Session 4: Architectural Risk Analysis (Threat Modelling)

Threat Modelling / Architectural Risk Analysis Styles

- What’s about the vocabulary?
  - “THREAT” and “RISK” are often used interchangeably; “threat modelling” and “(security) risk analysis” often mean the same, or similar, thing
  - Some authors define threats as actors (e.g., threat == threat agent), so check the meaning when reading a text
  - Some authors confuse threats, weaknesses and vulnerabilities
    - You can have a risk, or a threat of something, even though you do not necessarily know you have a weakness (and a resulting vulnerability)
    - “Someone will inject into our database” can be a valid risk even though we don’t know whether there is an injection problem

- We look primarily at technical analysis. There can be many other levels, and some will refer to quite high level business environment threat analysis as security risk analysis
  - For example, whether there will be a new paradigm of device use (such as “Bring Your Own Device” was) is a valid discussion … but not necessarily for the immediate technical analysis of a system.

- Here, during the next two lectures, our security risk analysis will assume:
  - We have a specific technical system or solution that we want to analyse
  - We actually have a clue about its architecture and design (or we’re going to design it very soon)

- Even so, there are two levels of risk analysis:
  - Business level, that is mainly about risks to the value creation
  - Technical (or architectural / design) level

A short overview of business level security risk analysis

- It greatly helps to frame the technical / design level security risk analysis if the actual business value creation logic is clear:
  - Some things that may be technically security risks might not be business risks at all - hence mitigation may not be necessary.
  - Some things that an attacker could aim at may not be evident from the architecture alone.
- Business level security analysis should be done with the customer representative. In many organisations, a direct customer contact is not available. In this case, a “Product Owner” or “Product Manager” is typically one person you want to talk to.

- A business level security risk analysis should:
  - Identify information flows and assets, as an input to the technical analysis, later. There may be some information flows that the business folks know about and that aren’t visible in your component but have an overall security effects, such as usage analytics data. It is also very useful to pinpoint personal data at this point.
  - Identify main use cases (user stories) and the misuse cases - what are the worries that business has. You could approach this by identifying the role of this system in the business value chain. What sort of issues in this system would affect the value stream (or actual revenue stream)?
  - Identify the users (including administrators) of the system, and extend this to all the other people who sell, service, or support the product. What sort of view do they have to the system? What kind of information do they receive? What sort of management interfaces do they have towards the system?
  - Identify whether the business has some specific regulations, customer requirements, or certifications that need to be met, and whether those have any software security impact.

- As it can be seen, a business level security risk analysis does involve business people. How exactly to structure this part depends; I would probably have a loose structure and have a face-to-face discussion with the business owner.

- The main takeaways that will help you later, are:
  - What is the system used for (business wise)
  - Information asset / flow list
  - List of persons / roles interacting with the system
  - What is the “top list” of things that should not happen (i.e., misuse cases)

**Visualising With Threat Trees**

- Many sources propose visualising threats with “threat trees”.
- However, most examples of those threat trees in the literature have significant flaws:
  - They are too simple and clean, tree-like (systems normally aren’t but issues have multiple causes and effects)
  - An unstructured discussion may not guarantee any specific coverage for analysis
  - They do not readily support one of the most critical processes of risk analysis, which is resolving any ambiguities and underlying assumptions in actual design

- Threat trees can be useful for noting down the business-level threats from the business-level analysis, but my suggestion is not to try to use them for technical analysis
- Real threat trees would look more like threat graphs. The most interesting issues would be probably at the graph cliques.

