

Threat Modeling and Security Test Planning

What Does a Security Lifecycle Look Like?



What is
Security?



Security Is Not a Property of Code



Outcomes
vs.
Assurance



Worse is
Better

Efficacy and Usability

Task Tools vs. Security Tools

Why are “security tools” not just “tools”?

What Does Secure Means Here?

Humans are trying to do different things with every system.

Security means something different in every system.

Security Objectives

Security objectives are what we decide we really care about at the requirements level.

They take the form of a statement:

When [person] attempts to [make a bad thing happen], the system will respond by [doing something sensible].

What is a Threat Model?

A threat model is a formal, complete, human-readable model of the human activities and priorities and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a **formal**, complete, human-readable model of the human activities and priorities and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, **complete**, human-readable model of the human activities and priorities and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, complete, **human-readable** model of the human activities and priorities and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, complete, human-readable **model** of the human activities and priorities and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, complete, human-readable model of the **human activities and priorities** and of the security-relevant features of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, complete, human-readable model of the human activities and priorities and of the **security-relevant features** of in-scope portions of a system.

What is a Threat Model?

A threat model is a formal, complete, human-readable model of the human activities and priorities and of the security-relevant features of **in-scope portions** of a system.

How Do You
Make a Threat
Model?

**ONE DOES NOT
SIMPLY**

**WRITE THEIR OWN
CIPHER**



Actions:

Things actors do to assets, from this controlled vocabulary:

Create Read
Update Delete
eXecute conFigure

Assets:

The concrete nouns we use to talk about the human problem we're solving: *Blog Post*, *Account Balance*

Passwords aren't assets unless you're a password management system.

Actors:

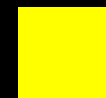
The people (and autonomous external agents) that interact with the system

	Comment			Post			Account		
Anonymous	C	R	X	C	R	X	C	R	X
	U	D	F	U	D	F	U	D	F
User	C	R	X	C	R	X	C	R	X
	U	D	F	U	D	F	U	D	F
Admin	C	R	X	C	R	X	C	R	X
	U	D	F	U	D	F	U	D	F

Always



Sometimes



Never



Threats:

A threat is a negative outcome in the human process a system is built to support. They come in positive (*elevation of privilege*) and negative (*denial of service*) version.

	Elevation			Denial		
Comment	C	R	X	C	R	X
	U	D	F	U	D	F
Post	C	R	X	C	R	X
	U	D	F	U	D	F
Account	C	R	X	C	R	X
	U	D	F	U	D	F

High



Med.



Low



Security

Objectives:

Now that we know what can go wrong with our system, we can create a list of security objectives:

When [person] attempts to [make a bad thing happen], the system will respond by [doing something sensible].

We draw the set of system responses from another controlled vocabulary:

