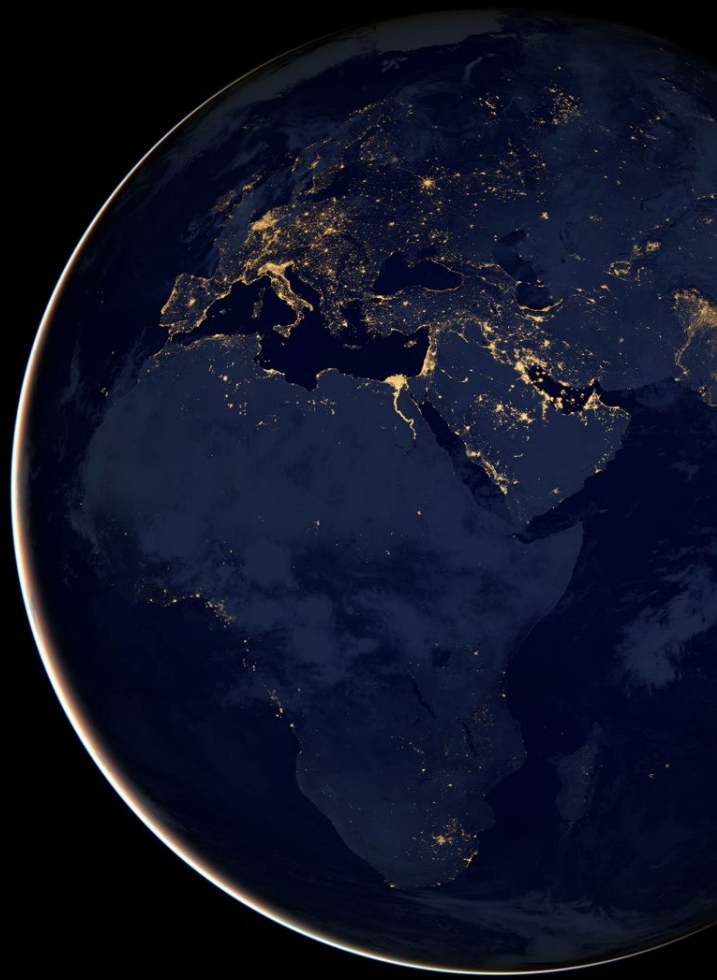
Getting Serious about Security



NYC LeadDev 2023

Eleanor Saitta
Systems Structure Ltd.

Part One: Thinking

What is a System?

Systems exist to do things in the world

To be useful, they need to have certain emergent properties

Whole-system properties which occur in a specific context

Require unified effort to deliver

Properties you care about:

- Correctness
- Performance
- Efficiency
- Reliability
- Observability
- Security
- Resilience

What is Security?

A secure system is one that:

- Enables a chosen set of people to predictably accomplish specific goals
- Does so in the face of actions by a chosen set of adversaries
- Predictably prevents that chosen set of adversaries from

What is Resilience?

The ability of a system to deal with unforeseen modes of failure without complete failure

Resilience is a property of humans, not code

Designing for Resilient Security

Designing both processes and technical systems in accordance with specific principles leads to desired emergent properties

Properties of technical artifacts vs. properties of human processes

Component Principles

A few useful system design principles:

- Statelessness/Logiclessness
- Immutability and Ephemerality
- Canonical Stores

State and Logic

Services should either do computation or hold state, not both

Complex components are unpredictable

Immutability and Ephemerality

Data, configuration, and memory are all state

Immutable systems eliminate unnecessary state

Respinning a cluster resets state

Minimal, Canonical State

Every piece of state should exist canonically in exactly one place

As few systems as possible should be stores of state

Any duplicated state must be validated

Process Principles

And a few for the human side of the org:

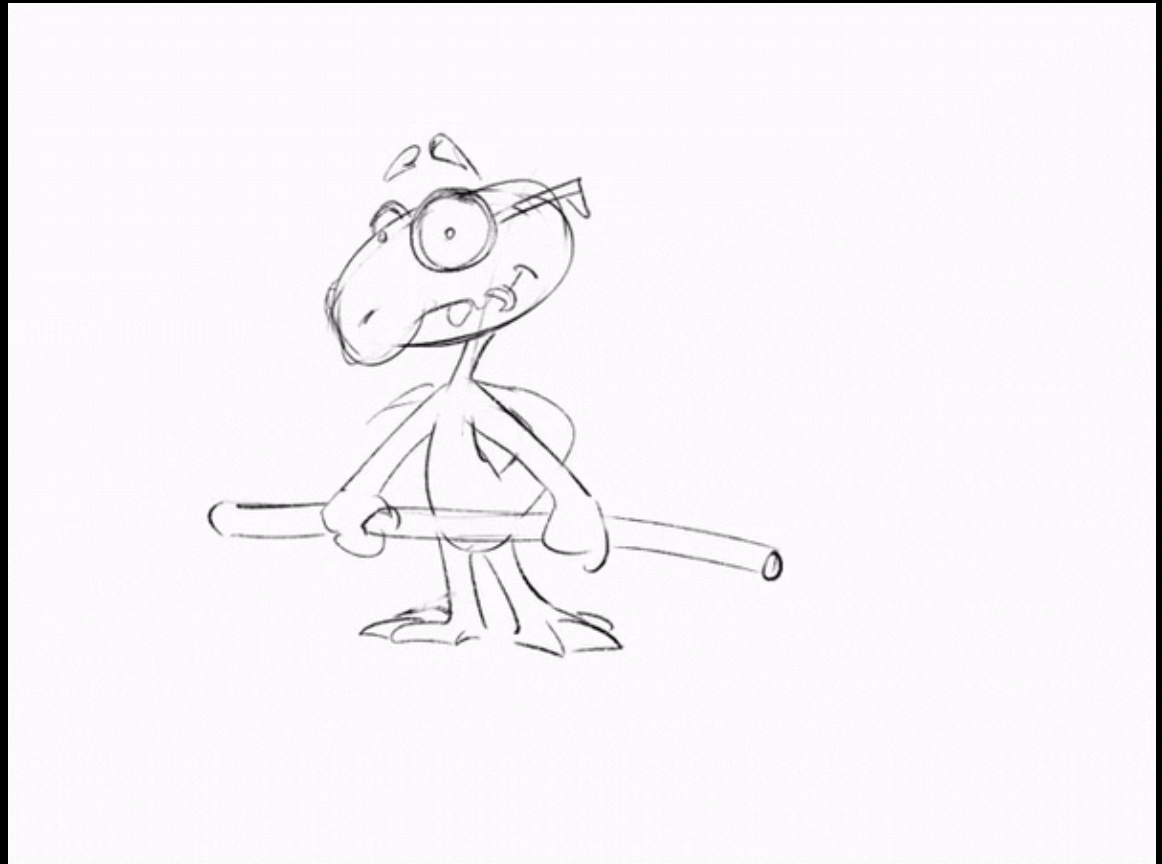
- Declare and Generate
- Design for Failure
- Decide at the Edge
- Slack

Declare, don't Program

Declarative configurations are easier for both humans and computers to create, compose, and validate

Use memory safe languages, parser generators, strongly typed languages, and state machine generators

Mitigations Always Fail



Kill Bug Classes

Security engineering changes that don't involve killing bug classes are emergency response work

…unless those changes kill traversal instead

Make a plan for each class and layer in advance and crosscheck

Design for Failure

Failure and compromise are inevitable

Design components and systems to handle both predictable and unpredictable types of failures

Think about security controls as a whole, assuming that some layers will always fail

Build the system you'd like to have during a compromise or outage

Decentralize Decisionmaking

Empower teams and engineers to work autonomously, so decisions can happen where people have full context

Focus on coordination and communication over control

Ensure teams have thick horizontal relationships outside of formal processes

Slack

Resilience requires teams to have downtime

Improving any emergent property takes more time than the bare minimum

Apply hard caps to feature velocity, ensure people take vacations, have large on-call rotations, and track out of hours work

Part Two: Doing

When to Start

For your product:

- Think about risks for users and the company early
- Make smart language and framework choices
- Let someone else do hard stuff like auth
- Pay attention to where data goes — maximal privacy is cheaper

Make sure it's a real product before going further

When to Start

For your company:

- Make it real first
- Not pre-A or before 10 technical staff
- Do start pre-B
- Keep SaaS systems simple until you start

Seven Immediate Actions

1. Hire at least one each ops and IT engineer
2. Make sure you have for-real tested backups
3. Easy SaaS tools on SSO; Yubikeys for 2FA
4. Get rid of your Office and Windows footprint
5. Laptop fleet management (e.g. Jamf)
6. Thinkst Canaries in your VPCs/network
7. Basic log centralization

