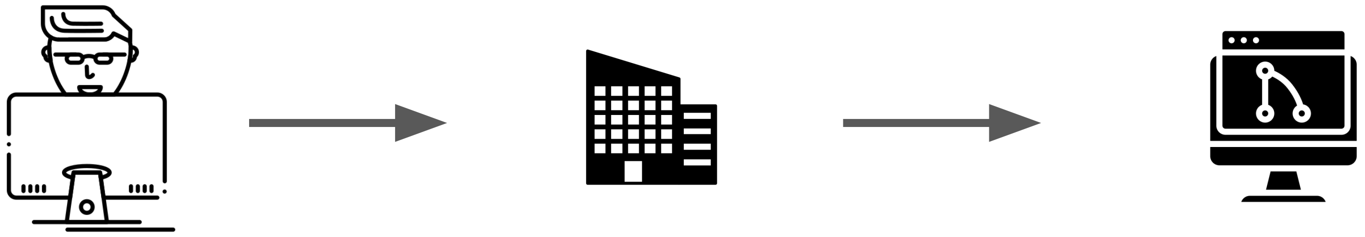
Web Application Security

Shishir Subedi(@shishirsub10**)**

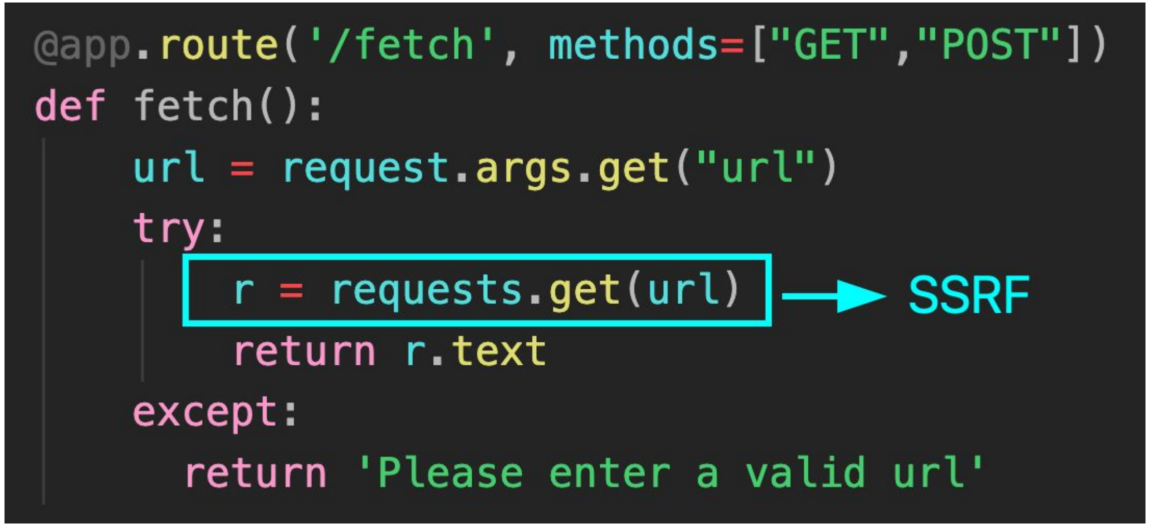
Scenario



Penetration Tester/
Bug-Bounty Hunter

Joined a new
company

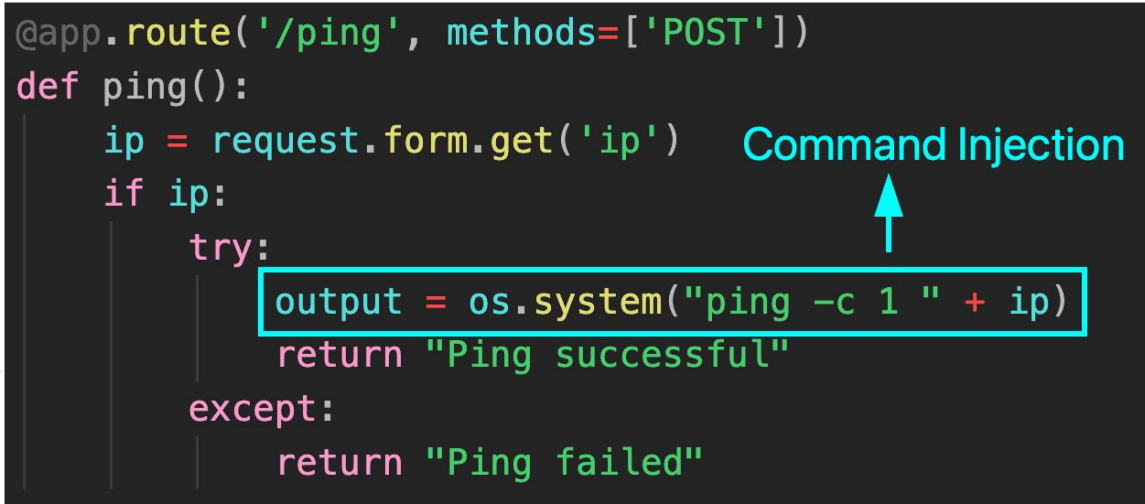
PR ready to be
merged



```python
@app.route('/fetch', methods=["GET","POST"])
def fetch():
    url = request.args.get("url")
    try:
        r = requests.get(url)
        return r.text
    except:
        return 'Please enter a valid url'
```

