Threat Modeling with STRIDE

Slides adapted from *Threat Modeling: Designing for Security* (Wiley, 2014) by Adam Shostack

Wouldn't it be better to find security issues before you write a line of code?

So how can you do that?

Ways to Find Security Issues

- Static analysis of code
- Fuzzing or other dynamic testing
- Pen test/red team
- Wait for bug reports after release

Ways to Find Security Issues (2)

- Threat modeling!
  - Think about security issues early
  - Understand your requirements better
  - Don’t write bugs into the code
  - And the subject of this lesson

So…how do you threat model?

Definitions

- What is a threat?
- How is it different from a
  - vulnerability,
  - risk,
  - or just a problem?
- What is a model?

Think Like an Attacker?

- Like thinking like a professional chef!
  - Even if you can, are you the chef at Olive Garden or Mario Batalli’s?
- Thinking like an attacker – or focusing on them is risky
  - What do they know? What will they do?
  - If you get these wrong, your threat modeling will go astray
- So don’t start from attackers!

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?

Focus On What You’re Building!

- Need an engineering approach
  - Predictable
  - Reliable
  - Scalable to a large product
- Can’t be dependent on one brilliant person
- Ideally, you understand it
- Concrete and testable?

How to Threat Model (Summary)

- What are you building?
- What can go wrong?
- What are you going to do about it?
- Check your work on 1-3

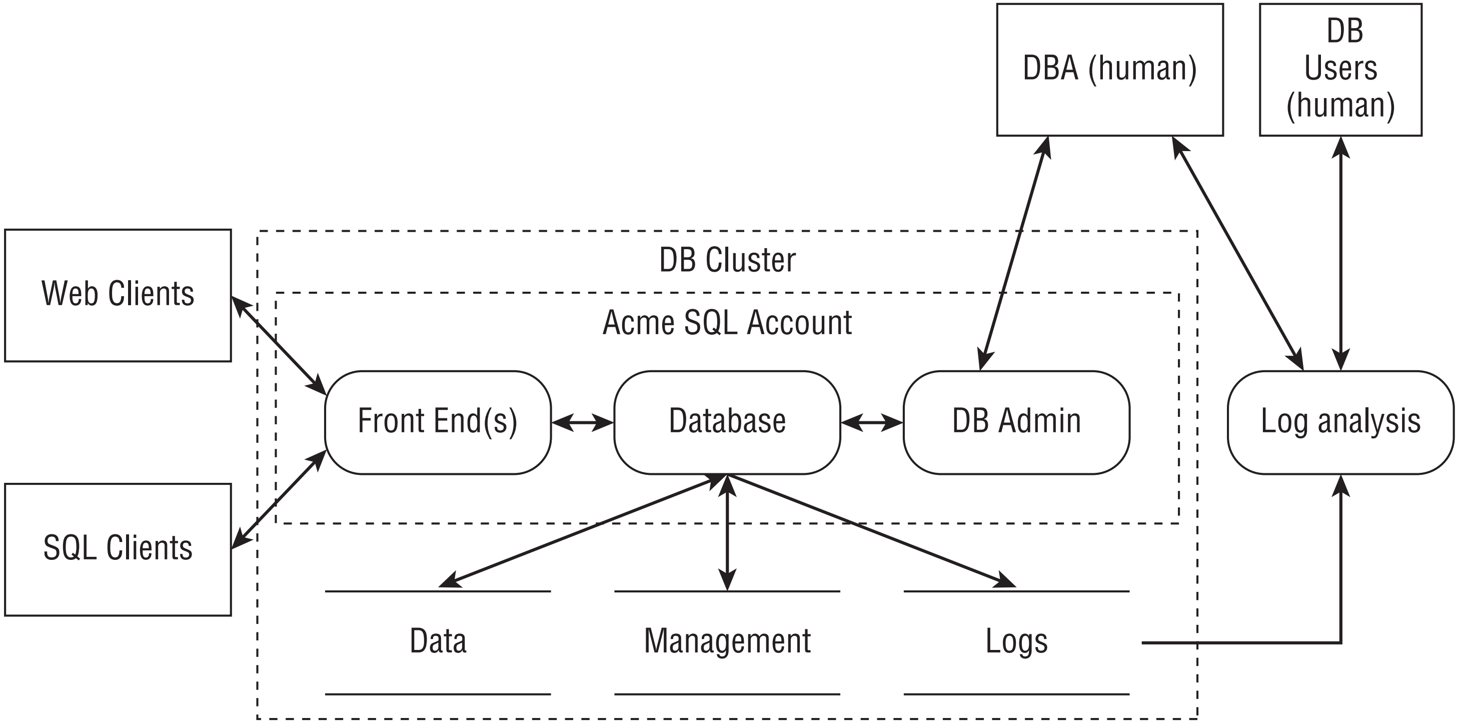
What Are You Building?

