Follow the Trace:

How Traditional AppSec Tools Have Failed Us

Kennedy Toomey

Application Security Researcher & Advocate @ Datadog

Meet Trace

Objective

Let the journey begin...

What is a trace?

What is a trace?

Detailed record of an application's actions and events

What is a trace?

Detailed record of an application's actions and events

Helps us understand the execution flow of an application

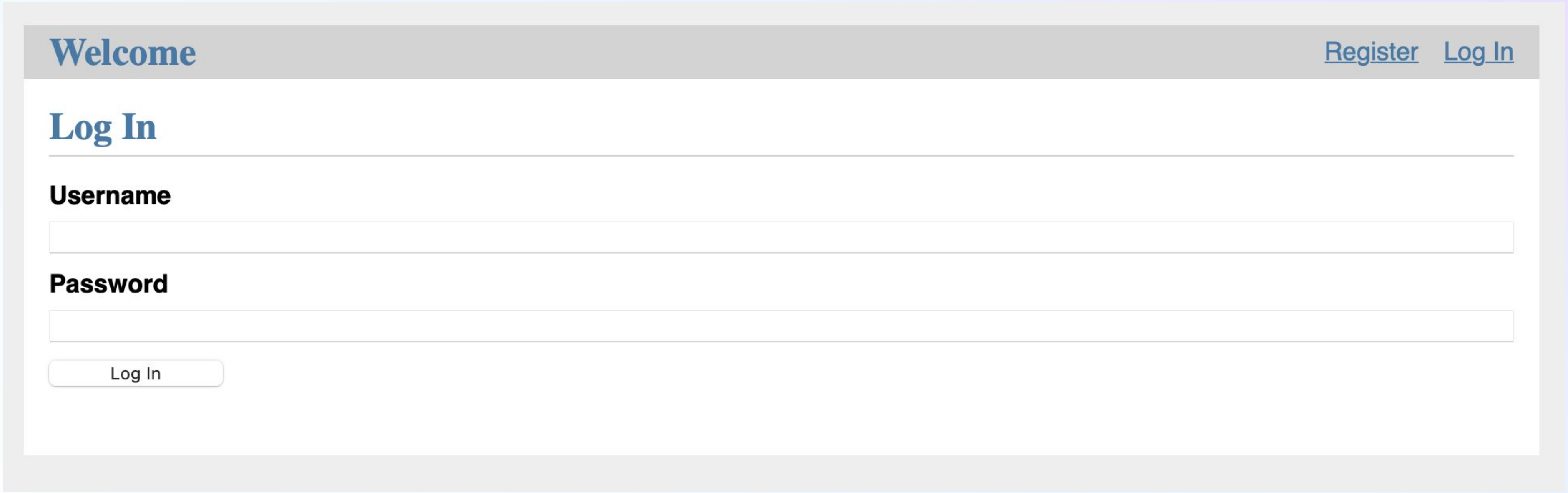
What is a trace?