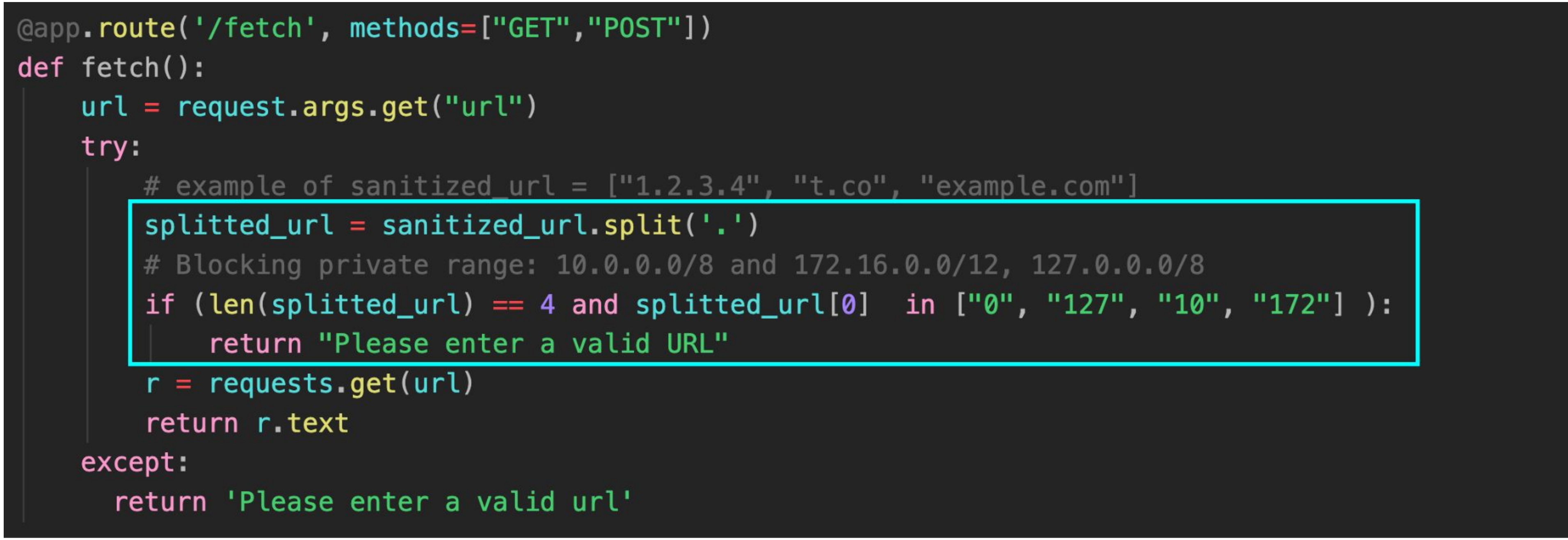
Server Side Request Forgery

Command injection



```python
@app.route('/ping', methods=['POST'])
def ping():
    ip = request.form.get('ip')
    if ip:
        try:
            output = os.system("ping -c 1 " + ip)
            return "Ping successful"
        except:
            return "Ping failed"
```

Preventing SSRF on Application

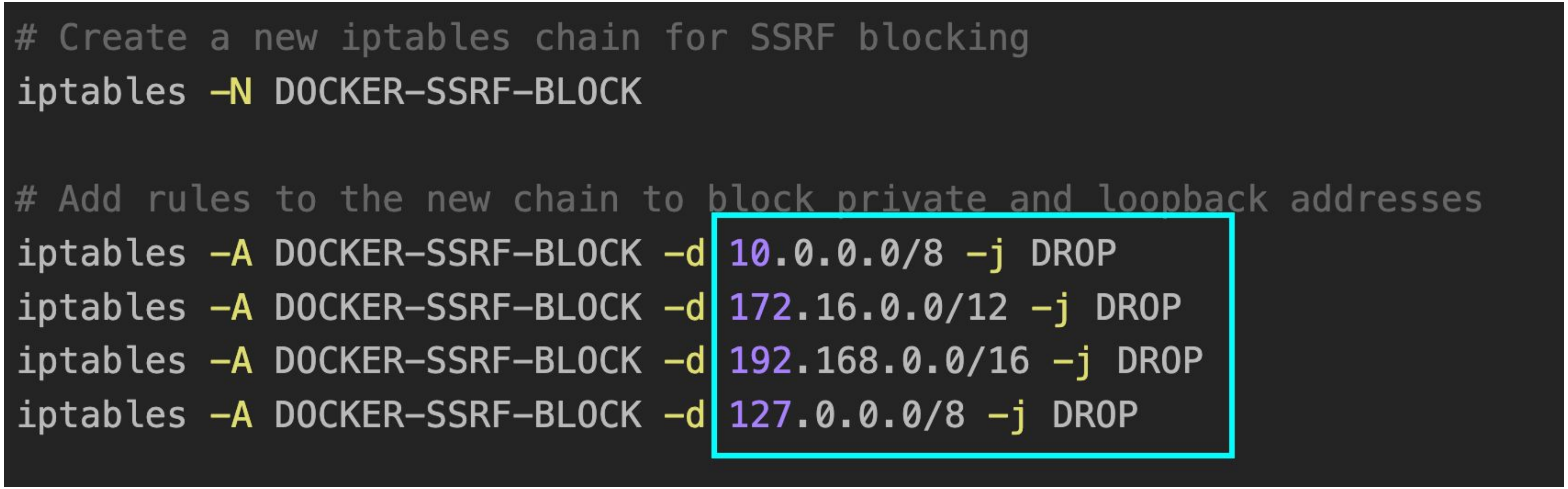


```python
@app.route('/fetch', methods=["GET","POST"])
def fetch():
    url = request.args.get("url")
    try:
        # example of sanitized_url = ["1.2.3.4", "t.co", "example.com"]
        splitted_url = sanitized_url.split('.')
        # Blocking private range: 10.0.0.0/8 and 172.16.0.0/12, 127.0.0.0/8
        if (len(splitted_url) == 4 and splitted_url[0]  in ["0", "127", "10", "172"] ):
            return "Please enter a valid URL"
        r = requests.get(url)
        return r.text
    except:
      return 'Please enter a valid url'
```

All the IP addresses starting from following number are blocked:

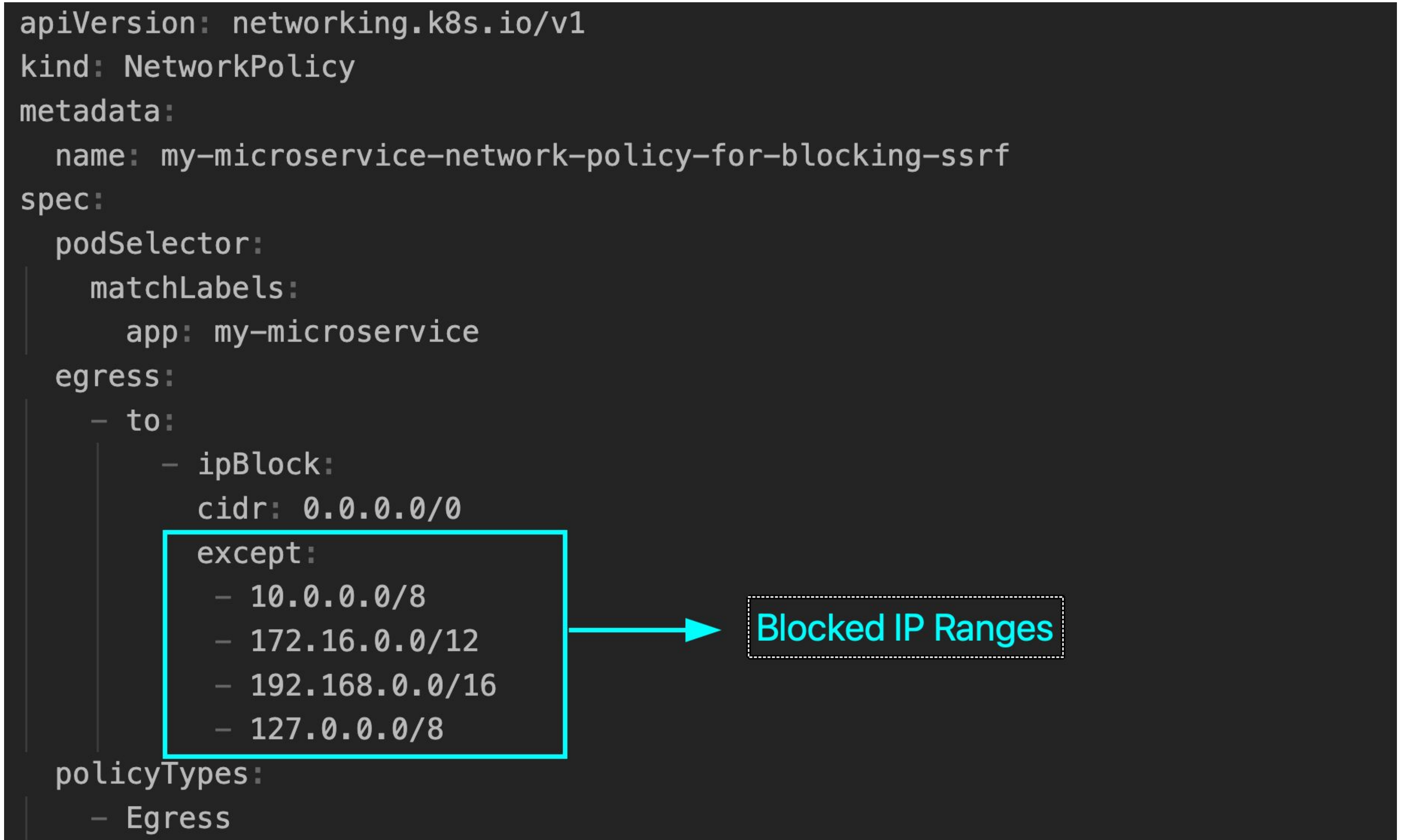
- 10 (Private Range 10.0.0.0/8)
- 0 (Localhost 0.0.0.0)
- 127 (Localhost 127.0.0.1)
- 172 (Private Range 172.16.0.0/12)

Preventing SSRF on Docker



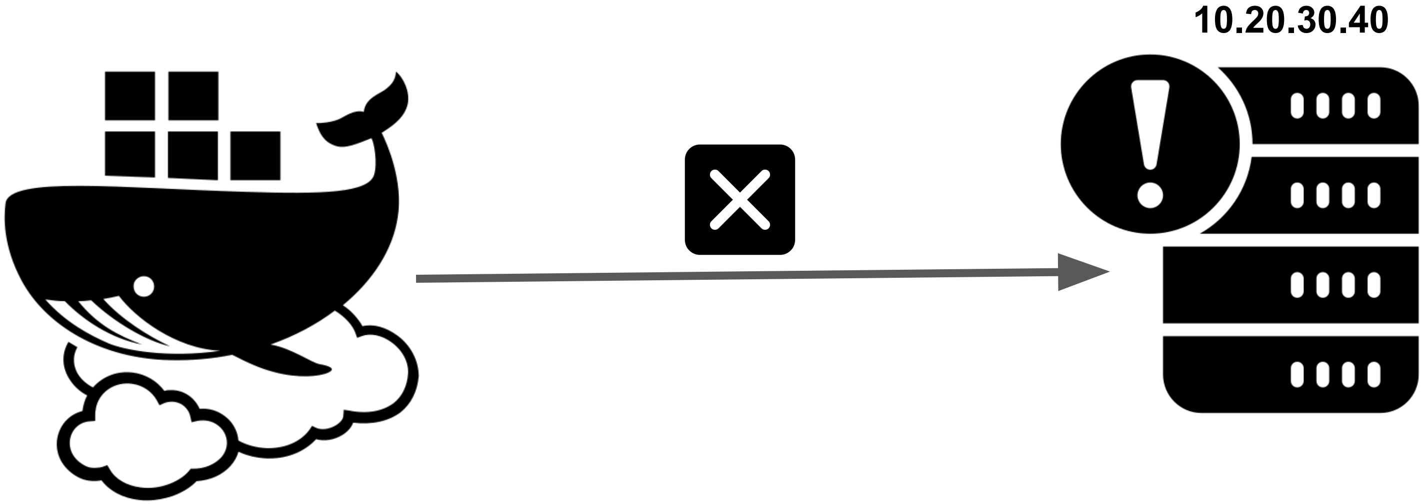
```
# Create a new iptables chain for SSRF blocking
iptables -N DOCKER-SSRF-BLOCK

# Add rules to the new chain to block private and loopback addresses
iptables -A DOCKER-SSRF-BLOCK -d 10.0.0.0/8 -j DROP
iptables -A DOCKER-SSRF-BLOCK -d 172.16.0.0/12 -j DROP
iptables -A DOCKER-SSRF-BLOCK -d 192.168.0.0/16 -j DROP
iptables -A DOCKER-SSRF-BLOCK -d 127.0.0.0/8 -j DROP
```