- Create a model of the software/system/ technology
- A model abstracts away the details so you can look at the whole

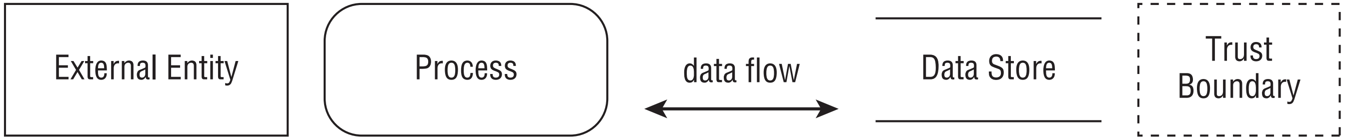
What Are Some Modeling Methods?

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

Data Flow Diagram (Example)



Key:

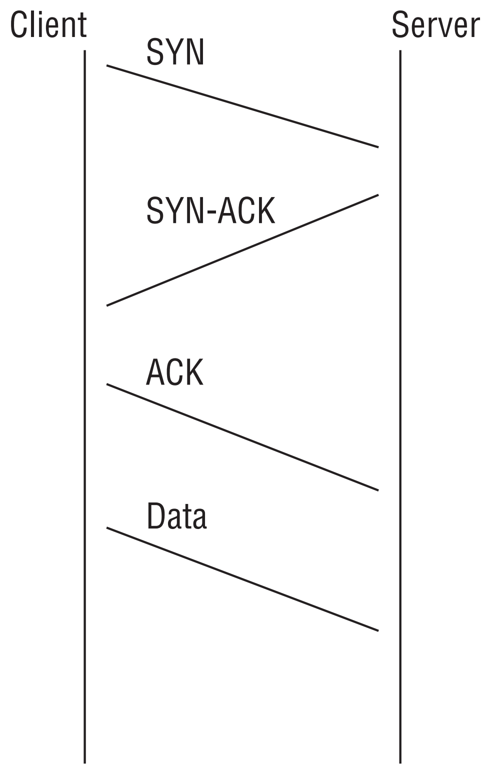


Trust Boundaries

- A trust boundary is everywhere two (or more) principals interact
- 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

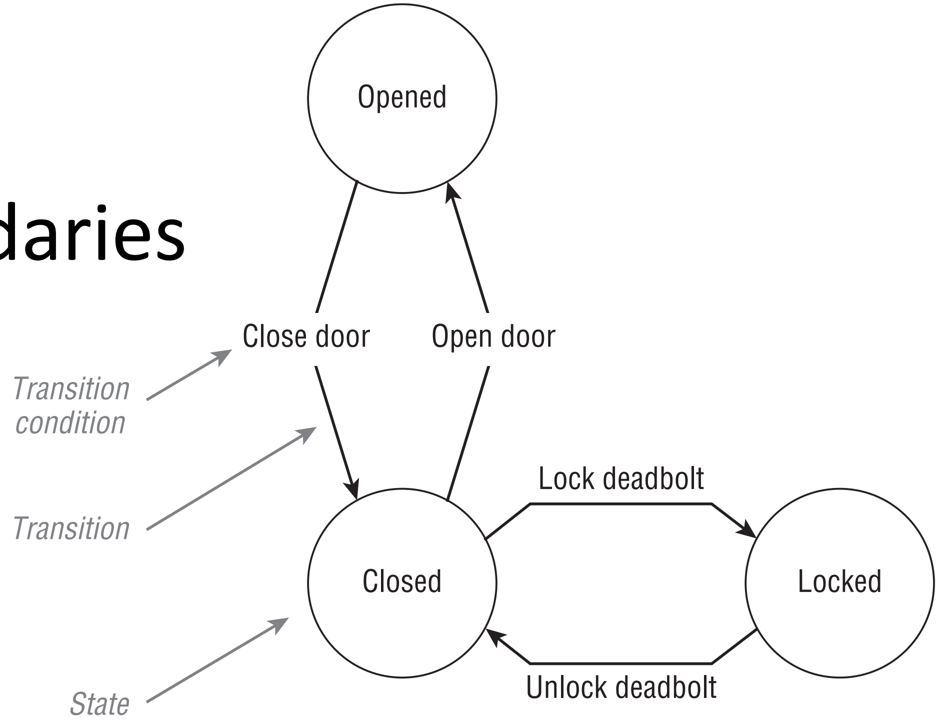
Swim Lane Diagrams

- 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



How to Threat Model (Summary)