**DEMO:** An actual threat graph from a security consultancy
Technical Analysis: Discovering the needs

- Moving on to the technical analysis, you may
- What sort of information you need to know about your system
  - Data processing
    - As we’ve seen from earlier sessions, processing data can go terribly wrong. And if processing breaks, the attacker may gain control of the processing entity (e.g., a process). It is useful to know exactly where in the architecture these potentially attacker-controllable blocks reside.
    - An attacker observing the state of data processing may break security assumptions. As an example, a process could have data unencrypted in memory. Knowing where processing takes place allows you to design protection.
  - Data in motion, data at rest
    - Data requires security services. Depending on the data, it might need, for example, confidentiality and integrity. How these can be offered depends on where the data is stored or being transferred.
    - Data “at rest” is actually in motion - but in time. It is travelling towards the future.
    - Of particular interest are cryptographic keys. These are data, but very specific and interesting type of data.
    - Configuration files are also data.
  - Interfaces
    - An attacker usually needs to interact with the system. This means that there is a data flow to/from the attacker. The place where the data flow crosses a system boundary is an interface.
    - Interfaces are natural places to conduct robustness or other types of testing.
    - The total of all interfaces is the attack surface.
- The process of “ambiguity analysis”
  - In most real-life situations, exactly what a system does may not be entirely clear.
  - For real-life software development, software developers necessarily speak in, and use, abstractions. You don’t really go through the whole HTTP stack when you define a new REST API.
  - Abstractions are necessary for effective work, but they can also mask wrong assumptions. Have you ever seen an architectural diagram that has a black box? Do you know what the black box really does? Does everyone around the table agree?
  - In security analysis, opening these black boxes and broken assumptions is very important.
  - In all its simplicity, it is useful to ask “how” and “what exactly” many times, until you arrive on a level where security analysis is meaningful.
  - What’s a suitable level? When have we resolved ambiguity well enough?
DISCUSS: Apart from the fact that Microsoft PowerPoint clearly is the wrong tool to do architectural risk analysis diagrams (this is why we do it on a whiteboard instead), even a simple assumption of a “web server” can turn up to be a can of worms. The picture above is pretty bad; it doesn't, for example, show data flows properly and only does a first stab at opening up the complexity of the processing blocks of the system. What is missing? How would we drill down to it more?

- The end result should be able to explain system in sufficient detail, meaning:
  - All data processing blocks have been divided down to process level.
  - Libraries that are statically linked have been identified.
  - Dynamic libraries and plug-ins have been listed and identified.
- We know what frameworks, virtual machines and other underlying technologies are being used, and in which languages stuff has been implemented in.
- Location of data stores is known - for example, where in a file system, or in which database, under which database user.
- Every data flow has the whole protocol stack under it clarified. Is it HTTP? Does it have TLS?
- Every protocol step is clear. In the web world, this is usually HTTP request/response, but there may be more complex ones.
- We know exactly what data is being stored or processed. What’s the content, format and encoding? What data came from outside the system (could be attacker controlled)?
- We know where configurations, keys and certificates are stored.
- We know where personal data (i.e., privately identifiable data) or otherwise sensitive data (e.g., credit card numbers) are stored.
  - You do not need to do all this up front. There can be other strategies. For example, you could do analysis one use case (“user story”) at a time.

A picture is a thousand words

- Architectural risk analysis is really, really helped by a picture.
  - Humans usually remember 5 +/- 2 concepts at a time. Most systems have more parts than this. A picture helps.
  - Resolving ambiguities is very effective if picture is drawn while others comment.
  - I don’t personally like pre-drawn pictures that a lone architect has produced; these contain assumptions and ambiguities that the one person has.
  - I’d rather draw the picture on a whiteboard and then (if an architectural picture is needed for something else) copy it off there than vice versa.
  - Do not worry about notation that much. Especially, forget trying to do clean UML.
  - A whiteboard works better than a drawing tool. This is why all the examples I’ve drawn below are drawn with an actual pen on actual paper. I also recommend you to do the same, because you will use a lot of time cursing at your diagramming software that you would better use doing analysis.