Governance

- If the C[EFOT]O isn't on board it won't work
- Someone has to own security
  - Not the CTO; ideally a peer
  - Probably fractional for the first 3 years
- Finish your vegetables
- Think about your incentives
- Qualitative metrics, not quantitative

Detection

- You need to log a lot of stuff somewhere
  - It will cost money
- You need someone to look at the logs
  - Hiring them will cost even more; outsource
- If your product means you have to deal with non-credit card fraud, that's a core competency

Code is Not an Asset

- You spend lines of code to buy features
- Every line of code is an ongoing cost
- Is your feature worth it?
- Tools that let humans write less code are good
- Every tool and library is also an ongoing cost
- Velocity averages out; technical debt is drag
- Most security debt is dark

The Front End

- You probably don't know what JS runs on your site
- Advertising = Malware
- Post-spectre web — CSPs, CORP/COOP/CORB
- Backend integrations are easier to control
- Beware GraphQL

The Supply Chain

- You also don't know what runs on your backend
- Need to be able to reproduce point in time
- Let someone else figure out a library was backdoored first
- Artifact management with configuration in git and logged deploys

Audits

- Red team reviews are for testing incident response if you already understand your environment
- Full access “grey box” testing with source and prod-like access
- Early test on an MVP once you frameworks are set
- Retest high-risk components or new approaches

Product Security

- You get to design your attacker's motivation level and the problems they have to solve
- Spend as much time designing unhappy paths as happy ones
- Know where each automated business or security decision in your flows
- Document this before each sprint and check it after

You are responsible for the impact of your work on people's lives.

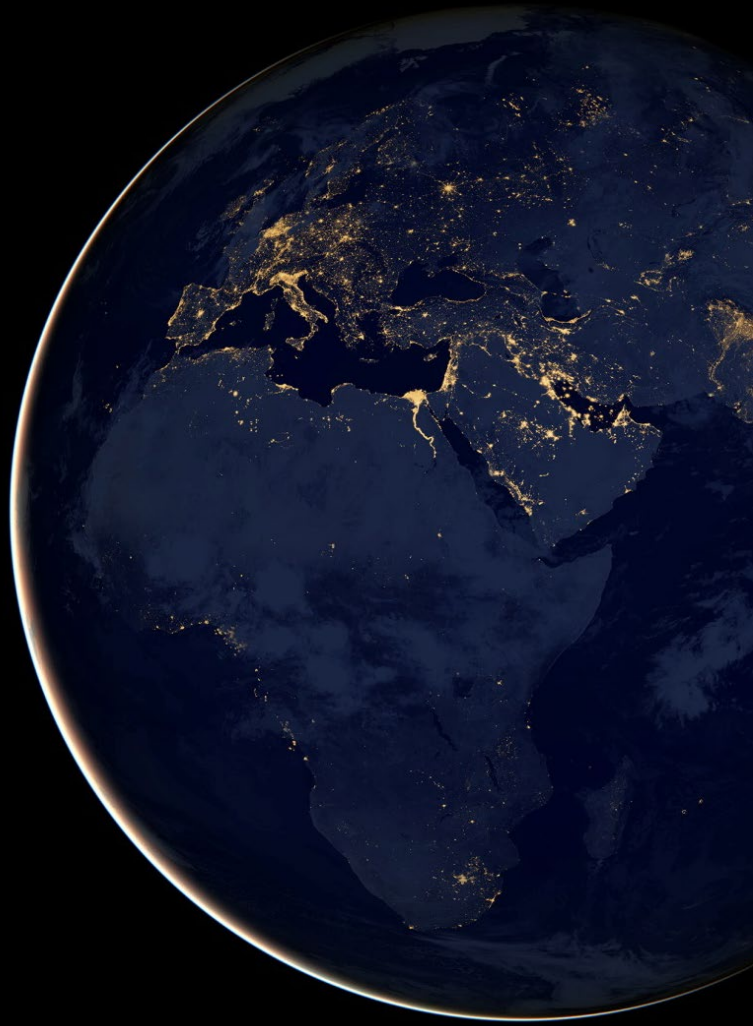
Personas to Examine

- A domestic violence victim seeking an abortion
- A queer teen
- A union organizer

Startup looking to get serious about security?

Let's talk.

ella@structures.systems



Eleanor Saitta
Systems Structure Ltd.