Security Engineering 101: When good design & security work together

Wendy Knox Everette - @wendyck

O’Reilly Software Architecture

June 13, 2019

Who am I?

Senior Security Advisor in the Risk Advisory Services group at Leviathan Security Group in Seattle. Software developer & security nerd.

@wendyck

What is security engineering, isn’t it just secure coding standards?

And how does secure development fit into a company’s broader security program?

What are some of the big issues we're concerned about?

- Authentication and authorization
- Information disclosure
- Tampering, Repudiation

- Authentication and authorization
  - Spoofing & impersonation
  - Evading permissions checks
  - Unauthorized access
  - Credential theft
  - Credential stuffing

- Information disclosure
  - From your service
  - Via XSS, from your users

- Tampering, Repudiation
  - Web Parameter Tampering
  - Modifying data within your system
  - Taking actions as another user

Security during the design & planning phase

Design your security requirements

Often these are partially defined by a space you're working within or a service that you integrate with.

But what else do you need to account for?

Threat modeling/ interestingness

Is your product interesting because of data it collects, who it connects, or what it can do? How can it be abused? Does it enable fraud?


Do you allow photo uploads?


i.e. Can I upload malware as a review attachment and just use your s3 hosting?

Beau Woods
@beauwoods

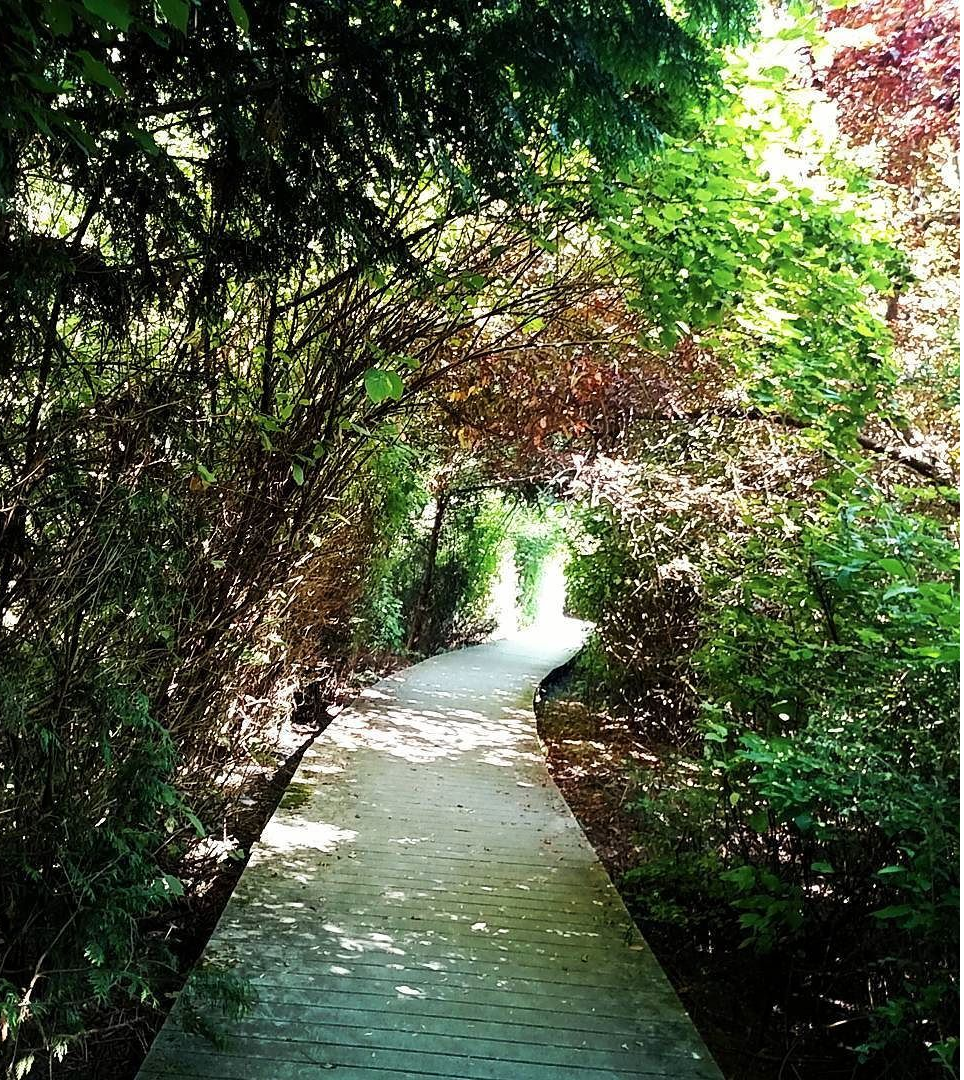
There's at least another thread or two on secure software development as a workforce issue. TL;DR: coders are taught, trained, and incentivized to write code that works under intended use - often not under adversarial abuse.

2:43 PM · May 6, 2019 · Twitter for iPhone

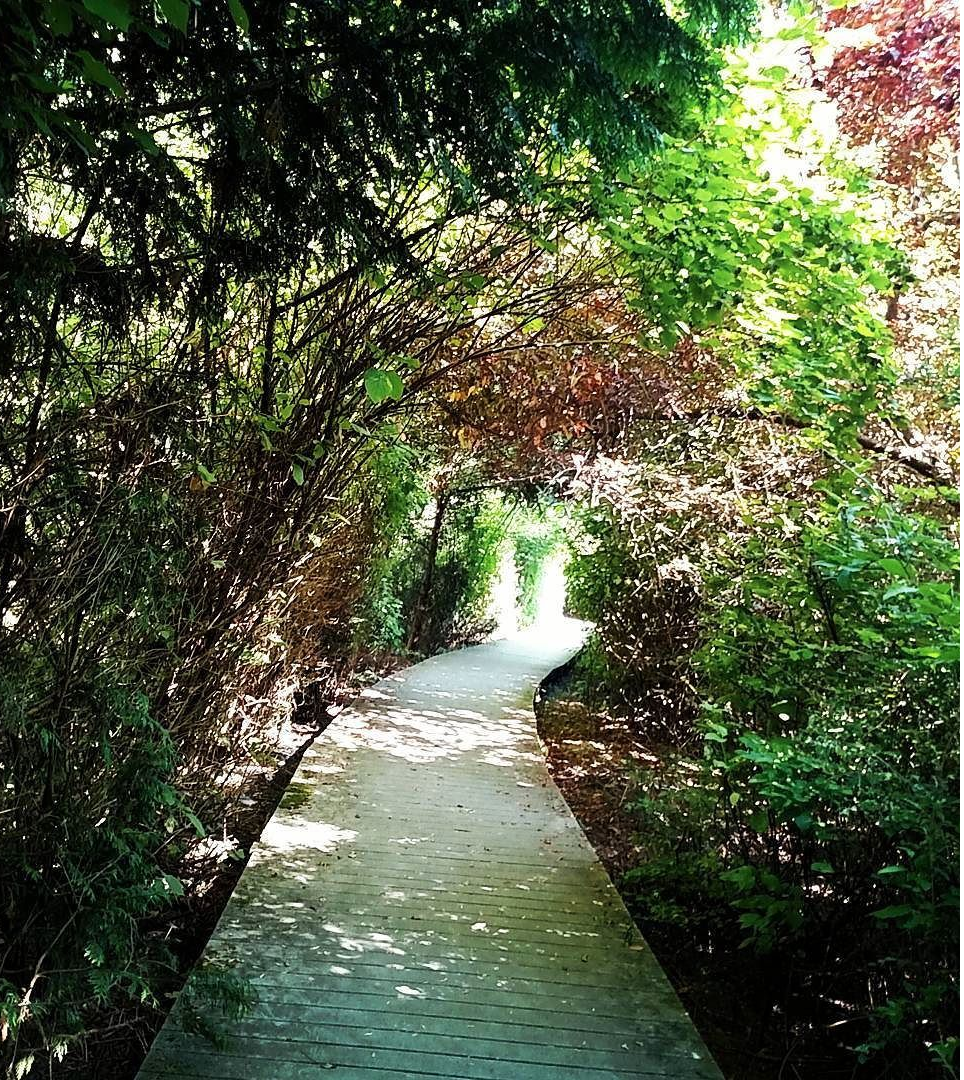
https://twitter.com/beauwoods/status/1125516366965092352