- The most useful things to draw
  - Data Flow Diagram
    - Shows processing blocks and data flows between them.
○ Message Sequence Chart
  ■ Shows message traffic between communicating entities
  ■ Useful especially for more complex protocols (complex = more than two parties, or more than two messages)
  ■ One way to work is to draw a DFD and for the most interesting data flows, draw an MSC

● Usually NOT VERY useful things to draw
  ○ Class diagrams, how source code is organised, etc.
    ■ Analysis is against a running instance of a system, not how it theoretically looks like as an abstraction.
    ■ Sometimes it is useful to know about classes - for example, when objects are serialised into data. But generally speaking, DFDs and MSCs are what is much more useful.
  ○ UML Use Case Diagrams
    ■ Mostly information that can be much better explained using plain text
    ■ Kids, just don’t do it

● “Neighbours”
  ○ You need to stop somewhere, unless you want to model the whole Internet
You can stop at the first processing block that you do not control (develop), and leave that as a black box. For example, if you have a pure server side web app, the browser is a black box. (If you have client-side JavaScript, it is not; then you need to model it.)

The data flow to/from that block still needs to be fleshed out in detail

All data flows in your DFD must have endpoints, they cannot end in a vacuum

- **Actors**
  - Sometimes, a data flow stops at a human being. These should be drawn. Your attacker is human.
  - Humans to be drawn include
    - Users
    - Admins (are humans too)
  - Any human could be an attacker, so a source of attacker-controlled data

- **Things people usually forget to draw**
  - Admins and admin interfaces
  - Key storage (including SSH keys, etc.)
  - Configuration files
  - Load balancers (even if they would terminate TLS)
  - Client-side code execution in web apps

**Understanding security boundaries**

- “Boxes” in your DFD should be security boundaries.
- A “security boundary” is some sort of barrier between two processing blocks that is enforced upon these blocks.
  - A process is within a security boundary, because (modern) operating systems ensure that processes cannot read or write each other’s memory. A memory protection boundary becomes a security boundary.
    - However, threads within a process do not have their own security boundaries.
  - A virtual machine is a security boundary, because another VM cannot magically jump into another VM.
  - Security boundaries form concentric layers. A process can be inside a VM, which is inside a physical box...

- **Beware of fake boundaries**
  - Just deciding to store stuff in different directories is not a boundary, unless directory access controls are enforced from outside
  - Processes owned by the same user are really in the same domain (because a user can control all processes of that user)

- **What an attacker control of a security boundary means**
  - An attacker that can execute code within a security boundary is usually thought of being able to control everything that happens within the boundary, and have access to all data within the boundary
  - Although this could sometimes be tricky, the safe assumption is to assume this
○ Example: If an attacker can execute code within a kernel, the attacker is thought to completely control the operating system - including data flows through it
○ Example: If an attacker can execute code on a physical host, the attacker is thought to completely control everything in all VMs running on that host

● Security boundaries as an optimisation for analysis
  ○ In some cases, it is useful to treat a security boundary as a blob. If you define everything within it is untrusted, you might not need to analyse its internals at all.
  ○ You could approach a security boundary as an abstract object, and for example, sandbox (isolate) a security domain with a lot of untrusted activity. Bad things can happen inside the box, but hopefully they can’t escape.
  ○ Many beginners make the mistake of trying to fix problems within an untrusted environment between components that are in the same security domain. It often happens that this results in a series of minimal security enhancements that make exploitation a bit harder, but still possible. Unfortunately, this sort of work may end up generating significant costs with very little benefit.
  ○ Example: (e.g., OS compromised => all apps within it compromised; apps cannot protect from threats in their “enclosing” security domain no matter what sort of obfuscation etc. they would do)

An example of embedded security domains (explanation during lecture)

