Secure Systems Engineering and Management

A Data-driven Approach

Threat Modeling

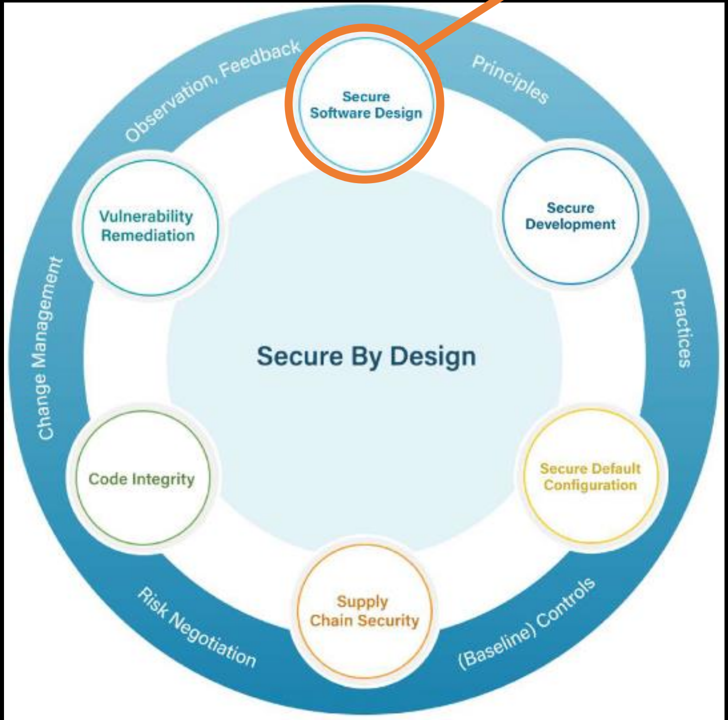
Michael Hicks

UPenn CIS 7000-003
Spring 2026

Six Considerations: Overview

Threat modeling and secure architecture design

1. Secure Software Design
2. Secure Development
3. Secure Default Configuration



4. Supply Chain Security
5. Code Integrity
6. Vulnerability Remediation

Why Threat Modeling?

- Secure design requires **systematically** identifying threats
- Not just intuition or experience – a **structured, repeatable process**
- Asking "What could go wrong?" *before* writing code

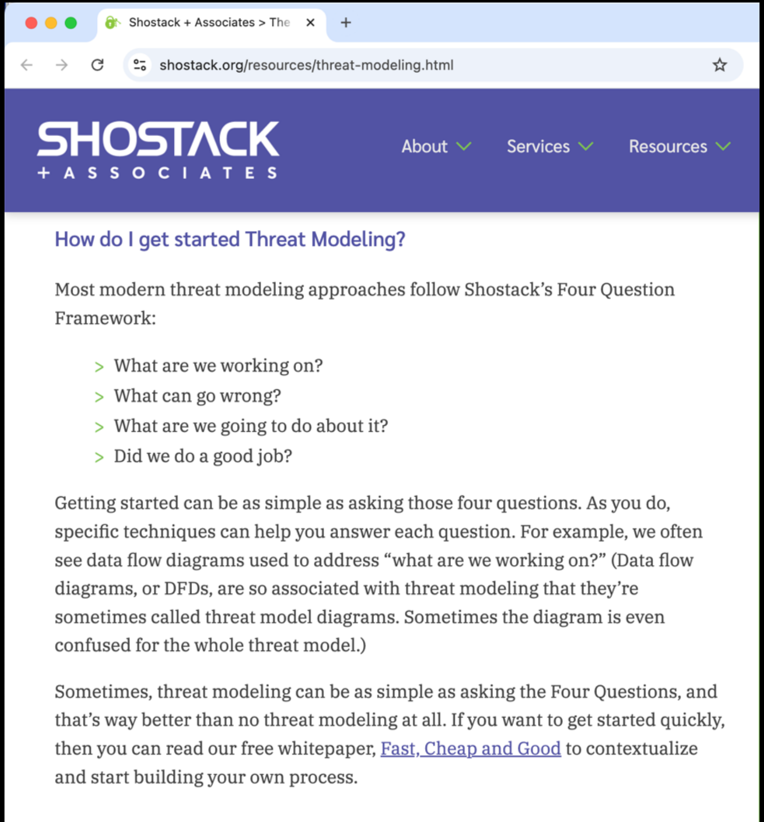
"Threat modeling is a family of structured, repeatable processes that allow you to make rational decisions to secure applications, software, and systems."

– Adam Shostack

The Four Questions of Threat Modeling

Shostack's four-question framework

1. **What are we working on?** Describe the system
2. **What can go wrong?** Identify threats
3. **What are we going to do about it?** Mitigate, accept, transfer, eliminate
4. **Did we do a good job?** Validate the analysis



How do I get started Threat Modeling?