Design flaws: little to big

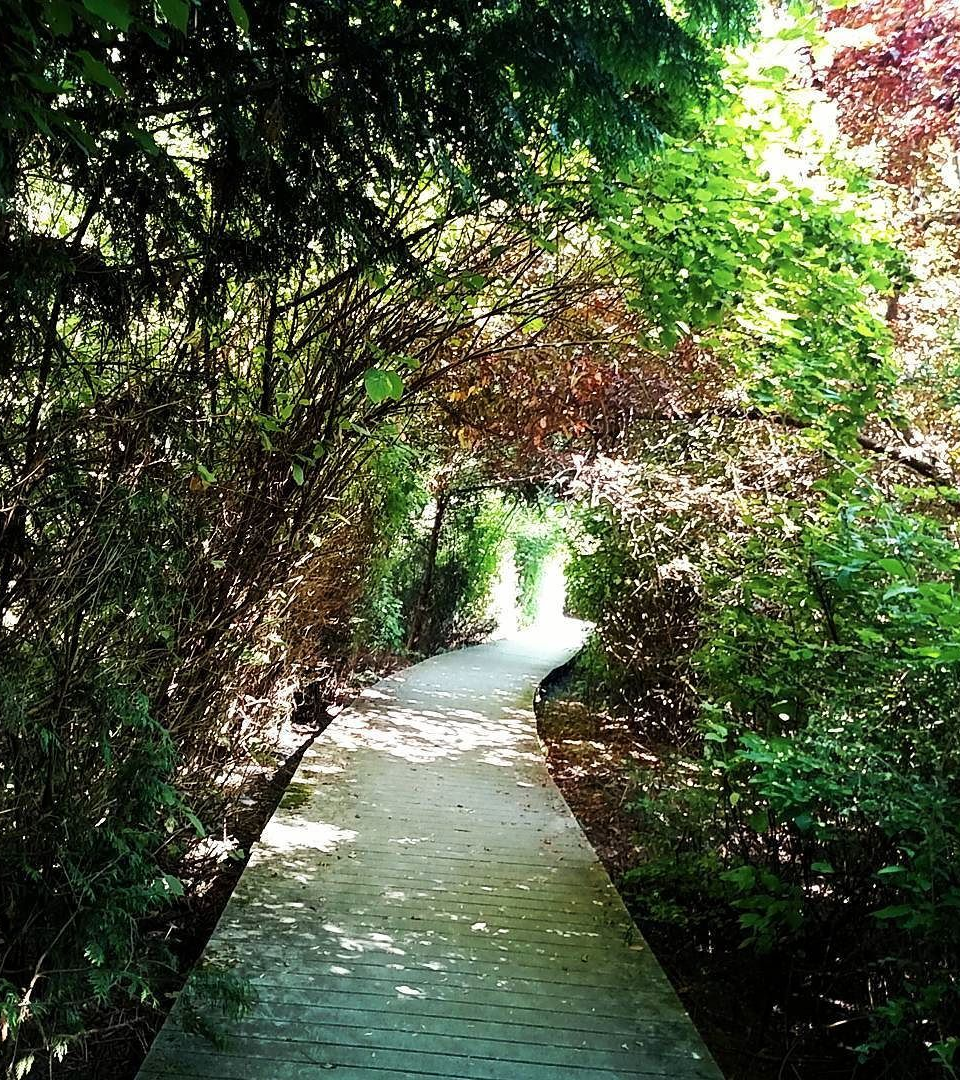
- Trusting user text input
- Trusting user headers sent in http requests
- That API can only be called internally…. Are you sure?
- Installing 3rd party libraries (innocuous now, can it turn malicious? How would you know?)
- Rolling your own crypto
- Authentication: make your own? (what are your authentication flows, how are you storing creds?) vs using an OAUTH (privacy risk for your user?)



