Threat Modelling

Getting from None to Done

John DiLeo, OWASP NZ Chapter

February 2019

About Me

- Dr. John DiLeo
- Born and raised in northeastern US
- Spent A LOT of time in school
- First career: Operations Research / Simulation
- Second career: Web Development
- Third Career: Application Security, since 2014
  - Focus on Software Assurance
  - Moved to NZ in 2017, joined Orion Health
  - Active in OWASP in US and NZ

And About You?

- Tell us about yourself
  - What's Your Name?
  - Where do you work?
  - What's your current role there?
  - What got you interested in Threat Modelling?

Today's Schedule

8:00	Registration Check-in (Main Lobby)
8:45	Class Session
10:10	Break for Morning Tea (Main Lobby)
10:40	Class Session
12:30	Break for Lunch (on your own)
1:30	Class Session
3:10	Break for Afternoon Tea (Main Lobby)
3:40	Class Session
5:30	Closing Time

Sources/Acknowledgements

- Adam Shostack
  - *Threat Modeling: Designing for Security* (2014)
  - Instructor's supplements provided by publisher
- Mark "pipes" Piper and Wade Winright
  - "A Cat, a Dog, and a Roast Turkey: What's in your Threat Model?"
  - Training presented 15 November 2018 @ Kiwicon
  - Slide deck provided by presenters, used with permission
- Irene Michlin
  - "Incremental Threat Modelling"
  - Presented at AppSec EU 2017, Belfast
  - Slide deck and video published by OWASP

About this Class

- I am NOT an expert on threat modelling
  - In fact, I'm still trying to figure it out…
- I'm going to talk about what I've learned, and what's worked for me so far
  - I want to hear your perspectives and experiences
- I've already realised value

  …and I've just gotten started

Agenda

- Introduction and Background
- The Five W's
- TM Approaches/Perspectives
- Identifying the Scope
- Identifying the Threats
- Identifying Mitigations
- **Sidebar: Threat Actor Personas**
- Selecting Mitigations
- Assessing the Model
- Getting Started
  - Incremental Threat Modelling
- Modelling Tools
- Integration with Software Development (SDLC)

How can we find security issues in our applications and systems?

Some Approaches

- Static analysis of code
- Fuzzing or other dynamic testing
- Penetration testing
- Production bug reports
- Incident response

“Wouldn’t it be better to find security issues before you write or deploy a line of code?”

--Adam Shostack

WHY Threat Model?

- Improve efficiency
  - Think about security issues early
  - Invest effort more wisely
- Understand requirements better
  - Bring security and development together
  - Shared, maintainable, understanding of risks
- Avoid writing bugs into the code
  - Avoid costs of rework
- Improved stakeholder confidence

A Quick Exercise

- Break into groups of four
- Come to agreement on definitions for:
  - Asset
  - Threat
  - Vulnerability
  - Risk
    - Impact
    - Probability
  - Model
  - Threat Modelling
- Then, we'll compare notes

Terms of Reference

- **Asset –** Anything we need to protect
- **Threat –** Anything that could let someone or something obtain, damage, or destroy an asset, if we fail to protect against it
- **Vulnerability –** A weakness or gap in our protection efforts
- **Risk –** The potential for loss, damage, or destruction of an ***asset***, due to a ***threat***'s having successfully exploited a ***vulnerability***

Terms of Reference

- **Model –** A representation or simplified version of a system. Objectives of a model include:
  1. to facilitate understanding by eliminating unnecessary components,
  2. to aid in decision making by simulating 'what if' scenarios, and
  3. to explain, control, and predict events on the basis of past observations.

  A model contains only those features that are of primary importance to the model maker's purpose.

  All models have a key feature in common: some elements of the actual 'thing' are abstracted.

  --Excerpted from businessdictonary.com

"All models are wrong, but some are useful" – *George Box*

WHAT Is a Threat Model?

A **Threat Model** is a conceptual representation of a *system*, and the *threats* to it that have been identified

- To be useful ***to more than one person***, the model must be captured in a persistent, shareable form
- To ***remain*** useful, the model must be kept up-to-date

WHAT Should Be in a Threat Model?

- Description of the system
- List of assumptions
- List of threats
- Decision on how to address each threat
- Validation approach

WHO Should Create the Model?

- All system stakeholders should take part
  - Security ”experts” play advisory role only
- Assign lifecycle roles:
  - Owner (Accountable)
  - Maintainer (Responsible)

WHEN to Create a Model?

- As early as possible
- Existing system without a Threat Model
  - Start NOW
  - Use Incremental Threat Modelling approach
- Review Threat Model every update cycle
  - Do the proposed changes affect the model?
  - If ‘yes,’ include model update efforts *in the cycle*

WHERE Should It Live?

- With other project/product documentation
  - Well-known location, with reliable backups
  - Ideally, place under revision control
  - Align model versions with product versions

HOW Do I Build a Threat Model?

Let’s start with an exercise

(Credit: Wade and pipes)

Threat Modelling Approaches

- Think like an attacker

Think Like an Attacker

- What does this mean in practice?
- Like thinking like a professional chef!
  - Even if you cook well, that's not nearly the same
  - What are all the things it takes to be the head chef at a popular restaurant?
- Thinking like an attacker – focusing on them – is risky
  - What do they know?
  - What will they do?
  - If you get these wrong, your threat modeling will go astray

Threat Modelling Approaches

- Think like an attacker
- Focus on assets

Focus on Assets

- Assets: valuable things – the business cares!
- But what's an asset?
  - Something an attacker wants?
  - Something you want to protect?
  - A stepping stone?

What would meet the need?

- Need an engineering approach
  - Predictable
  - Reliable
  - Scalable to a large product
- Can't be dependent on one brilliant person