Detailed record of an application’s actions and events

Helps us understand the execution flow of an application

Can identify performance issues or pinpoint the root cause of errors

Basic Login Page



Welcome

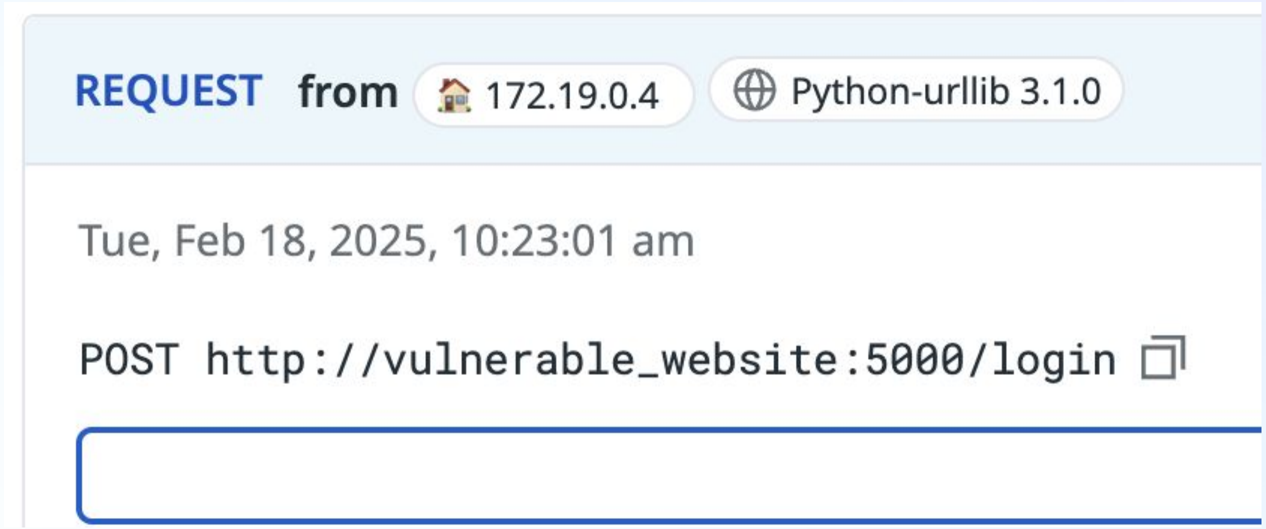
Register Log In

Log In

Username

Password

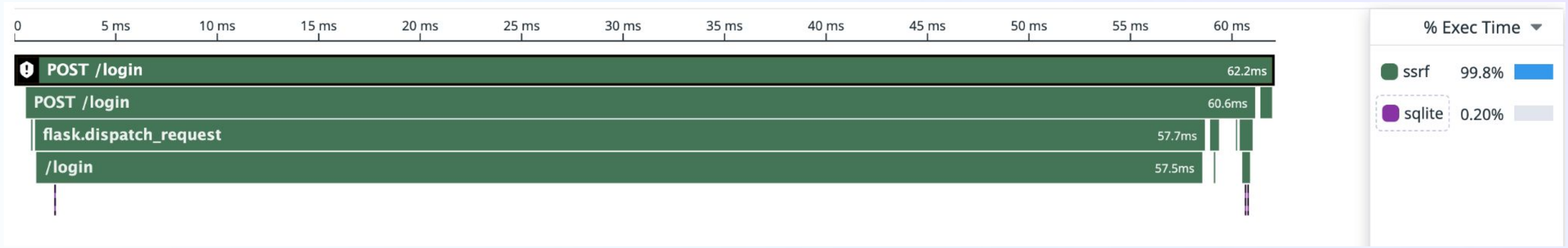
Our First Request


REQUEST from 🏠 172.19.0.4 Python-urllib 3.1.0

Tue, Feb 18, 2025, 10:23:01 am

```
POST http://vulnerable_website:5000/login
```

Login Flame Graph



Capturing Runtime Behavior & Interactions

Capturing Runtime Behavior & Interactions

Runtime behavior

- What your application is doing right now
- Tracks function calls and the code paths
- Latency, throughput, and error patterns

Capturing Runtime Behavior & Interactions

Runtime behavior

- What your application is doing right now
- Tracks function calls and the code paths
- Latency, throughput, and error patterns

Interactions across components

- Requests flowing through services
- Inbound & outbound traffic
- Distributed tracing

Capturing Runtime Behavior & Interactions

Runtime behavior

- What your application is doing right now
- Tracks function calls and the code paths
- Latency, throughput, and error patterns

Interactions across components

- Requests flowing through services
- Inbound & outbound traffic
- Distributed tracing

What gets captured?

- Timestamps
- Spans & traces
- Metadata

Capturing Runtime Behavior & Interactions

Runtime behavior

- What your application is doing right now
- Tracks function calls
- Latency, throughput, and error patterns

Interactions across components

- Requests flowing through services
- Inbound & outbound traffic
- Distributed tracing

What gets captured?

- Timestamps
- Spans & traces
- Metadata

Why it matters

- Performance tuning
- Troubleshooting
- Baseline for application behavior

Application Security Basics

Application Security Testing

SAST

S = Static

- Scans the code
  - Pattern matching

Application Security Testing

SAST

S = Static

- Scans the code
  - Pattern matching

DAST

D = Dynamic

- Scans the running application
  - Simulates malicious attacks on the application

Application Security Testing

SAST

S = Static

- Scans the code
  - Pattern matching

DAST

D = Dynamic

- Scans the running application
  - Simulates malicious attacks on the application

IAST

I = Interactive

- Analyzes the running application's behavior
  - Follows the data flow

Application Layer Protections

WAF

Web Application Firewall

- Outside the application
  - Great: Threat Landscape Visibility
  - Good: DDoS Protection
  - Bad: Exploit Prevention