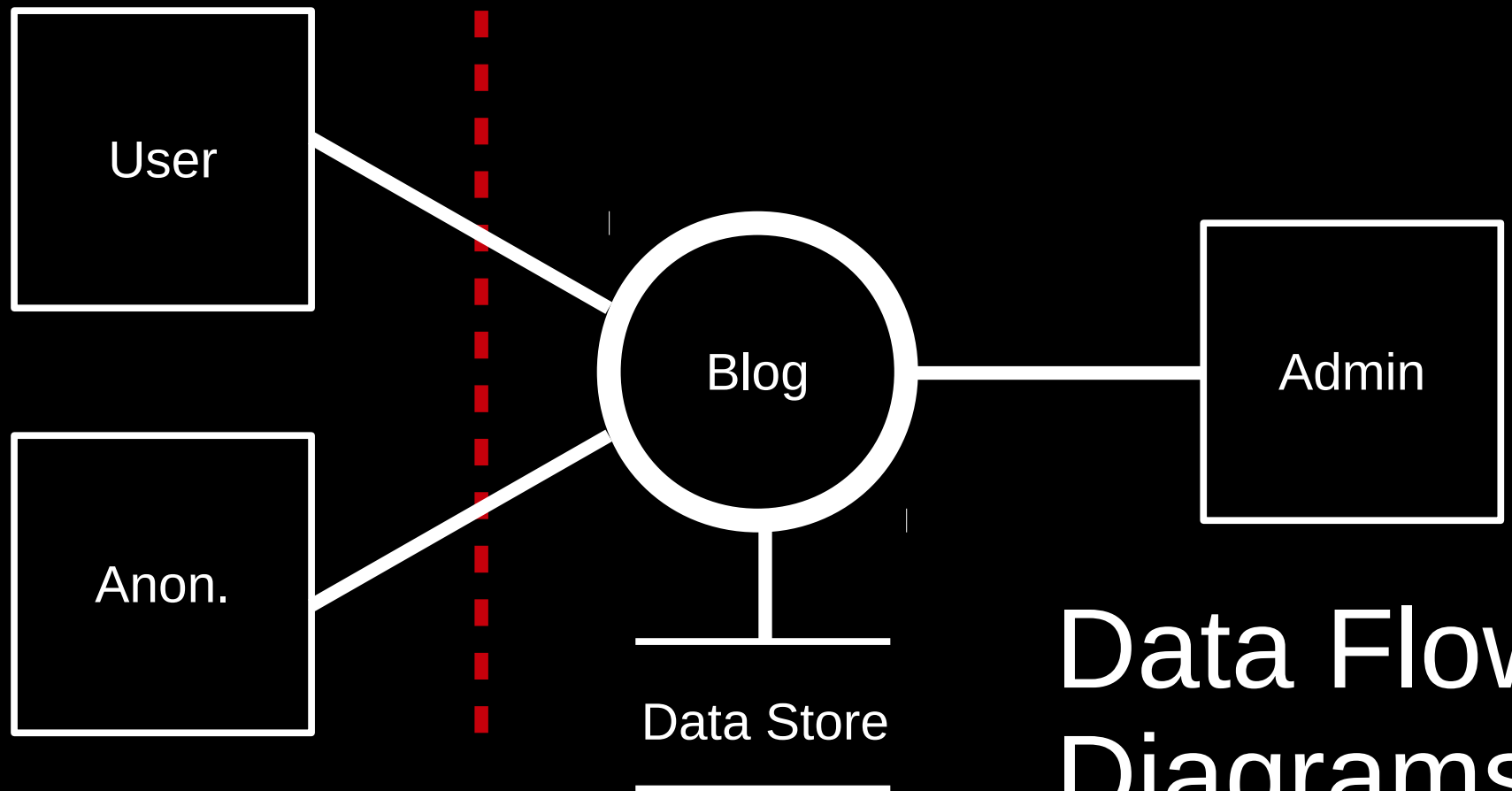
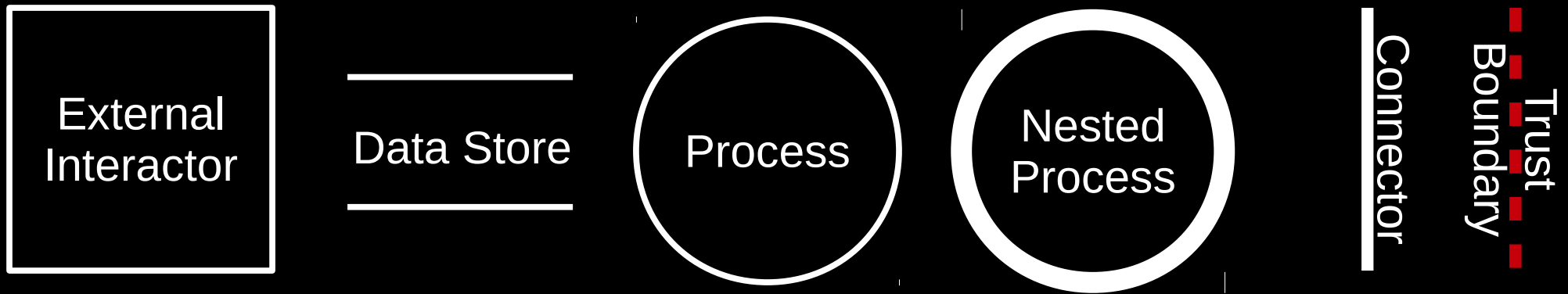
prevent the attacker from launching the attack