Preventing SSRF on Kubernetes



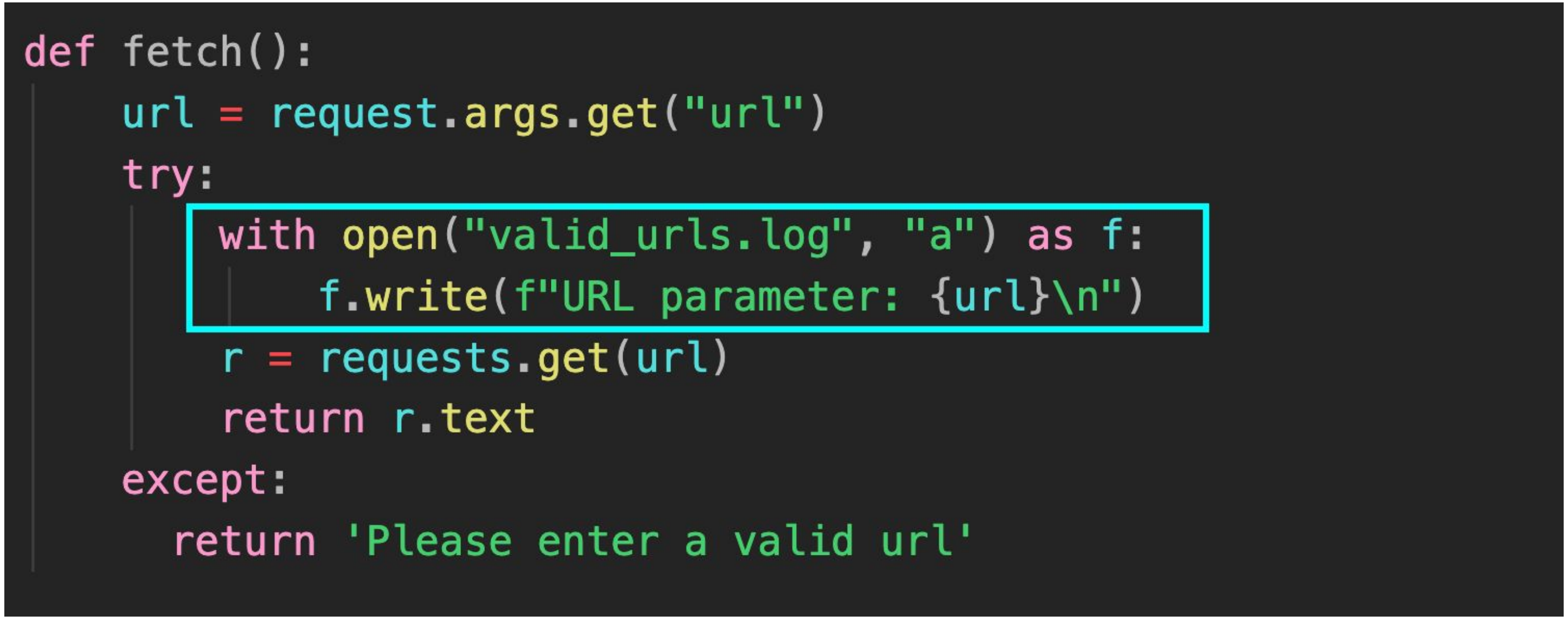
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-microservice-network-policy-for-blocking-ssrf
spec:
  podSelector:
    matchLabels:
      app: my-microservice
  egress:
    - to:
      - ipBlock:
        cidr: 0.0.0.0/0
        except:
          - 10.0.0.0/8
          - 172.16.0.0/12
          - 192.168.0.0/16
          - 127.0.0.0/8
  policyTypes:
    - Egress
```

ISSUE ON Production



Application was not able to talk to the database server

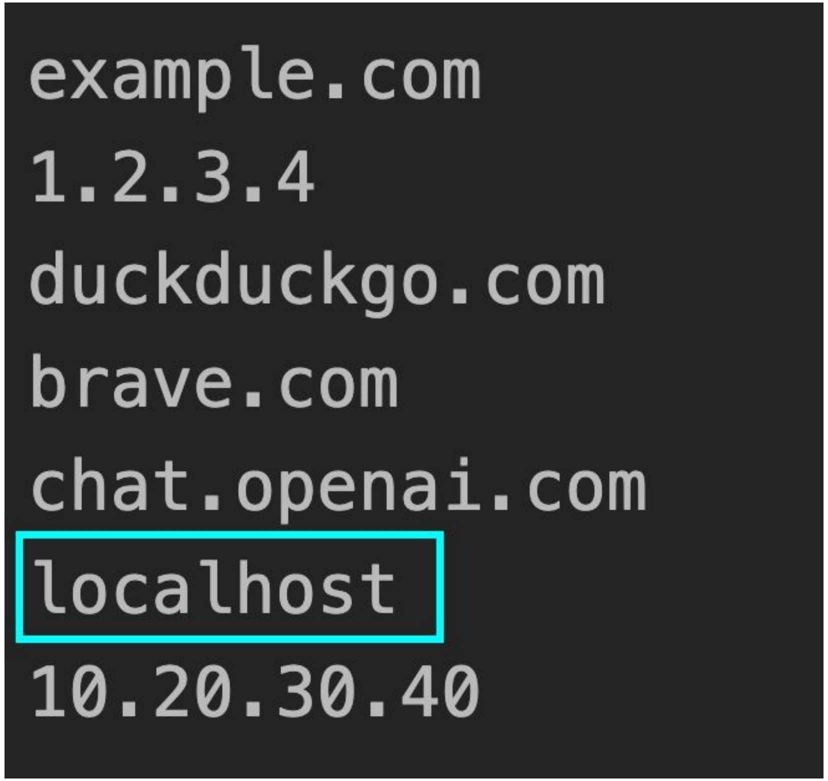
Logging and Monitoring