Most modern threat modeling approaches follow Shostack's Four Question Framework:

- What are we working on?
- What can go wrong?
- What are we going to do about it?
- Did we do a good job?

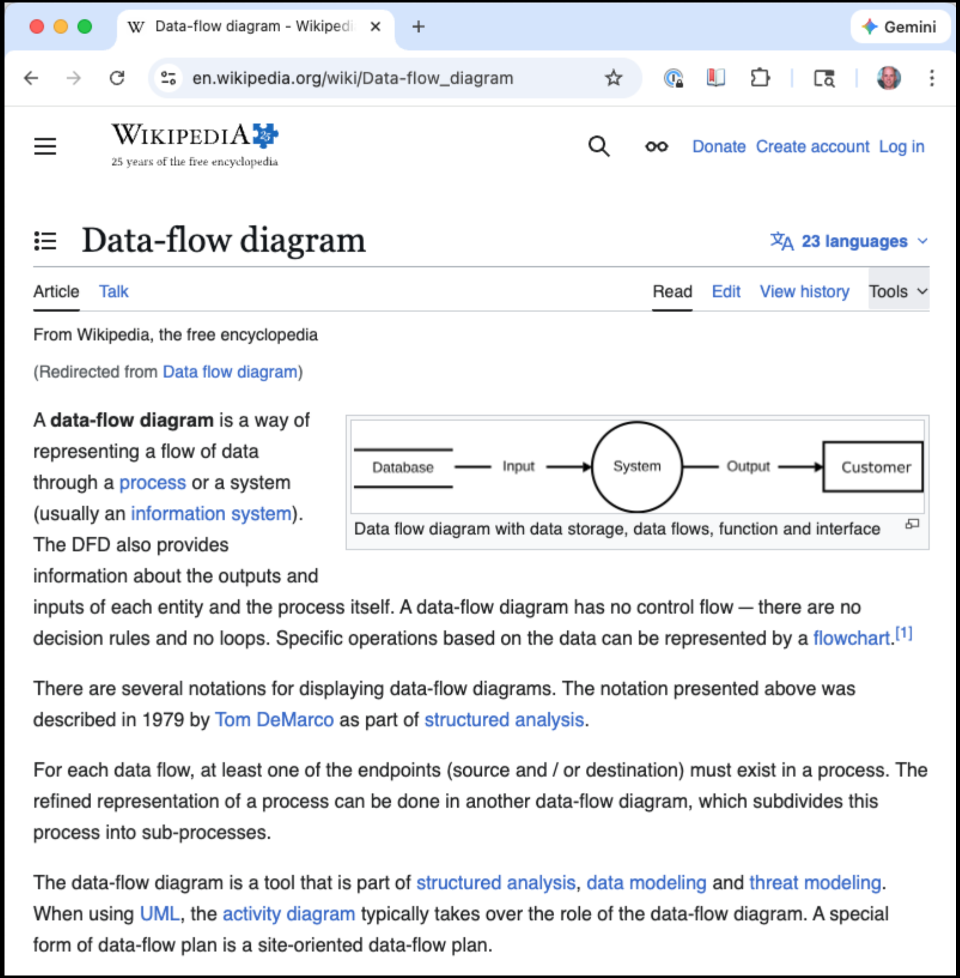
Getting started can be as simple as asking those four questions. As you do, specific techniques can help you answer each question. For example, we often see data flow diagrams used to address "what are we working on?" (Data flow diagrams, or DFDs, are so associated with threat modeling that they're sometimes called threat model diagrams. Sometimes the diagram is even confused for the whole threat model.)

Sometimes, threat modeling can be as simple as asking the Four Questions, and that's way better than no threat modeling at all. If you want to get started quickly, then you can read our free whitepaper, Fast, Cheap and Good to contextualize and start building your own process.