- What are you building?
- What can go wrong?
- What are you going to do about it?
- Check your work on 1-3

What Can Go Wrong?

- Fun to brainstorm
- Mnemonics, trees or libraries of threats can all help structure thinking
- Structure helps get you towards completeness and predictability
- STRIDE is a mnemonic
  - Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
  - Easy, right?

STRIDE

Threat	Property Violated	Definition	Example
Spoofing	Authentication	Impersonating something or someone else.	Pretending to be any of Bill Gates, Paypal.com or ntdll.dll
Tampering	Integrity	Modifying data or code	Modifying a DLL on disk or DVD, or a packet as it traverses the network
Repudiation	Non-repudiation	Claiming to have not performed an action.	“I didn’t send that email,” “I didn’t modify that file,” “I *certainly* didn’t visit that web site, dear!”
Information Disclosure	Confidentiality	Exposing information to someone not authorized to see it	Allowing someone to read the Windows source code; publishing a list of customers to a web site.
Denial of Service	Availability	Deny or degrade service to users	Crashing Windows or a web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole.
Elevation of Privilege	Authorization	Gain capabilities without proper authorization	Allowing a remote internet user to run commands is the classic example, but going from a limited user to admin is also EoP.

Using 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.”
- Track issues as you find them
  - “attacker could pretend to be a client & connect”
- Track assumptions
  - “I think that connection is always over SSL”
- Consolidate into an attack tree

Spoofing On the Local Machine

Threat Example	What the Attacker Does	Notes/Examples
Spoofing a process	Creates a file before the real process	Then your process relies on it
	Abuses names	Create a version of “sudo” and alter PATH
Spoofing a filename	Creates a file in the local directory	Library, executable or config file
	Creates a link, changes it	Also called ‘race condition’ or TOCTOU
	Creates many files in a target directory	Code can easily create all possible /tmp/foo.random

Spoofing Over a Network

Threat Example	What the Attacker Does	Notes/Examples
Spoofing a machine	ARP spoofing	
	IP spoofing	
	DNS spoofing	
	DNS compromise	Can be at the TLD, registrar or DNS server
	IP redirection	
Spoofing a person	Take over account	"Stranded in London"
	Set the display name	
Spoofing a role	Declares themselves to be that role	Sometimes opening a special account, setting up a domain/website, other "verifiers"

Tampering with a File

Threat Example	What the Attacker Does	Notes/Examples
Modifying a file…	… which you own and you rely on	
	… which they own and you rely on	
Modifying a file on a server…	…you own	
	…they own (or take over)	
Modifies links or redirects		Redirects are super-common on the web, and often rot away

Tampering with Memory

Threat Example	What the Attacker Does	Notes/Examples
Modifying code	Changes your code to suit themselves	Hard to defend against if the attacker is running code inside the trust boundaries
Modifying data they've supplied	Supplies data to a pass by reference API, then changes it	Works because of TOCTOU issues
	Supplies data into a shared memory segment, then changes it	

Tampering with a Network

Threat Example	What the Attacker Does	Notes/Examples
Redirects the flow of data to their machine	Uses an attack at some network layer to redirect traffic	Pakistan/YouTube
Modifies data flowing over the network		Easier (and more fun) with wireless networks
	Uses network tampering to improve spoofing attacks	

Repudiation

Threat Example	What the Attacker Does	Notes/examples
Repudiating an action	Claims to have not clicked	Maybe they did, maybe they didn't, maybe they're honestly confused
	Claims to not have received	1. Electronic or physical 2. Receipt is strange; does a client downloading email mean you've seen it? Did a network proxy pre-fetch images? Was a package left on a porch?
	Claims to be a fraud victim	
	Uses someone else's account	