detect and log the attack

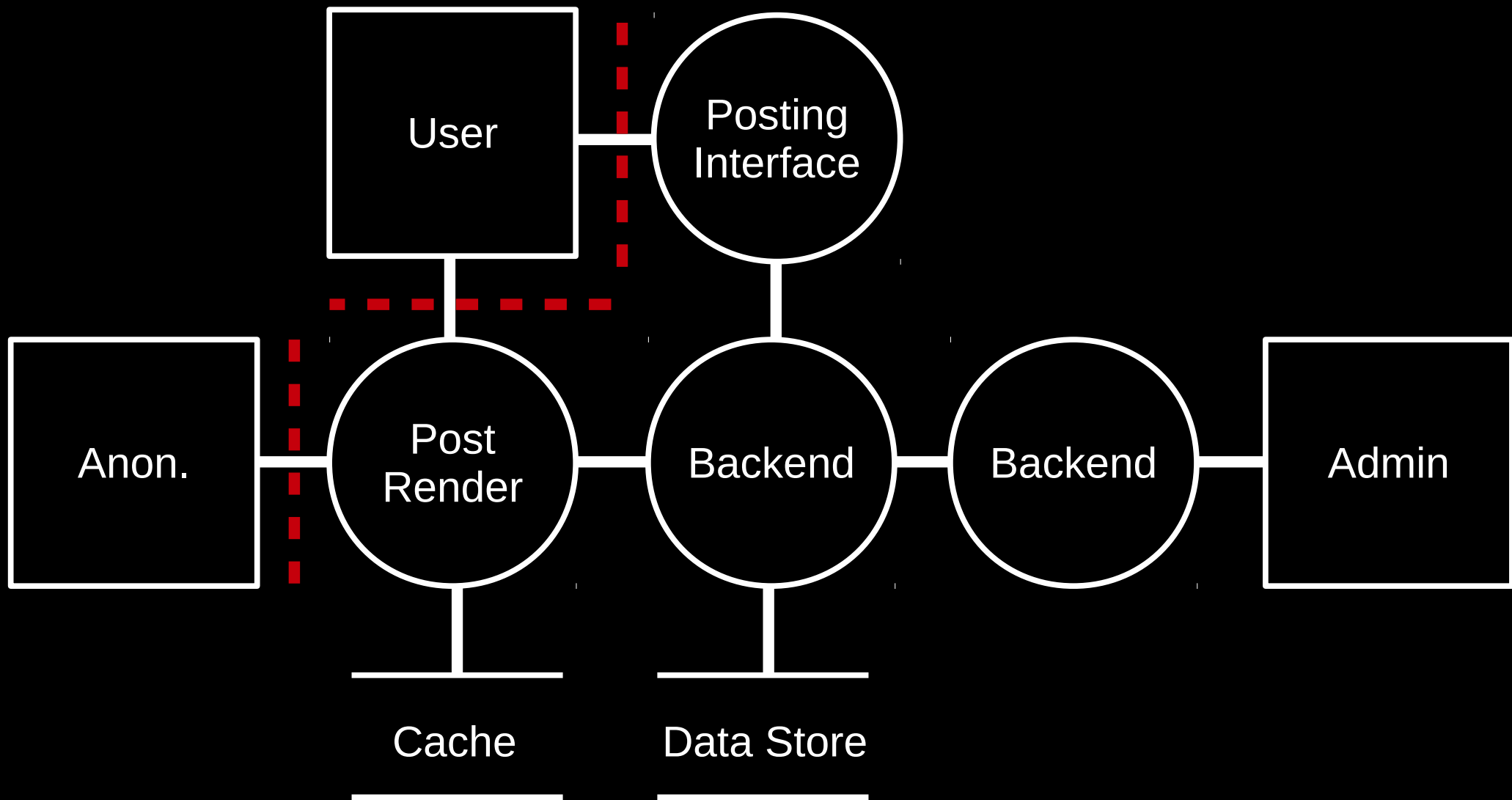
detect and alert some actor about the attack

rate limit the attack, as in DDoS response

thwart the attack from launching



Data Flow Diagrams



Use Cases

Use cases are how intended actions are actually implemented

Name	Description	Pre-Conditions for Success	Post-Conditions on Success
Create Post	A user creates a new post on their blog.		

Preconditions often tell us about things we forgot

Name	Description	Pre-Conditions for Success	Post-Conditions on Success
Login	A user or admin authenticates to the server.		
Create Post	A user creates a new post on their blog.	User can communicate with server. User is logged in.	Post exists on server and can be viewed by other actors.