Threat Modelling Approaches

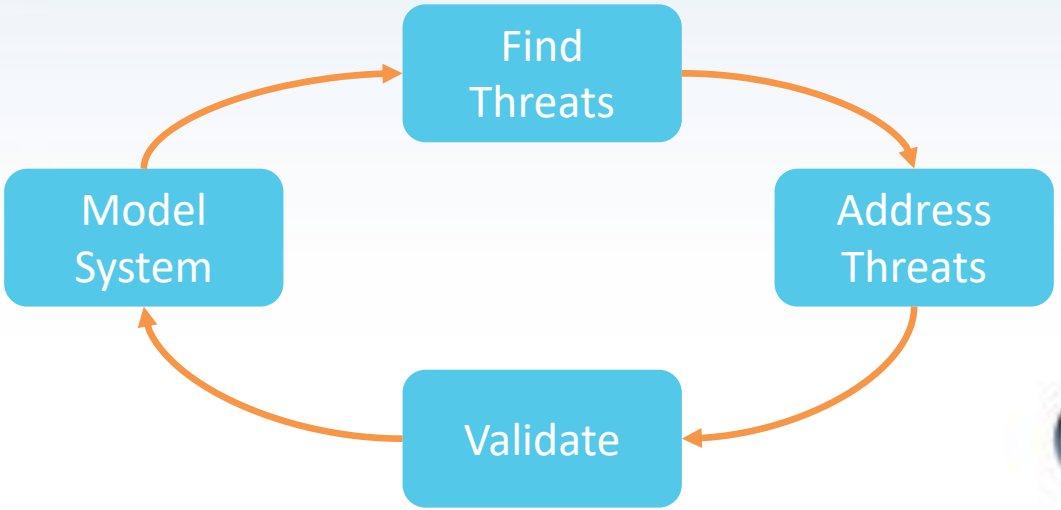
- Think like an attacker
- Focus on assets
- Focus on what's being built

What's Being Built

- Ideally, the team understands it
- Something concrete, with a given context
- We can test it

Shostack's Four Questions

1. What are we building?
2. What can go wrong?
3. What are we going to do about it?
4. How good a job did we do with 1 – 3?



What Are You Building?

- Create a model of the software/system/technology
- A model abstracts away the details so you can look at the whole
  - Diagraming is a key approach
  - Mathematical models of software are rare in commercial environments

What Are You Building?

- Whiteboard diagrams are a great way to start
- Software models for threat modeling usually focus on data flows and boundaries
- DFDs, “swim lanes”, state machines can all help (next slides)

What Are Some Modeling Methods?

- Whiteboard diagrams
- Brainstorming
- Structured ("formal") diagrams
  - Data flow diagrams
  - Swim lanes
  - State machines
- Mathematical representations of code

Trust Boundaries

- Sometimes left implicit in development
  - Must be made explicit for effective threat modeling
- A trust boundary is everywhere two (or more) principals interact
  - Principals are UIDs (unix)/SIDs (Windows) etc.
  - Apps on mobile platforms
  - (Two or more)
- Need to be enforced in some way
  - Best to rely on existing facilities – don't "roll your own"
  - Sometimes not possible (e.g., building a database)

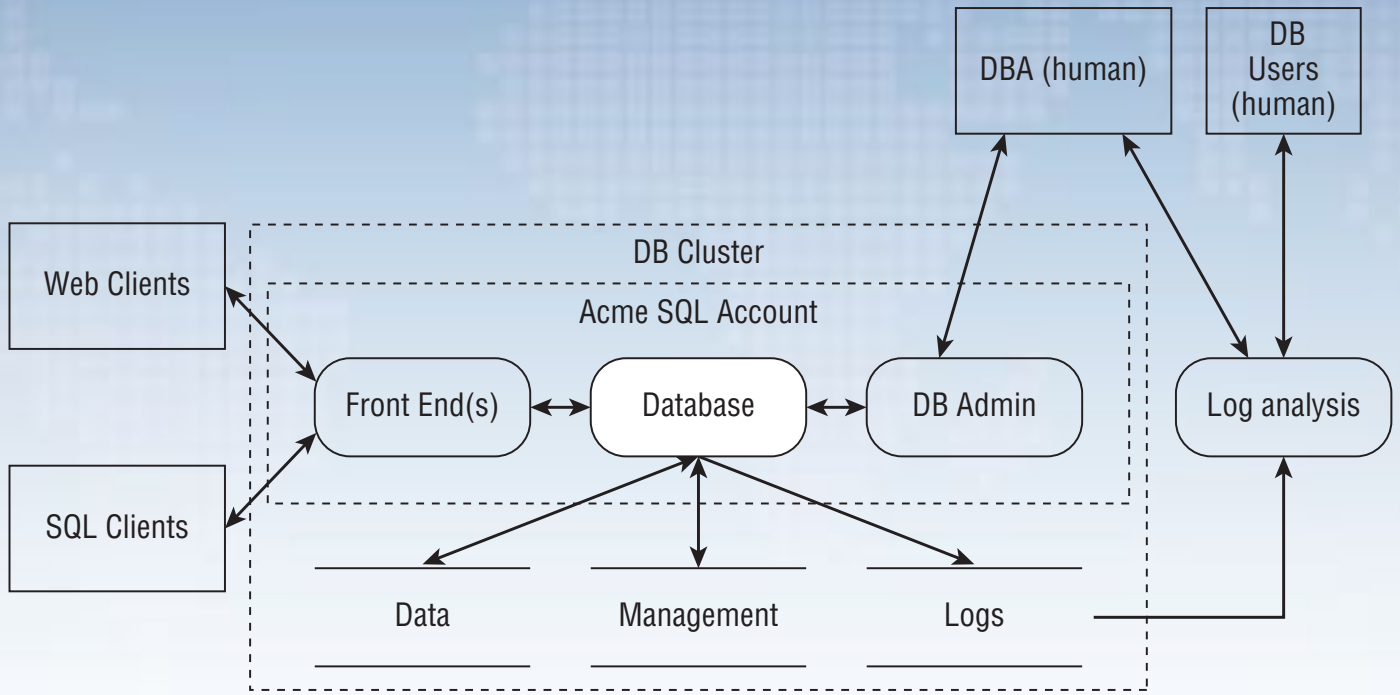
Trust Boundaries

- All interesting boundaries are semi-permeable
  - Air gaps
  - Firewalls
  - Require policy mechanisms (which are hard)
- Formal methods help build boundaries
  - Isolation
  - Type safety
  - Policy languages
  - Reference monitors/kernels

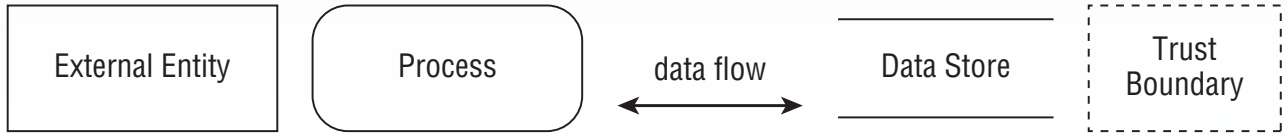
DFD (Data Flow Diagram)

- Developed in the early 70s, and still useful
  - Simple: easy to learn, sketch
  - Threats often follow data
- Abstracts programs into:
  - Processes: your code
  - Data stores: files, databases, shared memory
  - Data flows: connect processes to other elements
  - External entities: everything but your code & data
    Includes people & cloud software
  - Trust boundaries (now made explicit)

