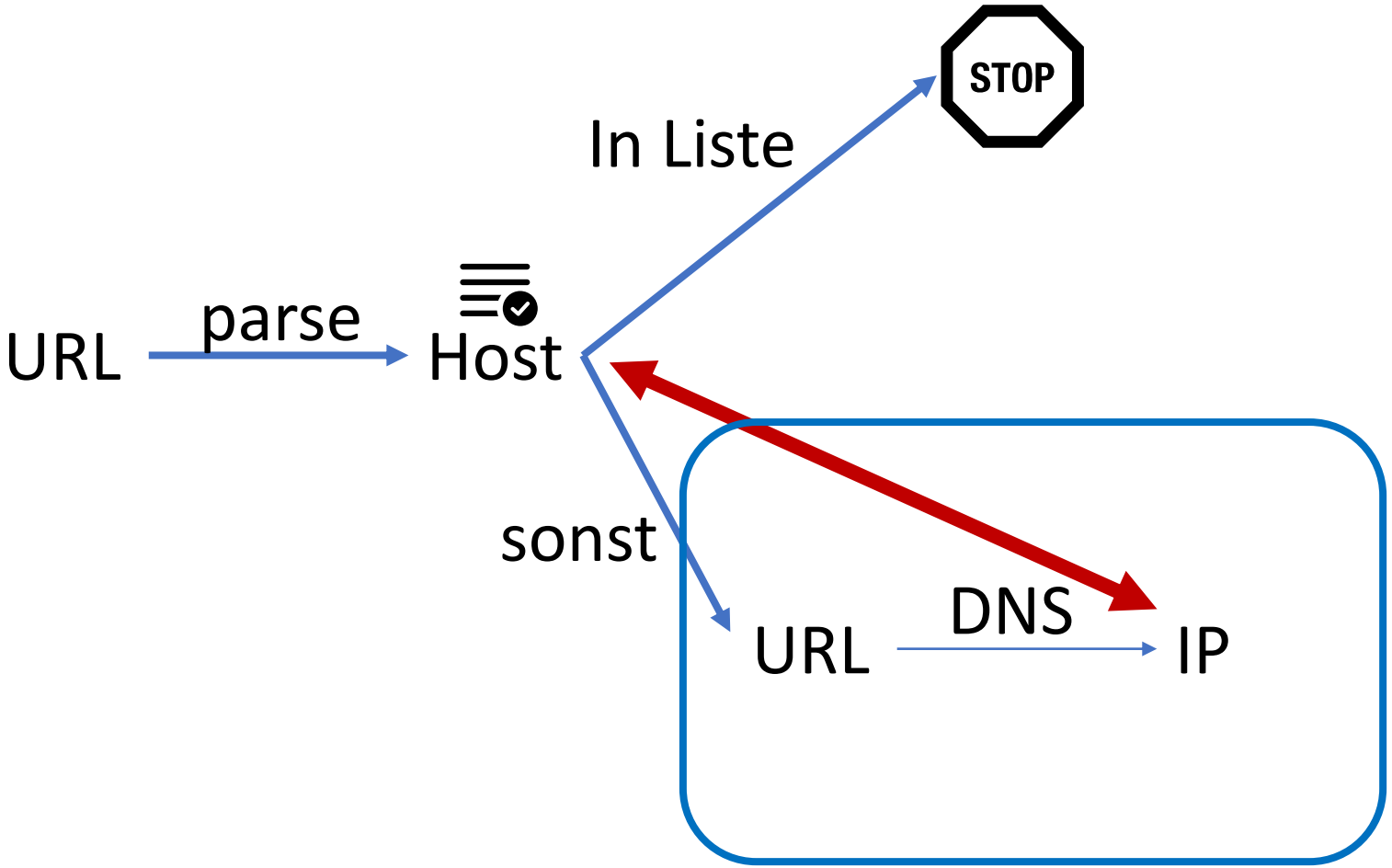
# Host Denylist



# Host Denylist



# Host Denylist



```
sh-5.2$ ping fbi.com
PING fbi.com (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.034 ms
```

```
sh-5.2$ ping fbi.com
PING fbi.com (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.034 ms
```

# IP-Denylist via DNS

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $ip             = gethostbyname($host);
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $ip             = gethostbyname($host);

4. $deniedIps      = ["127.0.0.1"]
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $ip             = gethostbyname($host);

4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)

3. $ip             = gethostbyname($host);

4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)

6. if (!$isDenied) {file_get_contents($input);}
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "fbi.com"
3. $ip             = gethostbyname($host);

4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)

6. if (!$isDenied) {file_get_contents($input);}
```

# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "fbi.com"
3. $ip             = gethostbyname($host);
                   = "127.0.0.1"
4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)

6. if (!$isDenied) {file_get_contents($input);}
```

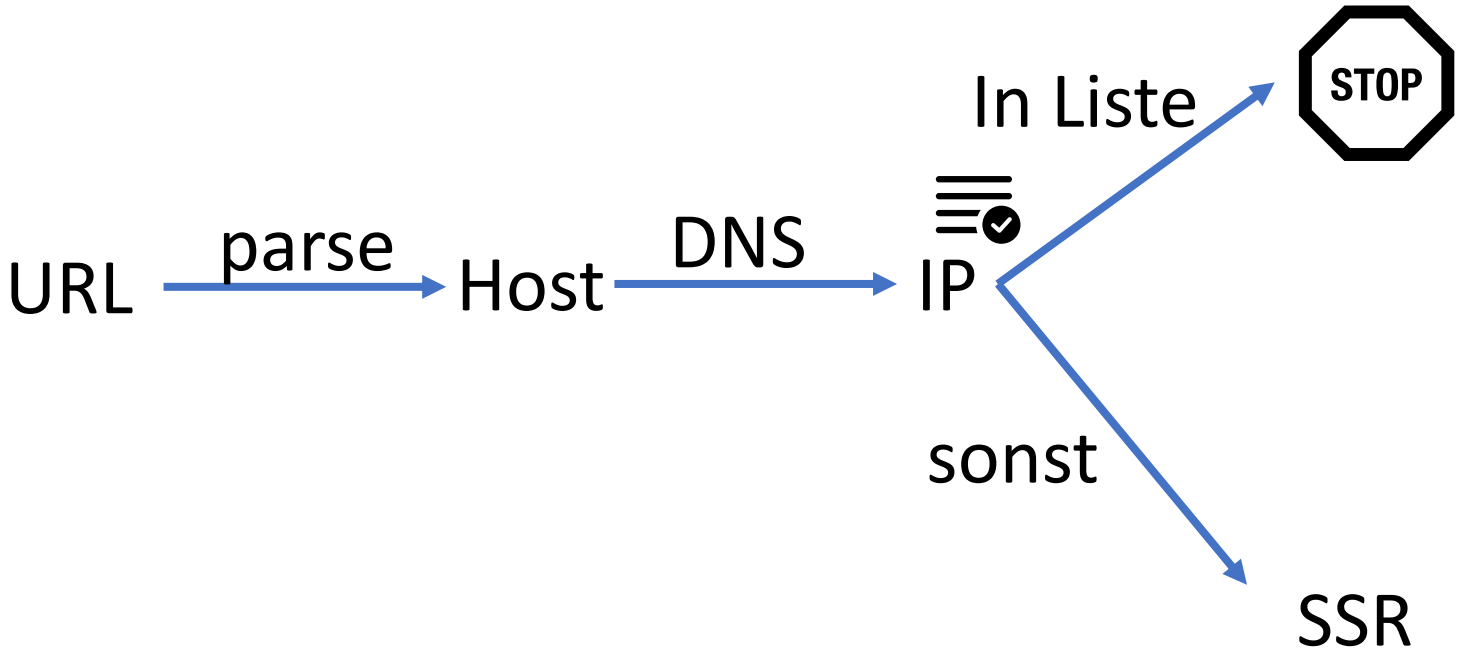
# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "fbi.com"
3. $ip             = gethostbyname($host);
                   = "127.0.0.1"
4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)
                   = true
6. if (!$isDenied) {file_get_contents($input);}
```

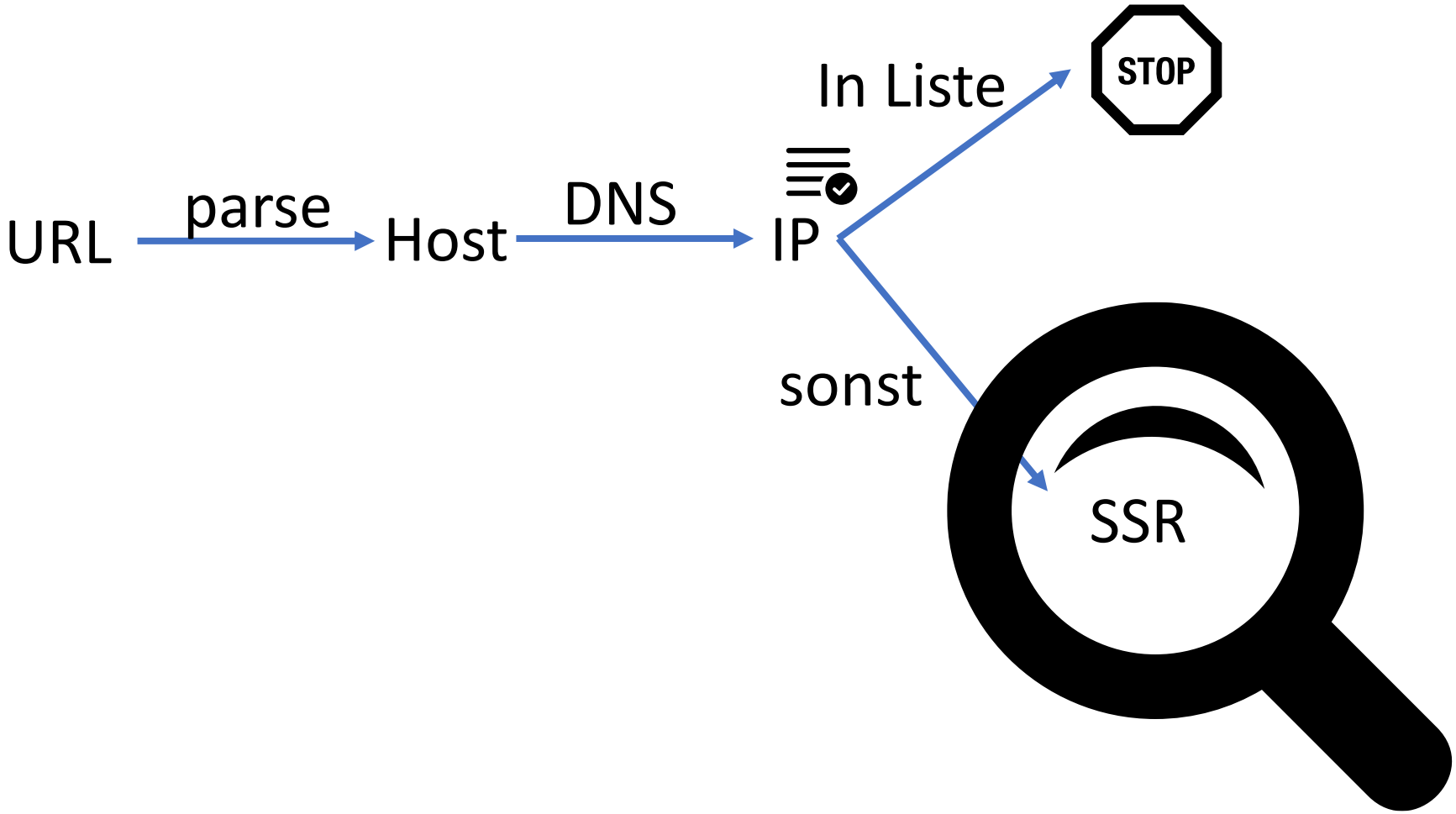
# IP-Denylist via DNS