Repudiation Attacks on Logs

Threat Example	What the Attacker Does	Notes/Examples
	Discovers there are no logs	
Modifies data flowing over the network	Puts data in the logs to confuse you	</tr></html>

Information Disclosure (Processes)

Threat Example	What the Attacker Does	Notes/Examples
Extracts user data	Exploits bugs like SQL injection to read db tables	Can find this by looking to data stores, but here the issue is the process returning data it shouldn't
	Reads error messages	
Extracts machine secrets	Reads error messages	Cannot connect to database 'foo' as user 'sql' with password '&IO*(^&'
	Exploits bugs	"Heartbleed"

Information Disclosure (Data Stores)

Sub-category	What the Attacker Does
Permissions	Take advantage of missing or inappropriate ACLs
	Take advantage of bad database permissions
	File files protected by obscurity
Security	Find crypto keys on disk or in memory
	Get data from logs/temp files
	Get data from swap files
	See interesting information in filenames/directory names
Network	See data traversing a network
Misc	Obtain device, boot in new OS

Information Disclosure (Data Flow)

Sub-category	What the Attacker Does
Network	Read data on a network
	Redirects traffics to enable reading data on the network
Metadata	Learns secrets by analyzing traffic
	Learns who talks to whom by watching the DNS
	Learns who talks to whom by analyzing social network information

Denial of Service

Threat Example	What the Attacker Does	Notes/Examples
Against a process	Absorb memory (ram or disk)	
	Absorb CPU	
	Uses a process as an amplifier	
	Against business logic	"Too many login attempts"
Against a data store	Fills the data store	
	Makes enough requests to slow the system	
Against a data flow	Consumes network resources	

Can be temporary (as the attack continues; fill the network) or persist beyond that (fill a disk)

Elevation of Privilege (“EoP”)

Threat Example	What the Attacker Does	Notes/Examples
EoP Against process via corruption	Sends inputs the code doesn’t handle properly	Very common, usually high impact
	Gains read/write access to memory	Writing memory more obviously bad
EoP via misused authorization checks		
EoP via buggy authorization checks		Centralizing checking makes consistency, correctness easier
EoP via data tampering	Modify bits on disk	

