Session 5: Reliability and Privacy Analysis; Examples of Security Patterns

NOTE: This session consists of two parts. 5a) is a continuation of Session 4, extending the analysis into information privacy. 5b) is about security design patterns that can be used as thinking or design aids during or after architectural discussion.

NOTE: The weekly exercise this week builds on week 4. There is a reference answer that will be made available for week 5, but only after the week 4 deadline has passed.

Session 5a: Extending data flow analysis beyond security: Reliability
- As it happens, a data flow diagram (session 4) can be used to determine other qualities than just security. There are two, reliability and privacy, that have security aspects.
- Reliability analysis
  - Reliability is related to being resistant to Denial of Service attacks - an externally induced failure can be a Denial of Service attack.
  - A data flow diagram can be used for reliability analysis by asking a series of questions for each data flow. These are considered from the caller's perspective (usually the source of a data flow).
  - Microsoft has an acronym for this as well; DIAL:
    - Discovery – can we find the resource we need (e.g., name resolution issues, etc.)
    - Incorrectness – callers of resources get errors back, or their messages are somehow corrupted or interoperable
    - Authentication/Authorization – caller cannot authenticate or authorize itself to a resource
    - Limits/Latency – There is a timeout, latency, or a limit that the caller meets.

Extending data flow analysis beyond security: Privacy
- Privacy, as a concept, comes from sociology and was created as a result of urbanisation. There was only a very limited form of privacy definition in rural and agricultural society; only masses provide relative anonymity which in turn is an enabler for privacy. You could go to a bar and drink and you could mostly count on nobody knowing who you were.
Information technology introduces concepts such as databases and rapid correlation of information, as well as machine-assisted identification. The old, “human”, privacy norms are difficult to enforce when the limits of human brain are no longer the limiting factor.

We are talking about information privacy here. This is about privacy-enabling treatment of information that is collected or processed through information technology. The information means information about a data subject, which is a person.

Also communications privacy is in our scope as far as it is done through information technology means. There is also other types of communication that is not electronic; that can be private, but is not in our treatment.

It is an interesting question whether non-natural persons would have anything like privacy - is there, for example, privacy for organisations or sentient neural networks that do not reside in human wetware. For the latter, the concept will have to evolve in the future. For organisatorial privacy, the general assumption is that it does not exist. Organisations have rights to trade secrets which are specifically regulated. However, human members of organisations have privacy rights as a part of them being humans, and their relationship to an organisation may be a privacy-related fact.

There are also other privacy aspects such as bodily privacy (right to one’s own body) and territorial privacy (right to be left alone, privacy of home). These sometimes spill over to information privacy. Bodily privacy could be violated by the release of health data, and spam (unsolicited email) or online bullying violate the right to be left alone. For the purposes of security risk analysis of an information system, we are treating these as system-specific requirements and risks, and these should be taken into account through their information privacy angle.

Privacy is based on security: There is a saying that you can have security without privacy, but no privacy without security.

Many privacy topics can be expressed in terms of the security needs that are required to implement the privacy topics. For example, a “leak of private data” from a system is in essence a confidentiality failure, and in STRIDE modelling, would be discussed under the “I” (Information Disclosure).

DISCUSS: What makes it then specifically a privacy issue? Why aren’t all confidentiality issues also privacy issues?

- What is a privacy issue is defined usually through legislation. The EU has a concept of Personal Data, and the data items that fall under privacy legislation are specifically listed. In the US, the term is usually Privately Identifiable Information or PII.
- The US (federal) legislation is “sectoral”, meaning that the US has multiple privacy laws and regulations that apply to specific contexts, such as HIPAA in health sector. EU privacy legislation is sector-neutral, and all use of personal data falls under the legislation irrespective of sector.