Preconditions often tell us about things we forgot

Name	Description	Pre-Conditions for Success	Post-Conditions on Success
Load Page	Any actor loads a page from the server.		
Login	A user or admin authenticates to the server.	Actor can communicate with server. Actor's account exists server-side.	User or Admin is logged in.
Create Post	A user creates a new post on their blog.	User can communicate with server. User is logged in.	Post exists on server and can be viewed by other actors.

A use case is a walk through the DFD:

- Adds elements representing the system's data model
- Examines how they're processed
- Defines where checks on "Sometimes" conditions happen

UC Name	Actor	Action	Object	Condition
Create Post	User	Sends	Post text	To Posting Site
	Posting Site	Parses	Post text	Checks for length, HTML tags
	Posting Site	Sends	Post text	To Backend
	Backend	Writes	Post text	To Data Store
	Backend	Sends	Success Message	To Posting Site
	Posting Site	Sends	Success Message	To User

Also must trace out all failure paths

HAZOP Analysis shows us what could go wrong:

Actor	Action	Object	Condition	Varied Element	Guide Word	Meaning
User	Sends	Post text	To Posting Site	Actor	No	User is not logged in
User	Sends	Post text	To Posting Site	Object	No	No post text is sent
User	Sends	Post text	To Posting Site	Object	As Well As	Something else is sent with it
User	Sends	Post text	To Posting Site	Object	Part Of	Incomplete post text
User	Sends	Post text	To Posting Site	...	...	...
User	Sends	Post text	To Posting Site	...	...	...

...and whether or not we care:

SO Violated?	Why is this useful?	Can Attacker Influence It?	Critical ?	Mitigated ?
#3, 4, 7	Bypasses restrictions on...	Users yes; not anons. (SSL)	Yes	No.
#1	Permits deletion of user...	No; admins only	No	Yes
#8	Can delete arbitrary post...	Users only	No	No
#6	Allows user to bypass...	Users only	No	Yes
#1, Maybe 2	Could violate rate limits on...	Any actor	No	Yes
#3	Could violate rate limits on...	Any actor	No	Yes...

Is This
Worth It?



```
graph TD; Requirements[Requirements] --> Architecture[Architecture]; Architecture --> Development[Development]; Development --> Test[Test]; Test --> Deployment[Deployment];
```

Requirements

No threat model, no idea what security means

Architecture

Detect issues at the business rules level

Development

Define security requirements in a standard form

Test

Align development and business around tradeoffs

Deployment

Make a case for secure development costs

```
graph TD; A[Requirements] --> B[Architecture]; B --> C[Development]; C --> D[Test]; D --> E[Deployment];
```

Requirements

Define a design that makes security sense

Architecture

Provides a framework for security analysis

Development

Format to communicate security invariants

Test

Allows for what-if feature trade-off analysis

Deployment

```
graph TD; A[Requirements] --> B[Architecture]; B --> C[Development]; C --> D[Test]; D --> E[Deployment];
```

Requirements

Formal list of everything each developer needs to check

Architecture

Developers can see impact & source of requirements

Development

Not every developer needs to understand all parts

Test

Deployment

```
graph TD; A[Requirements] --> B[Architecture]; B --> C[Development]; C --> D[Test]; D --> E[Deployment]; style D fill:#ffff00,stroke:#ff0000,stroke-width:2px
```

Requirements

Checklist for security tests

Architecture

Meaningful coverage guarantees for high-level tests

Development

Makes security testing in complex integration possible

Test

Deployment

```
graph TD; A[Requirements] --> B[Architecture]; B --> C[Development]; C --> D[Test]; D --> E[Deployment];
```