```
1. $input          = "https://fbi.com/attack.html"
2. $host           = parse_url($input, PHP_URL_HOST)
                   = "fbi.com"
3. $ip             = gethostbyname($host);
                   = "127.0.0.1"
4. $deniedIps      = ["127.0.0.1"]
5. $isDenied       = in_array($host, $deniedIps)
                   = true
6. if (!$isDenied) {file_get_contents($input);}
```

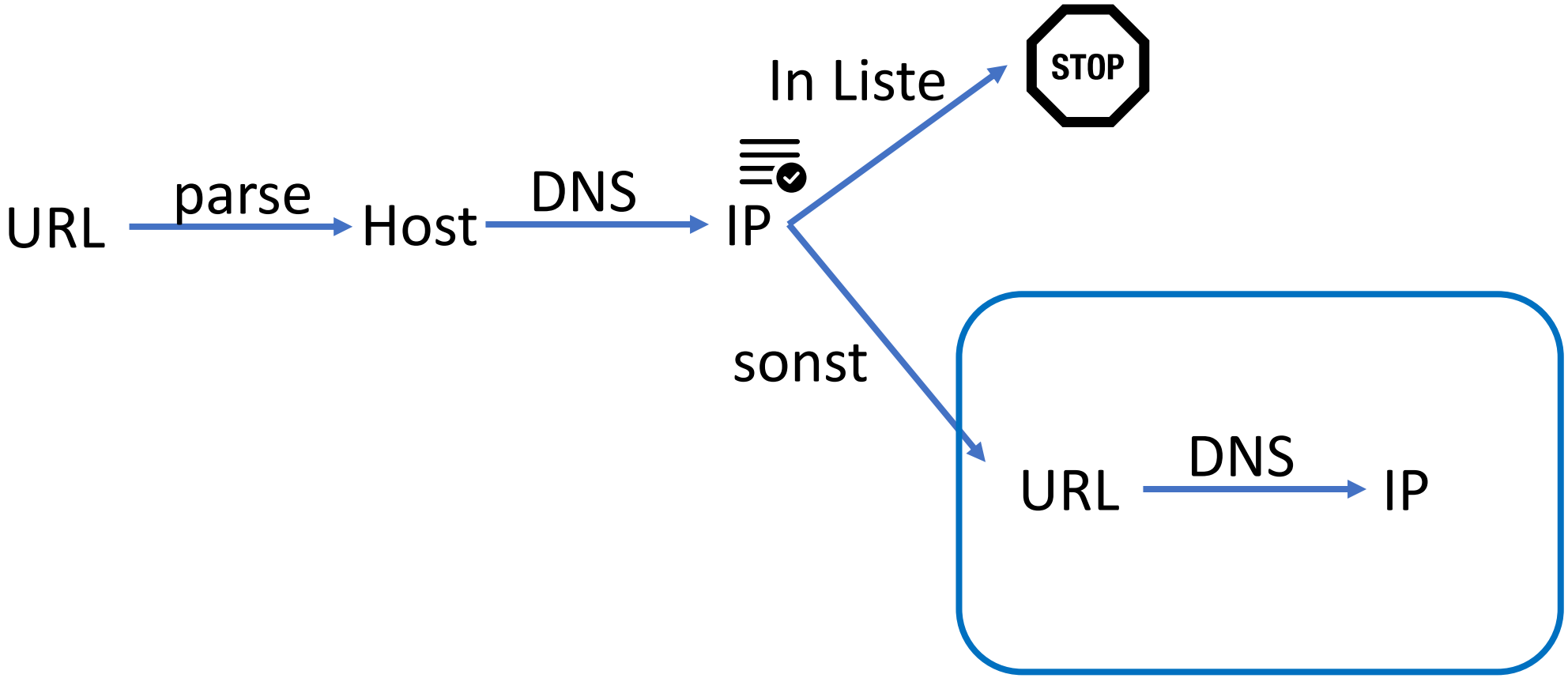
# Fix: DNS



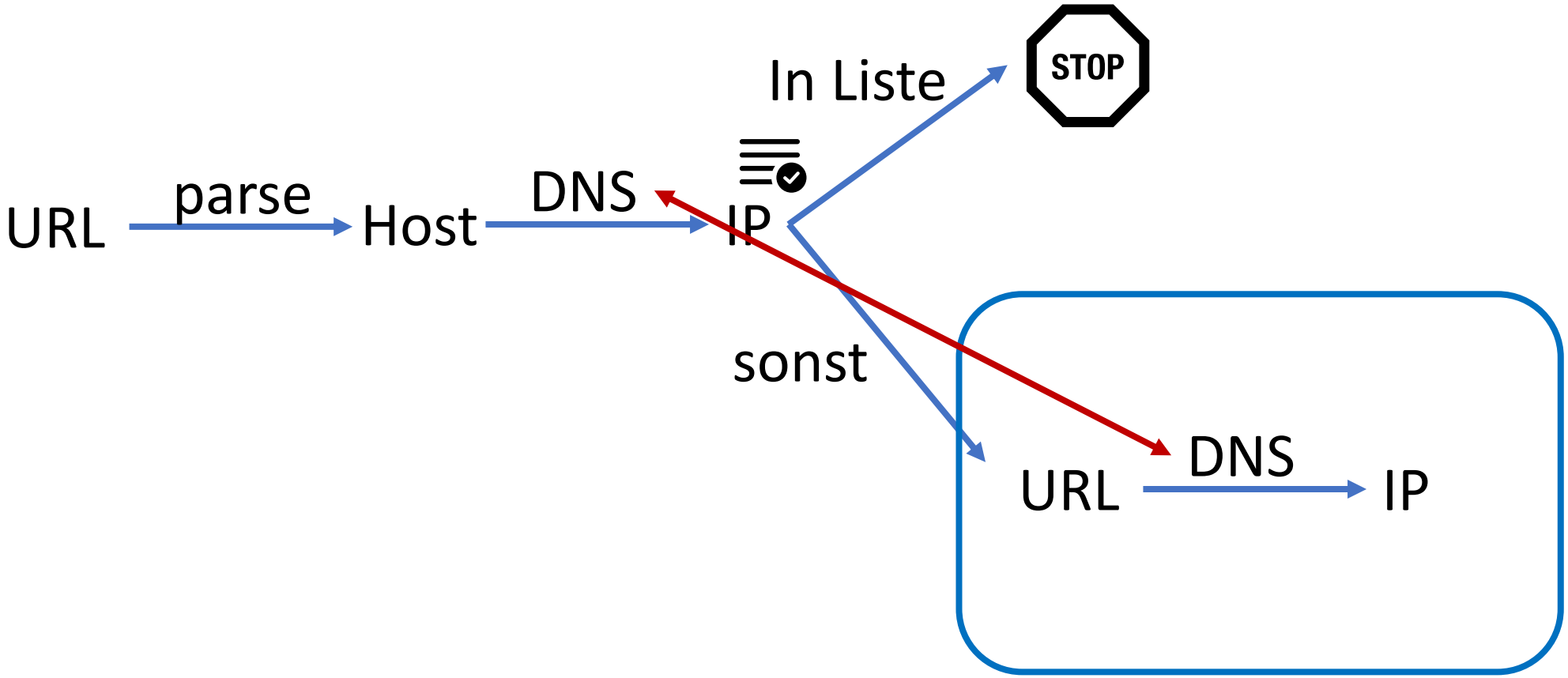
# Fix: DNS



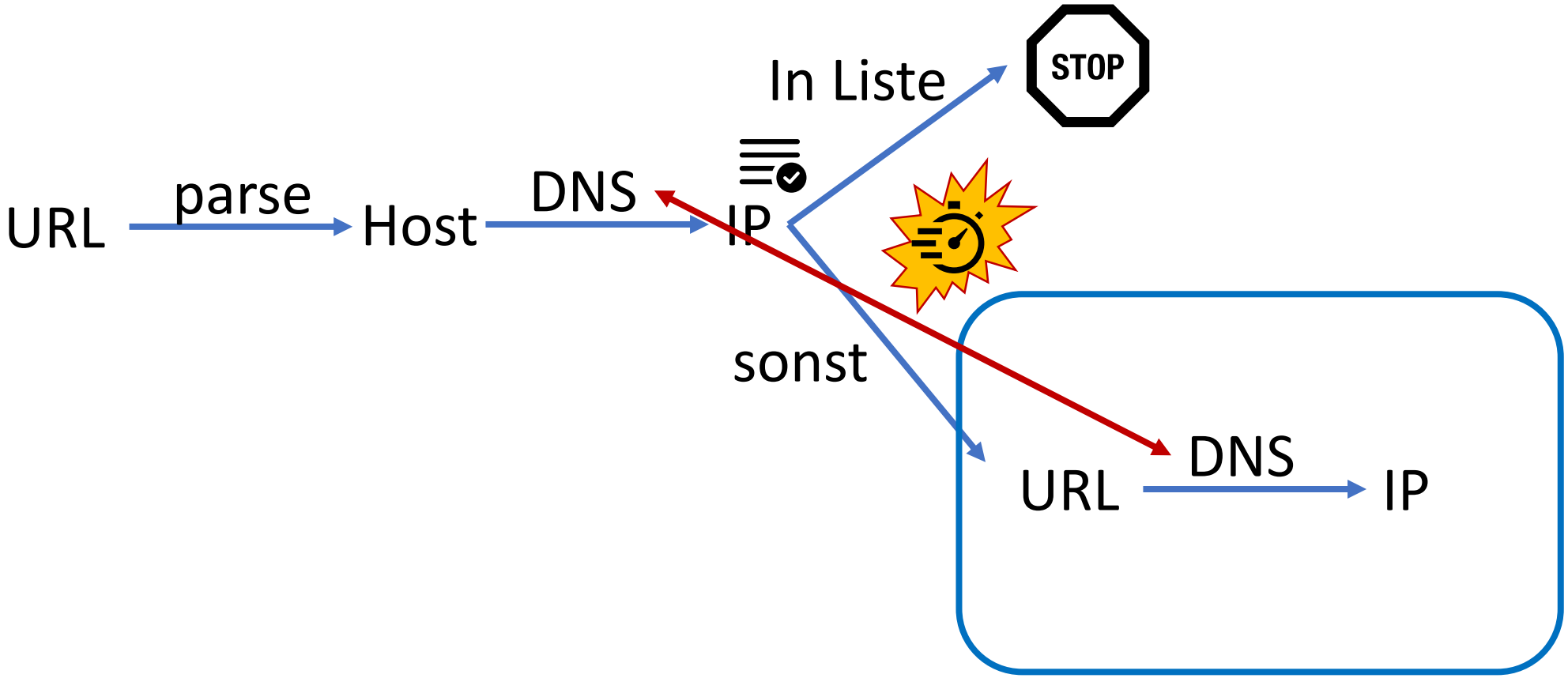
# Fix: DNS?

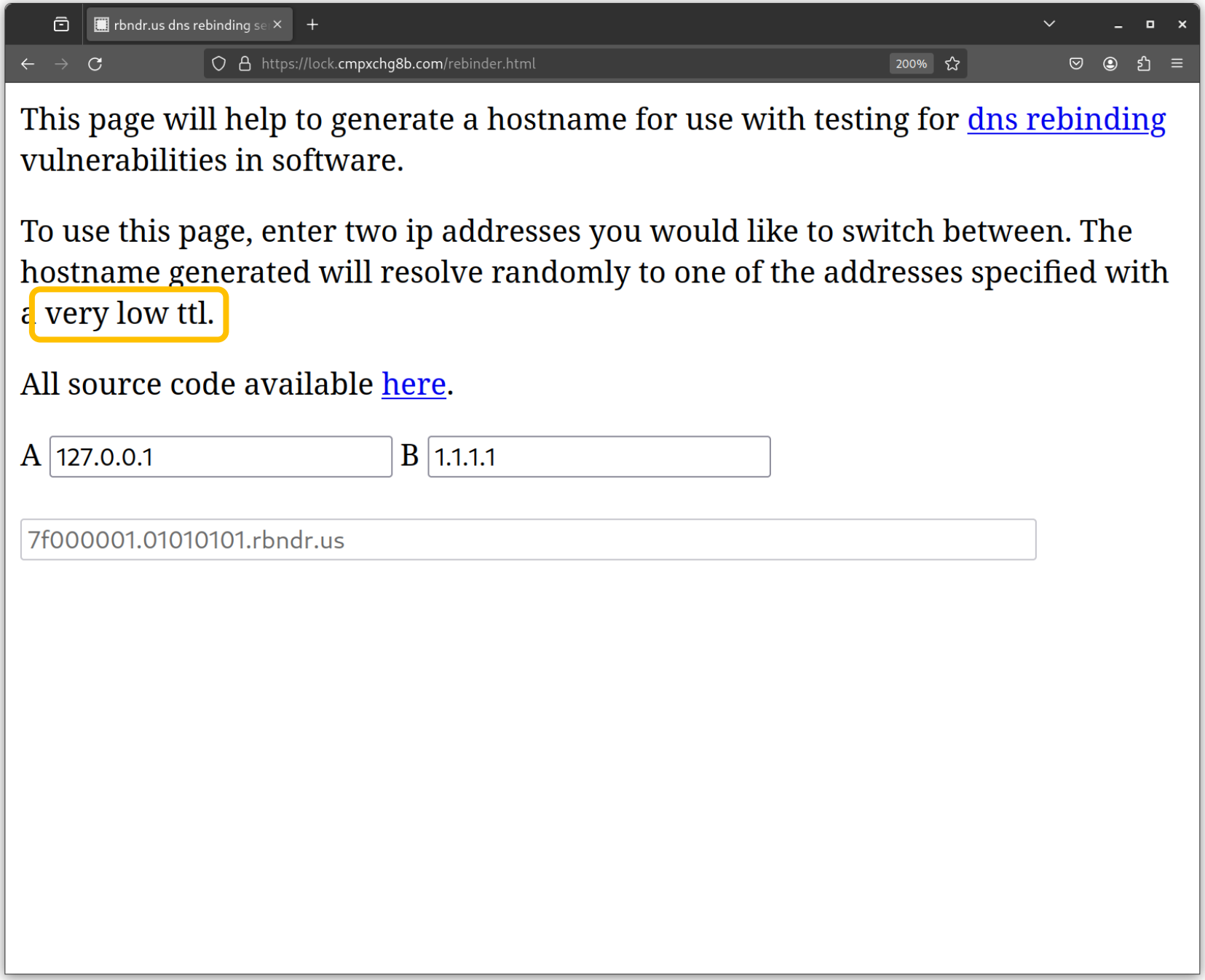


# Fix: DNS?



# Fix: DNS?




rbndr.us dns rebinding se
https://lock.cmpxchg8b.com/rebinder.html
200%
This page will help to generate a hostname for use with testing for dns rebinding vulnerabilities in software.
To use this page, enter two ip addresses you would like to switch between. The hostname generated will resolve randomly to one of the addresses specified with a very low ttl.
All source code available here.
A 127.0.0.1
B 1.1.1.1
7f000001.01010101.rbndr.us

# Beispielsangriff

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓
4. SSR wird ausgelöst

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓
4. SSR wird ausgelöst
5. Time-To-Live (TTL) des DNS-Cache-Eintrags läuft aus

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓
4. SSR wird ausgelöst
5. Time-To-Live (TTL) des DNS-Cache-Eintrags läuft aus
6. DNS Abfrage für SSR: IP = 127.0.0.1

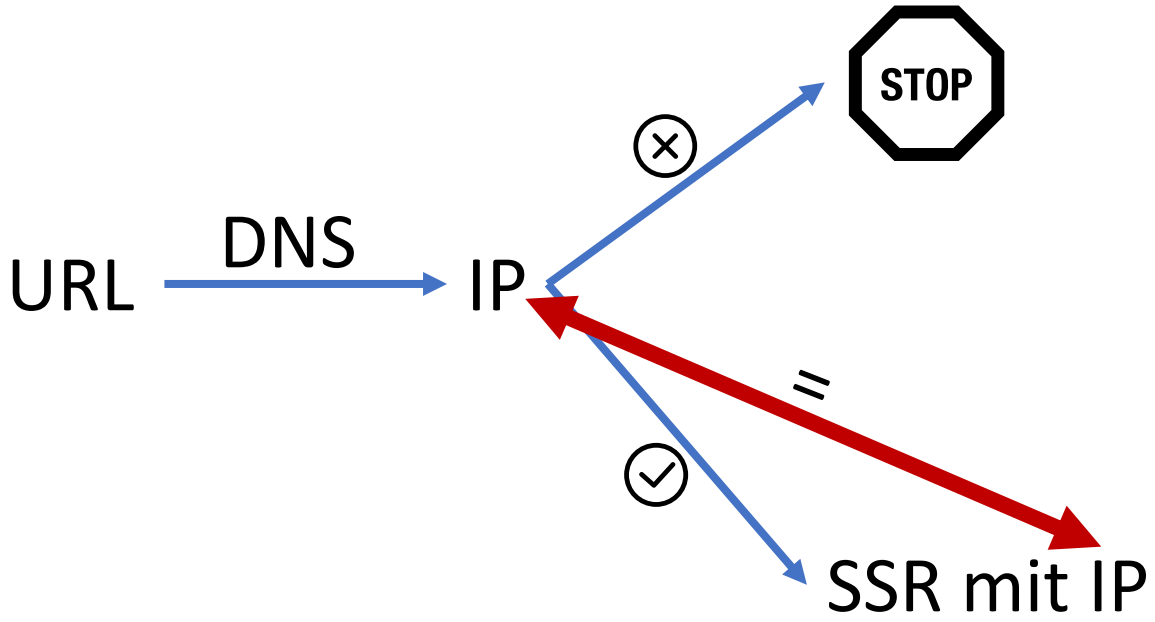
# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓
4. SSR wird ausgelöst
5. Time-To-Live (TTL) des DNS-Cache-Eintrags läuft aus
6. DNS Abfrage für SSR: IP = 127.0.0.1
7. SSR an 127.0.0.1

# Beispielsangriff

1. Domain: `7f000001.01010101.rbndr.us`
2. DNS-Abfrage für Check: IP = 1.1.1.1
3. Check bestanden ✓
4. SSR wird ausgelöst
5. Time-To-Live (TTL) des DNS-Cache-Eintrags läuft aus
6. DNS Abfrage für SSR: IP = 127.0.0.1
7. SSR an 127.0.0.1 

# DNS Pinning!



## SSRF (Server-side request forgery) Handling

SSRF allows an attacker to induce the backend application to make HTTP requests to an arbitrary domain. These attacks can also target the internal hosts and IPs of the attacked server.

If you use an HttpClient together with user-provided URIs, it is probably a good idea to decorate it with a NoPrivateNetworkHttpClient. This will ensure local networks are made inaccessible to the HTTP client:

```php
use Symfony\Component\HttpClient\HttpClient;
use Symfony\Component\HttpClient\NoPrivateNetworkHttpClient;

$client = new NoPrivateNetworkHttpClient(HttpClient::create());
// nothing changes when requesting public networks
$client->request('GET', 'https://example.com/');

// however, all requests to private networks are now blocked by default
$client->request('GET', 'http://localhost/');

// the second optional argument defines the networks to block
// in this example, requests from 104.26.14.0 to 104.26.15.255 will result in an exception
// but all the other requests, including other internal networks, will be allowed
$client = new NoPrivateNetworkHttpClient(HttpClient::create(), ['104.26.14.0/23']);
```

*https://symfony.com/doc/current/http_client.html#ssrf-server-side-request-forgery-handling*

README

# SafeCurl

CI no status | downloads 87.2 k | license MIT

SafeCurl intends to be a drop-in replacement for the curl_exec function in PHP. SafeCurl validates each part of the URL against a white or black list, to help protect against Server-Side Request Forgery attacks.

For more infomation about the project see the blog post 'SafeCurl: SSRF Protection, and a "Capture the Bitcoins"'.

## Protections

Each part of the URL is broken down and validated against a white or black list. This includes resolve a domain name to it's IP addresses.

If you chose to enable "FOLLOWLOCATION", then any redirects are caught, and re-validated.

*https://github.com/j0k3r/safecurl*

README

# SafeCurl

CI no status | downloads 87.2 k | license MIT

SafeCurl intends to be a drop-in replacement for the curl_exec function in PHP. SafeCurl validates each part of the URL against a white or black list, to help protect against Server-Side Request Forgery attacks.

## Optional Protections

In addition to the standard checks, two more are available.

The first is to prevent DNS Rebinding attacks. This can be enabled by calling the `enablePinDns` method on an `Options` object. There is one major issue with this - the SSL certificate **can't** be validated. This is due to the real hostname being sent in the `Host` header, and the URL using the IP address.

*https://github.com/j0k3r/safecurl*

# Denylist: DNS Pinning!

# Allow und Deny Listen

- Maßnahmen nur so gut wie die Listen
- Allowlist ist besser
- Falls ich vorher alles zu erreichende kenne
  - Aber sind die Domains unter meiner Kontrolle?....
- Falls nicht:
  - Alles „ unerwünschte“ ausschließen
  - Oft nicht nur lokale IPs
  - Nicht leicht: IP ranges, cloud IPs, IPv6

# SSRF Filmabend

```
https://app.com/?url=ftp://gratisfilme.to/film.mkv
```

# Unterstützte Protokolle und Wrapper

PHP bietet viele integrierte Wrapper für verschiedene Protokolle im URL-Stil auf die so mit Dateisystem-Funktionen wie fopen(), copy(), file_exists() und filesize() zugegriffen werden. Zusätzlich zu diesen eingebauten Wrappern ist es auch möglich eigene mit Hilfe der stream_wrapper_register() Funktion hinzuzufügen.

**Hinweis**: Die zur Beschreibung eines Wrappers genutzte URL-Syntax unterstützt nur URLs der Form `schema://...` mit zwei Schrägstrichen. Die `schema:/` und `schema:` Varianten werden nicht unterstützt.

## Inhaltsverzeichnis



*https://www.php.net/manual/de/wrappers.php*

# Curl Protocols

> PHP supports libcurl [...] libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols

*https://www.php.net/manual/en/intro.curl.php*

# Curl Protocols

> PHP supports libcurl […] libcurl currently supports the http, https, ftp, gopher, telnet, dict, file, and ldap protocols

*https://www.php.net/manual/en/intro.curl.php*

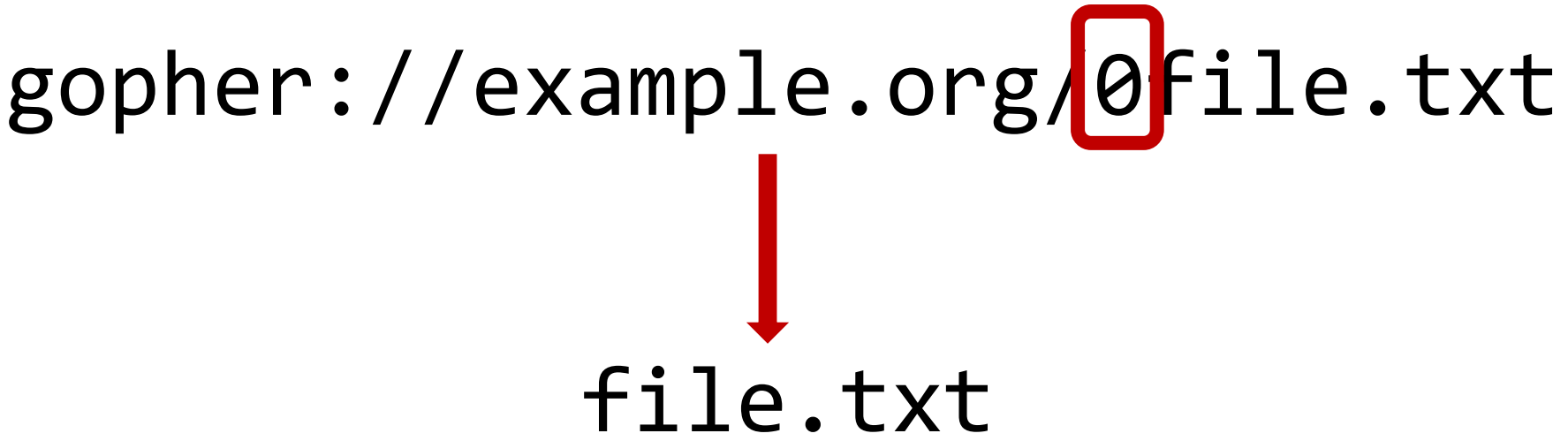
# Spaß mit Gopher

```
gopher://example.org/0file.txt
```

# Spaß mit Gopher

```
gopher://example.org/0file.txt
```

# Spaß mit Gopher



# Spaß mit Gopher

```
gopher://127.0.0.1:25/_MAIL%20FRO
M:hello%40%E2%82example.org%0ARCP
T%20To:target%40example.org%0ADAT
A%0AFrom:hello%40%E2%82example.or
g%0ASubject:SSRF-
Mail%0AMessage:Message-Body%0A.
```

# Spaß mit Gopher