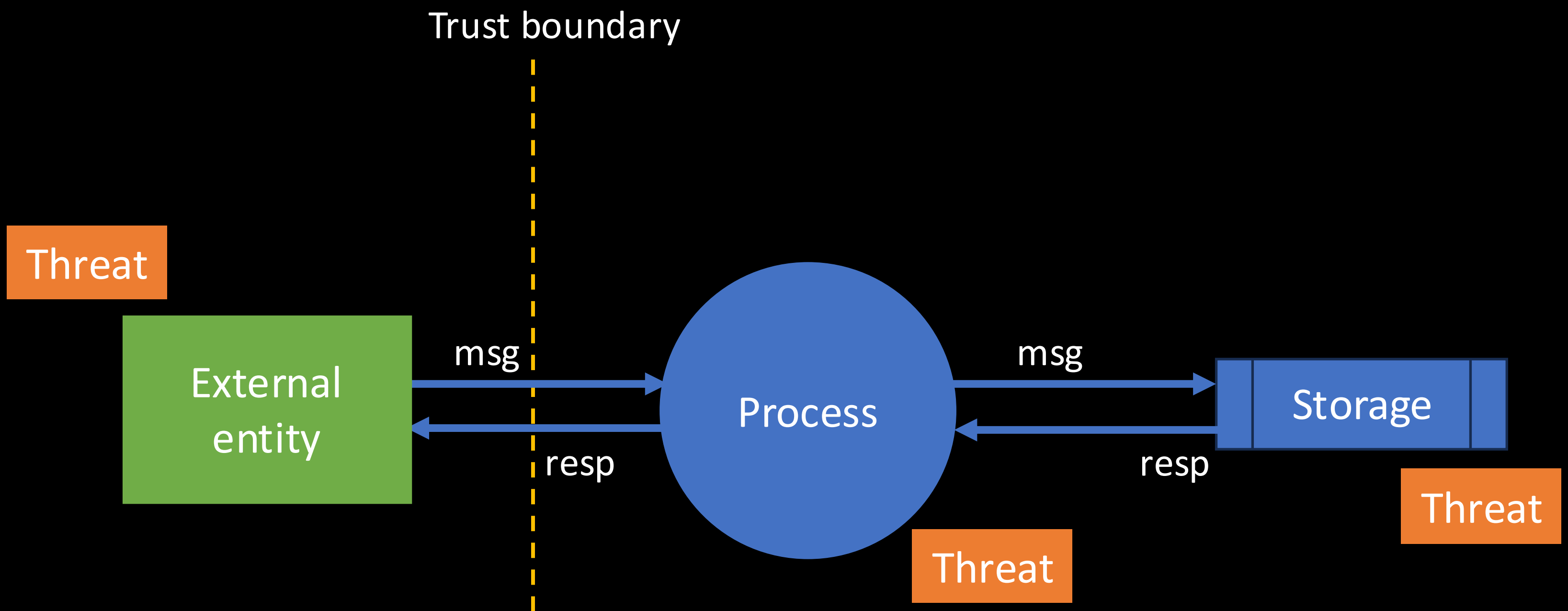
What are we working on? Data Flow Diagrams

Depicts the flow of data (indirectly: control) in a system

- For threat modeling –
  - **Level 0** diagram identifies the system and external entities that interact with it
  - **Level 1** diagram further identifies system components and flows between them