● Typical boundaries
  ○ Machine boundary
    ■ A physical machine boundary, or a virtual machine boundary, is usually a rather strong isolation. Modern operating systems (if kept up-to-date and properly hardened) are usually pretty good in isolating stuff that happens outside them.
    ■ The problem with a machine boundary is, of course, that an attacker can attack anything that runs within the machine. All applications and services enlarge the attack surface, and once the machine is “owned” (compromised), everything within that boundary is compromised.
  ○ Containers (e.g., created using Mandatory Access Control)
    ■ There are many types of containers that are usually run on top of an operating system. The Java VM is one example. Mandatory Access Control (MAC) frameworks such as AppArmor are other examples.
- Containers can be isolated execution environments, or they can just mediate accesses to resources.
- Example: Java VM and browser’s JavaScript engines are execution environments.
- Example: AppArmor is an example of a system where the operating system checks whether a process has a right to do specific things; it’s not a separate execution environment but how the process interacts with other parts of the OS are checked and enforced.
  - Processes
    - The process is the most common unit of isolation - modern operating systems do not let processes to alter the execution of other processes (that they do not somehow own).
    - However, most operating systems are pretty porous with regard to processes’ rights. A process has many channels through which it could have an impact on the other processes, or the container it is running in.

Taint analysis
- On “tainting” inputs, please refer to Session 2. We are now extending the concept of tainting from code to communication protocols.
- Refresh: Protocol stacks
  - A PROTOCOL STACK is a key concept of protocol engineering. Protocols are stacked on top of each other, and each layer is responsible for some aspect of the protocol.
  - An example being a typical web page load, where IP is responsible for getting its payload routed through the Internet; TCP creates a “stream” of octets by combining several IP packets; TLS on top of TCP provides confidentiality and integrity protection services; HTTP on top of TLS provides the web page request and response, including the web page itself; and the web application may then have an application level protocol that uses HTTP to exchange, for example, JSON objects.
  - There are protocol layers below IP, too. However, IP is the one layer that is end-to-end.
- Protocol layer termination and passthrough
  - When doing architectural risk analysis, it is necessary to understand where each protocol layer TERMINATES.
  - This termination point is part of your ATTACK SURFACE. You could think of it as a point of code that gets exposed to external inputs.
  - For example, if you have a HTTP proxy in your corporate firewall that does content filtering, seen from your desktop or the web server, HTTP probably terminates in that proxy. The application layer on top of the HTTP might not terminate there; it could be passed through as is.
  - Another example, if you have a load balancer in front of your cloud service that handles TLS, and passes the HTTP request in plaintext to your cloud instances, the TLS layer terminates in the load balancer, and HTTP is passed through.
The concept of termination is important because every location where a protocol is being parsed or acted on is a part of the attack surface. You could, for example, think of an illegal input that would cause processing to fail.

If data is passed through without acting on it or parsing it, then that passthrough location is not in the attack surface. For example, for most IP routers, anything on top of IP is just in a kind of opaque tube. The router just copies bytes from interface to another interface, so any malicious activity in that data flow does not have an effect.

Care should be taken not to confuse “real” pass-through (where data is really just being passed on) with protocol termination. If a proxy actually tries to look into traffic, parse it, and perhaps filter it, that actually terminates the layer, and thus creates a point in attack surface.

This is a very important aspect of the “ambiguity analysis”. If you have black boxes that seem to just be passing through traffic on a certain layer (i.e., do not seem to terminate a protocol), you really have to peek inside each black box to determine whether it really is a pass-through or does it actually actively do something to the data that is flowing through.

If it does something actively, treat it as a termination point (and as a new traffic source point).

A side note and a reminder from Session 1 & 2: Data as code

One layer’s data could be one layer’s code. Typically you would see JSON objects being passed back and forth in a web application. Those can be construed as data objects, but they could be evaluated in JavaScript context, becoming code.

Similarly vulnerabilities in a parser may cause parts of data being interpreted as code.

Especially when we approach the application layers, one layer’s data can be other layer’s code.