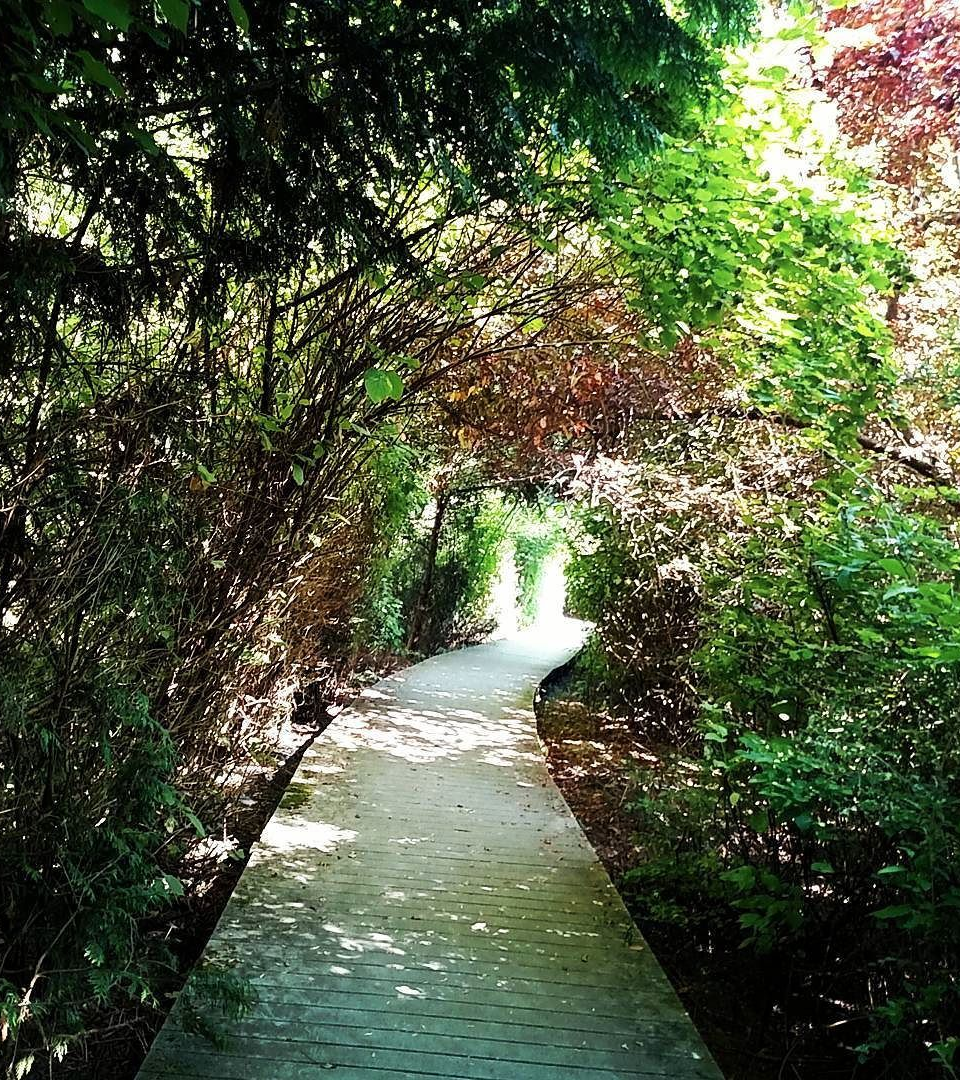
Not Building Granular Access Control



Granular access controls help avoid the anti-pattern of making every engineer at your company an admin in your platform



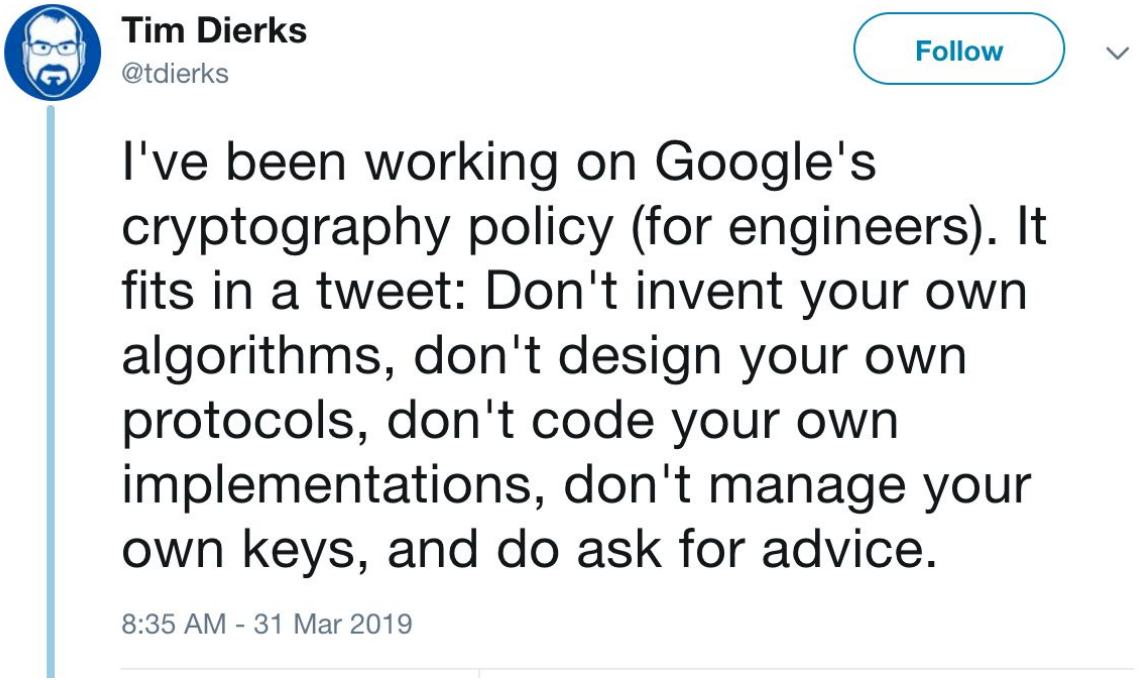
Begin with the expectation of multiple permissions levels, as bolting on a multi-level structure on a code base with a lot of “if user_type > 1” checks is painful.



Carefully encapsulating your auth code and permissions checks will help make your system more resilient

Don’t roll your own Encryption

Encryption



https://twitter.com/tdierks/status/1112377964942028805

Don’t roll your own crypto

Don’t write complex parsers

Language-theoretic Security, or langsec, is one of the causes of application security vulnerabilities today.

- Parsing or unparsing bugs
- Caused by software failing to correctly handle inputs
- You have these in your code base

Perry E. Metzger
@perrymetzger

Most programmers assume that they don't need proper parsers, and that's often the seed of later exploits. Also, some format and protocol designers make their input languages unnecessarily complicated, requiring excessively complicated parsers that are hard to get right. 5/

https://twitter.com/perrymetzger/status/1092525703021633538

Parse errors

1. “insufficient recognition, where input-checking logic is unfit to validate a program’s assumptions about inputs”
2. “Parser differentials, wherein two or more components of a system fail to interpret input equivalently”

From *The Seven Turrets of Babel: A Taxonomy of LangSec Errors and How to Expunge Them* http://langsec.org/papers/langsec-cwes-secdev2016.pdf

More Lang Sec papers & discussion: http://langsec.org/

Don’t roll your own parser

“But I just need to pull out this bit of data”..... 4 years later, we have a pile of exploits duct-taped together into an essential piece of code that I can’t take down.

What causes these issues?