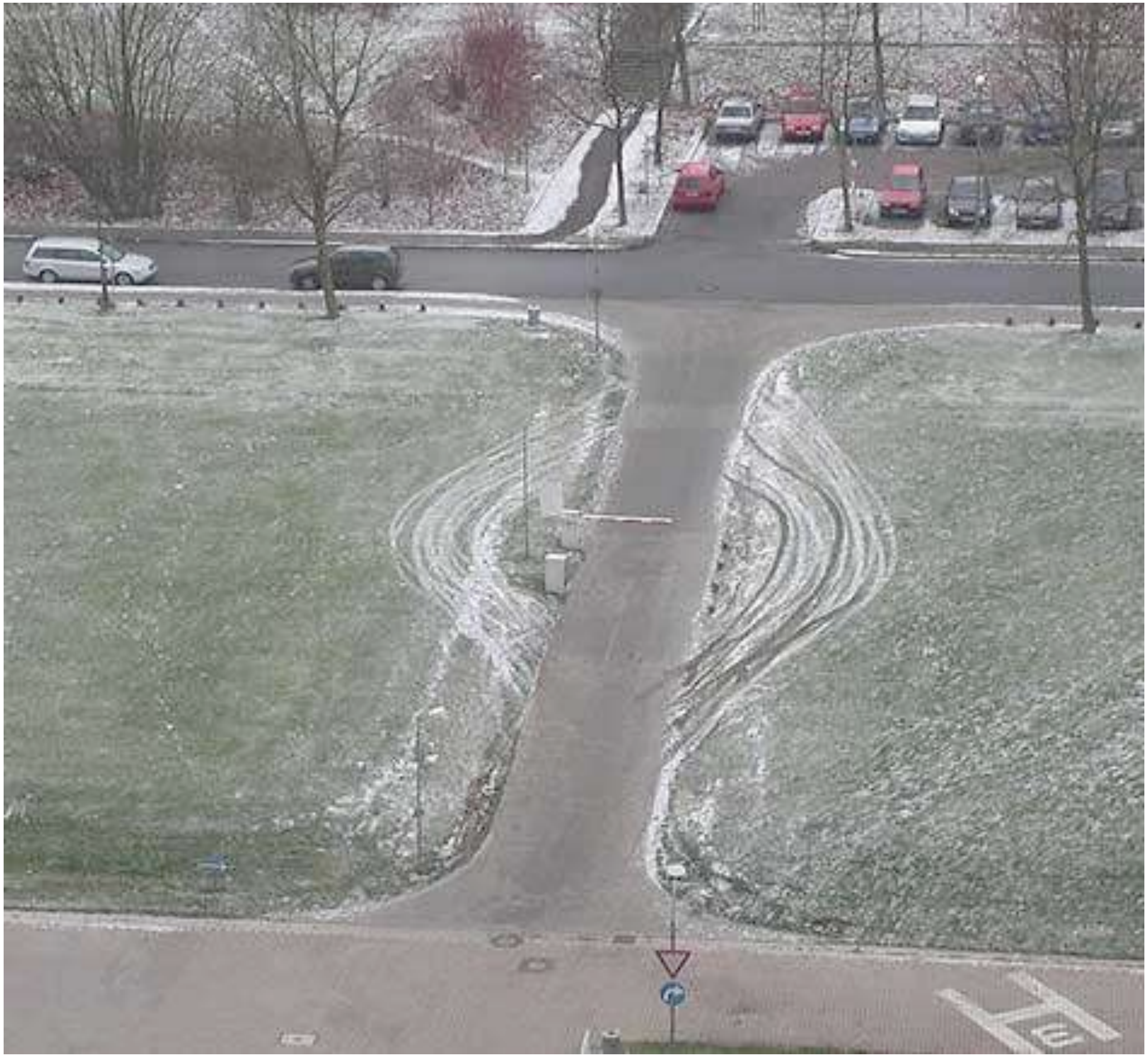
Using STRIDE

- 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."
- Track issues as you find them
  - "attacker could pretend to be a client & connect"
- Track assumptions
  - "I think that connection is always over SSL"
- Consolidate into an attack tree

When to Find 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 & model matches as you get close to shipping

Attackers Respond to Your Defenses



Playing Chess

- The ideal attacker will follow the road you defend
  - Ideal attackers are like spherical cows — they're a useful model for some things
- Real attackers will go around your defenses
- Your defenses need to be broad and deep

“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 the concept of interplay between mitigations & further attacks

How to Approach Software

- Depth first
  - The most fun and “instinctual”
  - Keep following threats to see where they go
  - Can be useful skill development, promoting “flow”

- Breadth first
  - The most conservative use of time
  - Most likely to result in good coverage

Tracking Threats and Assumptions

- There are an infinite number of ways to structure this
- Use the one that works reliably for you
- (Hope doesn’t work reliably)

Example 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

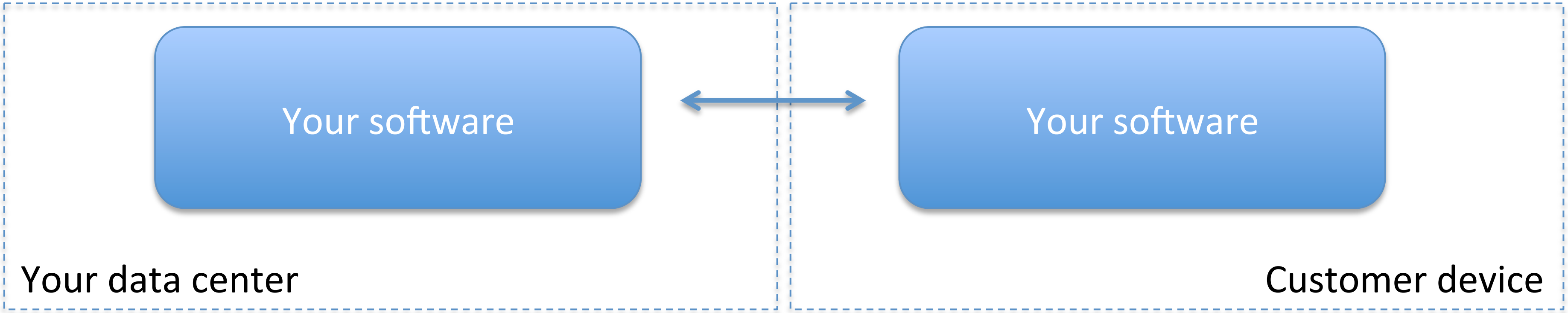
Example Assumption Tracking

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

The Customer/Vendor Boundary

- There is always a trust boundary when:
  - Your code goes to someone else's (device/premises)
  - Their data comes to your code
- Lawyers, pretending do not eliminate human trust issues
- You need to think about it while deciding what happens over the data flow shown