Taint analysis and security boundaries
In which boundary does tainted data end up in

- The taint analysis phase of architectural risk analysis follows all data flows, and all layers of a data flow, and identifies which component processes the data on that layer from that data flow.
- This is analogous to taint analysis in code review (static code analysis). A malicious input is assumed, and the data flow taint analysis tells you where (in which part of the attack surface) this malicious input will end up in.
- Assuming the malicious input causes a compromise (attacker control) of the component, then everything within that security boundary needs to be assumed to have been compromised. Although this may not be strictly true in every case - it may be technically difficult or impossible to pull it off - security-wise, this is the safe assumption.

Identifying robustness test needs based on taint analysis

- Taint analysis - understanding which parts of the attack surface bear the brunt of bad data - generates a list of places that need to be robustness tested.
- Typically, you could just make a list of protocol or data format parsers and consumers, and plan to do fuzz testing on them.
- Alternatively, you might flag these components (or parts of the components) for code review, static analysis, or other types of increased security quality control.

Prioritising findings

- It does matter where the data came from.
- If you can trust the sender of the data, and can authenticate the sender, you might be able to decide that this potential way to inject malicious data is not a risk. (If you actually authenticate only trusted senders, that is.)
- Technically, everything that is possible may not be a security risk.
- EXAMPLE: Let’s assume you have a Bluetooth-equipped system and you have identified two attack surfaces:
  - An endpoint that receives data objects over Bluetooth from anyone
  - An endpoint that receives audio data from a paired headset
  then it is most likely that the former one is a bigger risk, and the latter one isn’t, because in that case, only a paired (“trusted”) party can talk to you. You will then want to prioritise the former. Of course, the attack surface that is actually the Bluetooth pairing protocol is another question - if that can be subverted, then the second one becomes higher priority again.

Security services for data flows & stores; STRIDE

- Once we have discovered the data flows and data stores, we will look into security services we need to provide for each.
Not every data flow or data store requires all types of security services. Public information might not need confidentiality service, whereas it could benefit from integrity protection (so the recipient knows it is authentic information).

The traditional model for security services is "CIA triad", Confidentiality, Integrity and Availability. You could use that as a basis too, in which case you would discuss the CIA needs for each data flow and data store.

Microsoft, as a part of their Security Development Lifecycle, have come up with two other acronyms, STRIDE and DREAD. The latter has already been disowned by Microsoft themselves so we will also not discuss it.

STRIDE, however, seems to work rather well as a discussion facilitator for data flow analysis. One way to do STRIDE analysis is as follows:

○ Take one data flow, or data store, at a time. In the previous phases, you have already determined the specific protocols and data content, so you should be able to have a detailed technical discussion about it.

○ Consider each of the parts of STRIDE by asking a series of questions. If the people doing the analysis have a lot of experience, you may not need examples; however, when doing this for the first time, it is helpful to have a library of questions that you can ask.

○ It is usually helpful to start from the highest protocol layer (e.g., application data) and work your way down if necessary. If you apply a security measure on a higher protocol layer, that may already make discussion of lower layer protocols irrelevant, or change the type of security needs lower layers need to provide.

○ If all the layers of a data flow are not end-to-end (e.g., application data is passed with different underlying protocol stack in different parts of the system), you need to treat each different underlying stack separately. As an example, if application data is passed through a HTTP-over-TLS connection at one point, and a local domain socket at some other point, these are different cases - unless all necessary security services are provided on the application level.

STRIDE stands for:

○ Spoofing (questions related to authentication):
  ■ Do we need to know who we are talking to? What is the real business reason? As an example: Many web sites do not really care about who the user is, but the user may care a great deal about the fact the website is not a spoof site.
  ■ How can the components at either end of a data flow ensure that they are really talking to the “correct” other end?
  ■ How does a component know where data really came from, if it was received by some other component?

○ Tampering (questions related to the integrity of data):
  ■ Does it matter that the data arrives exactly how it was sent? In the right order? Is it complete?
  ■ If we store data, how do we know it stays that way until read back again?

○ Repudiation (discussions about rollback attacks & needs):
Does it matter if a transaction is claimed not to have happened (Example: Client buys something and later claims not to have done it)?