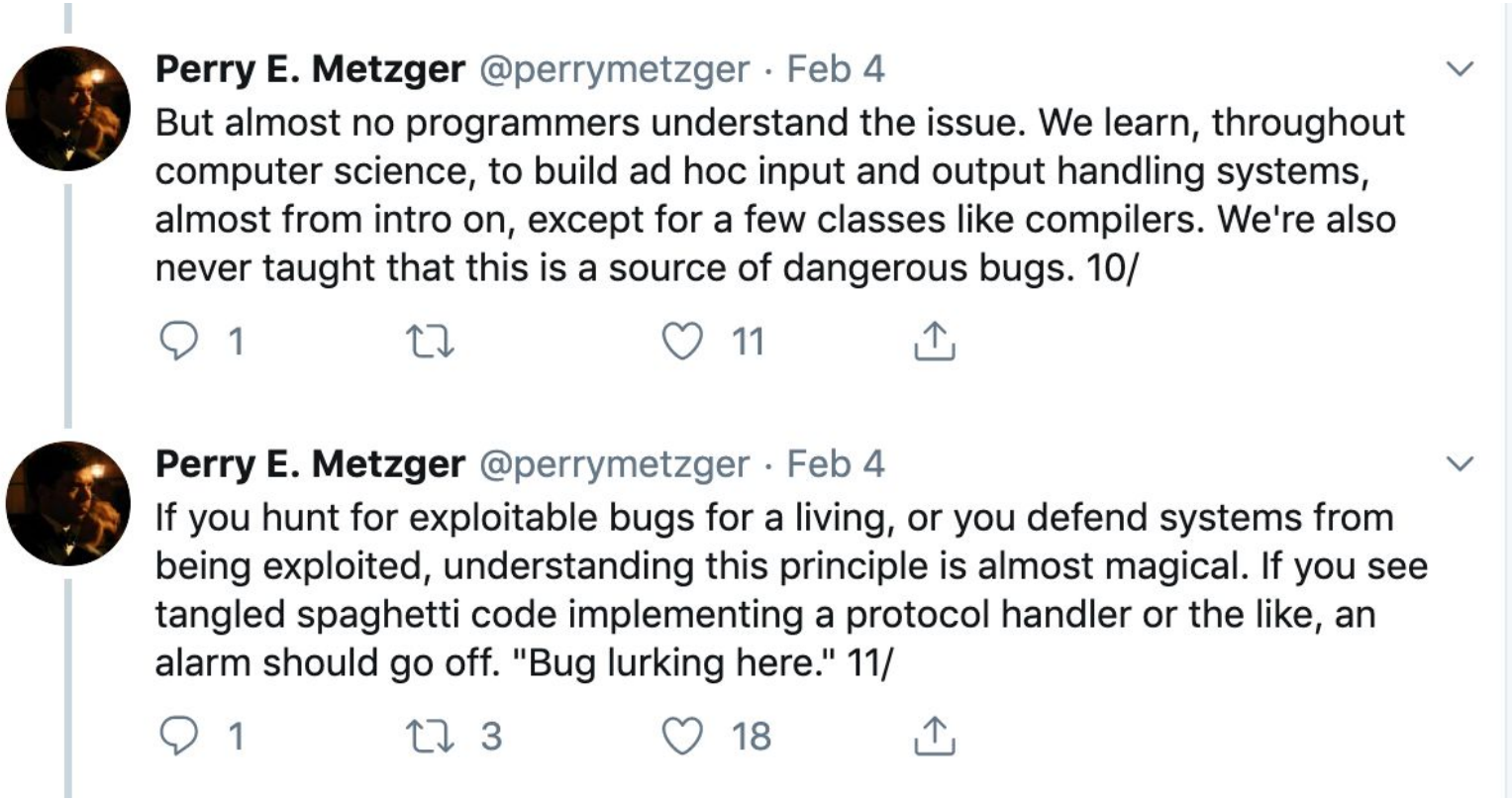
1. Input recognition and validation code scattered throughout your code
2. The code doesn’t match the programmers' assumptions about safety and validity of data

   -langsec.org

“for complex input languages the problem of full recognition of valid or expected inputs may be UNDECIDABLE, in which case no amount of input-checking code or testing will suffice to secure the program”

-http://langsec.org/

Please make our pen testers' lives harder



Perry E. Metzger @perrymetzger · Feb 4

But almost no programmers understand the issue. We learn, throughout computer science, to build ad hoc input and output handling systems, almost from intro on, except for a few classes like compilers. We're also never taught that this is a source of dangerous bugs. 10/

1 11

Perry E. Metzger @perrymetzger · Feb 4

If you hunt for exploitable bugs for a living, or you defend systems from being exploited, understanding this principle is almost magical. If you see tangled spaghetti code implementing a protocol handler or the like, an alarm should go off. "Bug lurking here." 11/