Data Flow Diagram (Example)

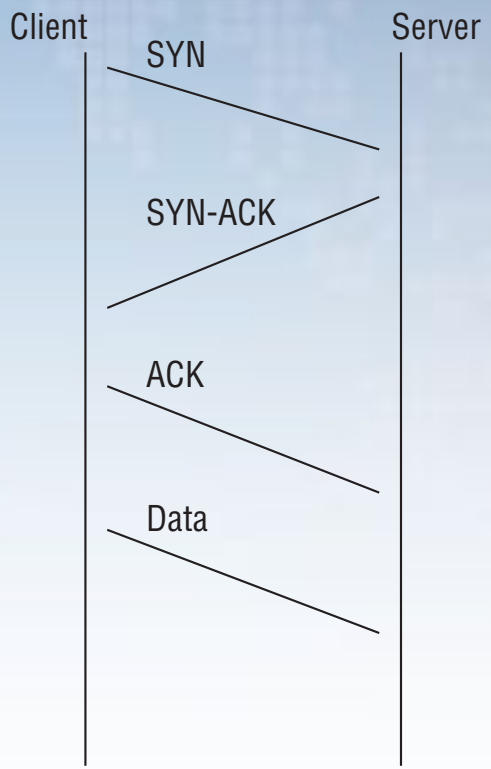


Key:



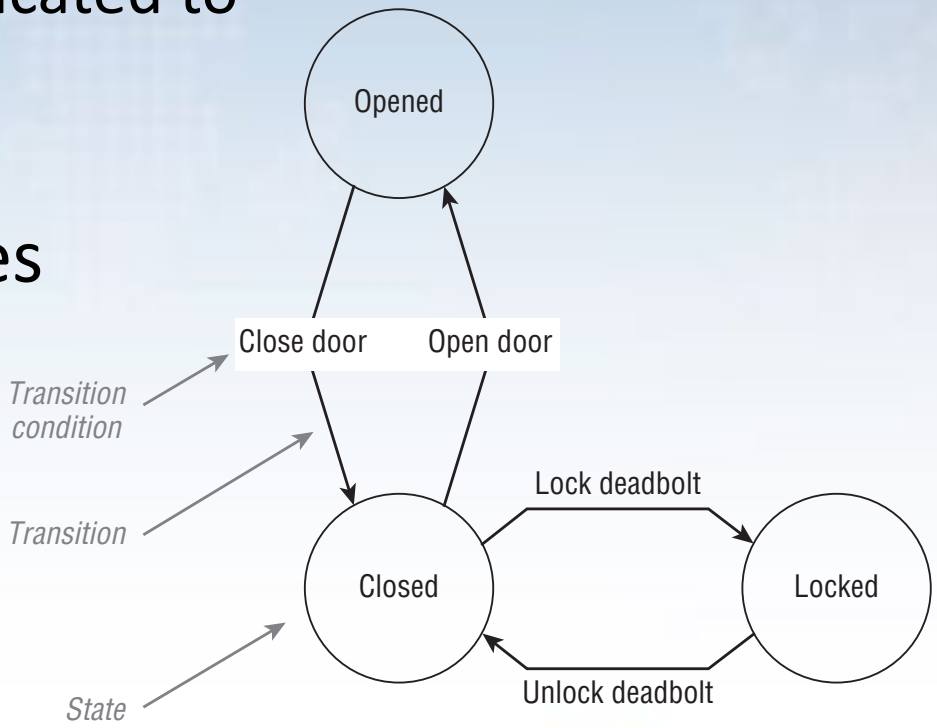
Swim Lane Diagrams

- Show two or more entities communicating, each “in a lane”
- Useful for network communication
- Lanes have implicit boundaries between them



State Machines

- Helpful for considering what *changes* security state
  - For example, unauthenticated to authenticated
  - User to root/admin
- Rarely shows boundaries



Case Study – Act One

The “Locker Situation”

Your organization is already employing “hoteling” and “hot desking.”

HR has contacted your group, saying they’re deploying a new integrated locker system for your offices. The system will allow employees to login to a Cloud-based portal, book a physical locker for a period of time, and assign it to their own ’device’ (BYOD).
Once on site, a user can login with their corporate credentials, on their BYOD device, to access the Cloud service, pair with their locker, and unlock or lock it on demand.
HR has advised that users will be granted a fixed number of “locker credits” per year, which they can use to pay for lease time on lockers. Their credits, usage, and balances will be tracked on the core HR platform. Users can purchase additional “locker credits” through payroll deduction, if needed - purchases are initiated from the app on their BYOD device.

HR has advised that this is non-negotiable, and the system will go love on August 1.

Question #1
What Are We Building?

What Could Go Wrong?

- Fun to brainstorm, but…
- Structured thinking can be more efficient
  - Mnemonics
  - Attack Trees
  - Threat Libraries
- Structure promotes completeness and predictability

What Can Go Wrong?

- STRIDE mnemonic
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege

Applying STRIDE

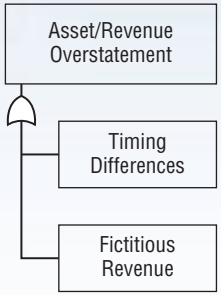
- Consider how each STRIDE threat could impact each part of the model
  - "How could a clever attacker spoof this part of the system?...tamper with?… etc."
- Easier with aids
  - *Elevation of Privilege* card game
  - Attack trees (see Appendix B of text)
  - Experience

STRIDE Variants

- Ways to focus on likely threats
  - STRIDE per element
  - STRIDE per interaction
- *Elevation of Privilege* game
  - Training, structure and execution

Attack Trees

- Structured relationship between attack details
  - Detail (This is a subcategory of that)
  - Present as outline or picture
  - Creation vs. use



1. Go through a door
   a. When it's unlocked:
      i. Get lucky.
      ii. Obstruct the latch plate (the "Watergate Classic").
      iii. Distract the person who locks the door at night.
   b. Drill the lock.
   c. Pick the lock.
   d. Use the key.
      i. Find a key.
      ii. Steal a key.
      iii. Photograph and reproduce the key.
      iv. Social engineer a key from someone.
         1. Borrow the key.
         2. Convince someone to post a photo of their key ring.