Application Layer Protections

WAF

Web Application Firewall

- Outside the application
  - Great: Threat Landscape Visibility
  - Good: DDoS Blocking
  - Bad: Exploit Prevention

RASP

Runtime Application Self-Protection

- Inside the application
  - Great: Exploit Prevention
  - Great: Threat Landscape Visibility
  - Bad: DDoS Protection

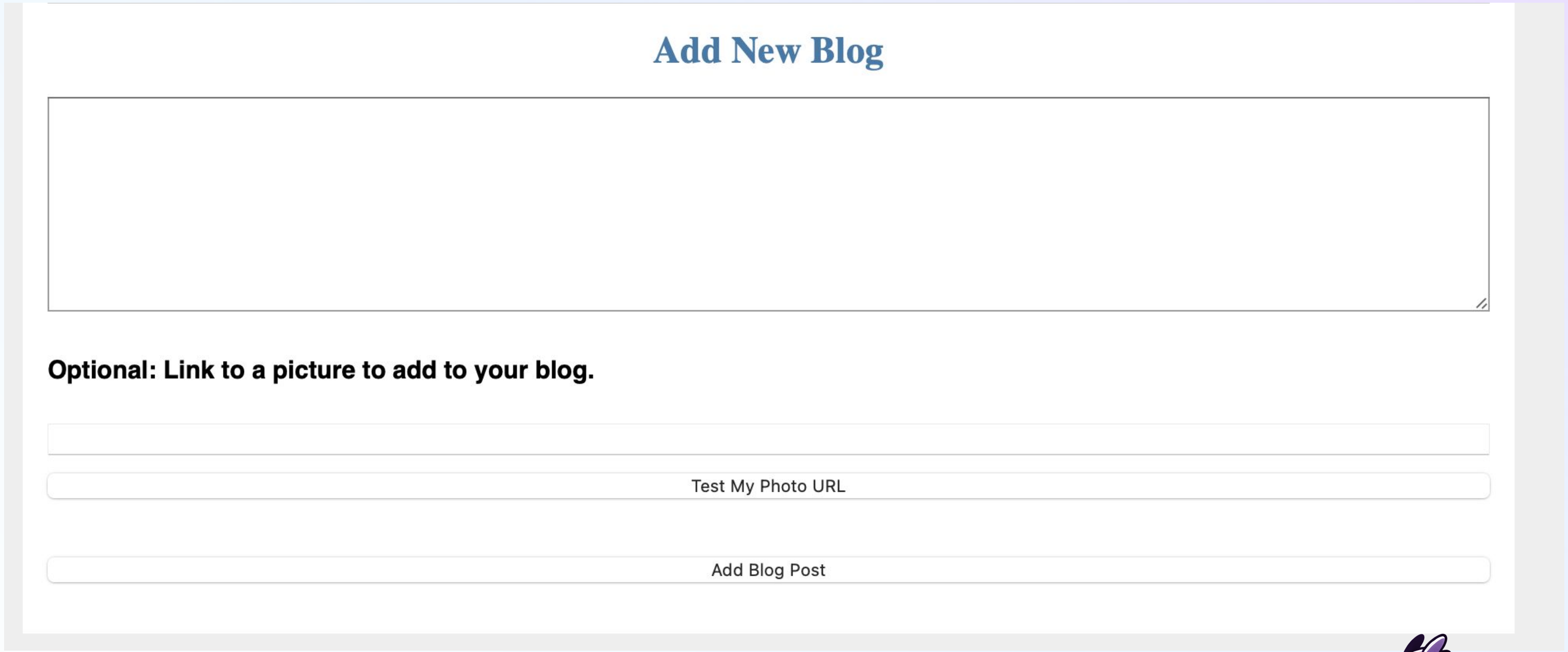
Why Runtime?

Added context allows for more precise results

18% of critical vulnerabilities are worth prioritizing

Step-by-Step Adventure

Targeted Service


Add New Blog

Optional: Link to a picture to add to your blog.

Oh no...