1 3 18

https://twitter.com/perrymetzger/status/1092528170354573312

Don’t use poor authentication

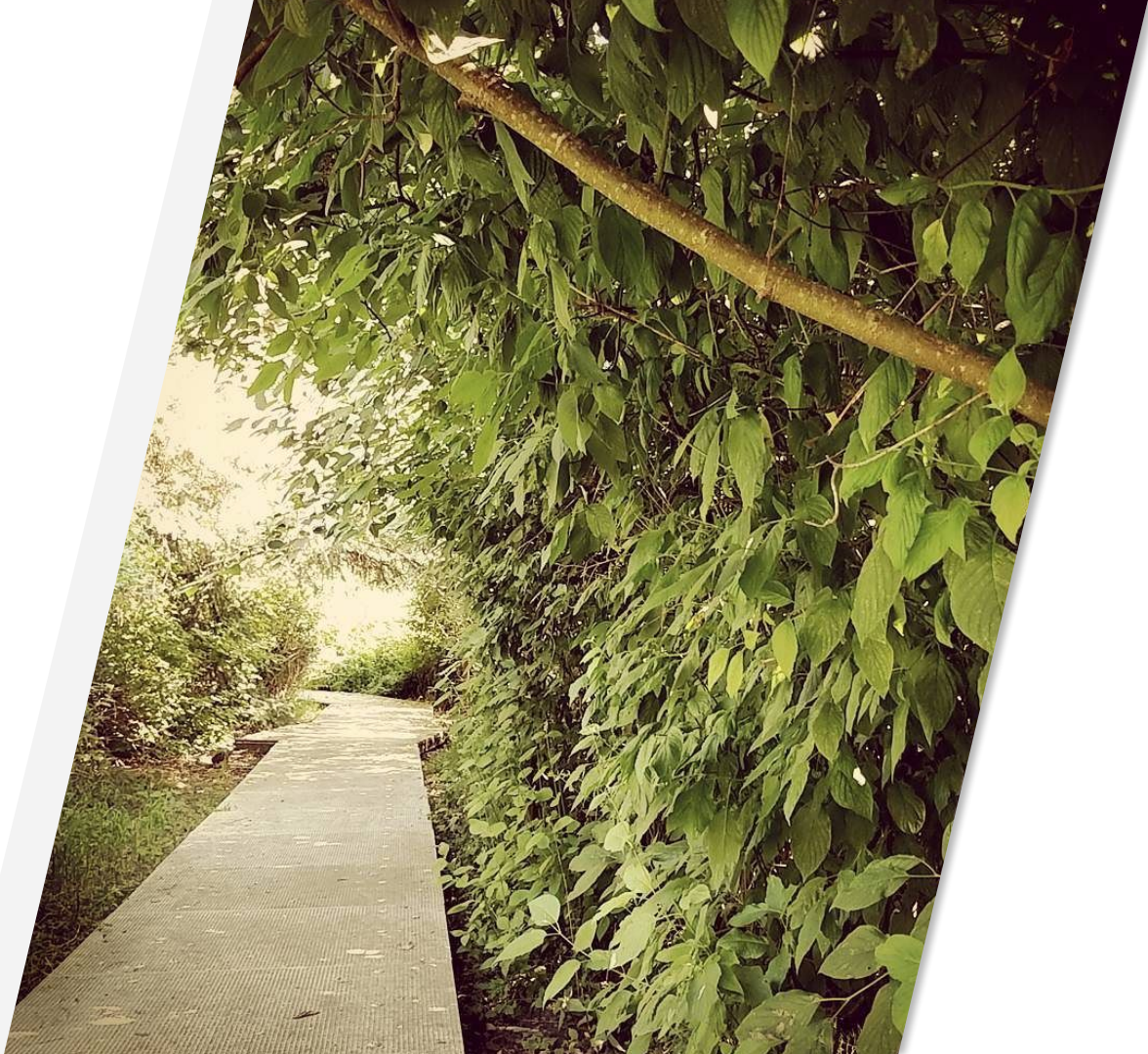
Do users log into your system? How do you secure their credentials and how do you help them protect their accounts?

Don’t ignore encapsulation

Not just a good design principle, it makes your code easier to update (to patch emergent security issues), helps you build good permissions models, and protects resources from each other.

Don’t ignore observability

Doesn’t just help you improve the software’s performance, it can be critical to discovering and remediating security incidents.



Language choice is not really security neutral

- Use after free and heap corruption vulnerabilities are super useful for building exploits
- Memory safe languages like Rust protect you in ways that C can’t
- If you’d like to regularly be appalled by these issues, here’s the Twitter account you want: https://twitter.com/@LazyFishBarrel

That's a lot to cover!

Think of this as an introduction to the types of security issues you should consider as you design software.

There are too many resources to list here, but the OWASP Top 10 and the OWASP Mobile Top 10 are decent places to start

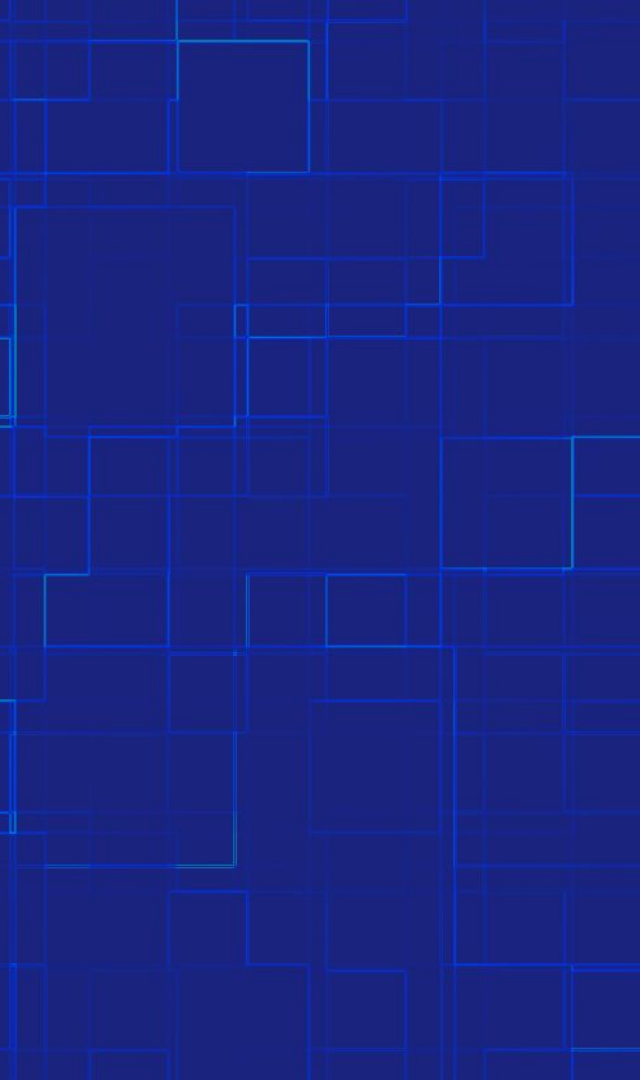
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