Threat Libraries

- Collections of knowledge for you to apply
- More structured than a mnemonic
- More detailed than a tree
- Common Attack Pattern Enumeration and Classification (CAPEC)
  - Most detailed library available today
  - Offers great structure

t CAPEC
ments
ary
IC List
CAPEC Dictionary
anisms of Attack
ains of Attack
rts
nit Content
mmunity
& Citations
ed Activities
ssion List
act Us
patibility
am
irements
ipants
a Declaration
s & Events
dar
Newsletter
ch the Site

CAPEC-66: SQL Injection

Attack Pattern ID: 66
Abstraction: Standard
Status: Draft
Completeness: Complete

Description

Summary

This attack exploits target software that constructs SQL statements based on user input. An attacker crafts input strings so that when the target software constructs SQL statements based on the input, the resulting SQL statement performs actions other than those the application intended.

SQL Injection results from failure of the application to appropriately validate input. When specially crafted user-controlled input consisting of SQL syntax is used without proper validation as part of SQL queries, it is possible to glean information from the database in ways not envisaged during application design. Depending upon the database and the design of the application, it may also be possible to leverage injection to have the database execute system-related commands of the attackers' choice. SQL Injection enables an attacker to talk directly to the database, thus bypassing the application completely. Successful injection can cause information disclosure as well as ability to add or modify data in the database. In order to successfully inject SQL and retrieve information from a database, an attacker:

Attack Execution Flow

Explore

1. **Survey application**:

   The attacker first takes an inventory of the functionality exposed by the application.

   Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	Spider web sites for all available links	env-Web
2	Sniff network communications with application using a utility such as WireShark.	env-ClientServer env-Peer2Peer env-CommProtocol

   Outcomes

ID	Type	Outcome Description
1	Success	At least one data input to application identified.
2	Failure	No inputs to application identified. Note that just because no inputs are identified does not mean that the application will not accept any.

Experiment

1. **Determine user-controllable input susceptible to injection**:

   Determine the user-controllable input susceptible to injection. For each user-controllable input that the attacker suspects is vulnerable to SQL injection, attempt to inject characters that have special meaning in SQL (such as a single quote character, a double quote character, two hyphens, a parenthesis, etc.). The goal is to create a SQL query with an invalid syntax.

   Attack Step Techniques

ID	Attack Step Technique Description	Environments
1	Use web browser to inject input through text fields or through HTTP GET parameters.	env-Web
2	Use a web application debugging tool such as Tamper Data, TamperIE, WebScarab,etc. to modify HTTP POST parameters, hidden fields, non-freeform fields, etc.	env-Web
3	Use network-level packet injection tools such as netcat to inject input	env-Web env-ClientServer env-Peer2Peer env-CommProtocol
4	Use modified client (modified by reverse engineering) to inject input.	env-ClientServer env-Peer2Peer env-CommProtocol

Finding the Threats

- Start at the beginning of your project
  - Create a model of what you're building
  - Do a first pass for threats
- Dig deep as you work through features
  - Think about how threats apply to your mitigations
- Check your design and model match as you get close to shipping

Attackers Respond to Our Defenses

OWASP
Open Web Application Security Project

Orders of Mitigation

By Example:

Order	Threat	Mitigation
1st	Window smashing	Reinforced glass
2nd	Window smashing	Alarm
3rd	Cut alarm wire	Heartbeat signal
4th	Fake heartbeat	Cryptographic signal integrity

- Thus window smashing is a first order threat, cutting alarm wire, a third-order threat
- Easy to get stuck arguing about orders
  - Are both stronger glass & alarms 1st order mitigations? (Who cares?!)
- Focus on interplay between mitigations and further attacks

Track Threats and Assumptions

- Lots of structures work for this
- Use what works reliably for you

Threat Tracking Tables

Diagram Element	Threat Type	Threat	Bug ID
Data flow #4, web server to business logic	Tampering	Add orders without payment checks	4553 “Need integrity controls on channel”
	Info disclosure	Payment instruments sent in clear	4554 “need crypto” #PCI

Threat Type	Diagram Element(s)	Threat	Bug ID
Tampering	Web browser	Attacker modifies our JavaScript order checking	4556 “Add order-checking logic to server”
	Data flow #2 from browser to server	Failure to authenticate	4557 “~~Add~~ enforce HTTPS everywhere”

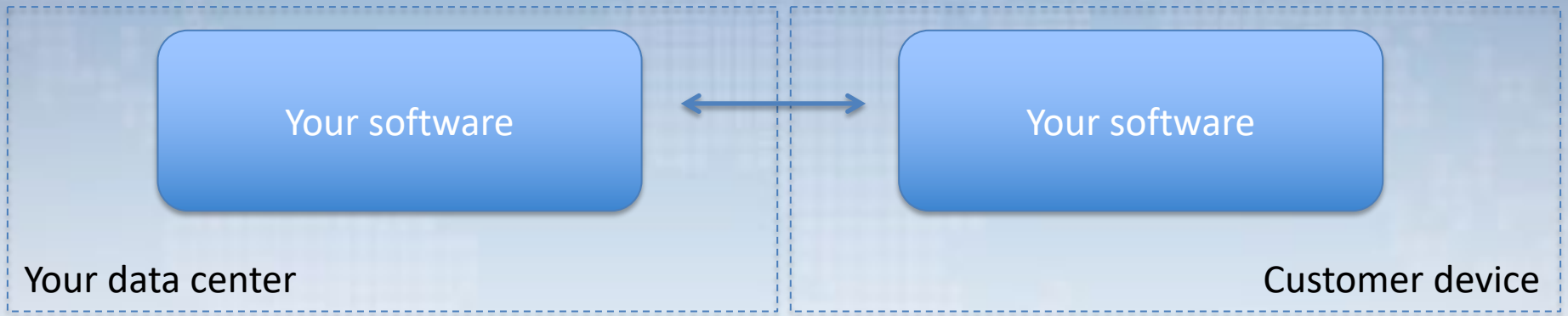
Both are fine, help you iterate over diagrams in different ways

Assumption Tracking Table

Assumption	Impact if it's wrong	Who to talk to	Who's following up	Follow-up by date	Bug #
It's ok to ignore denial of service within the data center	Availability will be below spec	Alice	Bob	April 15	4555

- Impact is sometimes so obvious it's not worth filling out
- Who to talk to is not always obvious, it's ok to start out blank
- Tracking assumptions in bugs helps you not lose track
  - Treat the assumption as a bug – you need to resolve it

Customer/Vendor Boundaries



- There is always a trust boundary when:
  - Your code goes to someone else's (device/premises)
  - Their data comes to your code
- You need to think about it while deciding what happens over the data flow shown

Case Study – Act Two

Question #2
What Can Go Wrong?

What Are We Going to Do about It?