```
def fetch():
    url = request.args.get("url")
    try:
        with open("valid_urls.log", "a") as f:
            f.write(f"URL parameter: {url}\n")
        r = requests.get(url)
        return r.text
    except:
      return 'Please enter a valid url'
```

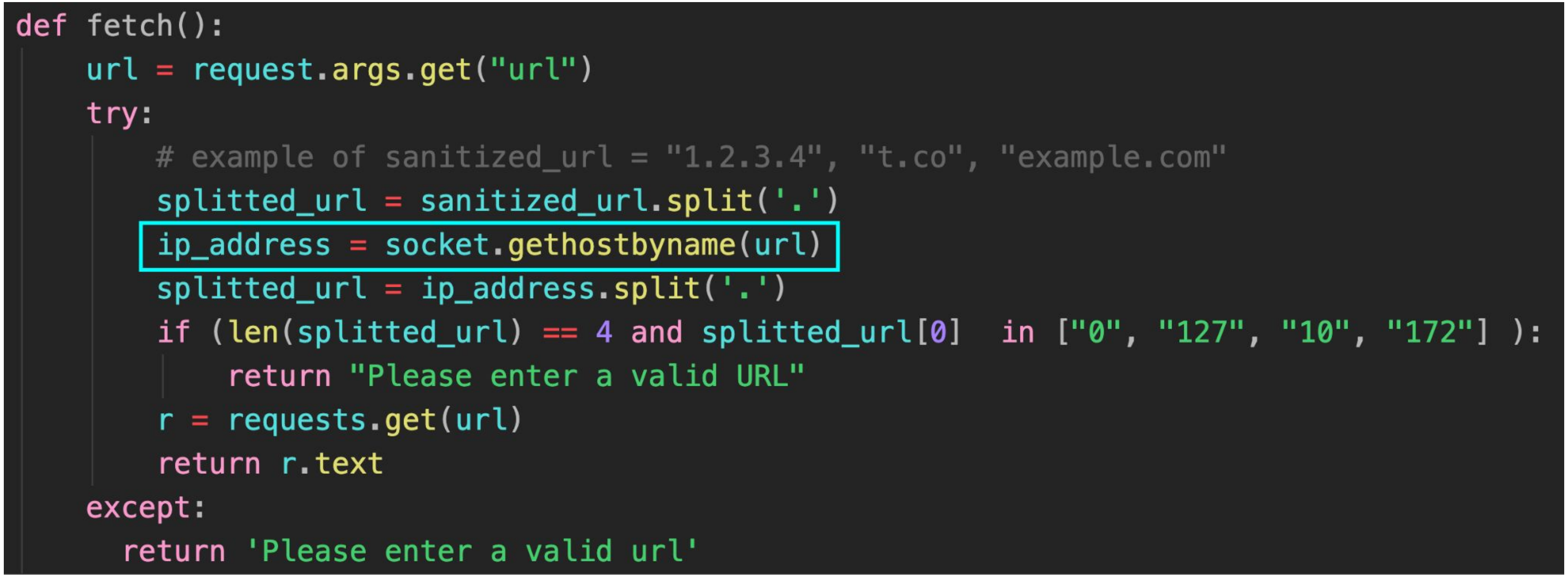
- All the URL are logged and imported to SIEM solution
- Alert is set to detect SSRF issues

Alert from SIEM Solution at 2 AM