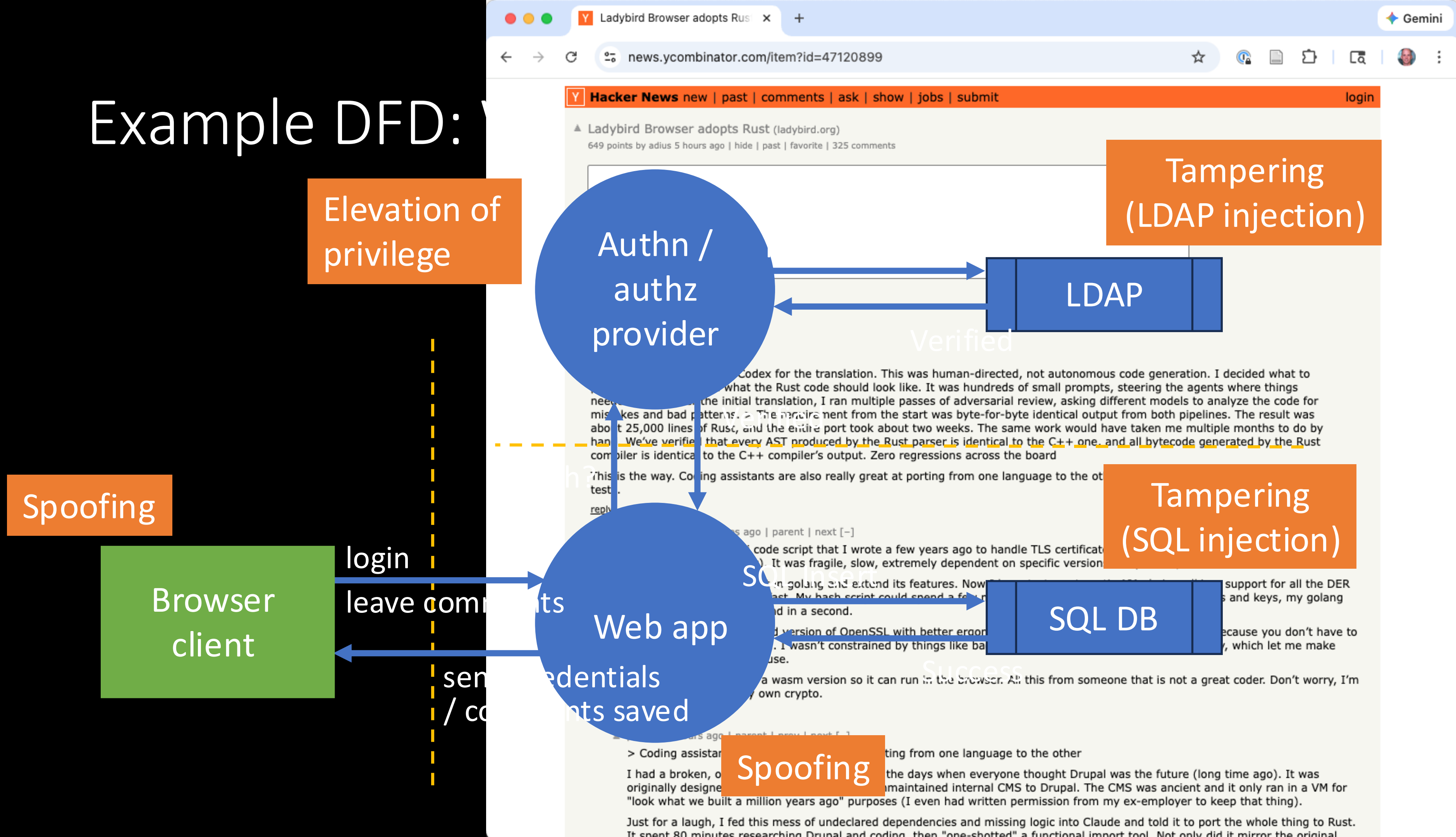


What are we working on? Data Flow Diagrams



(**threats** part of step 2)

Example DFD:



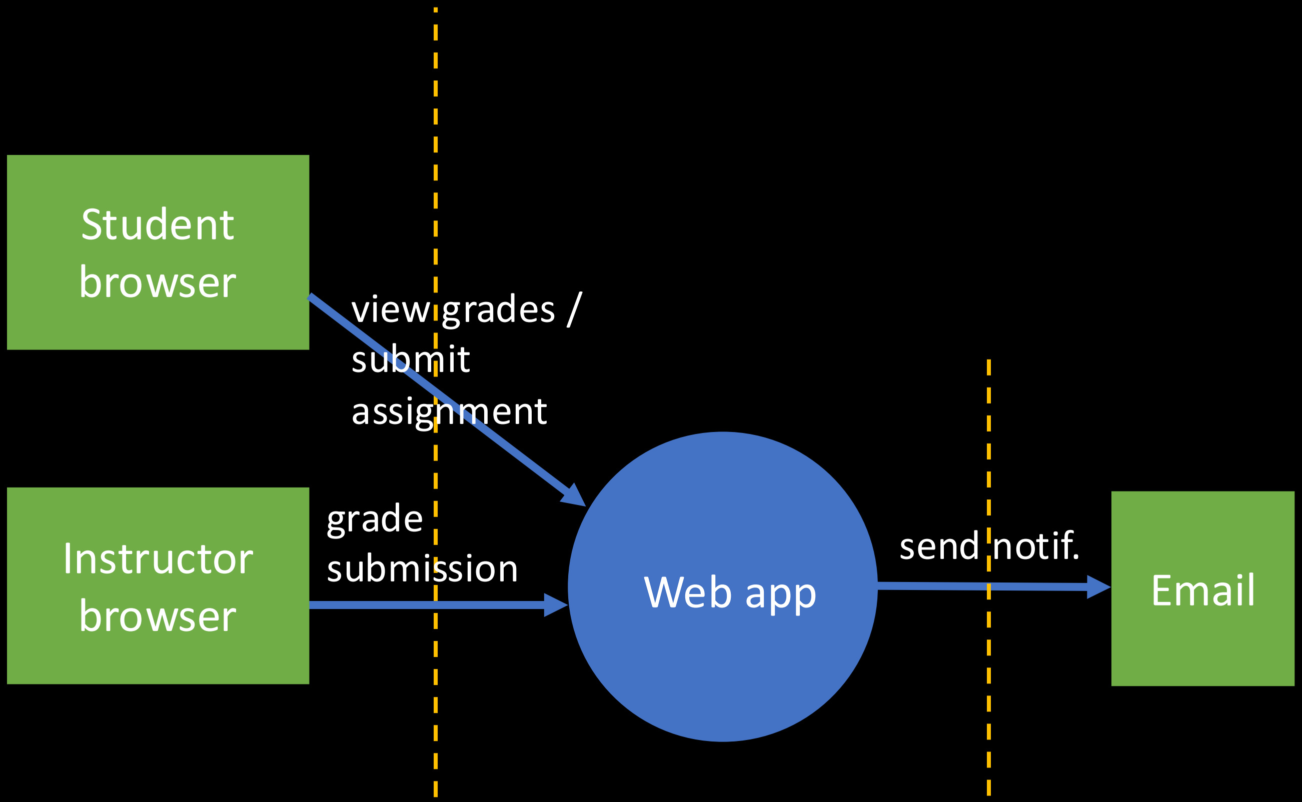
Ladybird Browser adopts Rus
news.ycombinator.com/item?id=47120899
Hacker News new | past | comments | ask | show | jobs | submit
login
Ladybird Browser adopts Rust (ladybird.org)
649 points by adius 5 hours ago | hide | past | favorite | 325 comments
Elevation of privilege
Authn / authz provider
Tampering (LDAP injection)
LDAP
Verified
Spoofing
Browser client
login
leave comments
send credentials / comments saved
Web app
Tampering (SQL injection)
SQL DB
Success
Spoofing