Requirements

List of invariants required from the deployment environment

Architecture

Basic documentation of system flows

Development

Communicates criticality of different systems

Test

Deployment

Requirements

Quickly communicates system details to staff

Architecture

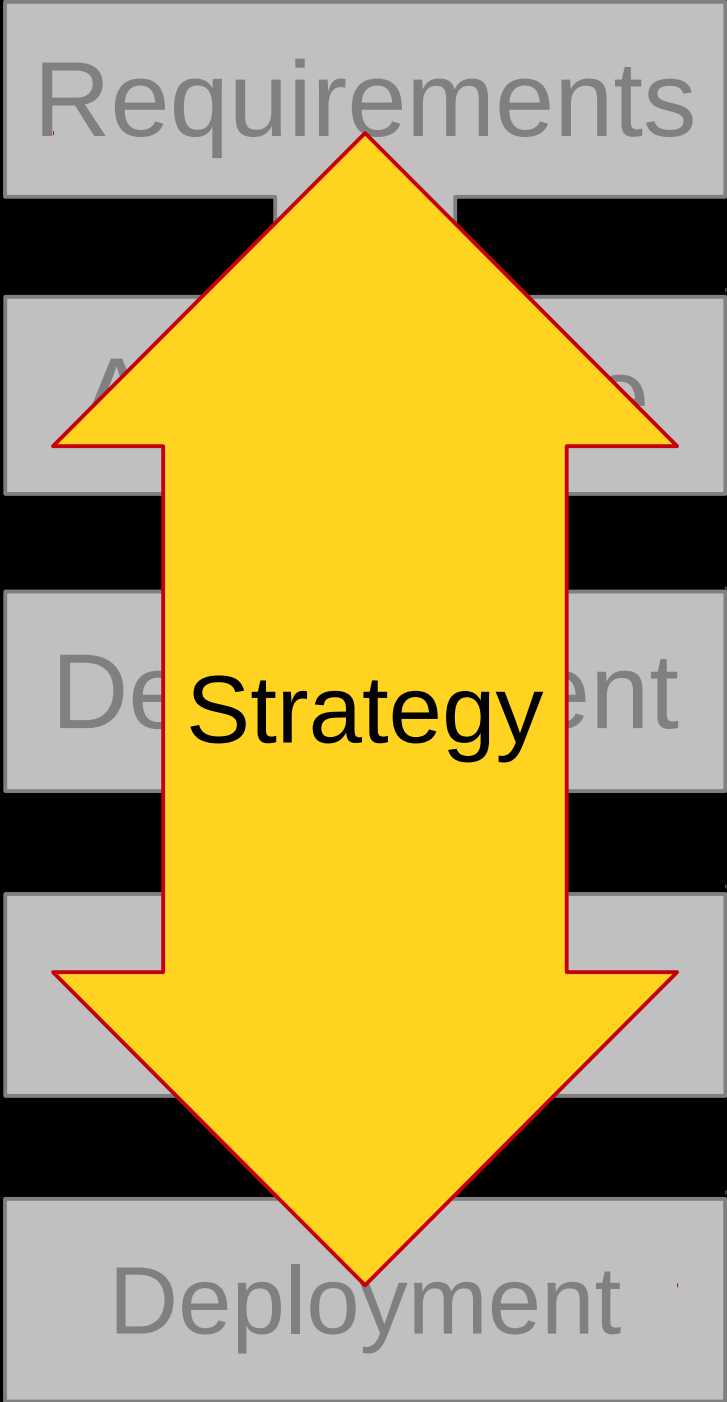
Built-in triage of potential negative business outcomes

Incident Response

What-if scenario analysis for vulnerabilities

Test

Deployment



Requirements

Provides a high-level understanding of system impact

Allows business side to understand security tradeoffs

Integrates security requirements with other business risk

Strategy

Deployment

But My Process Doesn't
Look Like That!

I USED TO BE AGILE

BUT THEN I TOOK A SCRUM MASTER TO THE KNEE

Don't worry, you'll be fine.