- For each threat, we could:
  - Remove it (Avoid the Risk)
  - Mitigate with standard or custom approaches
  - Accept it
  - Transfer the risk
- For each assumption:
  - Check it
  - Discovering incorrect assumptions requires reconsidering what can go wrong

Remove the Threat

- The most effective way to address a security threat is to remove the functionality
  - For example, if SSL doesn't have a "heartbeat" message, the "heartbleed bug" couldn't exist
  - You can only take this so far
  - Risk trade-offs are more common

Mitigate the Threat

- Add/use technology to prevent attacks
- For example, prevent tampering:
  - Network: Digital signatures, cryptographic integrity tools, crypto tunnels such as SSH or IPsec
- Developers and SysAdmins each have toolkits for mitigating problems
- Standard approaches are available
  - Tested, well-studies, supported
- But…sometimes you need a custom approach

Technical Mitigations

Threat	Mitigation Technology	Developer Example	SysAdmin Example
Spoofing	Authentication	Digital signatures, Active directory, LDAP	Passwords, crypto tunnels
Tampering	Integrity, permissions	Digital signatures	ACLs/permissions, crypto tunnels
Repudiation	Fraud prevention, logging, signatures	Customer history risk management	Logging
Information disclosure	Permissions, encryption	Permissions (local), PGP, SSL	Crypto tunnels
Denial of service	Availability	Elastic cloud design	Load balancers, more capacity
Elevation of privilege	Authorization, isolation	Roles, privileges, input validation for purpose, (fuzzing*)	Sandboxes, firewalls

* Fuzzing/fault injection is not a mitigation, but a testing technique

Custom Mitigations

- Sometimes the standard technologies don't work for your situation
- Custom (home-grown) mitigation is an option
- Easy to get custom mitigation wrong
  - And…you don't have a larger community supporting it
- Testing is difficult and expensive

Accept the Risk

- Works best when it's your risk
  - Your organization can accept risk
  - Be careful about "accepting" risk for your customers.
- Customer risk acceptance
  - Via user interface
  - Sometimes the customer has details you can't have (is this network your work or a coffee shop?)

Transfer the Risk

- Via license agreements, terms of service, etc.
- Silently
- Both can lead to unhappy customers
  - Threat that no one reads ToS
  - Surprise!
  - Media blowups

Common Mistakes

- Custom mitigations because they're fun
- Fuzzing as a mitigation
- Not covering all threats

Case Study – Act Three

Question #3
What Are We Going to Do?

Issue Prioritisation

- Wait and see
- Easy fixes first
- Threat ranking - Bug Bar
- Cost/damage estimation approaches
  - Risk “traffic light”

Wait and See

- Can be risky
- Requires some way of seeing
  - Change detection
  - Signature-based detection
  - Anomaly-based detection
  - Impact detection

Easy Fixes First

May be helpful when getting started

- Benefit: demonstrate value early
- Risk: fixing the wrong things

Bug Bar

- Assess threat against criteria, assign priority from where it fits
- Microsoft Example

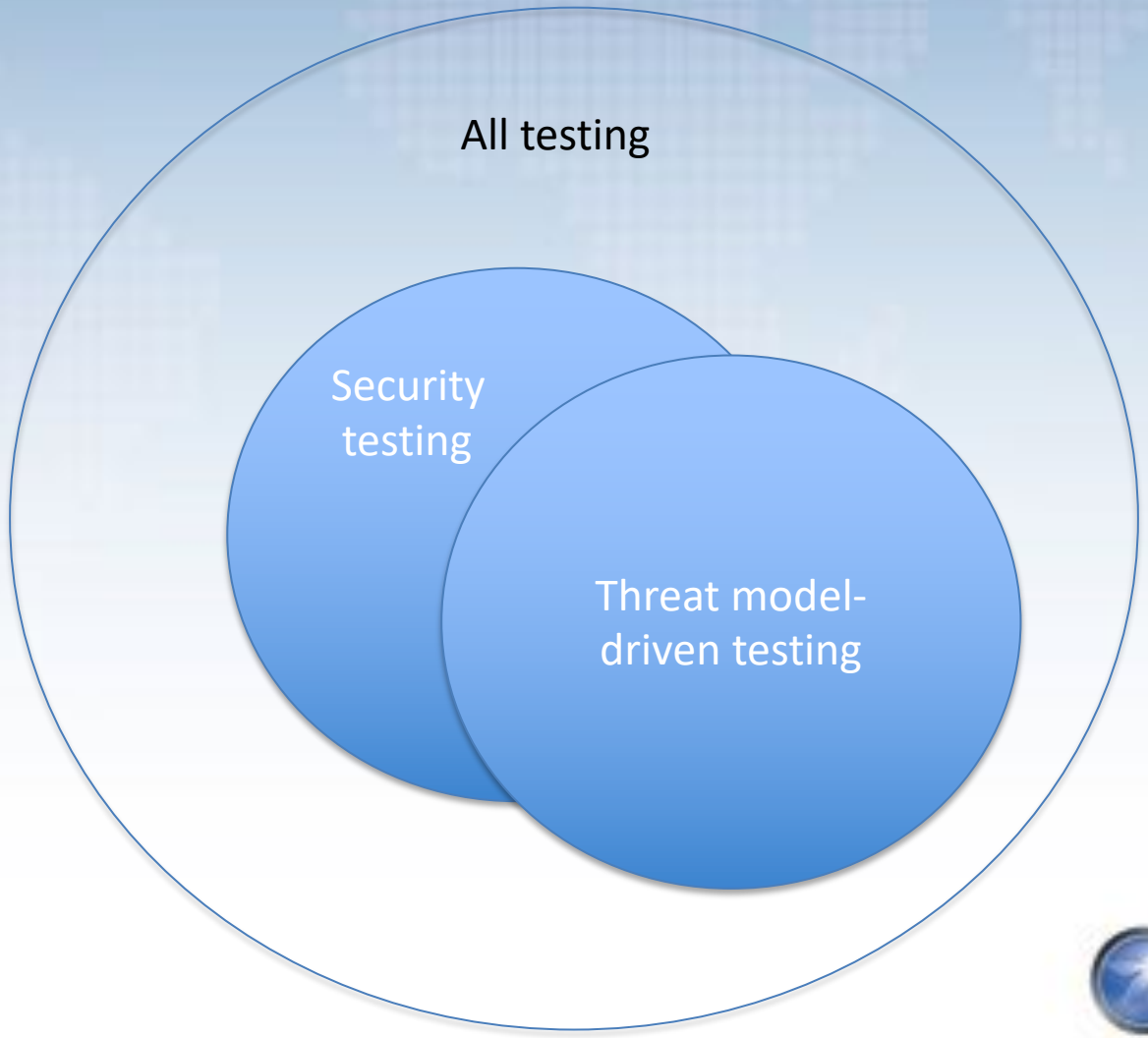
Cost/Damage Estimation

- Likelihood/Impact
  - Hard to do well
  - Predicting odds and difficulty is challenging
- Factor Analysis of Information Risk (FAIR)
  - Useful
  - Can be more time consuming than a bug bar

How Did We Do?