OWASP Top 10

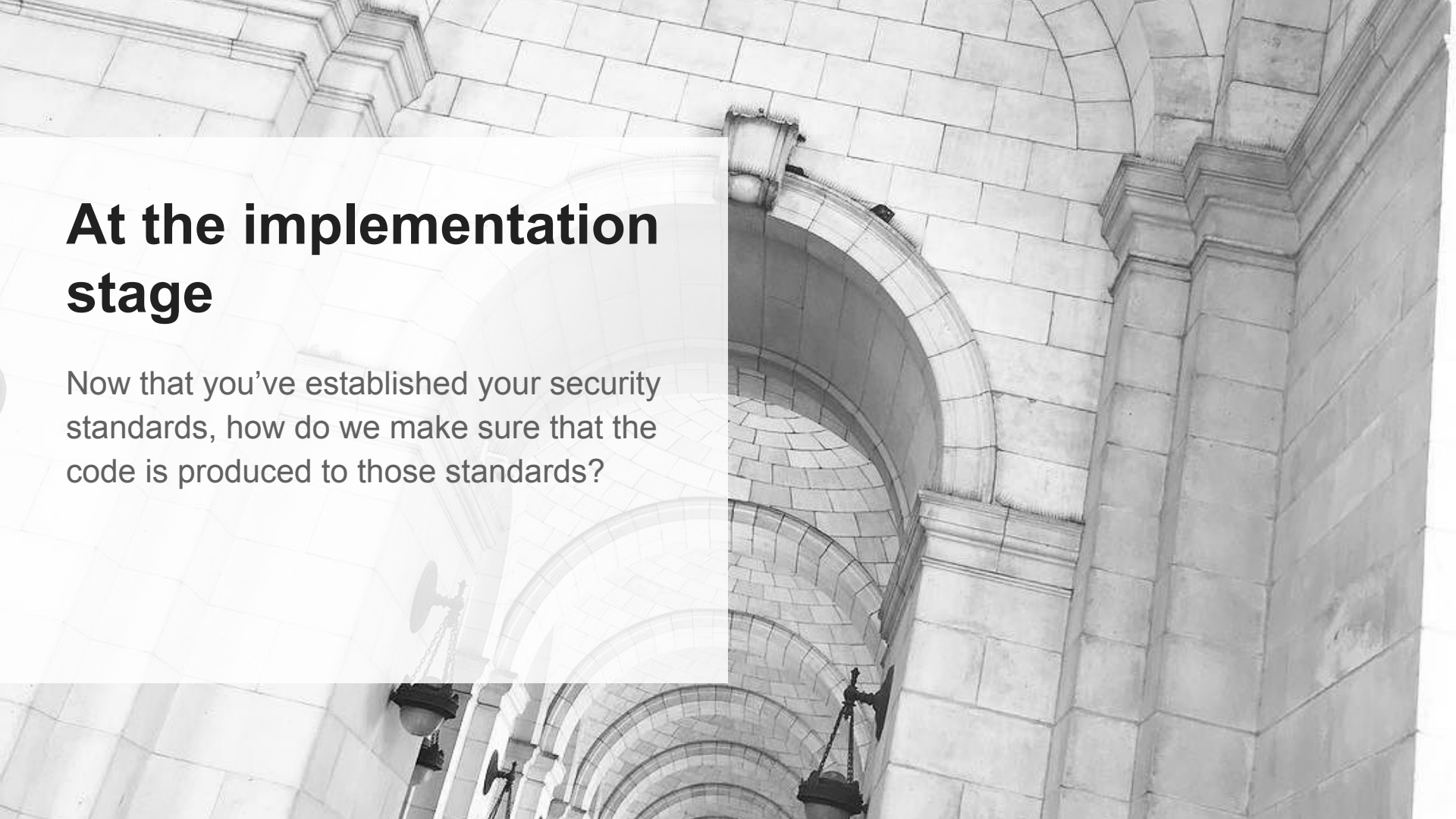
1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Getting it right at this stage is critical



Writing secure code



At the implementation stage

Now that you've established your security standards, how do we make sure that the code is produced to those standards?

Addressing security early and often

Do developers in your organization know what areas of your code are security sensitive?

Are your engineers empowered to ask for security reviews?

Build good working relationships

How are interactions between developers and security?

Do they think you just write vulns all day long?

Do you think they always say no?

How do you talk about risks that you need to take?

What do your conversations about risk look like?

Document and train

How do you enforce consistent escaping of user input?

What are your access control standards?

Do you have style guides that help prevent LangSec-style parsing issues?

Role specific security training

Start with some basic application security training for everyone

Can your application security engineers offer ongoing classes, send people out to classes?

We've found that teaching PMs some basics can be a force multiplier- they will recognize security sensitive areas and engage resources.

Ownership

Are engineers empowered to request changes when they see a security impact?

Do your teams see customer reports? Do you share pen test report findings? Bug bounty findings?

Familiarity

Can you play OWASP JuiceShop or another CTF to learn what security issues look like?

- https://www.owasp.org/index.php/OWASP_Juice_Shop_Project

What tools can we use?

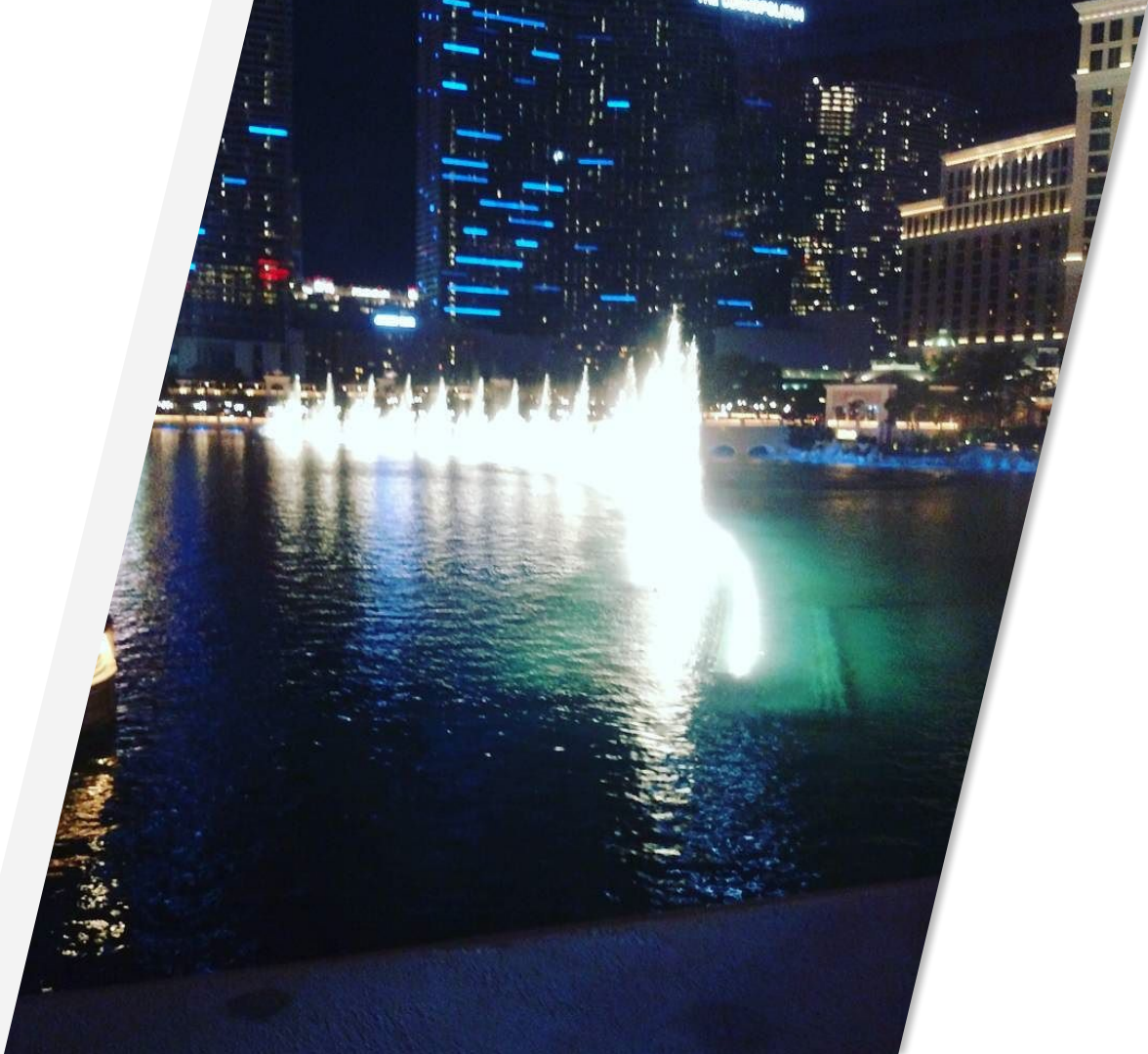
Dynamic & Static analysis

Static analysis tools

- Start here
- Source-code analysis that hunts for errors in the source code

Dynamic analysis

- Injecting flaws into the executing source code
- Fuzzing inputs
- Verifying authentication state