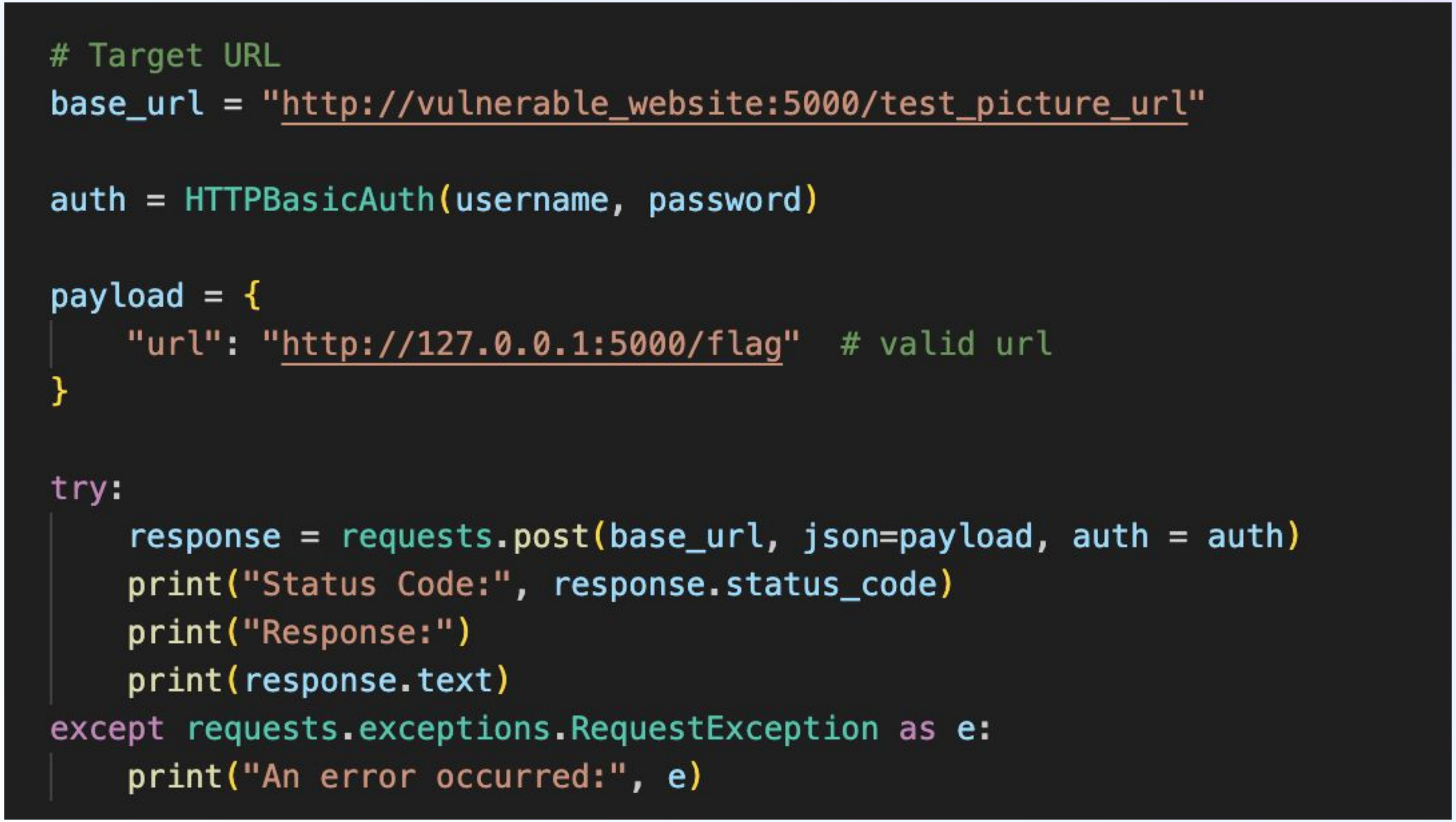
We’ve encountered a threat!

Server-Side Request Forgery (SSRF)

Allows the attacker to cause the **server-side application** to make requests to an **unintended location**

Common entry points include sources that do not check user input for unexpected data