Generic API Threat Model

- Perform security checks inside the boundary
- Copy before validation for purpose
  - Is http://evil.org/pwnme.html “valid”?
- Define the purpose for data, validate near that definition
- Manage error reporting
- Document what checks happen where
- Do crypto in constant time
- Address the security requirements for your API

How to Threat Model (Summary)

- What are you building?
- What can go wrong?
- What are you going to do about it?
- Check your work on 1-3

What Are You Going to Do About It?

- For each threat:
  - Fix it!
  - Mitigate with standard or custom approaches
  - Accept it?
  - Transfer the risk?
- For each assumption:
  - Check it
  - Wrong assumptions lead to reconsider what goes wrong

Fix It!

- The best way to fix a security bug is to remove functionality
  - For example, if SSL doesn’t have a “heartbeat” message, the “heartbleed bug” couldn’t exist
  - You can only take this so far
  - Oftentimes end up making risk tradeoffs
- Mitigate the risk in various ways (next slide)

Mitigate

- Add/use technology to prevent attacks
- For example, prevent tampering:
  - Network: Digital signatures, cryptographic integrity tools, crypto tunnels such as SSH or IPsec
- Developers, sysadmins have different toolkits for mitigating problems
- Standard approaches available which have been tested & worked through
- Sometimes you need a custom approach

Some Technical Ways to Address

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 great testing technique

See chapter 8, *Threat Modeling* for more

Custom Mitigations

- Sometimes the standard technologies don't work for your situation
- Requires custom mitigations (or risk acceptance)
- Easy to get a custom mitigation wrong
- Hard and expensive to test (page 176)

Accepting 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?)

Transferring Risk

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

Some Technical Ways to Address

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 great testing technique

See chapter 8, *Threat Modeling* for more

Understanding Authentication

- *To prove or show (something, esp. a claim or an artistic work) to be true or genuine*
- Applies to all sorts of things
  - Programs or libraries on disk
  - Remote machines
  - People (a complex subject, covered later in the course)

Tactics for Authentication

- Local
  - Leverage the OS/program (database, web server, etc)
  - Defaults are not always secure
- Remote machines
  - Cryptographic methods (more reliable)
  - Consistency checking DNS, IP, route (less reliable)
- Cryptographic key exchange
  - DNSSec, PKI, etc: All involve trust delegation
  - Manual: expensive, sometimes worthwhile for existing business relationships

Developer Ways to Address Spoofing

- Leverage the OS
  - Use full pathnames (what does open("foo.txt") find?)
  - Make pathnames canonical
    - Resolving links including ../ or symlinks
    - Remove %20 or other encoding
  - Check permissions
  - Shared directories are usually troublesome
- Cryptographic identifiers & validation

Operational Ways to Address Spoofing

- Difficult to improve local (on-system) name resolution when the code is done
- Possible to use SSH or IPSec or other crypto tunneling to reduce spoofing issues over the network

Technologies for Addressing Spoofing

- Authenticating computers
  - IPSec, DNSSec, SSH Host keys
  - Kerberos
  - Windows Domain authentication
  - PKI with SSL/TLS
- Authenticating bits (files, messages, etc)
  - Digital signatures
  - Hashes (appropriately managed)

Technologies for Addressing Spoofing (2)

1. Something you know, like a password
2. Something you have, like an access card
3. Something you are (or are measured to be)
   - "Biometrics"
   - Fingerprints, vein patterns, photographs
4. Someone you know who can authenticate you

- The first three are traditional, #4 is new
- "Multi-factor authentication" usually means more than one from the list
  - Some people call channels a factor
  - Many of them should threat model better

Understanding Integrity

- *To interfere with (something) in order to cause damage or make unauthorized alterations*
- Can apply to data wherever it is, including:
  - Disk
  - Network
  - Memory

