STRIDE and security threat modelling

Arran Stewart

What we'll cover

- What is threat modelling?
- What is STRIDE? Why use it?
- How can we do threat modelling with STRIDE?

What we won't cover

- *How* to implement mitigations of threats
  - (e.g.: cryptography; privilege separation; access control; authentication frameworks; secure coding standards)
- Those are an entire subject to themsevles.

Risk management

Running a software project is about managing risks.

No system is perfect, and no project ever goes completely according to plan.

But we can try to ensure that we bring the *risk* of serious problems down to a tolerable level.

Risk management

Risk management is basically just asking the question:

- What can go wrong?

So that we can do something about it, *before* things go wrong.

Threat modelling is one form of risk management: it focuses on risks arising from information security threats.

What is threat modelling?

- It's always conducted in the *context* of wanting to protect something of value
- It's a process whereby potential threats (e.g. security vulnerabilities) can be identified, enumerated, and prioritized
- It includes the process of then understanding and communicating those threats, their mitigations, and reviewing the effectiveness of those mitigations

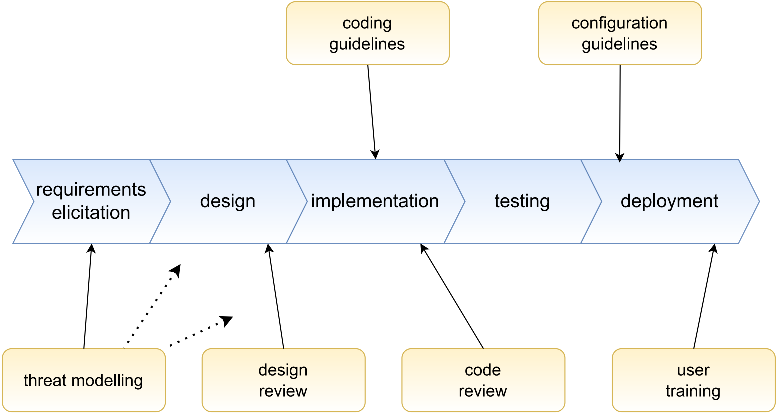
An aside: incorporating security

Approaching security as something you can simply “add on” to existing systems or processes as an extra phase or step is doomed to failure.

The aim should be to *incorporate* security into existing processes, at all stages of the software development life cycle:

- analysis/requirements elicitation
- design
- implementation and testing
- deployment/operation
- disposal

An aside: incorporating security



Incorporating security

- Requirements stage: identify security goals
- Design stage: threat modelling
- Maintenance/operation: how are reported vulnerabilities handled?
- Disposal: when the product is no longer needed – what happens to any sensitive data?

Threat modelling with STRIDE

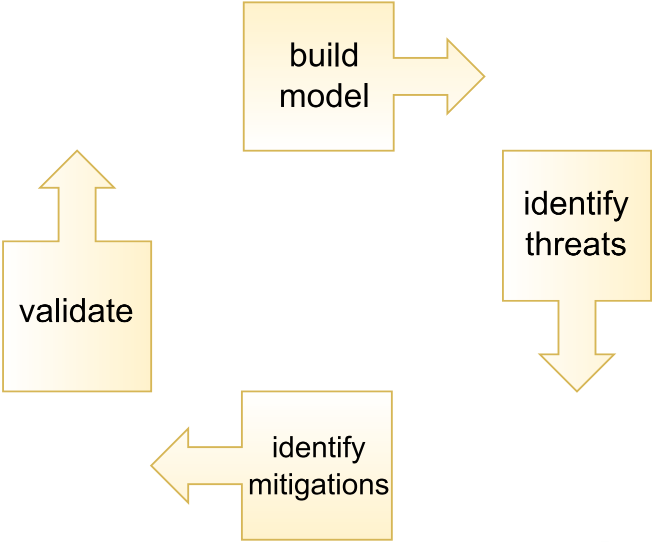
Four questions

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

Four questions – activites

1. What are we building?
   - Outcome: a *model* or diagram of the system (and identified assets)
2. What can go wrong?
   - Outcome: prioritized list of threats
3. What are we going to do about it?
   - Outcome: prioritized list of mitigations or countermeasures
4. Did we do a good job?
   - Outcome: validation of prior steps; tests; gaps identified; improvements to process

Iterable


build model
identify threats
identify mitigations
validate

Scope

“Small and often” is better than “comprehensive, but never finished”.

- *Full* inventory of all potential threats for a large, complex system could be huge
- But it’s better to do *something than nothing*, and it’s better to identify the *most critical* threats than to aim for completeness

First pass = focus on biggest, most likely threats, to high-value assets

- Other assets and threats can be dealt with later; scope can be increased

A precursor question

- Do we have security requirements?
  - These can be user stories, or more traditional requirements documents

The STRIDE taxonomy

STRIDE is technically just a taxonomy (plus mnemonic) for threats, developed by Praerit Garg and Loren Kohnfelder at Microsoft.

But it also gets used as the name of the 4-phase model for threat modelling.

The name is a mnemonic for categories of threats:

- **Spoofing:** attacker pretends to be someone else
- **Tampering:** attacker alters data or settings
- **Repudiation:** user can deny performing an action/making attack
- **Information disclosure:** leakage of confidential info
- **Denial of service:** preventing proper site operation
- **Elevation of privilege:** user gains power of another, more privileged user

Threat modeling

STRIDE is intended to be *developer*-friendly

- doesn’t assume we know about the end-user’s risk appetite
- doesn’t emphasise detailed risk/impact assessment (developers may not be able to do so)

What are we building?