- Testing Our Software
- Validating Our Threat Modelling Work

The Testing Context



Testing Our Software

- All threats you find can be tested
- In Test-Driven Development (TDD), threat modeling is a great way to design tests
- Start with a test to execute the threat
- Continue with tests that bypass mitigations (aka, 2nd order attacks)
- Automated vs. manual

Penetration Testing

- Improve security of your code, by breaking it
- Differs from threat modeling
  - Done late
  - Hard to judge scope
  - Sometimes “black box” where testers start without knowledge of system
  - But…often required for compliance (e.g., HITrust)

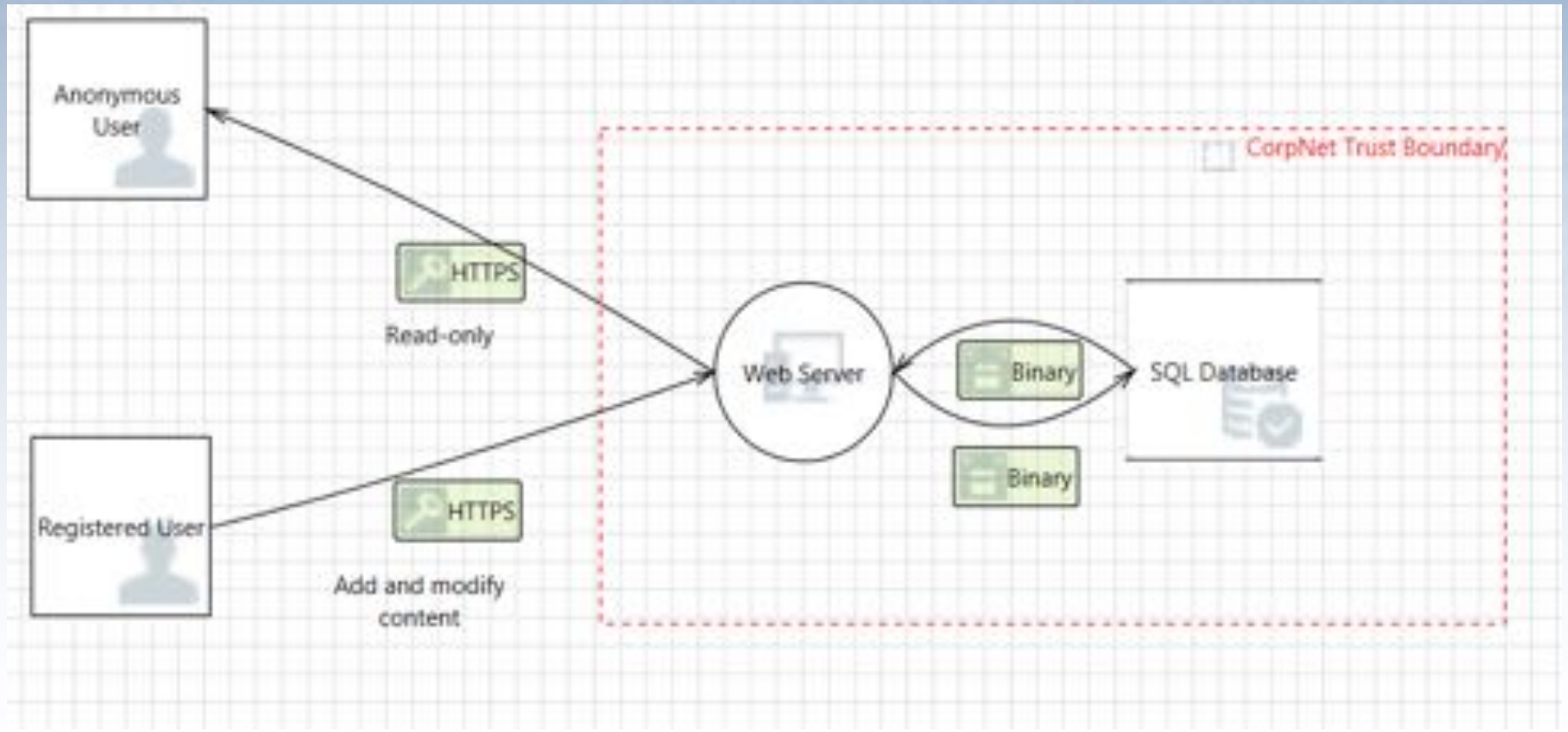
Validating Our TM Work

- Check software model/reality conformance
- Check that each task and process is done
- Bug checking: Look at each TM bug
  - Is it closed properly ("Fixed," and not "Won't Fix")?
  - Is there a test case?
  - Tags/labels on bug tickets really helpful here

Incremental Threat Modelling

- Existing system
- No – or out of date – threat model
- New work is proposed on the system, and threat modelling is required

Example

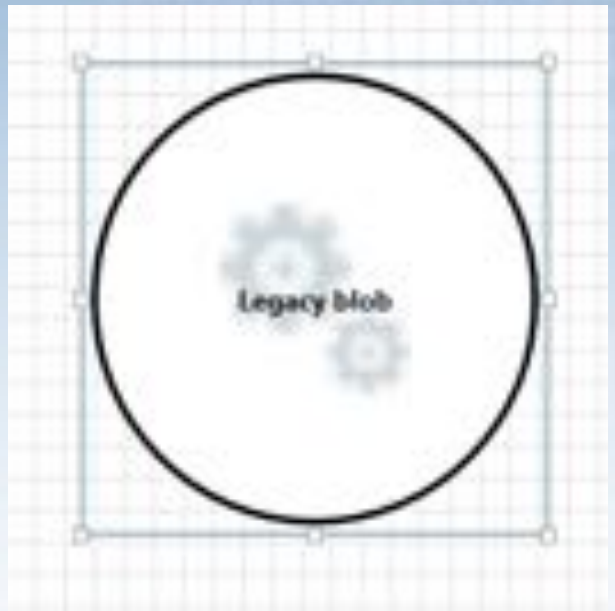


Credit: Irene Michlin (2017)