Tactics for Integrity

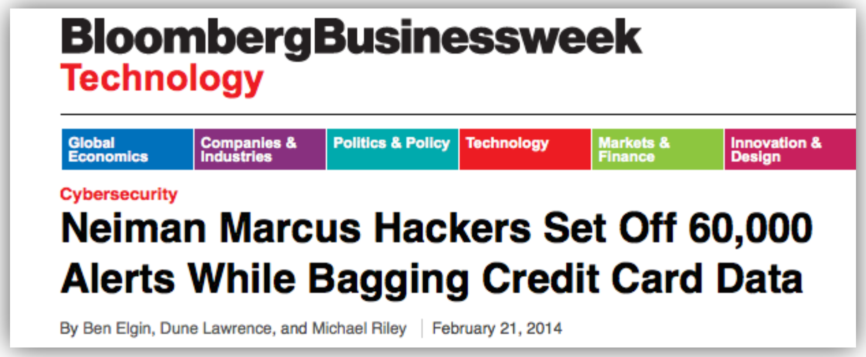
- System defenses
  - Permissions (operating system/program)
- Cryptographic defenses
  - Digital signatures
  - Hashes/MACs
- Logging and audit
  - These do not prevent, but may deter
  - Generally used as a fallback or defense in depth

Developer Ways to Address Integrity

- Use permissions as provided
- Cryptography is required over a network
- Implementing a permission system is hard
  - Lots of mistakes have been made & documented

Operational Ways to Address Integrity

- Add additional protections
  - Tripwire-like systems on local machine
  - Tunneling over network
- Tripwire: acting on alerts is key!
  - Don’t be these folks ->



Cybersecurity

Neiman Marcus Hackers Set Off 60,000 Alerts While Bagging Credit Card Data

By Ben Elgin, Dune Lawrence, and Michael Riley | February 21, 2014

- Good alert design is a pre-requisite
  - Too many alerts, people will be overwhelmed
  - Too few, they’ll miss stuff

Technologies for Addressing Integrity

- Protect files with
  - Digital signatures
  - ACLs/permissions
  - Hashes
  - Windows Mandatory Integrity Control features
  - Unix immutability
- Protect network traffic with
  - SSL
  - SSH
  - IPSec
  - Digital signatures

Understanding Non-Repudiation

- *Repudiation: To refuse to accept or be associated with; deny the truth or validity of some statement*
- Non-repudiation are the tools & technologies to establish what happened — ideally to the satisfaction of everyone involved or impacted
- Bridges business & technical levels
- Repudiation can be a feature
  - "Off The Record"

Tactics for Non-Repudiation

- Fraud prevention
  - Internal fraud such as embezzlement
  - “Customer” fraud prevention
- Logs
  - As much as you can, keep for as long as you can
- Cryptography

“Customer” Fraud Prevention

- Alice’s account is taken over & abused (or)
- Bob creates an account for fraud
- Must manage both
- Stable customers are good, predictable
- Technologies/services
  - Validation services
  - Customer history sharing
  - Multi-merchant data
  - Purchase device tracking

Developer Ways to Address

- Log business logic
  - Eg “For this transaction, we saw that geolocate(ip) was ‘Seattle,’ which is typical for this account.”
- Cryptographic digital signatures
  - Most useful today between business partners, not consumer-usable

Operational Ways to Address

- Operations get stuck investigating
  - Table-top exercises may expose issues that the logs don’t exist
- Scaling
  - Logs may end up in diverse places
  - Dedicated people
  - Specialized tooling

Technologies for Addressing Repudiation

- Logs
  - Logging
  - Log analysis tools
  - Secured log storage
- Digital signatures
- Secure time stamps
- Trusted third parties

Understanding Confidentiality

- *To ensure that information is only disclosed to authorized parties*
- Secrets in data
  - Yours: financial results, new product plans
  - Entrusted to you: private data
  - Complex rules: Who can see that Facebook post?
- Secrets also exist in metadata
  - “Layoff letter for Alice.docx”, “Janlayoff/alice.docx”
  - Calls to an STD clinic (repeatedly?!)