Recall Our Running Example: Limes

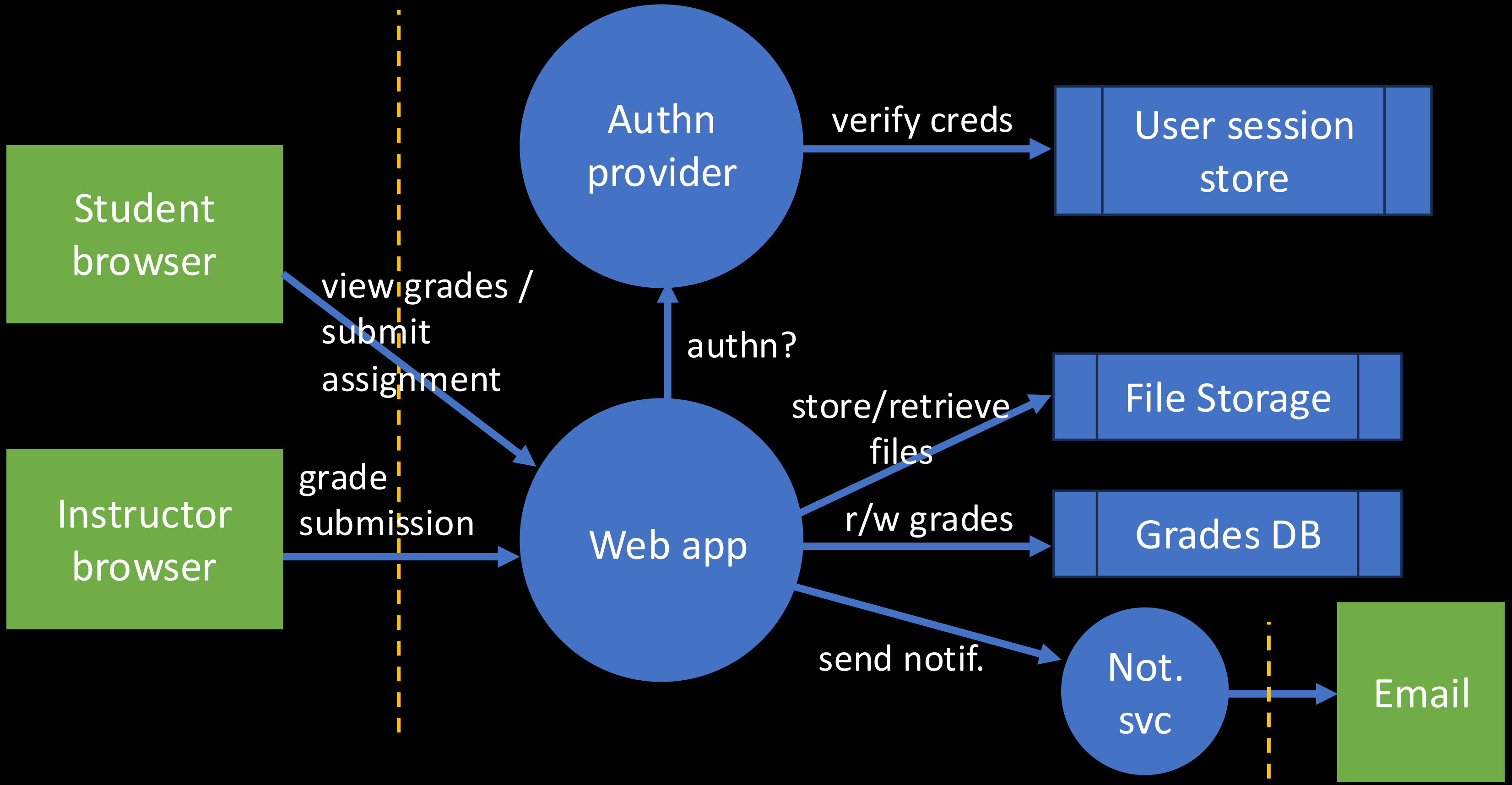
A university course management system (Canvas, Blackboard, …)

- **Users:** Students, TAs, instructors, admins
- **Data:** Grades, submissions, personal records (FERPA)
- **Features:** File upload, discussion forums, quizzes, video integration
- **Integrations:** SSO, plagiarism detection, LTI plugins, email

DFD: CMS Assignment Submission (Level 0)



DFD: CMS Assignment Submission (Level 1)



What can go wrong?