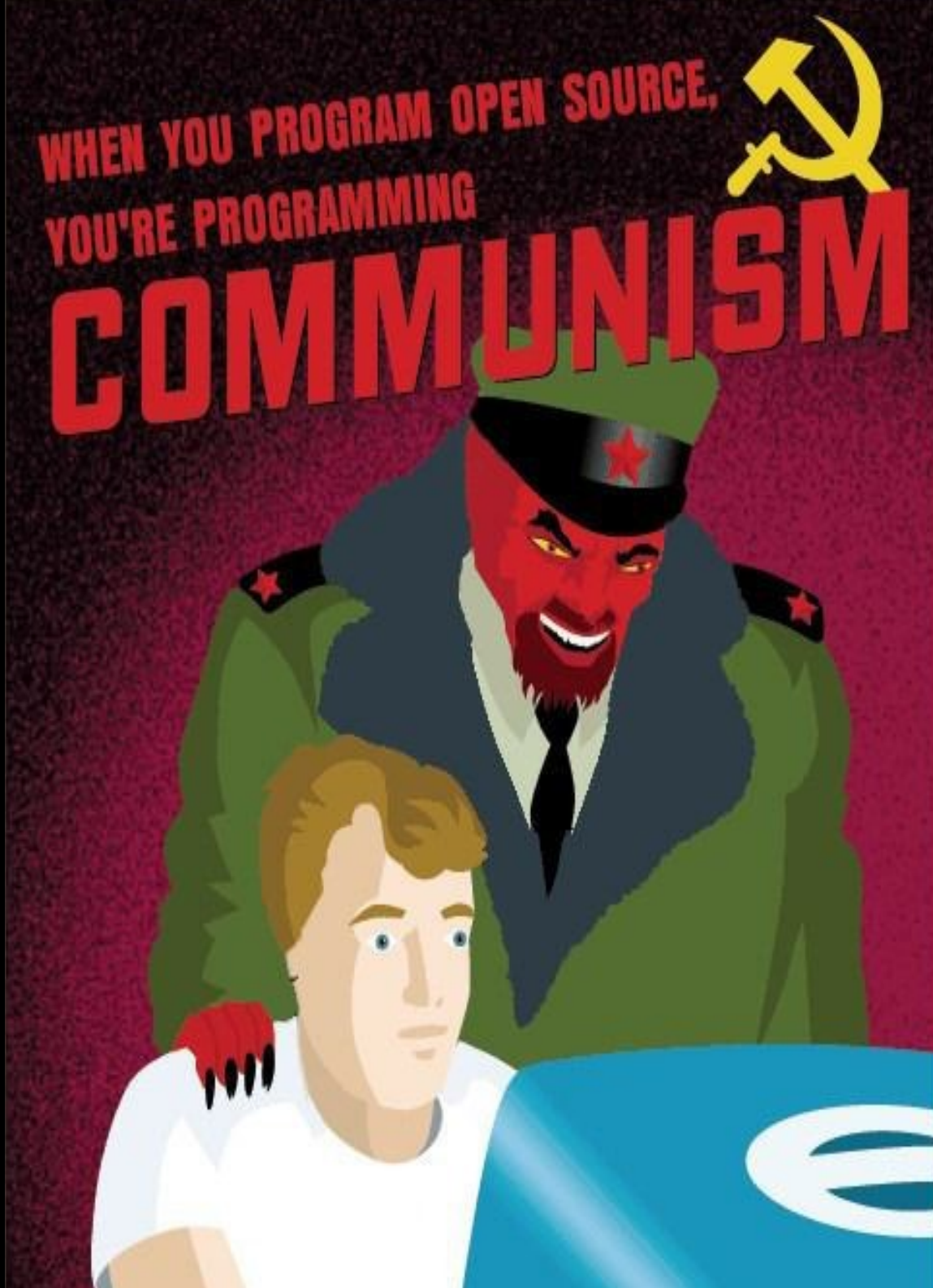
Open Source is Different

Requirements come
later

More exploratory

More of a living
model

Still works the same



Bug Classes

Understand your bug classes.

Kill classes instead of bugs when you can.

Frameworks are your friend.

Parsers, Protocols, and State Machines

Human-generated parsers considered harmful

Formal specifications are mandatory.

Threat model-driven, requirements oriented tests are less likely to catch errors here.

Fuzzers

If you didn't pay attention to the last slide, you're going to have to pay for it here.

Fuzz any interface, protocol, or API endpoint you can't fully explain to an outsider in five minutes.

Cryptography

We all know you don't write your own crypto, right?

Except when you do. Then you need help.

Normal audit teams can test implementations of normal protocols.

When to Bring In Experts

You need other people to read your code

Field outcome sanity checks during requirements/exploration

Architecture review as soon as it settles

Code review by beta and again before release

Mitigation review

Peer review and specific cryptographic audits for anything novel

Communicating Threat Models to Users



Security practice requires failure detection

Why Aren't We Doing It Right?

everything is

broken

everyone go home



Resources

Money is always tight

We have a duty to our users

Threat models vastly reduce security costs in the long term

Judicious use of outside experts helps too

Security as a Negative Feature

Conception that security is just a cost

In an efficacy- and outcome-centric mindset,
this is clearly false

Red Team

The 0day market is killing blue team

It's making us culturally unable/unwilling to look at outcomes

Security requirements in particular is underdeveloped

“Community” security folks less likely to want think about structured work in their “fun time” projects

“Why should I care about Usability?”