Code reviews

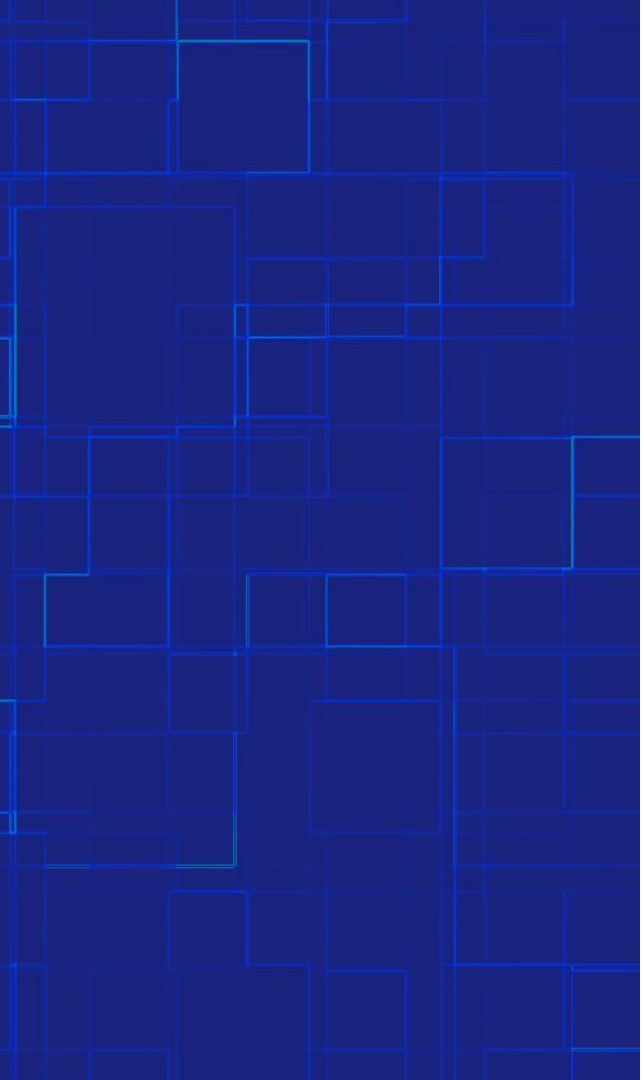
Requiring security specific code reviews for some types of changes will help catch some classes of bugs

Readable & modular code is more secure

Your users are probably going to find the best bugs

Canaries & launch flags

You want to see over time that the CLASSES of bugs you're still shipping are getting more complex



It’s not just code

Accounts, Secrets, Configuration Defaults

Clouds are so secure!

Private keys & Data access - these are both critical to the security level of your cloud hosted system

GitHub’s blog on their scanning for auth tokens:

https://github.blog/2018-10-17-behind-the-scenes-of-github-token-scanning/

Backup files & configurations still around?

Do you store user uploaded data?

How often do you patch- how old is the longest running process you have?

Are you running VMs when you should run lamdas?

AWS Configurations & Risk

Is a "bastion" pattern enforced for SSH access?

I don't know.

Is your AWS account and / or product source code repositories protected by more than simple password authentication?

I don't know.

Are AWS root credentials secured with a hardware two-factor token?

I don't know.

Is infrastructure in this account usually managed by "configuration as code"?

I don't know.

Do you manage "secrets" so they are less accessible to engineers, source code, or configuration files?

I don't know.

Are long lived IAM keys used within production infrastructure?

I don't know.

Have you improved CloudTrail logging beyond default settings in this account?

I don't know.

If a vulnerable configuration change or suspicious API activity is escalated, is it investigated?

I don't know.

https://magoo.github.io/model-risk-aws/

Is a "bastion" pattern enforced for SSH access?

Yes. There shouldn't be exceptions that bypass it.

Is your AWS account and / or product source code repositories protected by more than simple password authentication?

No.

Are AWS root credentials secured with a hardware two-factor token?

Yes.

Is infrastructure in this account usually managed by "configuration as code"?

Yes, partially.

Do you manage "secrets" so they are less accessible to engineers, source code, or configuration files?

Yes. However, unmanaged secrets are not a rare exception.

Are long lived IAM keys used within production infrastructure?