- Want to identify possible threats *systematically*
- 1999 paper: Identified the STRIDE approach to threat modeling
  - For each element or data flow in the DFD, you ask a set of questions

The threats to our products

April 1, 1999 — By Loren Kohnfelder and Praerit Garg

The growing use of computer systems to store data critical to businesses, as well as users' personal data, makes them very attractive targets for security attacks. Successful attacks can lead to loss of privacy, disclosure of sensitive data, and disruption or denial of service—losses that can cost millions of dollars. The Microsoft Security Task Force has defined a security threat model that it recommends all Microsoft product teams adopt to secure our products for our customers.

The **S.T.R.I.D.E.** security threat model should be used by all MS products to identify various types of threats the product is susceptible to during the design phase. Identifying the threats is the first step in a proactive security analysis process. Threats are identified based on the design of the product. The next steps in the process are identifying the vulnerabilities in the implementation and then taking measures to close security gaps.

S.T.R.I.D.E. stands for:

- **Spoofing of user identity**
- **Tampering with data**
- **Repudiability**
- **Information disclosure (privacy breach)**
- **Denial of Service (D.o.S.)**
- **Elevation of privilege**

Some attacks can be very sophisticated and have several steps. In such attacks, one minor break-in leads to another, and eventually substantial system damage is done. In most such cases, one of the links is the weakest, and the security of the entire system typically is no better than its weakest link. Finding and improving such weak links is how threat analysis helps improve the security of our products and services.

This article describes various threat categories in the S.T.R.I.D.E. model and provides examples of vulnerabilities that may be exploited by various kinds of attacks to make a threat a reality. This is intended to help you identify potential vulnerabilities in your product during a security analysis.

Each threat is discussed in the context of Microsoft products, which fall into the following five categories:

- **Server operating systems** Windows NT/2000 Server
- **Client operating systems** Windows NT/2000 Workstation, Win9x, WinCE, Internet Explorer
- **Client/server applications** Exchange, SQL, etc.
- **Desktop applications** Office, etc.
- **Web and media applications** WebEssentials, portal Web sites, etc.

But first, let's define some important terms that we'll use throughout this piece and that have very precise meaning in security discussions.

STRIDE: A Systematic Way to Find Threats

Threat	Violates	Ask yourself…
Spoofing	Authentication	Can someone pretend to be another user or system?

STRIDE: A Systematic Way to Find Threats

Threat	Violates	Ask yourself…
Spoofing	Authentication	Can someone pretend to be another user or system?
Tampering	Integrity	Can data be modified without detection?
Repudiation	Non-repudiation	Can someone deny performing an action?
Information Disclosure	Confidentiality	Can unauthorized parties access data?
Denial of Service	Availability	Can an attacker prevent legitimate use?
Elevation of Privilege	Authorization	Can someone gain permissions they shouldn’t have?

Special Mention: The Human in the Loop

What are we trusting users to do?

- Pick a strong, unique password (or enroll in MFA)
- Not share their credentials
- Recognize phishing / fraudulent requests
- Review transaction notifications and spot unauthorized activity
- Keep their device and app updated

How confident are we that real users will actually do these things? Part of the risk calculation

A Framework for Reasoning About the Human in the Loop

Lorrie Faith Cranor
Carnegie Mellon University
lorrie@cmu.edu

Abstract

Many secure systems rely on a "human in the loop" to perform security-critical functions. However, humans often fail in their security roles. Whenever possible, secure system designers should find ways of keeping humans out of the loop. However, there are some tasks for which feasible or cost effective alternatives to humans are not available. In these cases secure system designers should engineer their systems to support the humans in the loop and maximize their chances of performing their security-critical functions successfully. We propose a framework for reasoning about the human in the loop that provides a systematic approach to identifying potential causes for human failure. This framework can be used by system designers to identify problem areas before a system is built and proactively address deficiencies. System operators can also use this framework to analyze the root cause of security failures that have been attributed to "human error." We provide examples to illustrate the applicability of this framework to a variety of secure systems design problems, including anti-phishing warnings and password policies.

"Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. (They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.)"
– C. Kaufman, R. Perlman, and M. Speciner, 2002 [20]