```
gopher://127.0.0.1:25/_MAIL
FROM:hello@example.org
RCPT To:target@example.org
DATA
From:hello@example.org
Subject:SSRF-Mail
Message:Message-Body
.
```

# Spaß mit Gopher

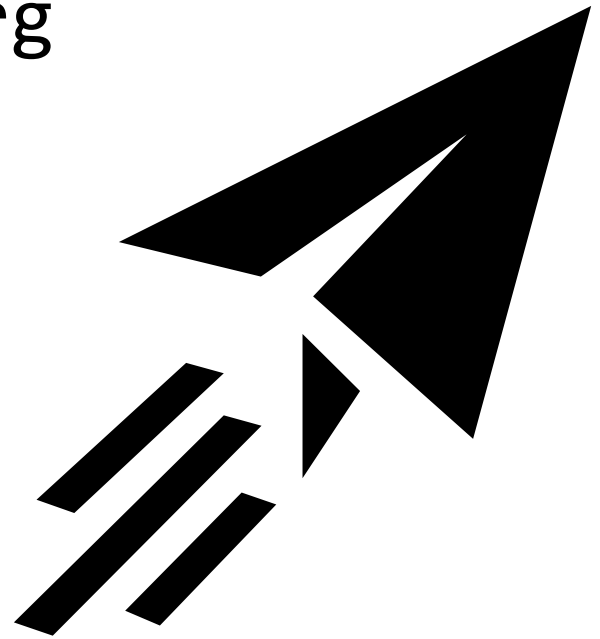
```
gopher://127.0.0.1:25/_MAIL
FROM:hello@example.org
RCPT To:target@example.org
DATA
From:hello@example.org
Subject:SSRF-Mail
Message:Message-Body
.
```

# Spaß mit Gopher

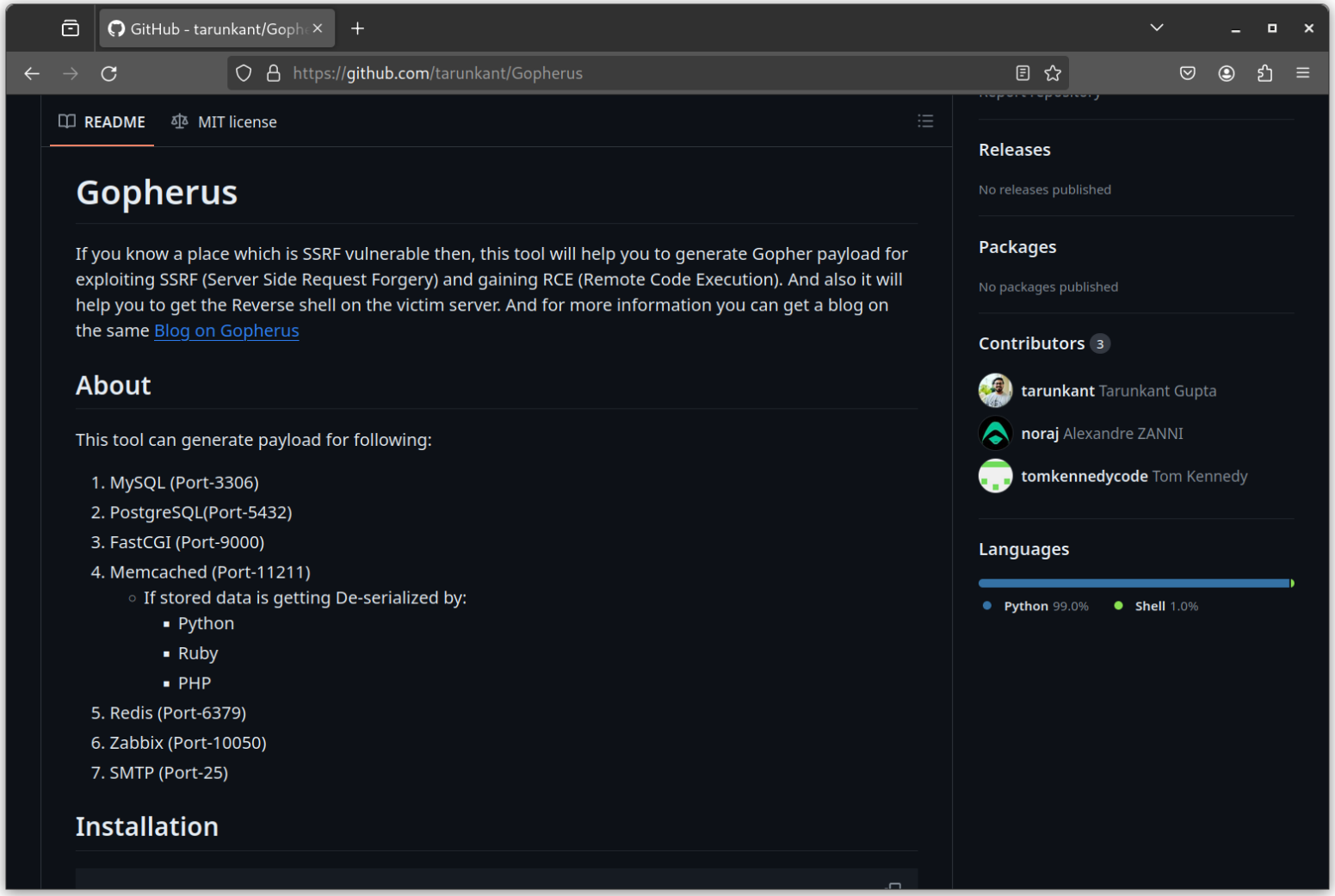
```
MAIL FROM:hello@example.org
RCPT To:target@example.org
DATA
From:hello@example.org
Subject:SSRF-Mail
Message:Message-Body
.
```

# Spaß mit Gopher