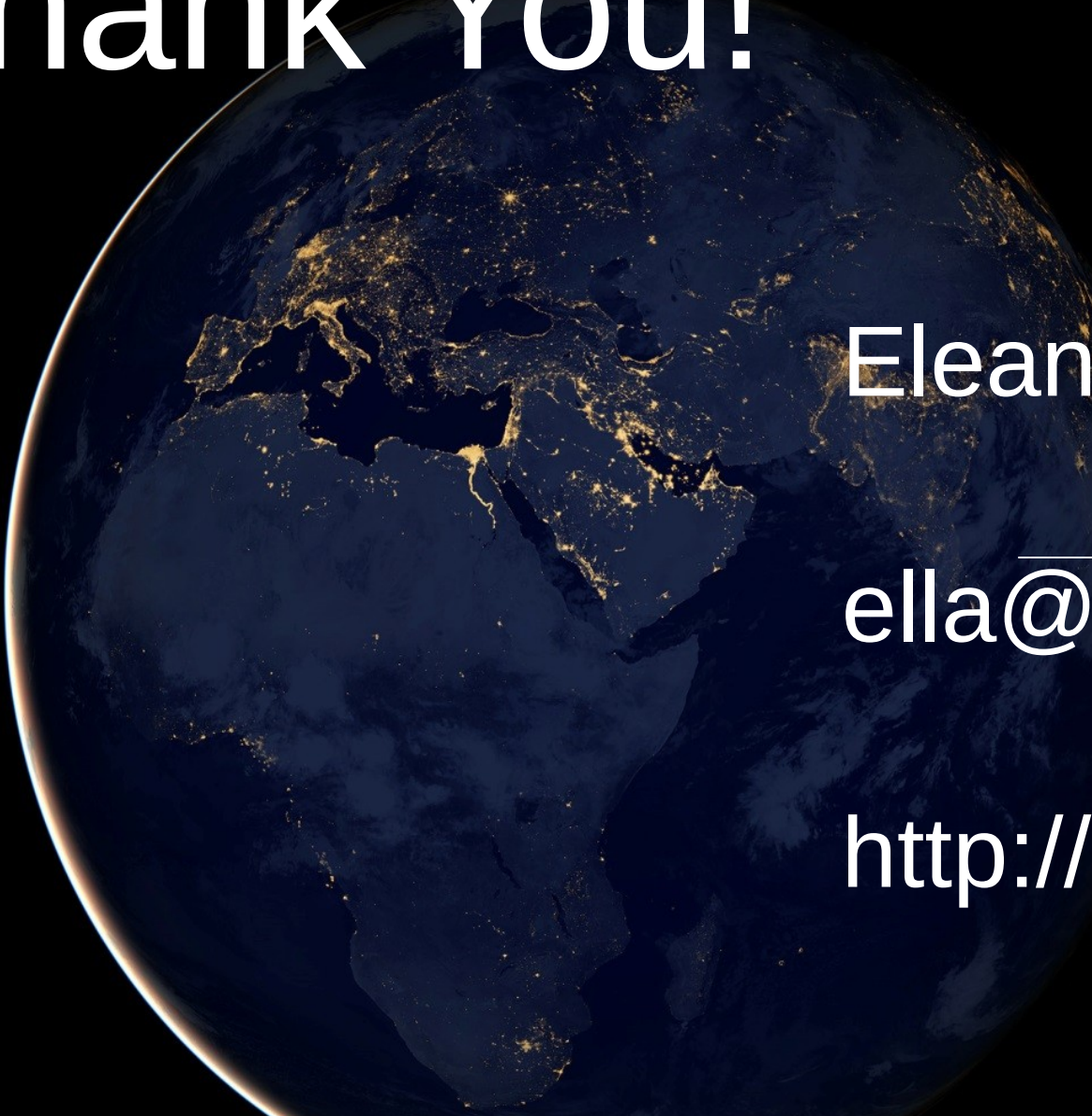
Common, tragic attitude

If you don't understand system goals and the usability impact of mechanisms you can't give rational security advice

Next huge issue is folks thinking they understand

None of us understand the perspective of other users until we go ask

Thank You!



Eleanor Saitta

twitter

ella@dymaxion.org

http

<http://octotrike.org>

HOPE X

Threat Modeling and Security
Test Planning