The Attack



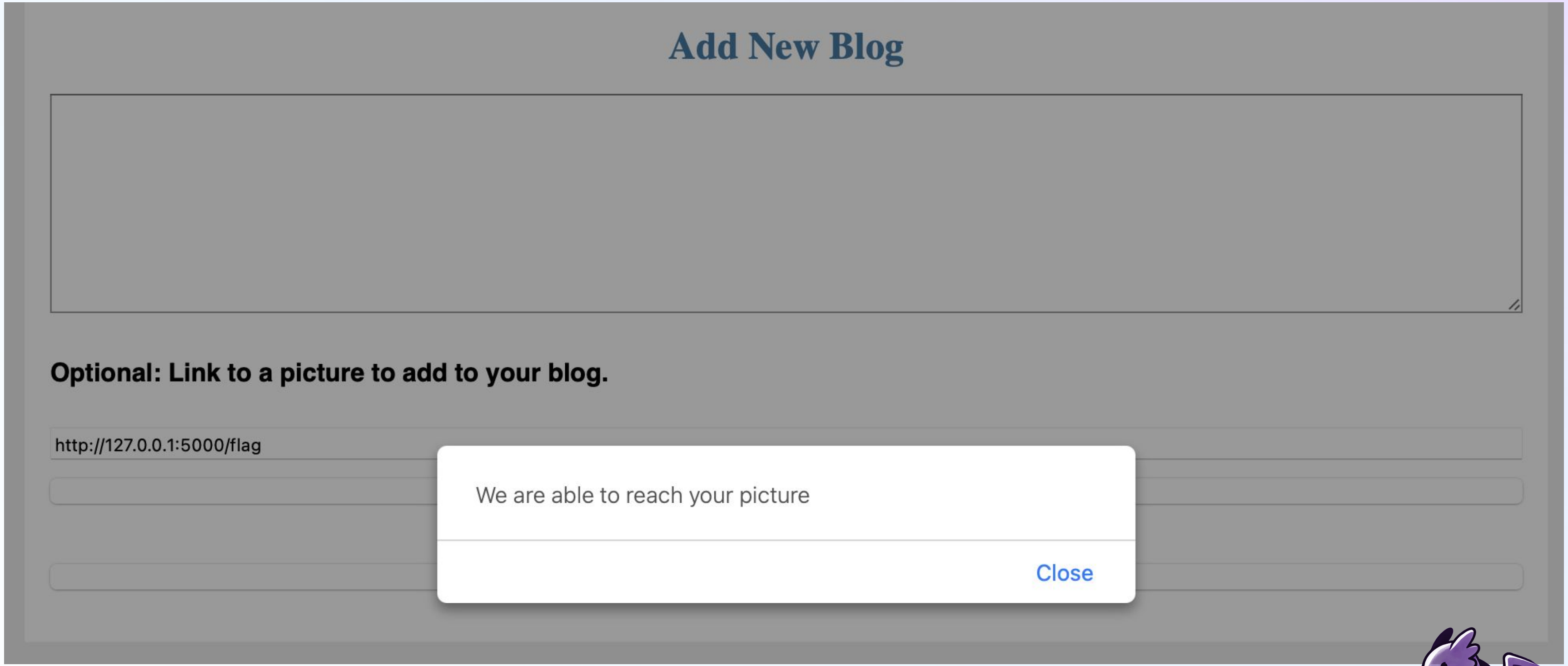
```
# Target URL
base_url = "http://vulnerable_website:5000/test_picture_url"

auth = HTTPBasicAuth(username, password)

payload = {
    "url": "http://127.0.0.1:5000/flag"  # valid url
}


try:
    response = requests.post(base_url, json=payload, auth = auth)
    print("Status Code:", response.status_code)
    print("Response:")
    print(response.text)
except requests.exceptions.RequestException as e:
    print("An error occurred:", e)
```