```
MAIL FROM:hello@example.org
RCPT To:target@example.org
DATA
From:hello@example.org
Subject:SSRF-Mail
Message:Message-Body
.
```



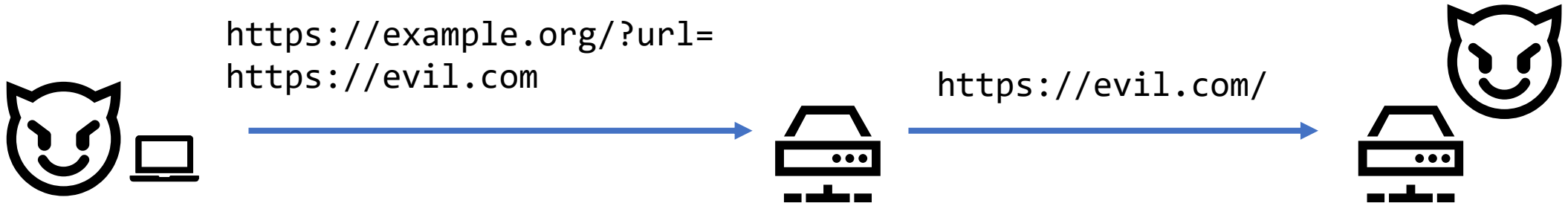
# Spaß mit Gopher



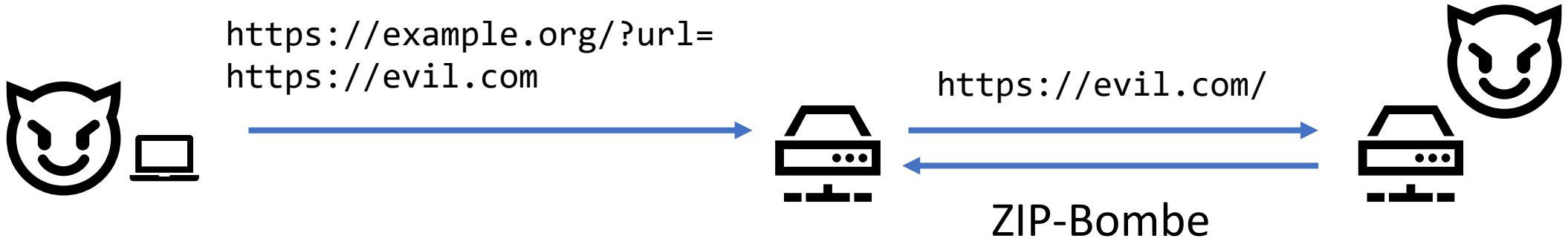
# Protokolle festlegen!

Meist: Nur HTTP(S)

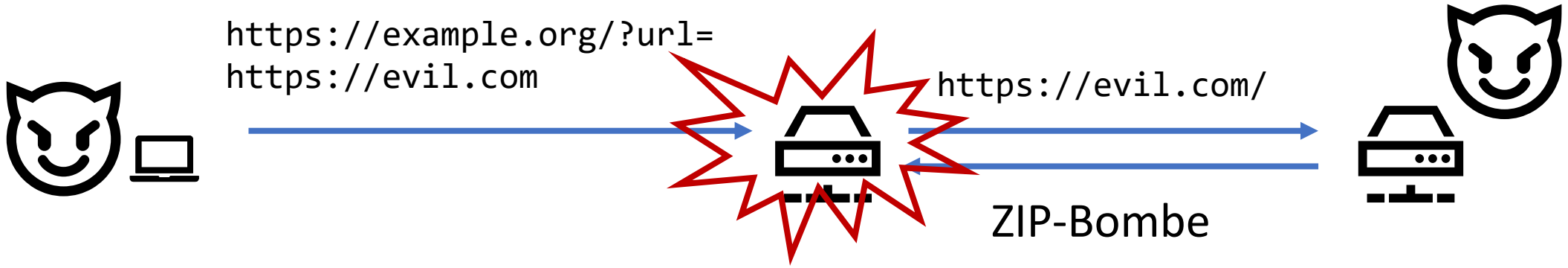
# Denial-Of-Service (DoS)



# Denial-Of-Service (DoS)

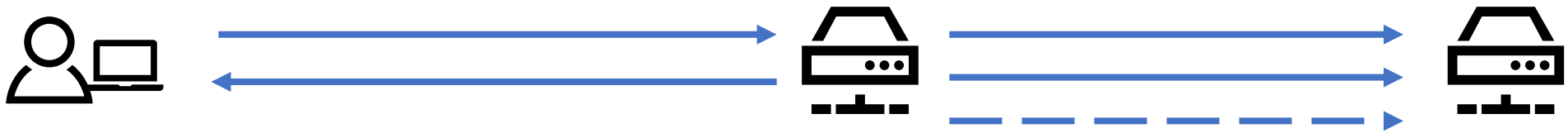


# Denial-Of-Service (DoS)



# Andere DoSen

# Andere DoSen

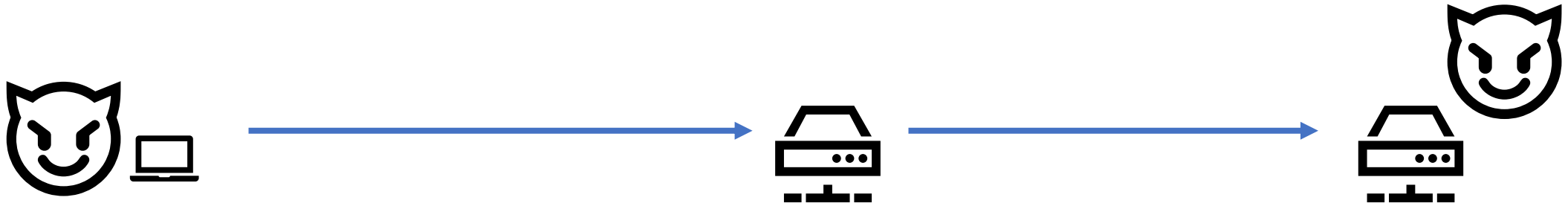


# Rate-Limits!

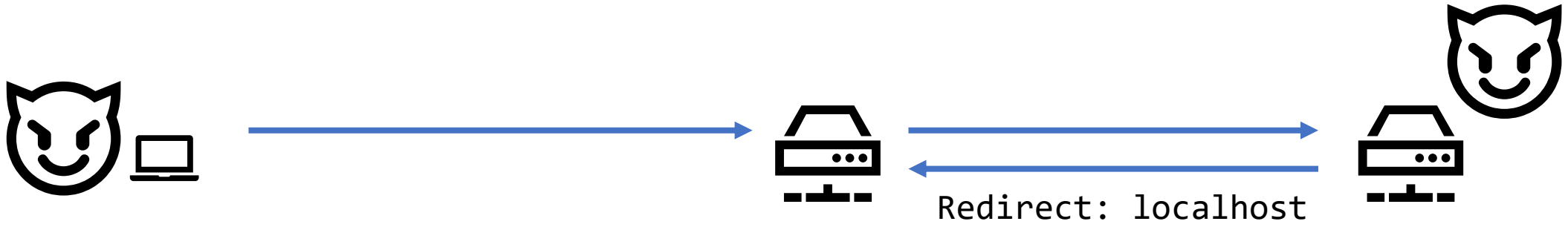
Für Input, Output, sowie Rechenzeit.

# Redirects

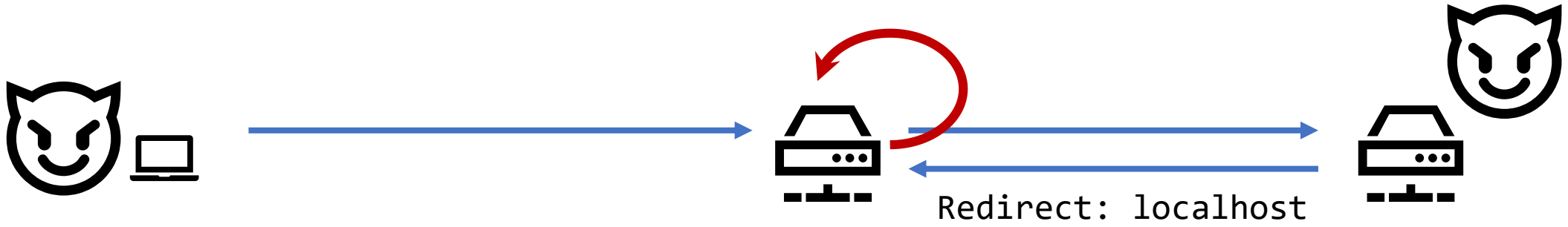
# Redirects



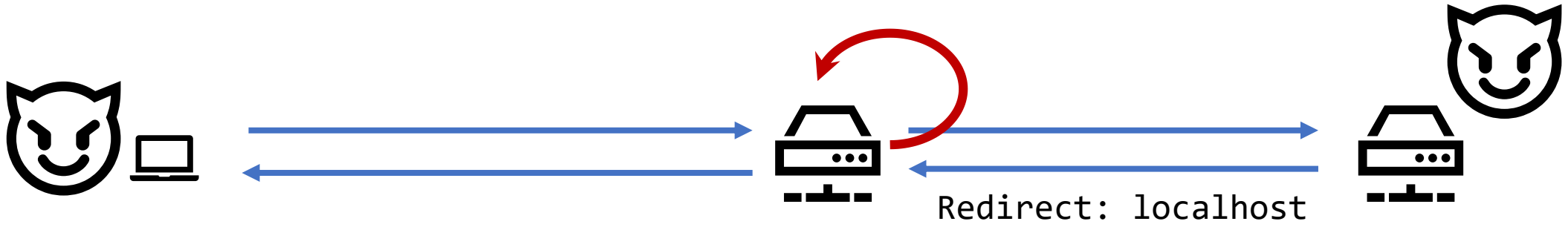
# Redirects



# Redirects



# Redirects



# Redirects ausschalten

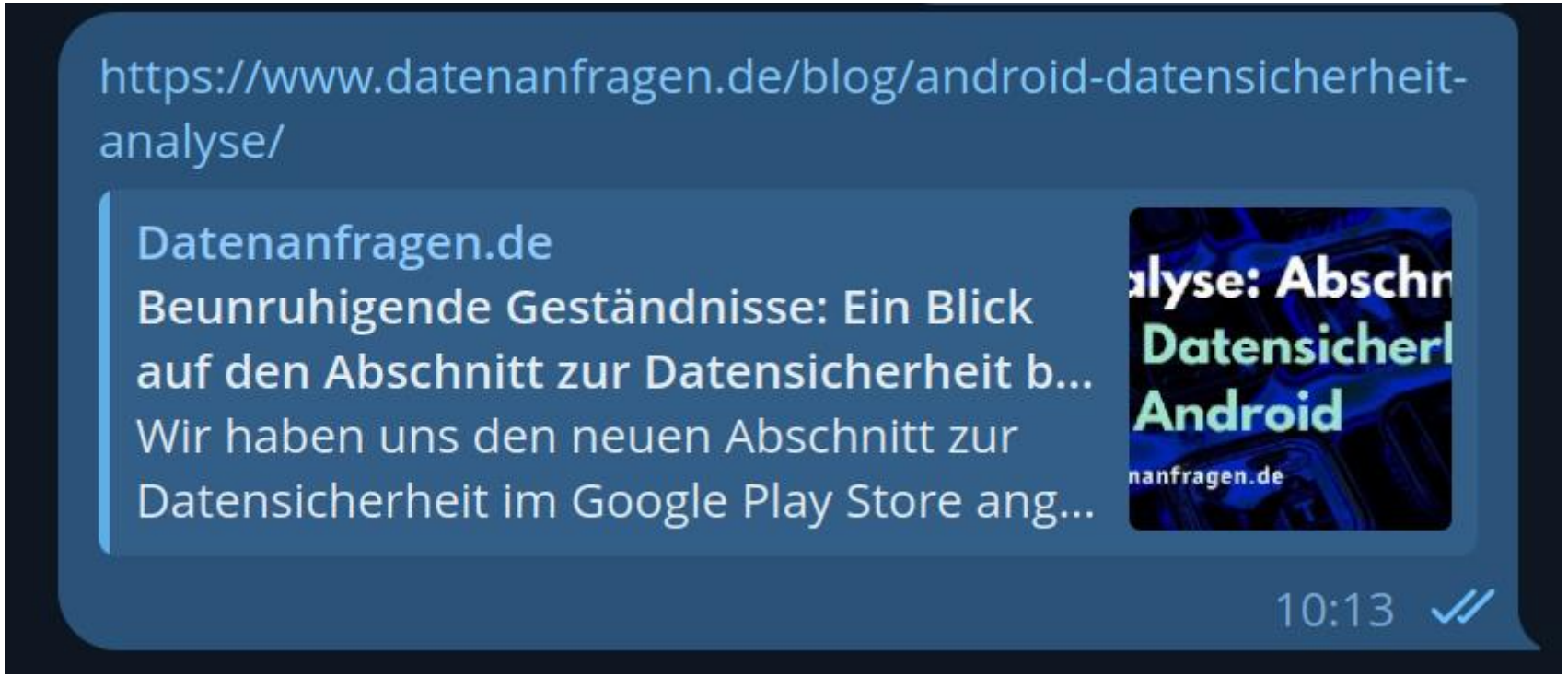
Oder immer wieder erneut prüfen

# ~~Geld~~Origin-Wäsche

- Wir sollten nicht zum Verstecken des Ursprungs von Content genutzt werden können