Tactics for Confidentiality

- On a system
  - ACLs/permissions
  - Cryptography
- Between systems
  - Cryptography
- To hide the existence of information
  - Steganography

Developer Ways to Address

- Permissions/ACLs
- Cryptography
  - Data (file on disk, email message)
  - Container (volume encryption, email connections)
  - Requires proper key management
  - Remember: Encryption doesn't provide authentication or integrity

Operational Ways to Address

- Add permissions/ACLs
- Volume encryption
  - Protects if the machine is stolen and powered down
  - Doesn’t protect against an attacker who breaks in
- Network encryption (SSH, SSL, IPSec)

Technologies for Confidentiality

- Protecting files
  - ACLs/Permissions
  - Encryption
  - Appropriate key management
- Protecting network data
  - Encryption
  - Appropriate key management
- Communication headers/act of communication
  - Mix networks
  - Onion routing
  - Steganography

Understanding Availability

- *Being able to meet a defined or implied SLA*
- Attacks can absorb any resource
  - Disk, network, CPU
- Attacks can be transient or require intervention
  - Network flooding stops when attacker does
  - Fork bomb (eg: while(1) {fork();}) might need reboot
  - Full disk might require human intervention

Tactics for Availability

- Have enough resources to serve requests
- Proof of work
  - … “Proves Not to Work”
  - Bitcoin uses high cost proofs
- Proof of communication

Developer Ways to Address

- Avoid fixed-size buffers
  - For example, 5 half-open TCP connections
- Consider
  - Resources you consume per request
  - How many requests you'll serve
  - Clever attacks that balloon resource use
  - Recovery

Operational Ways to Address

- Quotas
- Elastic cloud systems to add more resources

Technologies for Addressing DoS

- ACLs
- Filters
- Quotas (rate limits, thresholding, throttling)
- High availability design
- Extra bandwidth
- Cloud services

Understanding Authorization

- *Elevation of Privilege* is one class of authorization bypass
  - The only one covered here
  - Authorization systems are their own sub-field

Tactics for Authorization

- Limit the attack surface
  - For example, small number of setuid programs
  - Use sandboxes for network-exposed code
  - Don’t run as root/admin
  - Be aware that there’s often elevation paths for semi-privileged accounts
- Comprehensible, manageable permissions systems

Developer Ways to Address

- Limit the attack surface
- Carefully define purpose & validation rules for inbound data
- Define what you'll accept, not what you reject
- Reject bad input, don't try to sanitize
- Looped canonicalization routines
- Transform from one form to another (e.g., markdown to html)

Operational Ways to Address

- Defense in depth
- Run each target as its own unique limited user
  - Unix "nobody" account ended up quite privileged
- Sandboxes

Technologies for Addressing

- ACLs
- Groups or role membership
- Role based access controls
- Windows privileges (runas)/Unix sudo
- Chroot, apparmor, other unix sandboxes
- MOICE Windows sandbox
- Input validation for defined purposes

How to Threat Model (Summary)

- What are you building?
- What can go wrong?
- What are you going to do about it?
- Check your work on 1-3

Check Your Work

- Requirements engineering and quality assurance
- Check that you covered all the threats & assumptions
- Check that each is covered well

Testing Software You Make

- All threats you find can be tested
- In agile shops that rely on 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)
- Automation vs manual

Penetration Testing

- Aka “ethical hacking,” “red teaming”
- Improve the 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

Testing Software You Acquire

- Build a software model
  - Use the documentation and actual software
  - See if they include a threat model or security operations guide
- Look for threats
- Address the issues you find

Build a Software Model

- Components
  - Start with the binaries, databases, dependencies
  - Some will likely merge into a single process for threat modeling purposes
- Trust boundaries
  - Account(s) used
  - Sockets, RPC
  - Admin interfaces
- Look at platform changes on install
- Diagram as you find things

Look for Threats

- Use the model you've created
- This is similar to looking for threats in any other software
  - You're less familiar with it
  - It may include relevant documentation
  - (If not, what does that tell you?)
- Use STRIDE, CAPEC, attack trees, etc.

Address the Issues You Find

- Ask the creator to fix them
  - Be ready to discuss views of requirements, tradeoffs
  - Some backwards vendors will threaten you (this is a red flag they don't understand security)
- Look for an alternative
  - Easier if you TM early
- Mitigate yourself
  - Using operational security techniques from earlier classes on "what to do about it"

QA’ing the Threat Modeling Process

- Another aspect of checking your 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, not wontfix)?
  - Is there a test case?
  - Tags on bugs really helpful here

Recap

- Think like an attacker isn't repeatable
- Focusing on assets and attackers doesn't work for most people
- 4 questions
  - What are you building?
  - What can go wrong?
  - What are you going to do about it
  - Checking your work
- For more, *Threat Modeling Designing for Security*