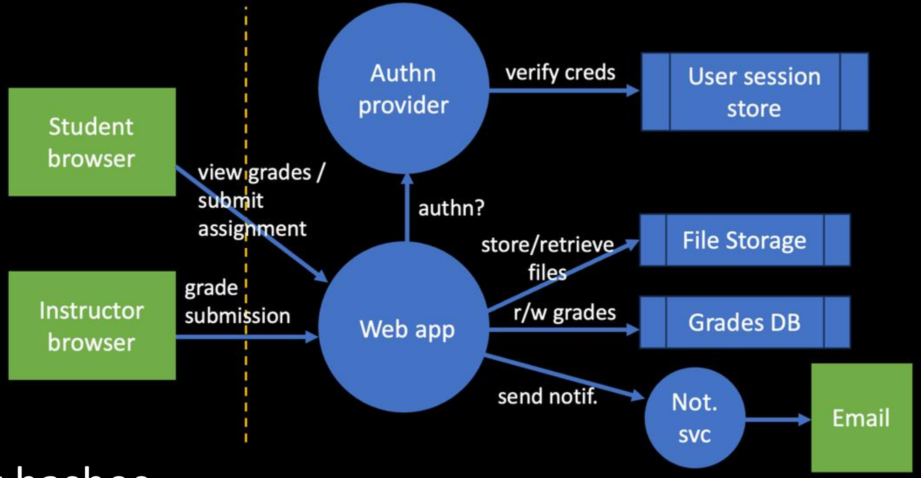
1. Introduction

It is becoming increasingly apparent that humans are a major cause of computer security failures. In the context of secure systems, humans are often thought of as "the weakest link in the chain" [31]. In 2006 the SANS Top 20 Internet Security Vulnerabilities report began listing human vulnerabilities along with software vulnerabilities [25]. A 2006 Computing Technology Industry Association survey found that security managers attribute approximately 60 percent of security breaches to human error, up from 47% the previous year [6]. Social engineering attacks and lack of compliance with organizational security policies are increasingly cited as security concerns.

With so many security failures attributed to humans, secure systems that do not rely on a "human in the loop" to perform security-critical functions are attractive. Automated components are generally more accurate and predictable than humans, and automated components don't get tired or bored [14]. Indeed, in some areas we have seen significant progress towards secure systems that "just work" without human intervention. For example, while early anti-virus programs prompted users to make a decision about every detected virus, today many anti-virus programs automatically repair or quarantine infected files in their default mode of operation. Thus, anti-virus software no longer relies on inexperienced users to make security-critical judgments. When software is likely to be able to make a better security decision than a human, removing the human from the loop may be wise. Likewise, when a user is unlikely to have relevant insights into which configuration options to choose, well-chosen default settings may result in better security-configurations than most humans would achieve on their own.

In some cases we don't seem to be able to avoid relying on humans to perform security-critical functions. There are a number of reasons why it may not be feasible or desirable to automate these functions completely [11], [14]. Some tasks rely on human knowledge that is currently difficult for a computer to reason about or process. For example, today humans tend to be better than computers at recognizing faces in crowds or noticing other humans who are acting suspiciously [30]. When tasks rely on human knowledge about context, it may be difficult to capture the entire necessary context in a way that a computer can reason about it. For example, a human may be a better judge than a computer about whether an email attachment is suspicious in a particular context. We also rely on humans to make some security-related configuration decisions and to apply policies when it is difficult to encode all of the nuances of a policy or program a computer to handle special cases.

Data flow: Student submits assignment via browser to web application



S: Submit as another student

- Stolen session cookie, weak/stolen credentials

T: Submission modified improperly

- MITM without TLS; authz bypass to modify submission post-deadline

R: “I never submitted that”

- No logging of timestamps or content hashes

I: Other students view submission

- Broken access control, IDOR (forged DB object ref) on download URLs

D: Deadline flood crashes server

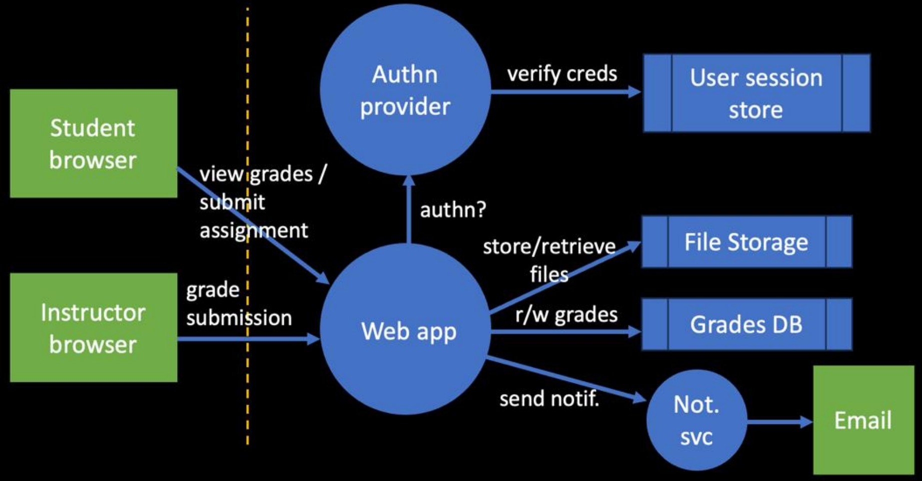
- No rate limiting; malicious file upload

E: Student gains instructor role

- Parameter tampering on role field

STRIDE Applied to CMS Example (generally)

Attack Class
Spoofing
Tampering
Repudiation
Information Disclosure
Denial of Service
Elevation of Privilege



STRIDE Applied to CMS Example (generally)

Attack Class	Intent	CMS Example
Spoofing	Impersonate a user or system	Faking a login as another student
Tampering	Unauthorized modification	Changing a grade in transit
Repudiation	Deny having performed an action	Instructor denies changing a grade
Information Disclosure	Unauthorized access to data	Viewing another student's submissions
Denial of Service	Disrupt system operation	Overwhelming the system during finals
Elevation of Privilege	Gain unauthorized capabilities	Student gaining instructor access

What Are We Going to Do About It?