The aim is to produce a *model* or high-level description of the system, including assets (valuable data and resources) that need protection.

- Traditionally:
    - data flow diagram (DFD), or
    - Unified Modelling Language (UML)

But *any* sort of model will do.

Could be a design document or a box-and-arrows whiteboard sketch.

Level of detail

No model is perfect – it is a useful *simplification* of reality.

- Needs enough granularity that we can analyse it, identify assets and threats
- Always possible to iterate the process later with more detailed models if necessary
- Too little detail ⇒ details will be missed
  Too much detail ⇒ the work will take too long

Identify assets

These are things we want to protect.

Usually data.

But could also be:

- hardware
- information technology resources (like bandwidth, computational capacity)
- physical resources (electricity)

Can you think of threats targeting these?

Identify assets

Assets should be *prioritized* – which are most important?

- We could try to hide *everything* about our server, for example
  - But is the best use of our time?
- Compare server details with (e.g.) financial data, password hashes, cryptographic keys

Identify assets

Don't try and put a dollar value on assets. Avoid superfluous and unrealistic granularity.

- One idea: categorize with "T-shirt sizes"
- "Large" (major assets), "Medium" (valuable assets, but less critical), "Small" (minor consequence).
  - . . . maybe your project needs "Extra-large" (super-critical)?

Remember other parties/stakeholders' viewpoints – something *you* think is of "minor consequence" could be much more important to (e.g.) a customer, CEO, finance, etc.

What can go wrong? – Identify threats

Methodically go through the model, component by component, flow by flow, looking for possible threats.

Identify

- *attack surfaces* (places an attack could originate)
  - Points where an attacker could interpose themselves
- *trust boundaries* (the borders between more-trusted and less-trusted parts of the system)
  - These will intersect data flows
- threats in each of the possible STRIDE categories.

Tip: threats often lurk at trust boundaries.

Identify attack surfaces

These are an attacker's "points of entry", or opportunities for attack. (For example: communication over a network.)

- When we look at mitigations – try to completely remove, or at least reduce, opportunities for attack

Identify attack surfaces

Physical example: we have a building we want to secure.

- What's better – many exits and entries?
- Or: just a single exit and entry, which we can monitor carefully, and have (e.g.) security screening, metal detectors at.

Identify threats

For each of the STRIDE categories – e.g. tampering – we ask, What advantages could an attacker gain if they did/subverted X?

A suggested approach: brainstorm first – come up with ideas quickly, without critiquing or judging them yet

Analyse, understand and prioritize threats

For each identified threat:

- flesh out the details
- try to assess the chance of them happening
- assess what the impacts would be

Analyse, understand and prioritize threats

- For probability and impact – no need for exact numbers – just use a point/level system (e.g. 1 to 3, 1 to 5)
    - Give your levels labels – “likely”, “unlikely”; “minor impact”, “showstopping / enterprise-destroying”
- Be cautious of unrealistic levels of granularity –
can you *really* distinguish “5%” versus “7.5%” probability, or “3/10” from “4/10”?

Ranking threats

Microsoft “DREAD” model:

- Damage: How great would the damage be if the attack succeeded?
- Reproducibility: How easy is it to reproduce an attack?
- Exploitability: How much time, effort, and expertise is needed to exploit the threat?
- Affected users: If a threat were exploited, what percentage of users would be affected?
- Discoverability: How easy is it for an attacker to discover this threat?

What are we going to do about it? – mitigations

- Propose ways of dealing with each threat – usually called “mitigation” or “countermeasures”.
- But in full: either mitigate, remove, transfer, or accept.

Mitigations and other approaches

- *Mitigate* risk by redesigning or adding defenses.
  - The aim is either to reduce the chance of the risk occurring, or lower degree of harm to an acceptable level
- *Remove* a threatened asset if it is not actually needed
  - Or if removal is not possible – seek to reduce the exposure of the asset, or limit optional features of your system that increase the threat.

Mitigations and other approaches

- *Transfer* the risk – offload responsibility to a third party.
    - Example: Insurance is a common type of risk transfer
    - Example: Outsource responsibility for e.g. processing payments, or processing sensitive data, to an enterprise with expertise in the area.
- *Accept* the risk (once it's well understood) as being reasonable to incur.

Mitigations – questions to ask

- Can we make the attack less likely to work?
- Can we make the harm less severe – perhaps only some of the data is accessible?
- Can we make it possible to undo the harm – e.g. backups?
- Can we make it more obvious when harm has occurred – e.g. by ensuring we have comprehensive logging and monitoring?

Did we do a good job? – validation, review and testing

- Validate previous steps, act upon them, look for gaps missed
- Test the efficacy of mitigations, from most to least critical

Validation

- For a model – does it match what has actually been implemented?
- For threats – have we describe them properly? missed any?
  - do they: describe the attack, the context, the impact?
- Other stakeholders – have testing/quality assurance staff reviewed the model?
- Mitigations – is each threat mitigated (or otherwise dealt with)
- Are the mitigations done correctly? Have they been tested?