The Result



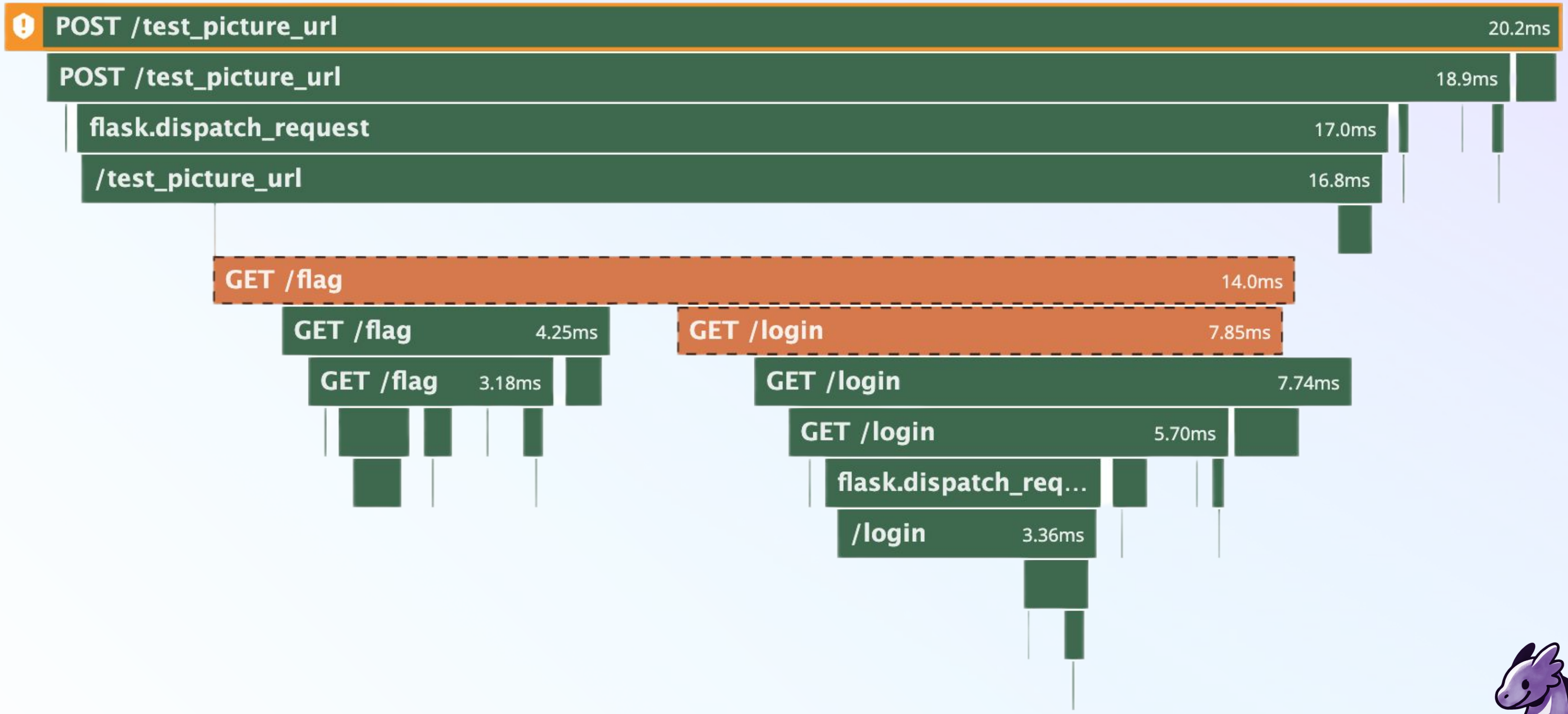
Add New Blog

Optional: Link to a picture to add to your blog.

http://127.0.0.1:5000/flag

We are able to reach your picture

Flame Graph from the Attack



RASP to the Rescue!

