Secure Software Design

Andey Robins

Spring 23 - Week 3

The World of Threats

Outline

- Threat Modeling
- Attack Surfaces
- Risk
- Mitigation and Prevention
- Example Threat Modeling

Threat Modeling

“Understanding the potential threats to a system is the essential starting point in order to bake solid defenses and mitigations into your software designs.” - Loren Kohnfelder

An Extension of “Think like an adversary”

- Specific details and techniques are unimportant
- Focus on high level questions
  - What would they be after?
  - What would we think is important?
  - How do these things differ?
- They only need to succeed once

The Four Questions

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

What Are We Working On?

This is the primary question of software design:

- gather requirements
- identify stakeholders
- propose and plan for features

Going Even More Specific

- What components are we using?
- How do these components interact with each other?
- By what method do we ensure interaction?

Example: TCG-client

Consider that we are writing a client application for a trading card game. We are on the team tasked with developing a system for writing the system for loading players into a game, and you are specifically tasked with handling deck initialization.

Problem: players provide us with their decks and we then tell them to shuffle those decks. How do we prevent “stacked decks?”

What Are We Working On?

A deck shuffling tool which requires the deck to be shuffled

What Can Go Wrong?

- A user says they've shuffled when they haven't
- A user shuffles, but poorly
- A user doesn't even provide a valid deck

What Are We Going to Do About It?

In othe words: “how do we mitigate the threat?”

Threat mitigation is discussed in further depth later and in Chapter 3 of the textbook.

Did We Do a Good Job?

This question is a retro-spective question. It's impossible to answer up front, and often may be nearly impossible to answer definitively after the fact too. Did we do a good job or was this part of the system just never the target of an attack? The only situation where we can clearly answer this is when an attack occurs and we have information on this component's role in the attack.

What Can We Do Better?

If the answer to the previous question is “no,” this is the follow up question I propose.

1. Use the previous attack to create a proof of concept “attack”
2. Brainstorm alternative designs
3. Integrate the PoC into a regression test suite while working on fixes
4. Ensure the same attack won’t work a second time ever again

Returning to Our Example

Create some interfaces:

```
interface Deck {
    cards: Card[],
    shuffle(d: Deck): ShuffledDeck,
}

interface ShuffledDeck {
    readonly cards: ReadonlyArray<Card>,
    draw(d: ShuffledDeck): Card,
}
```

Subsequent functions can only accept a `ShuffledDeck` as an argument, and we can design the system so that the only function which returns a `ShuffledDeck` is the `shuffle` function on the `Deck` interface. This addresses our first concern of ensuring that a user has actually shuffled their deck.

Addressing Poor Shuffling

```
interface Deck {
    cards: Card[],
    shuffle(d: Deck): ShuffledDeck,
}

interface ShuffledDeck {
    readonly cards: ReadonlyArray<Card>,
    shuffle(d: ShuffledDeck): ShuffledDeck,
    draw(d: ShuffledDeck): Card,
}
```

We can now design a workflow where a user shuffles their opponent's deck and then the opponent shuffles their own deck. Similarly, they shuffle our deck before we also shuffle it. We now have redundant randomization which can serve to eliminate the concern that you could have a non-randomizing shuffle function. You're incentivized to present a true randomizer because it will ensure that both decks are truly randomized.

Addressing Valid Decks

We've already ensured it! The type based aproach to interface design makes it so that we can only shuffle a deck and a deck must be made of a list of cards.

Attack Surfaces

Attack Surfaces

Attack surfaces are an attacker's first point of entry to a system.

- Internet
- USB ports
- I2C bus
- Config file
- Physical device

Attack Surface Identification

- Identify assets
- Characterize asset access
- Identify trust and trust boundaries
- Model potential threats

Identify Assets

Our ultimate goal is to be able to protect everything, however this is impossible, and therefore it makes sense to somehow prioritize what we protect.

1. Enumerate assets
2. Identify stakeholders
3. Assign value level

Iterative Approach to Security

Kohnfelder identifies a general application of the agile methodologies from the field to security. Perform incremental passes, seeking to improve the overall system each step.

Some Example Assets and Levels

Asset	Value
Customer post DB	Mid
Frontend source code	Low
Backend source code	Mid
Financial backend	Mid
Moderation backend	Low
Subscriber DB	High

The level of value placed on an asset will often vary between stakeholders

Asset	CEO	Advertiser	DB Admin	User
Customer post DB	Mid	High	High	Mid
Frontend source code	Low	Low	Low	High
Backend source code	Low	Low	Mid	Low
Financial backend	Mid	Low	High	Low
Moderation backend	Mid	Mid	Mid	Low
Subscriber DB	High	High	High	Mid

Caution: Avoid performing specific calculations to assess the threat to an asset. Even though we can define levels, assigning them some sort of specific numeric score is misleading and may mischaracterize the threats against a system.

Characterize How We Access Assets

- When do we read to a DB?
- Who is allowed to commit code?
- How does releasing code work?
- Where are our servers?
- What does normal trafic and use look like?

Limiting Anomalous Access

- Lock DB writes outside hours
- Reduce role access to create more “sign-offs” for code
- Ensure we have forward and backward models
- Limit geographical access
- Reject anomalous traffic

Trust vs Privilege

Privilege is how much you are able to do.

Trust is how much you are expected to be able to do.

Identify Trust and Trust Boundaries

1. Identify the privilege we wish to assign to an asset
2. Identify locations where communication between different privilege levels occurs
3. Document and plan for the ways in which privilege may be upgraded
4. Define these lines as trust boundaries
5. Assert a trust level betweeen boundaries

Model Potential Threats

Start with the first two of the four questions.

- What are we doing?
- What can go wrong?

Building on the previous section we covered, this should be something we're familiar with. If we need a higher degree of specificity because we're returning to an asset, this process can continue until we have a degree of granularity befitting the asset under question.

Attack Surfaces

As a rule, attack surfaces should be minimized. Performing this type of analysis is how we identify susceptible levels of an attack surface. The next step is to assess the risk and mitigate that risk as appropriate.

Risk

Mitigation and Prevention

Example Threat Modeling of Software

1. Identify attack surfaces
2. Model threats on the system
3. Assess risk posed by attacks
4. Mitigate and prevent where possible