New Functionality to Be Added

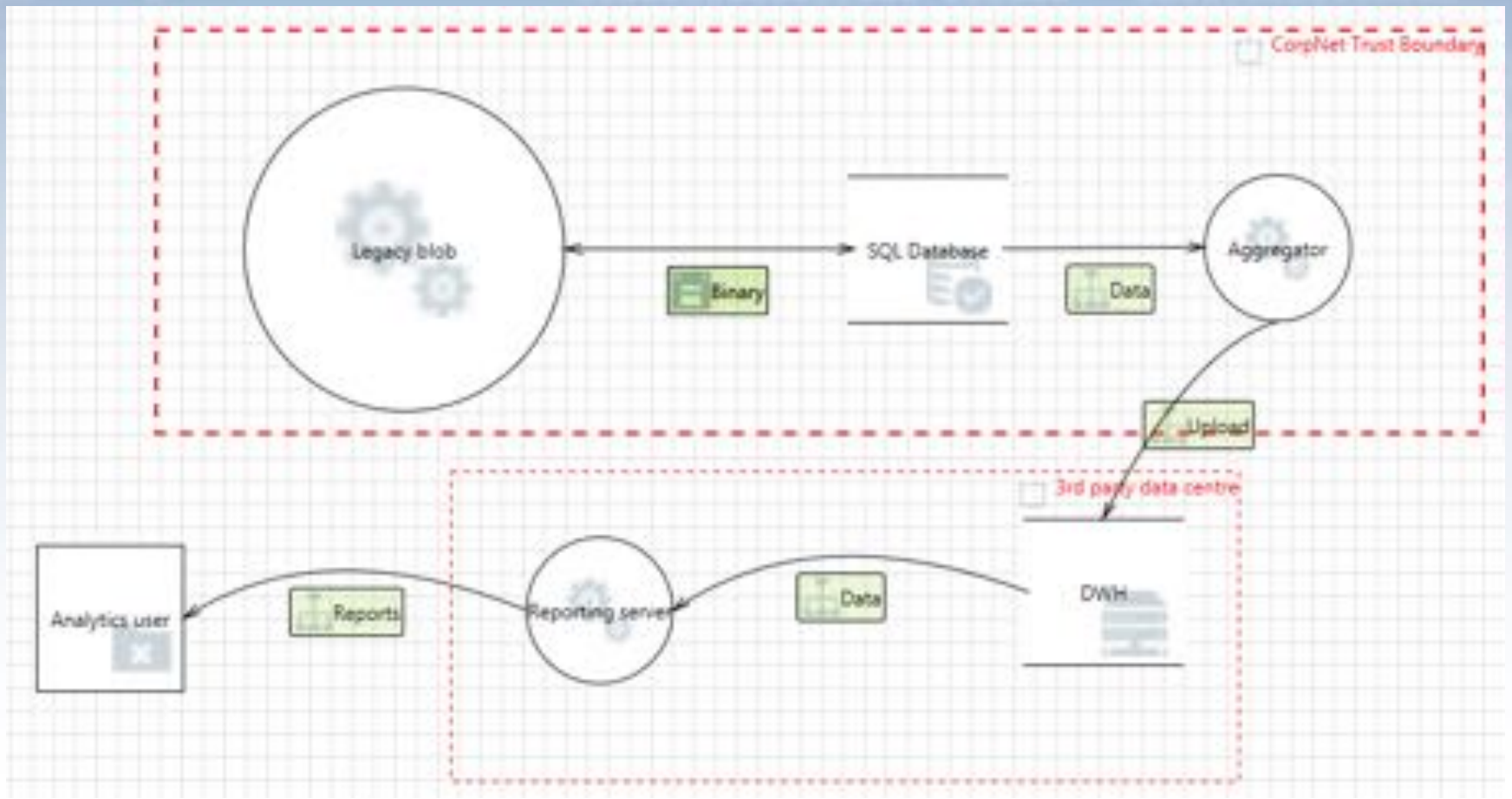
- Add third-party reporting and analytics to existing system
  - They host a data warehouse and reporting server
  - License to use Web-based analytics app, which can query reporting server
- We'll build an aggregator process in-house
  - Extract data from our database
  - Aggregate and upload to DWH (using their API)

Pretend the Model Doesn't Exist



- Tease out the elements necessary to model NEW functionality
  - ”data warehouse”
  - “our database”
  - “aggregator process”
  - “upload to DWH”
  - “reporting server”
  - “Web-based analytics app”