RASP Process

Embedded in the app

Monitors behavior

Analyzes in real time

Detects attacks

Blocks attacks and sends an alert

RASP Configuration

Deployment Mode

- Choose whether you want to monitor or block attacks

Policy Tuning

- Use OOTB detections or create your own

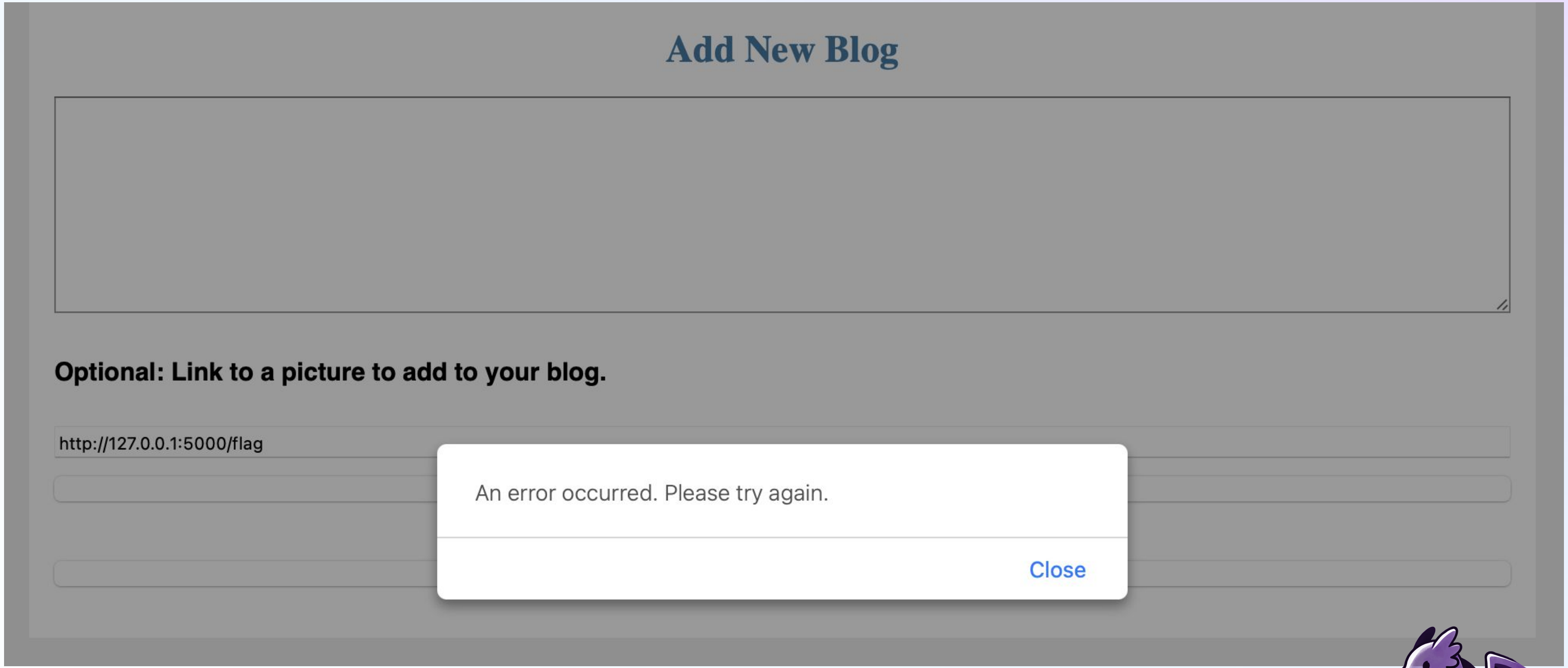
Exclusion Rules

- Allowlist trusted sources to reduce noise

Alerting

- Set up notifications for different attacks

✋ Blocked by the RASP ✋

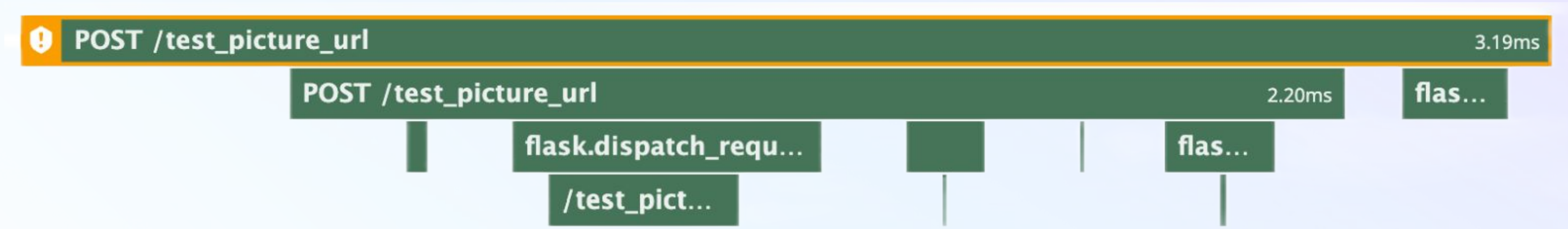

Add New Blog

Optional: Link to a picture to add to your blog.

http://127.0.0.1:5000/flag

An error occurred. Please try again.

Flame Graph from the Blocked Attack



Security Insights

Context Makes All the Difference

Puts less stress on developers

Empowers security teams

Challenges & Opportunities

Limitations of Runtime Tools

Many tools require an agent running within the application

Setup is more complex and RASP rules need continuous tweaking

Other tools are still required for early detection

The Evolving Runtime Security Landscape

Runtime appsec tools are becoming more available

Agentless tools are slowly emerging to allow for easier setup

Recapping our Journey

Runtime security is changing the game in application security

IAST scans have the advantage of runtime context but do not replace SAST scans

RASPs have added benefits over WAFs but do not prevent DDoS attacks

Get a copy of these slides

Join our community newsletter

Check out the vulnerabile website repo

Connect with me

Questions?

Thank You!