```
example.com
1.2.3.4
duckduckgo.com
brave.com
chat.openai.com
localhost
10.20.30.40
```

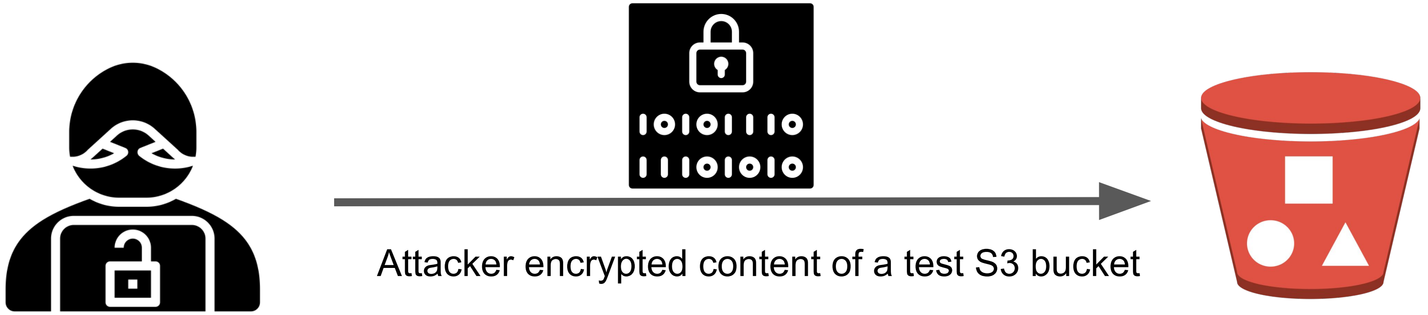
Fixing the bypass on the Application



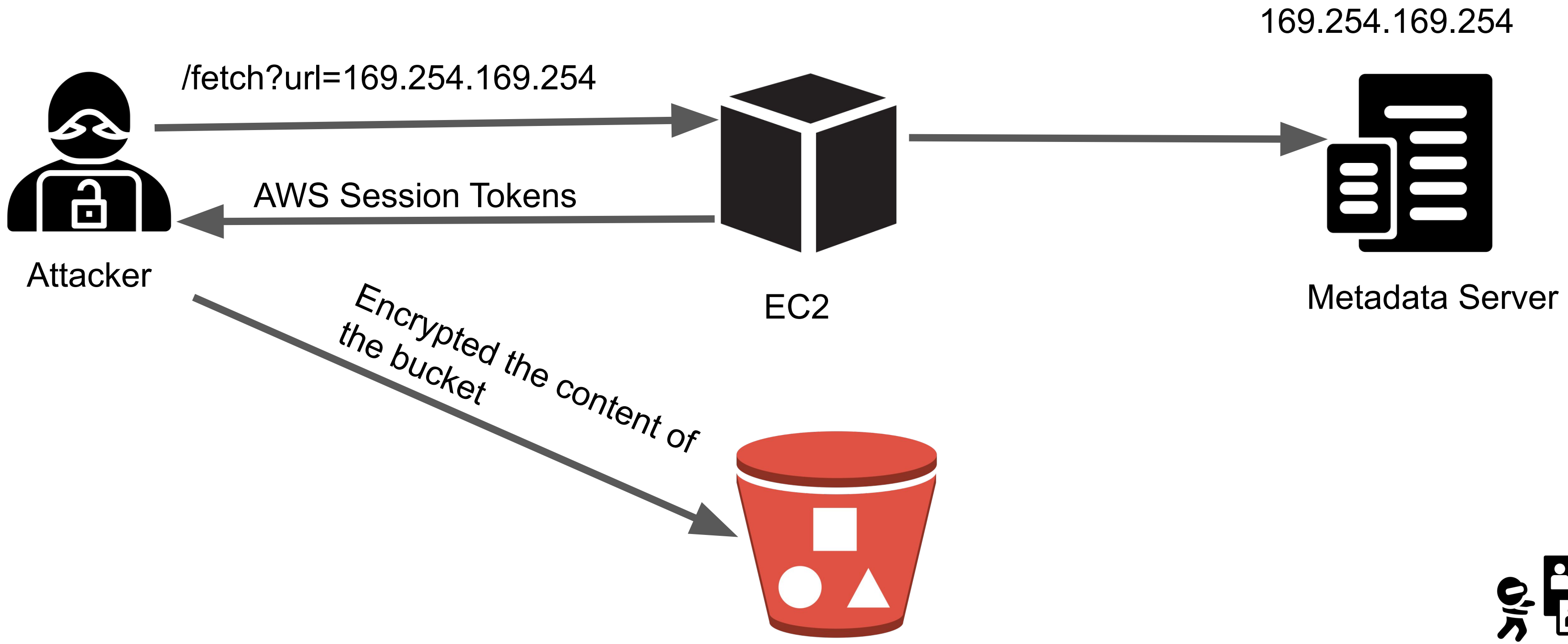
```
def fetch():
    url = request.args.get("url")
    try:
        # example of sanitized_url = "1.2.3.4", "t.co", "example.com"
        splitted_url = sanitized_url.split('.')
        ip_address = socket.gethostbyname(url)
        splitted_url = ip_address.split('.')
        if (len(splitted_url) == 4 and splitted_url[0]  in ["0", "127", "10", "172"] ):
            return "Please enter a valid URL"
        r = requests.get(url)
        return r.text
    except:
      return 'Please enter a valid url'
```

All the domain are converted to IPs before checking for SSRF.

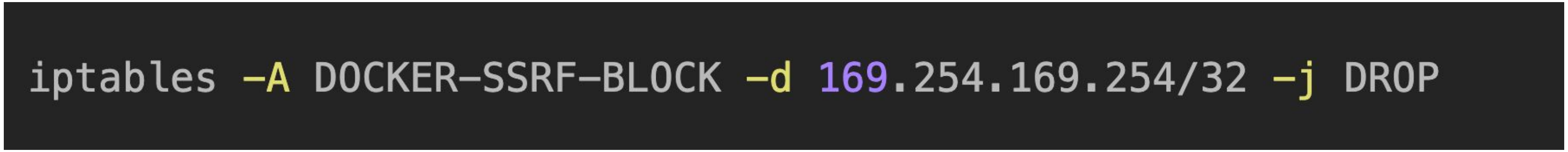
