Security from Zero

Agile on the Beach 2024

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

These 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
- Kill Bug Classes
- Segmentation

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

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

Segment All The Things!

After a compromise, the adversary still needs to reach their target

- Horizontally between systems/instances/environments

  Event busses, service meshes, service-level firewalls, per-instance creds
- Vertically within a system

  Least privilege, AppArmor, traditional hardening
- Temporally across execution lifetimes

  Credential dropping, ephemeral workers, short-lived secrets
- Across the lifecycle from supply chain to dev to prod

Process Principles

And a few for the human side of the org:

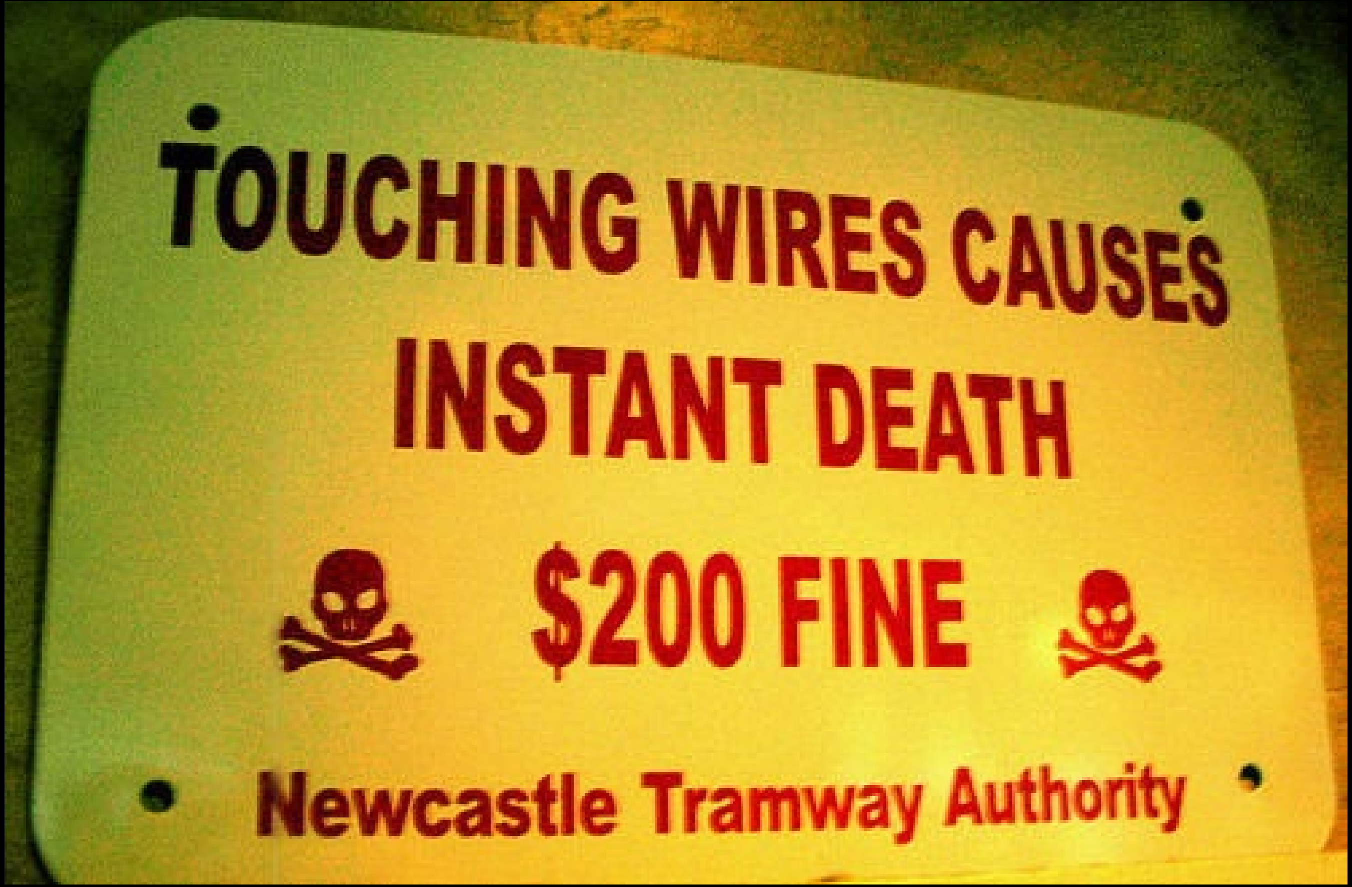
- Declare and Generate
- Design for Failure and Error
- Decide at the Edge
- Slack