- Web-Security-Mechanismen
- Diverse Best-Practice-Regeln
  - Result der SSR nicht 1:1 zurück geben, also z.B. wrappen
  - ~~`Access-Control-Allow-Origin: *`~~
  - `Content-Disposition` Header auf `attachment` setzen

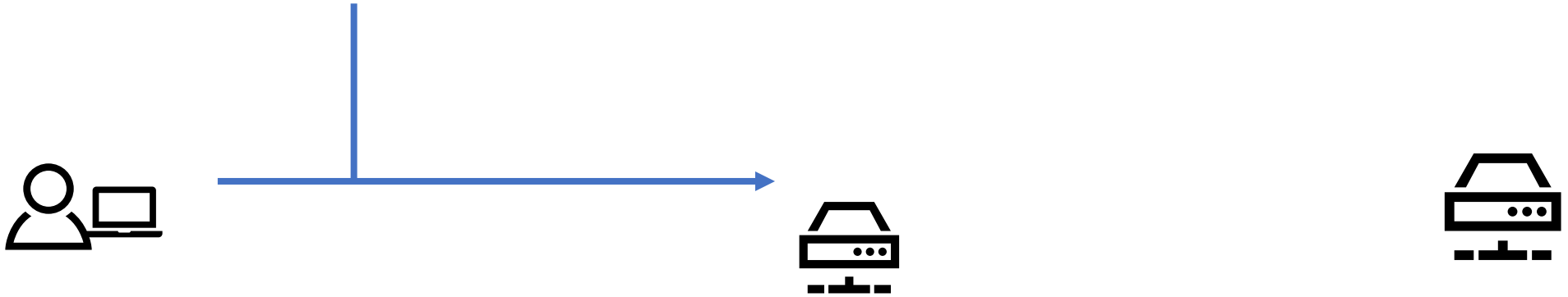
# Antwort sicher ausgeben

# Zurück zur Linkvorschau



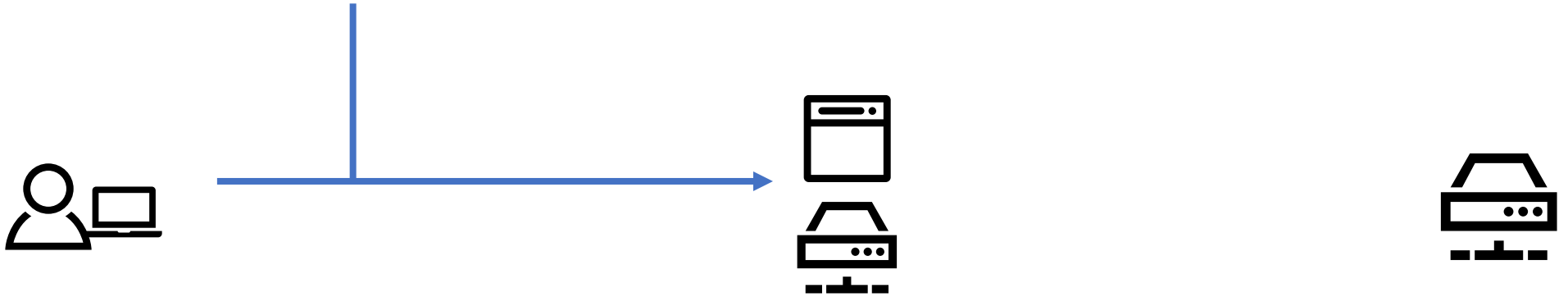
# Szenario: Screenshot

`https://example.org/?url=https://winterkongress.ch`

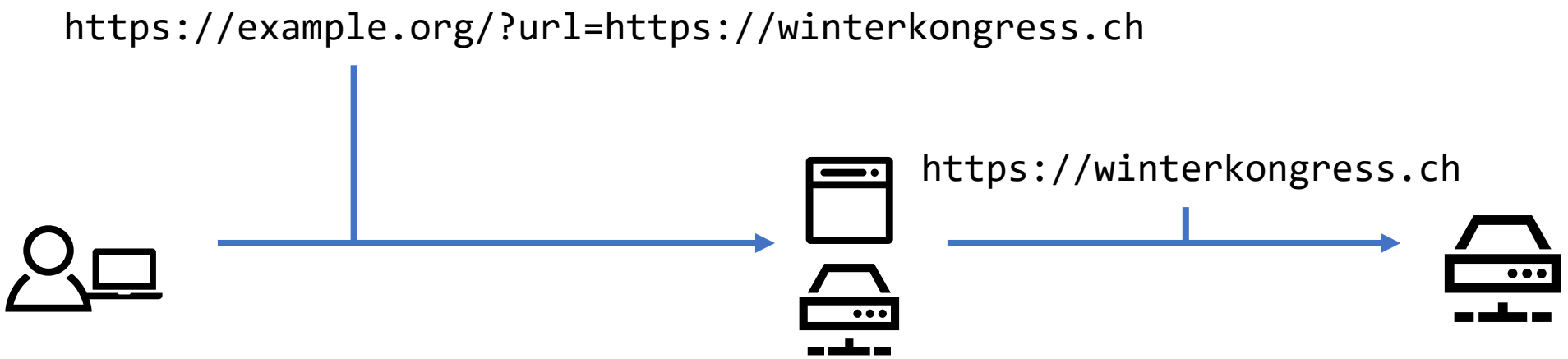


# Szenario: Screenshot

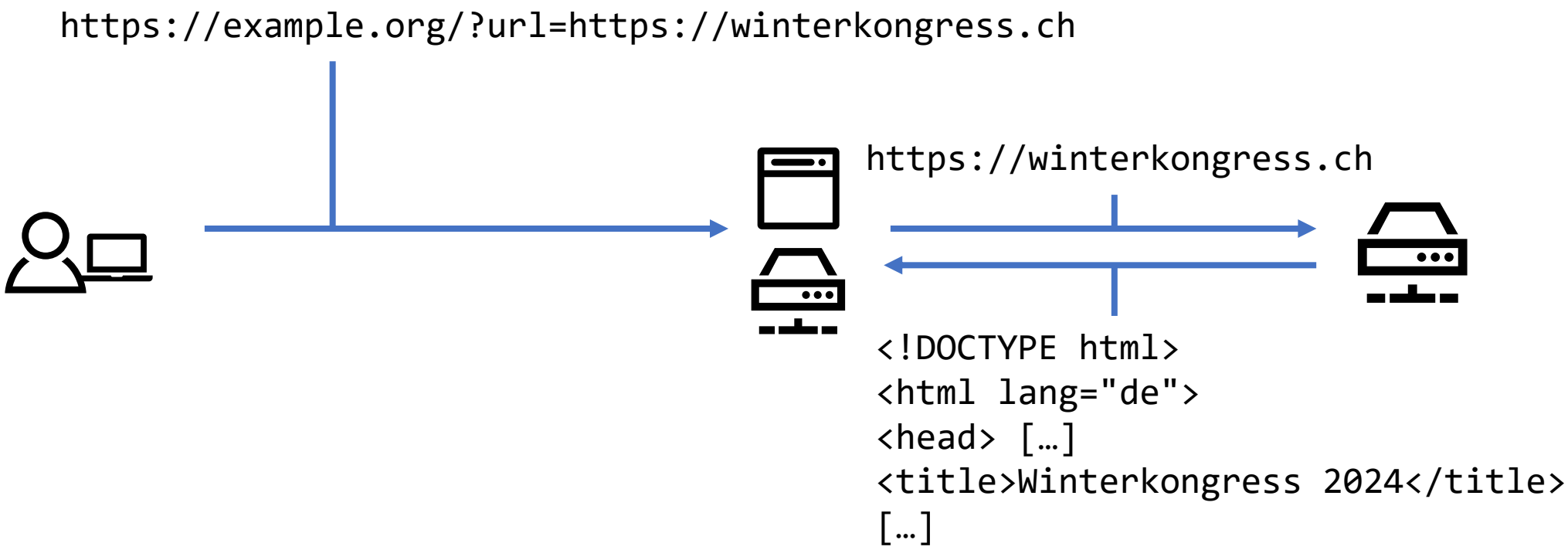
`https://example.org/?url=https://winterkongress.ch`



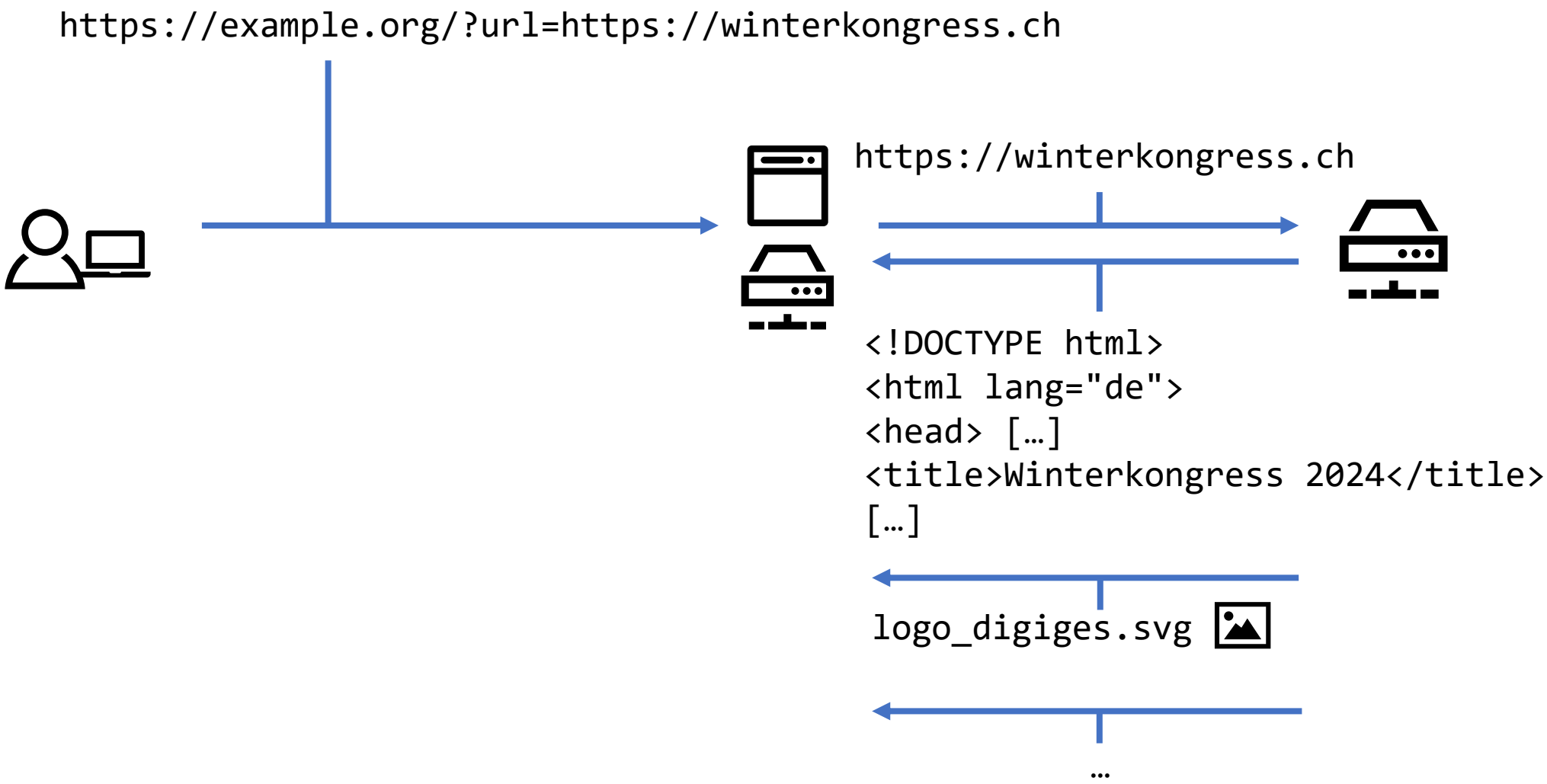
# Szenario: Screenshot



# Szenario: Screenshot

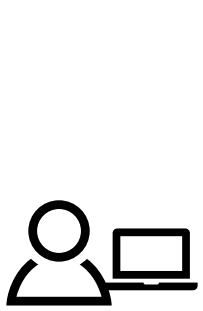


# Szenario: Screenshot

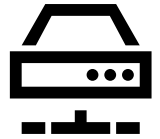


# Szenario: Screenshot

https://example.org/?url=https://winterkongress.ch



https://winterkongress.ch