Another Security incident



Forensics of the incident

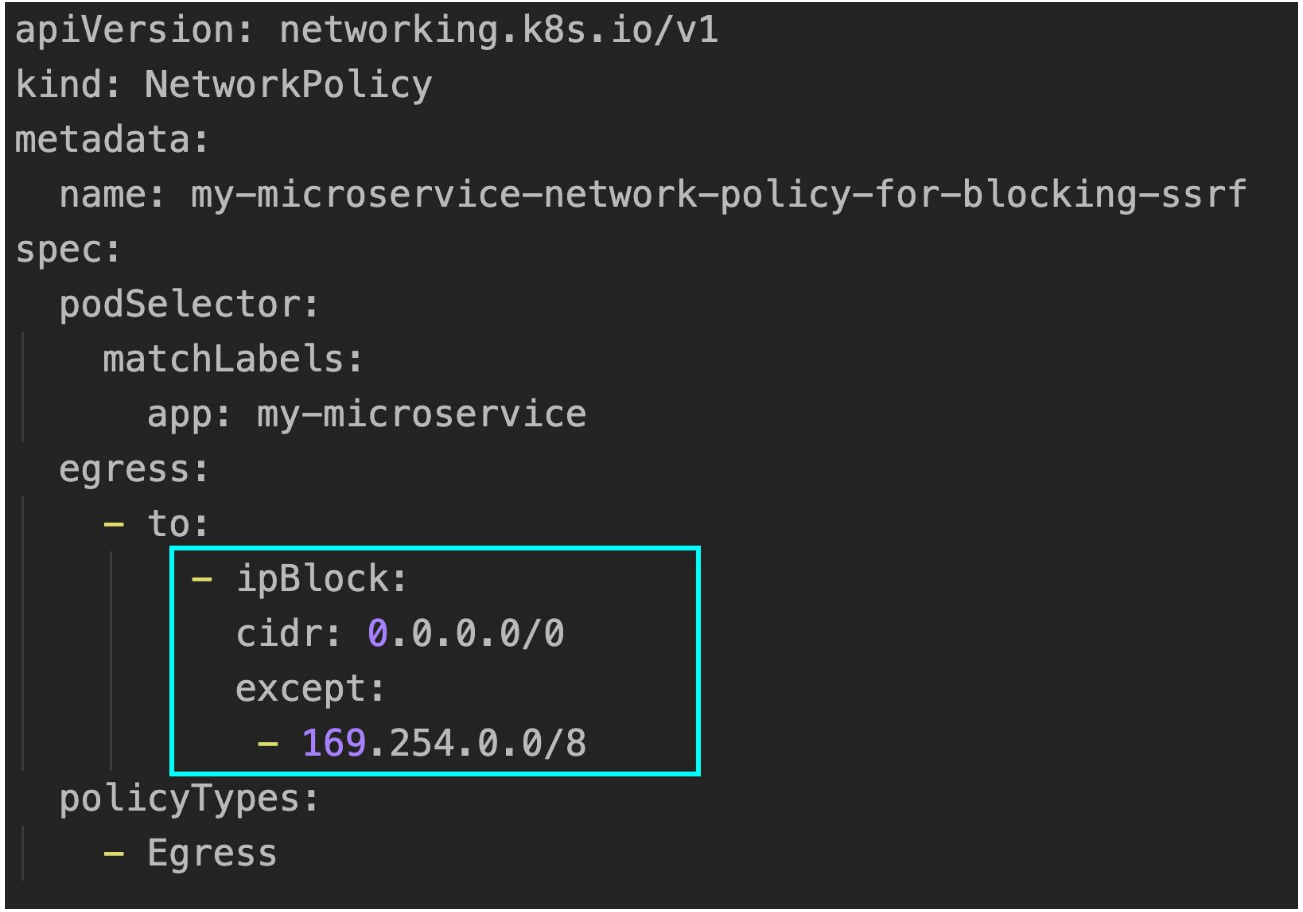

169.254.169.254
/fetch?url=169.254.169.254
AWS Session Tokens
Attacker
EC2
Metadata Server
Encrypted the content of the bucket

Restricting Access to metadata Server on Docker



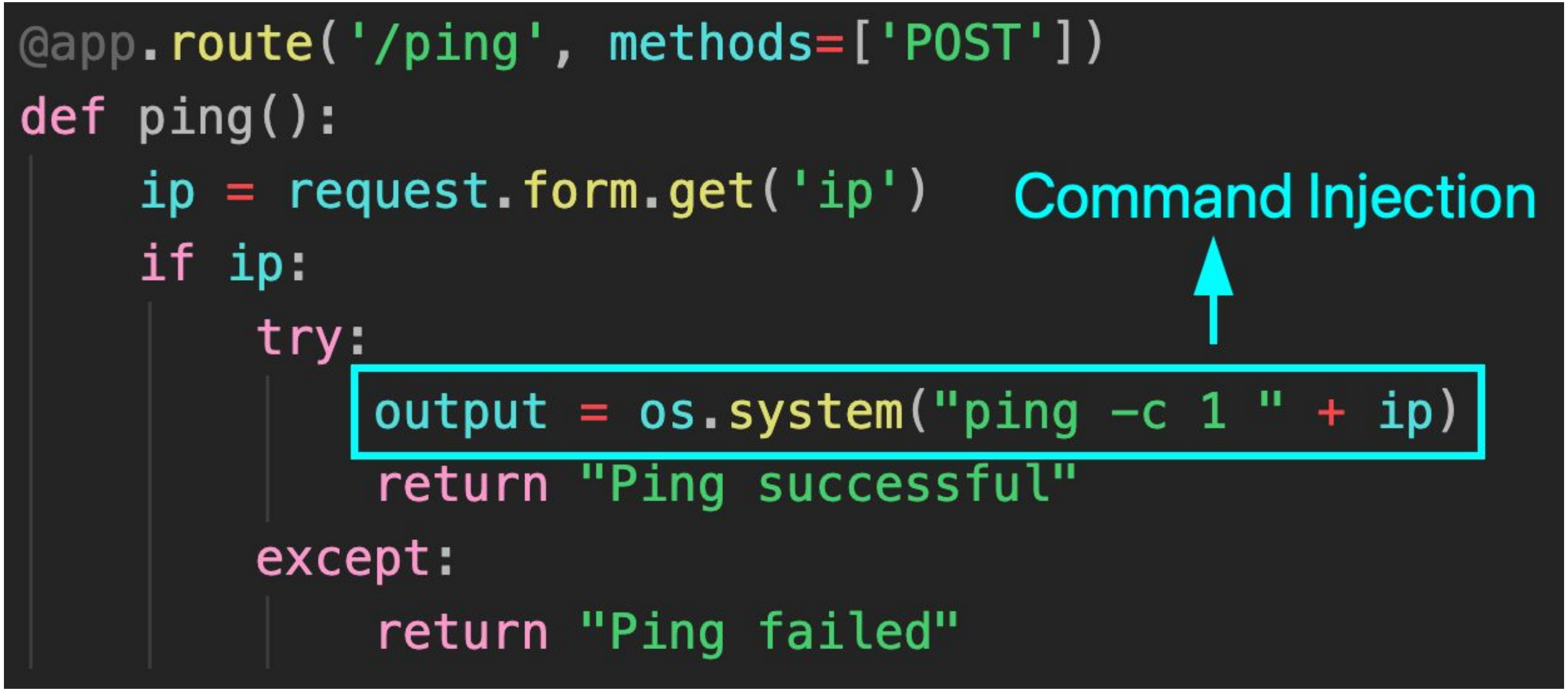
```
iptables -A DOCKER-SSRF-BLOCK -d 169.254.169.254/32 -j DROP
```

Restricting Access to metadata Server on Kubernetes



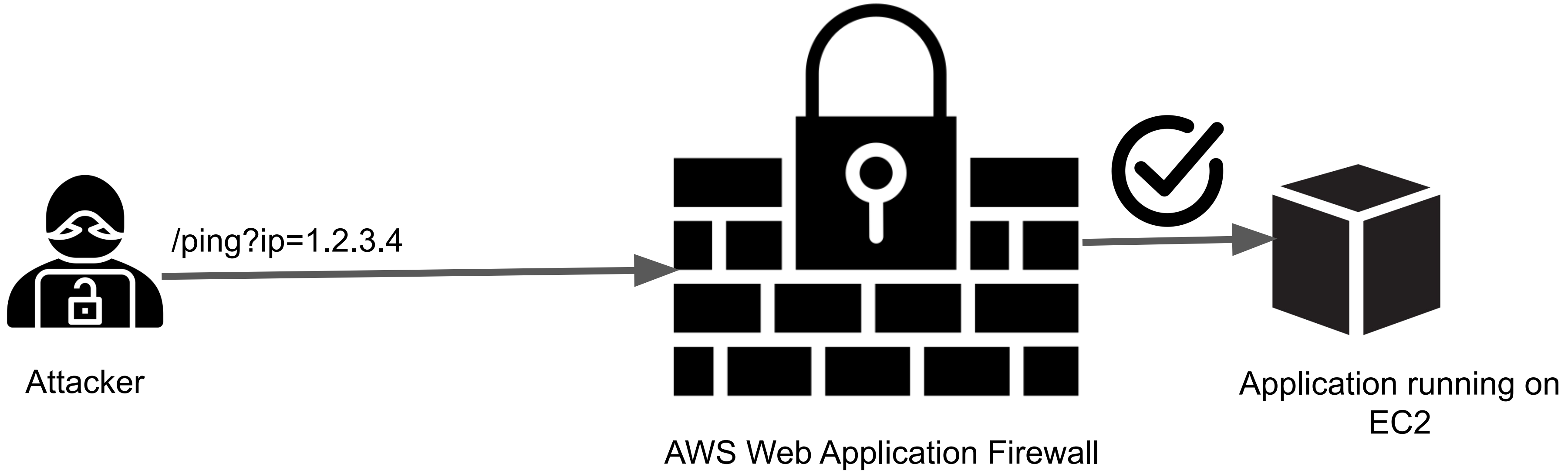
```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-microservice-network-policy-for-blocking-ssrf
spec:
  podSelector:
    matchLabels:
      app: my-microservice
  egress:
    - to:
        - ipBlock:
          cidr: 0.0.0.0/0
          except:
            - 169.254.0.0/8
  policyTypes:
    - Egress
```