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, state machine generators, and declarative IaC languages

Compromise is Inevitable



Design for Failure

Something will eventually fail
Some machine will eventually get owned

Design systems to handle predictable and unpredictable failures

Think about controls as a whole
  Assuming some layer will always fail

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

Make risk events visible

Design for Human Error

Design for Human Error

Your staff (and users) click on things for a living

The cannot do their jobs without clicking on stuff, quickly

Do not try to yell^Wcajole^Wtrain them out of this

Make clicking on things safe:

- U2F and WebAuthN (Yubikeys) solve phishing
- Burn Word, Excel, and Acrobat out of your company like it's poison

If you can't, consider goat farming

The goats will still screw up, but it's funnier

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

Scope

- The software you write and your production infrastructure
- Your laptop fleet and IT systems
- All the “cloud stuff” you use, especially bits you’ve forgotten
- The tools your employees use to get work done
- Your products & services, from the customer’s perspective
- How you run your operations

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

Include maintenance when costing SaaS tools

Getting Work Done

Security work will sometimes need to delay product work

If it doesn't, you won't have secure products

Your planning process is security critical

Segment high-importance, low-urgency work from urgent work

Make qualitative decisions and plan beyond single quarters

Get better at cross-team coordination — security needs a lot of it

Build decision-making/execution capacity on technical arch.

Unify working processes and tools across teams

Productive Team Relationships

Productive Team Relationships

Security’s job is not to say no to things

Make friends — rotations are great once you’re staffed up

If you have an office, have free candy

Hire for social and comms skills as much as technical ones

Security will need everyone else to do a lot of extra work

Make sure the security team can give back directly

Do not make the security team do all the work

Compliance

Compliance

Compliance regimes give useful incentives and useless guidance
Compliance doesn't get to design technical architecture

Security makes rules to satisfy business needs and technical reality
Compliance figures out how to map them to regulations

This isn't about dodging compliance duties, it's about satisfying them in ways that help real-world outcomes

Security compliance must report to the CISO

Risk is a Lie

Risk is a Lie

Do you know your adversaries personally?
Do you have enough of them that you can conduct meaningful statistical analysis on their behavior?

If not, you have exposure, not risk

Exposure and cost structures satisfice for decision support, but most useful metrics will be qualitative, not quantitative

Make sure your team agrees on exposure tolerance

Detection

You need to log a lot of stuff somewhere

- It will cost money

Average time to detection is 210 days

You need someone to look at the logs

- Hiring them will cost even more; outsource
- ML cannot replace a good engineer

You can't review what wasn't logged to track down the entry point for a breach or what was accessed

Capability is a Liability

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

Solutions to Avoid

Solutions to Avoid

There's a lot of snake oil out there

Do not let the CTO buy things he sees in airports

Good security tooling rarely replaces skilled engineers

Black boxes are rarely useful

Exceptions: spam filters, DDoS protection via your CDN

Not an exception: Antivirus systems

Favor systems that work like the internet, and that get integrated into your engineering processes

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

If your product means you have to deal with non-credit card fraud, that's a core competency

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

Personas to Examine

A domestic violence victim seeking an abortion

A trans teen not out at home

A union organizer

Startup looking to get serious about security?

Let's talk.

ella@structures.systems

Eleanor Saitta
Systems Structure Ltd.