```
<!DOCTYPE html>
<html lang="de">
<head> […]
<title>Winterkongress 2024</title>
[…]
```

logo_digiges.svg 

…

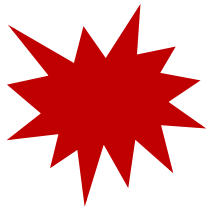
# Server-Side Browser

# Server-Side Browser

- Parsen und führen Antwort der SSR aus!

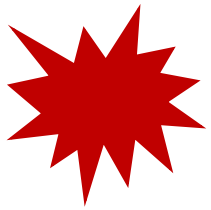
# Server-Side Browser

- Parsen und führen Antwort der SSR aus!
- Browser haben sehr viel Angriffsfläche



# Server-Side Browser

- Parsen und führen Antwort der SSR aus!
- Browser haben sehr viel Angriffsfläche





## Server-Side Browsers: Exploring the Web's Hidden Attack Surface

Marius Musch
TU Braunschweig
Germany

Robin Kirchner
TU Braunschweig
Germany

Max Boll
TU Braunschweig
Germany

Martin Johns
TU Braunschweig
Germany

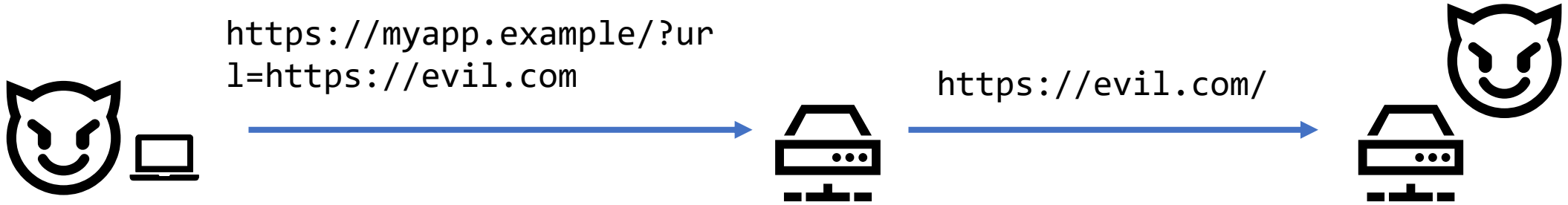
**ABSTRACT**

As websites grow ever more dynamic and load more of their content on the fly, automatically interacting with them via simple tools like *curl* is getting less of an option. Instead, headless browsers with JavaScript support, such as *PhantomJS* and *Puppeteer*, have gained traction on the Web over the last few years. For various use cases like messengers and social networks that display link

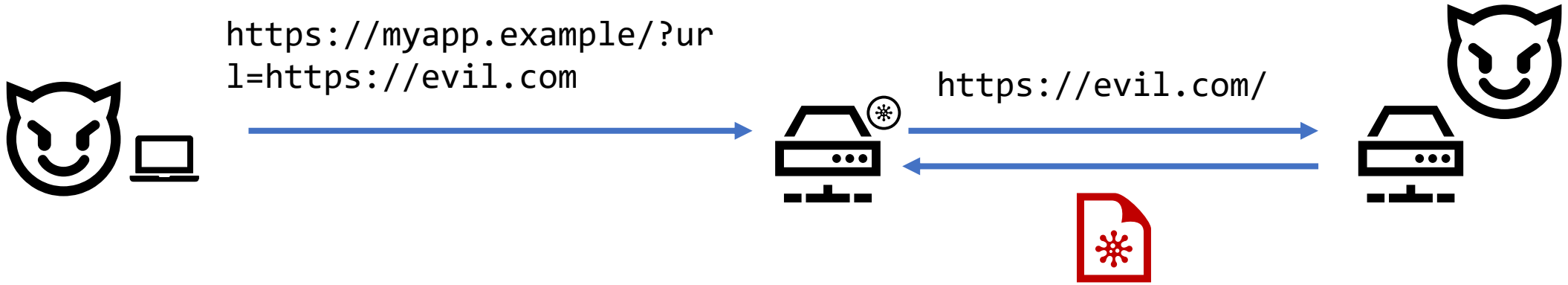
implemented to support this transition. One of these are *server-side requests* (SSRs), i.e., when the web server also acts as a client to fetch additional information from other locations on the Web. Such SSR implementations can often be triggered by unprivileged users of a website, e.g., social networks showing a preview for all user-submitted links.

Traditional SSR implementations resemble tools like *wget* or *curl* and only fetch the main HTML document in a single request without

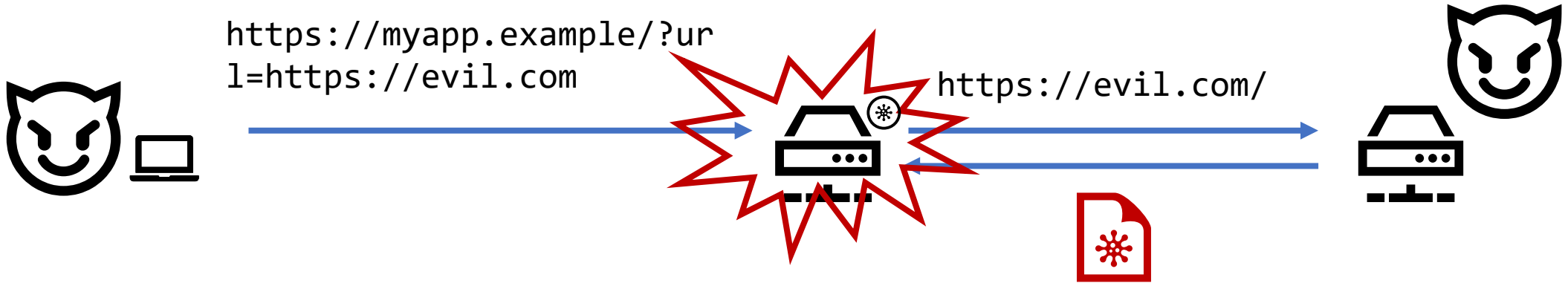
# Generalisiert



# Generalisiert



# Generalisiert



# Clients up-to-date halten!

Insbesondere Browser.

# Takeaways

- Allgemeine Einstellungen beachten (HTTPS only, no redirects, etc.)
- Solide [allow|deny] Liste bauen
- Szenario A: Allow list → Relativ sicher
- Szenario B: Deny list → IP pinning
- Non-application level
  - Auf Netzwerkebene trennen
  - …

# Danke für Eure Aufmerksamkeit!


 @maltee:chaos.social

 malte.wessels@tu-braunschweig.de


Icons:

- CC BY 4.0 DEED / Fontawesome
- Iconmonstr