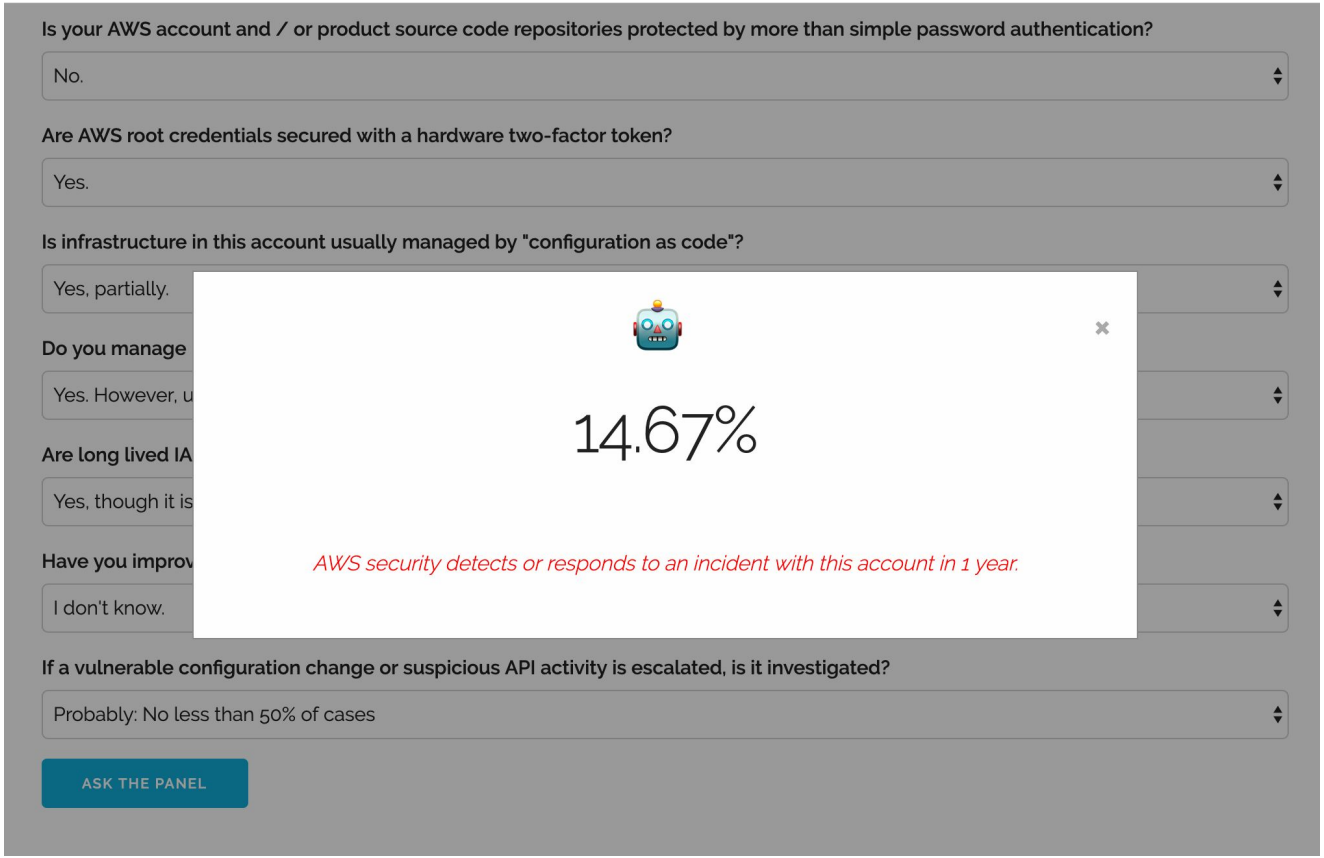
Yes, though it is not an typical pattern.

Have you improved CloudTrail logging beyond default settings in this account?

I don't know.

If a vulnerable configuration change or suspicious API activity is escalated, is it investigated?

Probably: No less than 50% of cases



https://magoo.github.io/model-risk-aws/

How do you secure authentication materials?

https://twitter.com/QuinnyPig/status/1120780385460391939

The AWS Permissions Underwear Theorem

Deployment

Monitoring & alerting

How do you know the health of your system?

Do you have enough logging to for useful incident response capabilities?

What are you logging? Where?

Do you have logs of user access to sensitive data?

Do you know where people connected from?

What timezone are your logs in? Are all your logs in the same timezone?

Who has write access to logs?

Change Management

How does your organization know what’s changed in your production environment?

Who has the ability to make those changes?

“You can’t secure what you don’t know you own”

- What’s the process for creating new internet exposed endpoints? Don’t make this so onerous that people route around it - the Shadow IT problem can make things worse
- What tools do people in your organization use that can create internet exposed endpoints (Salesforce?)

Post Launch

Maintainability (patching your code)

Can you update your code easily? Can other people who use your code update it easily?

How are non-emergent security issues fixed?

If it’s not a sev 1 emergent issue, what’s your mean time to rolling out a patch?

Do you track whether security issues on your backlog are being exploited or impacting users?

How is technical debt prioritized and tracked?

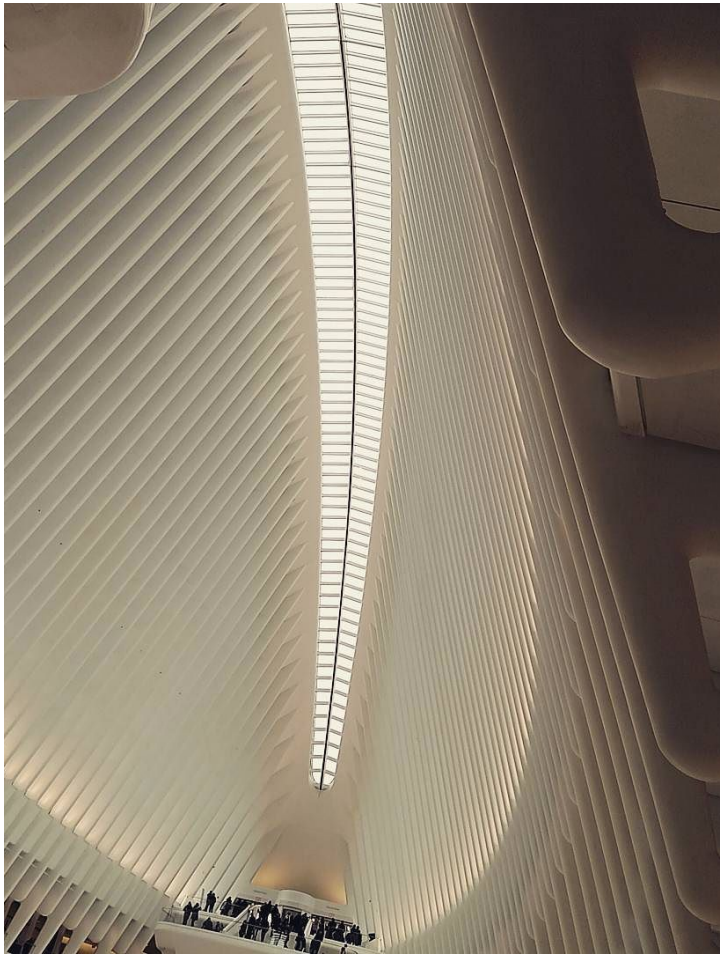
Do you constantly push out new features and ignore older sections of your code?

Who is assigned maintenance work - is it the least experienced developer on your team? Or do you have one engineer who is always saddled with fixing the oldest code?

Vulnerability management (applying their patches)

How do you find out about updates to libraries you incorporate?

- Github vulnerability alerts if you're on Github
  - https://help.github.com/en/articles/about-security-alerts-for-vulnerable-dependencies



Patching timelines

Can you commit to timelines to roll out critical patches?

In summary….

Hopefully your team is doing a lot of these things, and hopefully you've gotten some new ideas to dig into.

The biggest force multiplier we've found is to educate & empower everyone in your organization to be responsible for the security of your environment.

Thank you & Questions

Please don’t forget to rate this session!

Wendy Knox Everette

@wendyck

Leviathan Security Group

https://www.leviathansecurity.com/