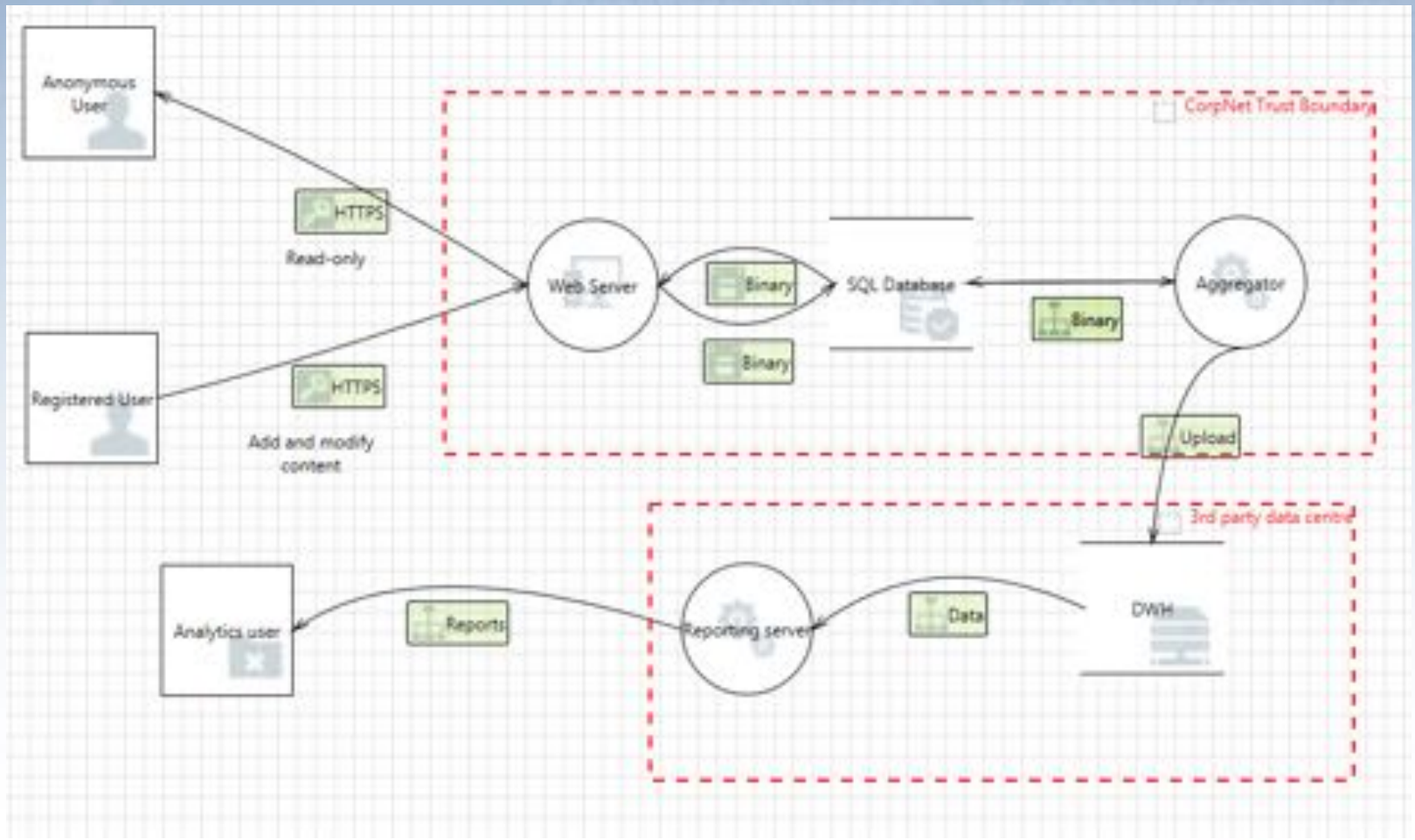
Our (Incremental) Model



Relevant Threats

- Spoofing
  - Can an attacker upload data on our behalf?
  - How do we authenticate to the destination before uploading?
- Tampering / Information Disclosure
  - Can an attacker sniff data, or tamper with it?
- Repudiation
  - Can the DWH claim we didn't send that data?
  - Or sent data exceeding our quota?
- Denial of Service
  - Is there an availability SLA for uploads?
- Privacy
  - Can our data aggregation be reverse engineered?
  - Are we required to notify users a third party is involved?

What about the Rest of the System?



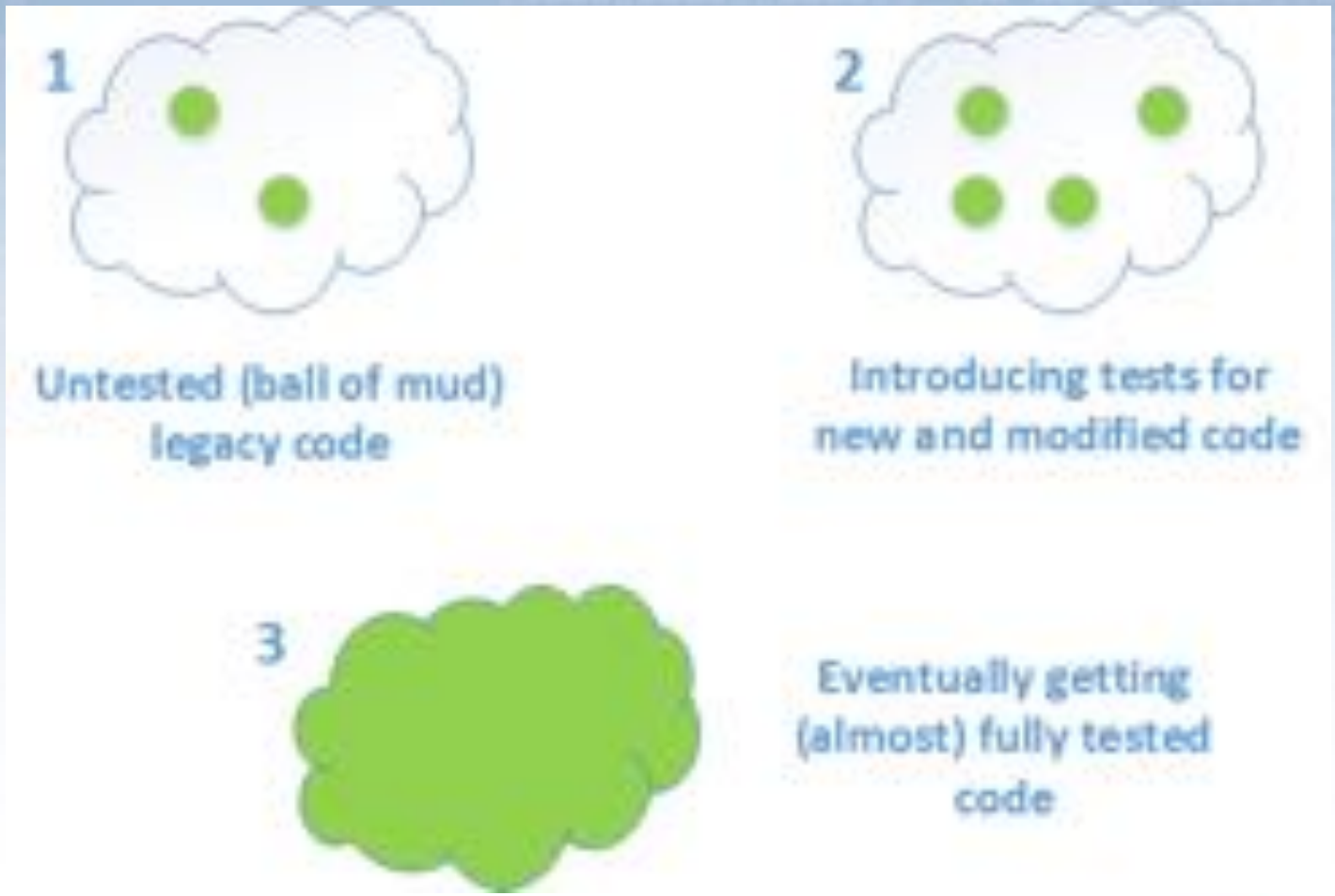
“Legacy Blob” Threats - Irrelevant

- Can a registered user inject malicious content?
  - We’re not making it worse
- Can an anonymous user bypass access controls and modify something?
  - We’re not making it worse
- Is our datacenter infrastructure secure?
  - We’re not making it worse (are we sure?)
- Can an analytics user abuse licensing?
  - Not our problem

Caveats

- "Not our problem"
  - But…what if our task is to evaluate several third-party providers?
- "We're not making it worse"
  - But…if we discover a catastrophic flaw in the "legacy blob," we have to address it

Does This Look Familiar?



Eventually…We Need the Full Picture

- What we don't know CAN hurt us
- Every system is greater than the sum of its parts
  - At some point, we have to model holistically

Eventually is Better than Never

If we never start, because doing it all at once is "too hard," we'll never have the full picture.