Four response strategies for each identified threat:

- **Mitigate** – Implement a control that reduces the risk *(most common)*
- **Eliminate** – Remove the feature or component that creates the threat
- **Transfer** – Shift risk to another party (insurance, outsourcing, user accepts ToS)
- **Accept** – Acknowledge and *document* the decision not to act

From Threats Mitigations to Security Requirements

Structure each requirement in three parts (SAFECode):

The Problem:

Unauthorized access to another student's submitted file (CWE-639: IDOR)

The Control:

All file download requests must verify that the requesting user is authorized to access the specific file

Implementation Guidance:

Use indirect references (per-session token mapped to file path); validate enrollment and role on every request

Threat Modeling in the SDLC

The threat model's security requirements feed into:

- **Design review:** Are the controls designed correctly?
- **Test plan:** Are we testing for the threats we identified?
- **Bug bar:** Which threat-derived bugs block release?

Today's Exercise: Payment App

System scope:

- Users create accounts, link a bank account or debit card
- Send money to other users, request money
- View transaction history
- Social feed showing transactions (with privacy settings)
- Authentication via username/password + optional MFA
- External banking/payment processor API

Our job: figure out what could go wrong *before* we write the code.

STRIDE Reference Card

	Threat	Violates	Ask…
S	Spoofing	Authentication	Who are you, really?
T	Tampering	Integrity	Has the data been changed?
R	Repudiation	Non-repudiation	Can they deny it?
I	Info Disclosure	Confidentiality	Who can see this?
D	Denial of Service	Availability	Can they shut it down?
E	Elev. of Privilege	Authorization	Should they be allowed to do that?

Keep this visible as we work through the DFD.

Prioritize: Impact x Likelihood

	Low Likelihood	Medium Likelihood	High Likelihood
High Impact	Medium	High	Critical
Medium Impact	Low	Medium	High
Low Impact	Informational	Low	Medium

As a class, pick the top 5 threats from our table.

- **Impact:** How bad is it? (Financial loss, privacy breach, downtime)
- **Likelihood:** How easy to exploit? (Anyone on the internet, or requires insider access?)

Developing Mitigations

For each top threat, define:

Problem: What weakness are we addressing?

Control: What must be done? *(technology-agnostic)*

Implementation Guidance: How do we build it?

For each mitigation, also ask:
“Is the mitigation itself introducing new attack surface?”

Did We Do a Good Job?

Collect your responses into security requirements, and a new DFD

- Helps you see the whole, not just the parts

Now ask: **What did we miss?**

- Insider threats? (rogue employee with database access)
- Supply chain? (compromised push notification SDK)
- Device as threat vector? (malware on the phone, rooted device)
- Regulatory/compliance? (PCI-DSS, state money transmitter laws)

Threat modeling is iterative – you don’t get everything on the first pass, and that’s OK.

Common Failures in Threat Modeling

Mindset failures (SAFECode Section 6):

- "We already do pen testing, so we don't need threat modeling"
- "The system is already deployed, so there's no reason"
- "We did it once; we don't need to do it again"
- "Threat modeling is too complicated"
- "We don't have security experts, so we can't do it"

Process failures:

- Failing to control scope ("boiling the ocean")
- Focusing only on areas the team already knows well
- Not defining what "success" looks like

Key Takeaways

Threat modeling is:

- **A team sport** – diverse perspectives find more threats
- **Iterative** – you refine it as the system evolves
- **A living document** – update when the system changes
- **High-ROI** – cheaper to find flaws in design than in production
- **A practical skill you can develop with practice**

> "You can walk into a design meeting, draw a DFD on a whiteboard, and systematically ask 'what can go wrong?' using STRIDE."

References

Assigned reading:

- Shostack, A. “Threat Modeling.” https://shostack.org/resources/threat-modeling
- SAFECode. “Tactical Threat Modeling.” 2017.
- Kohnfelder, L. & Garg, P. “The Threats to Our Products.” Microsoft, 1999.

Further reading:

- Cranor, L.F. “A Framework for Reasoning About the Human in the Loop.” UPSEC, 2008.
- OWASP Threat Modeling Cheat Sheet
- Microsoft Threat Modeling Tool (free)