Revisiting Command Injection



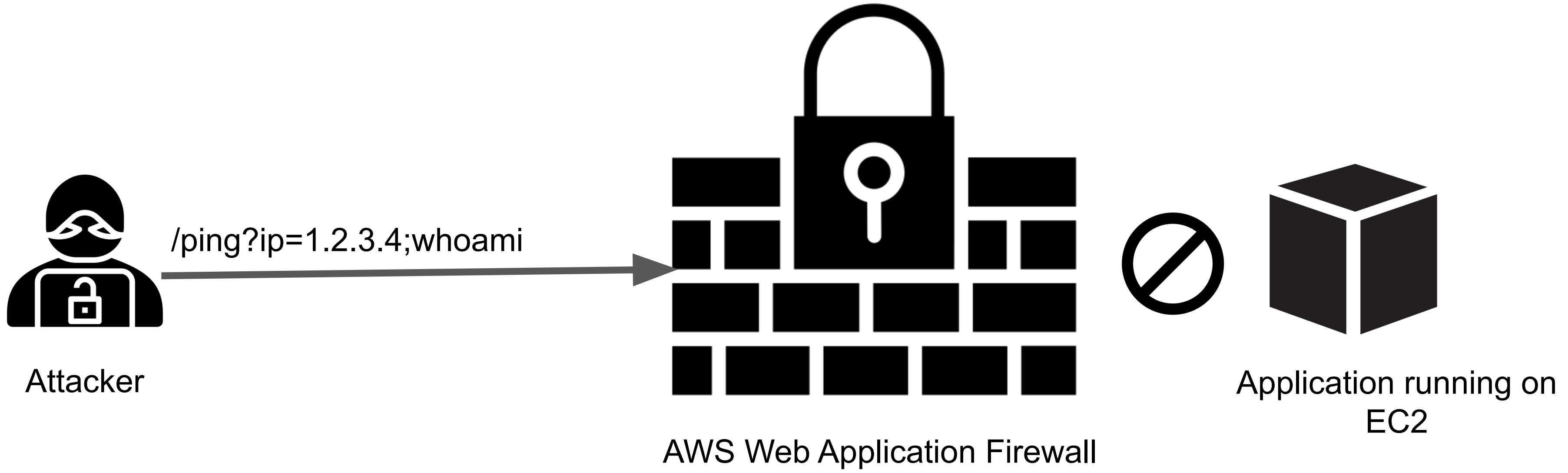
```
@app.route('/ping', methods=['POST'])
def ping():
    ip = request.form.get('ip')
    if ip:
        try:
            output = os.system("ping -c 1 " + ip)
            return "Ping successful"
        except:
            return "Ping failed"
```

Partial Fix



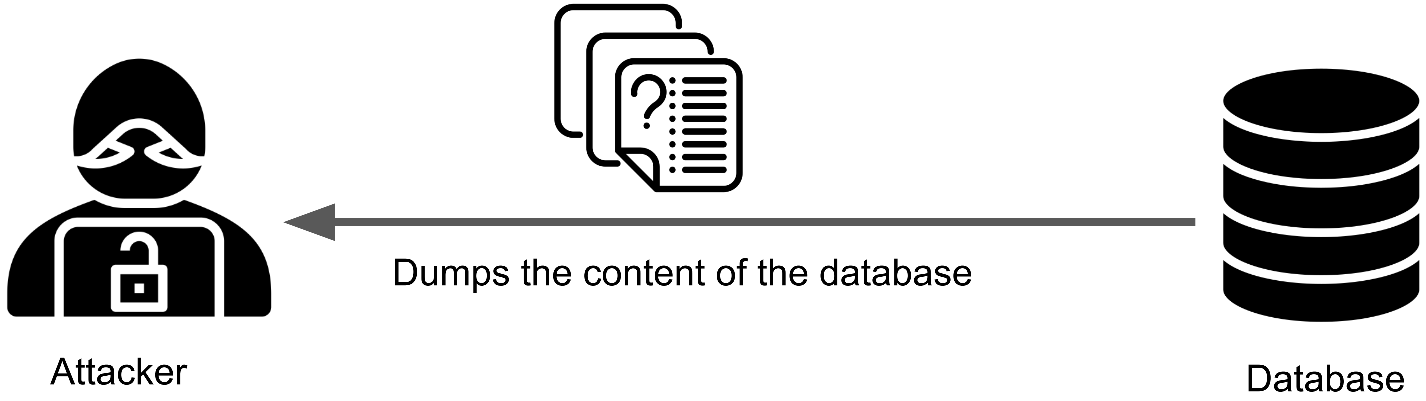
Attacker
/ping?ip=1.2.3.4
AWS Web Application Firewall
Application running on EC2

Partial Fix

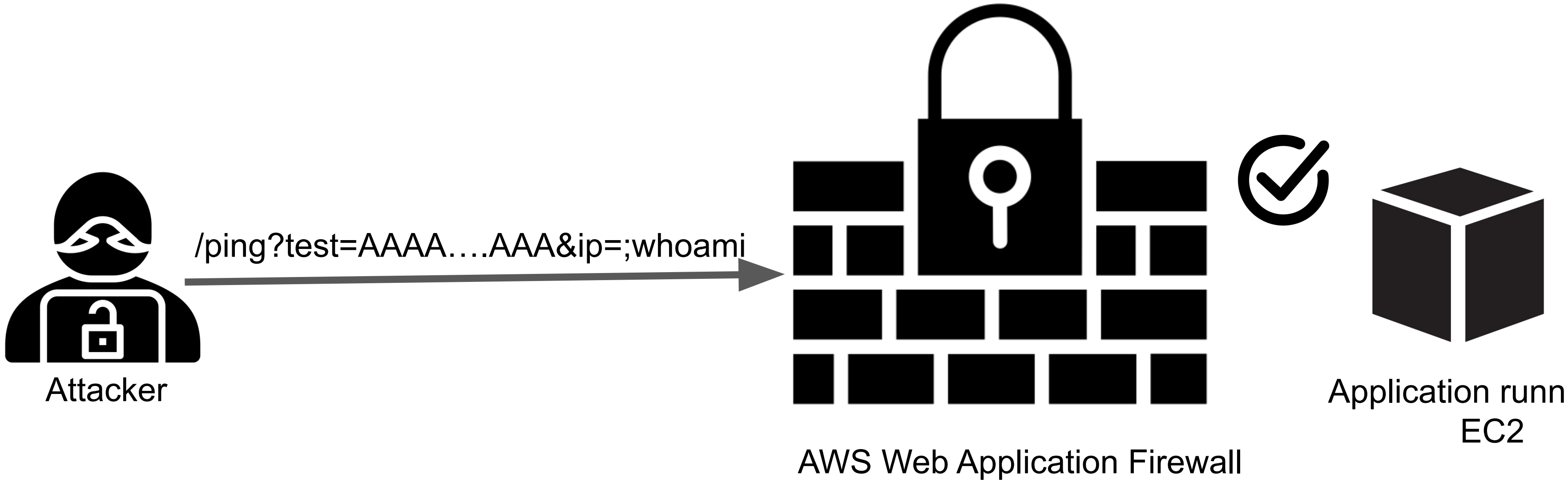


/ping?ip=1.2.3.4;whoami
Attacker
AWS Web Application Firewall
Application running on EC2

Yet Another Security incident



Forensics of the incident



Know your tools' limitations:

- AWS WAF only checks for 8KB of POST data

Never run web application as root user

THANK YOU!!!

Any Questions?