Or is it actually a good thing if clients can say “didn’t do it” because it gives them privacy-preserving possibility of plausible denial? (Example: “It wasn’t me who was using this whistleblower site”)

Does your system create a log file for audit purposes? How can it be ensured that audit log entries cannot be reversed (edited/deleted)? (Example: Cannot trust an audit log because it was stored on a box that got compromised)

Does your system use a database? What sort of rollback capabilities you need to have, if an attacker would cause bad data to be injected? (Example: A user account got compromised; can we roll back the activities the attacker made, to get the user into the pre-compromise stage?)

- Information Disclosure (questions about keeping data secret, or confidentiality):
  - Who are those who are authorised to see the data being stored or transferred? How can we determine who has been looking at data and whether all those were authorised?
  - What are our long-term confidentiality needs? (Example: If we are using encryption now, does it matter if it won’t hold for 20 years?)

- Denial of Service (Availability of service to authorised users):
  - In which ways could an attacker cause service not to be available for those who are authorised to use it?

- Elevation of Privilege (getting the system to perform an action that the attacker / user was not authorised to do, for example through injection of bad data)
  - These issues have already been discussed to some length in taint analysis, but it is useful to explicitly ask the question.

DISCUSS: The list of considerations, above, is of course not complete. What other types of sources would you use for getting ideas of aspects to discuss about?

- Where to find worries?
  - Inherent technical & security knowledge of engineers? What sort of experience and traits would you want to see in an engineer doing STRIDE analysis?
  - Checklists? What are good and bad sides of checklists?
  - Lists of attack scenarios customised for your organisation? Why would they be useful? Is it realistic to build such lists? How would you make them not to be checklists?
  - The Elevation of Privilege card game is an example of facilitating questions is the Elevation of Privilege card game by Microsoft.
    - What would be the benefits and downsides of using such a game?
  - IEEE S&P article (see reading list) documents one way how the questions could be distilled into a list.
What do you think about this sort of list that actually cuts out STRIDE altogether?

What do we do with the results and findings? - Risk analysis

- Everything you find may not be a problem.
- Some problems you find may not be big enough risks.
- You need to decide whether the cost of mitigation (fixing the issue) is greater than the risk (likelihood of it happening, and its impact). How this process is ultimately driven depends on your organisation and whether you have some sort of mandatory risk management process. However, from experience, here is some practical advice to you as a software engineer.
  - Risk is impact x likelihood. A typical thing to calculate is an “annual loss expectancy”, that is, how many times per year the issue is going to cost you, times how much it is going to cost you each time.
  - The problem is that in many cases, you have no data to base your likelihood estimate on. It’s mostly guesswork. Impact is usually easier to estimate.
  - In many cases, the cost of mitigation in planning stage is very small, so you can actually make architectural changes that are cheap and quick. However, sometimes there could be a large cost involved, and then you may need to ask someone who decides on how money should be spent. In corporate biz speak, you would “escalate”, i.e., ask the boss.
  - If you are an engineer (without a monetary “approval limit”), make very clear that you do not make any large risk decisions! You are most likely not compensated (=paid) enough to make large risk decisions, and if things go sour, you might end up as the culprit.
  - If someone (your management) tells you that you should not fix an issue you identified, require them to explicitly sign off on the risk. This means, at a minimum, them sending an email that explicitly tells that the risk is acceptable. If you think that the risk is large (typically, a product liability risk), it would be a good idea to print the email and store it personally.
  - If you end up at a position where you need to accept risks, ensure you are compensated for the responsibility you take. If you aren’t, provide first-class technical opinions, but do not allow yourself to become the one who gets fired or jailed.

Licensed under Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International Public License (see https://creativecommons.org/licenses/by-nc-sa/4.0/).

Attribution: Antti Vähä-Sipilä, avs@iki.fi. Thanks to Sini Ruohomaa and Prof. N. Asokan.