Privacy is a fairly large topic and would be a topic of a course of its own. The “OECD Guidelines”, a well-known information privacy guideline from 1980, was updated in 2013.
The Part 2 of the guidelines lists a number of principles that are usually understood as defining what privacy is. The current EU directive, and as a result, the national legislation, also echo the same OECD guideline principles. The principles are as follows:

- Collection limitation principle - Private information cannot be collected without limits, and the person about whom the data is, should be aware (or even consent) to the collection.
- Data quality principle - Private information must be accurate and up-to-date.
- Purpose specification principle - Private information needs to be collected for a specified purpose, and collected information cannot be used for another purpose that was not known at the collection time.
- Use limitation principle - Private information must not be used or disclosed to others than authorised parties.
- Security safeguards principle - Private information must be protected by appropriate information security controls.
- Openness principle - Private information collection must be transparent.
- Individual participation principle - Individuals whose information has been collected have the right to know this fact, know what data has been collected, and request corrections (or even deletion) of the data.
- Accountability principle - There needs to be accountability for following these principles, that is, enforcement of some sort.

One can extend a data flow diagram by superimposing a set of privacy domains in addition to security domains. Often these are the same, for example, in a simple web application, the server can be its own security domain and it could also be the privacy domain (if the server is controlled by a single entity).

However, sometimes the security and privacy domains can differ. A particular example could be when the server is a cloud service. The cloud server is in the possession of a third party (not the application vendor), so even though the data is security-wise in the application, privacy-wise it is in the cloud provider’s domain.

It is very useful to draw the privacy domains on the data flow diagram, annotated with the regulatory domain (e.g., European Union, United States, or something else).

Methods that are useful for privacy analysis with DFDs are LINDDUN and TRIM. We will look at TRIM here, being the one that the lecturer came up with, and probably easier to digest.

- Transferring data across borders - When a data flow extends from a privacy domain to another, does either the source or destination privacy domain consider this data to be personal data according to their legislation or contractual needs? If so, have you fulfilled the requirements for such transfer? For example, the European Union by default forbids transfer of personal data out of EU/EEC unless you have a permission for that.
- Retention periods - This is applicable to any data storage component. The question is that whether you have actually specified a maximum retention period for the data that is being stored. In most cases, you will have to define a maximum storage period and enforce purging of the data after the period has expired. If you have not defined a retention period, you may be in violation of the
purpose specification principle - you might not have a valid cause to store data indefinitely if its purpose is limited in time.

- Informed disclosure - When data is transferred out from a privacy domain, you would have to discuss whether the person about whom the data is has been made aware of the data transfer, and whether a consent is required or has been given. Unlike most information security data flows (where the responsibility of correctness is on the recipient) this question needs to be discussed at the source.

- Minimisation - Again, when data is being transferred out from a privacy domain, you should discuss whether the data that is being transferred is the minimum set that is technically required to fulfill the usage scenario. This is related to the collection limitation and purpose specification principles. If the data that is being transferred contains more data than what is technically necessary, it should not be transferred.

Visualising privacy domains on top of a DFD

Session 5b: Some Security Design Patterns

Note: This part begins a new theme on security design patterns. The previous part, 5a), was an extension to Session 4.

- We will have a look at three security design patterns. These are definitely not a complete treatise on secure design, but these aspects are perhaps the most common topics that help you to re-architect systems while doing security risk analysis.
- The patterns are described here. We will show “bad” architecture examples and then show how they can be made better by applying one or more of these three principles.
- There are many patterns that are technology specific (dependent on some aspect of a used platform or framework, for example). One researcher has catalogued almost a
hundred security design patterns and also classified them according to the STRIDE classification; please see the reading list.

Kerckhoffs’ Principle

- The principle is today known as the Kerckhoffs’ Principle (note: the person’s last name was Kerckhoffs, so the possessive is thusly spelled; the guy had seven first names plus a nobiliary particle and a second surname, but let’s just cut corners here).
- The actual principle, as it is currently understood, is from his La Cryptographie militaire from 1883, and a synthesis of two of his six requirements for encryption systems:
  - Requirement 2: If the attacker learns the inner workings of the system, that should not be a problem. In modern cryptographic terms, this means that the algorithm must be secure even if it is public.
  - Requirement 3: (actually the latter part of the requirement only:) The key should be easily changeable. Back in his time, this mainly meant that the key needs to be distinct from the algorithm, that is, the algorithm itself is not the key.
- Today, “Kerckhoffs’ Principle” means that the only thing that needs to be kept secret about an encryption algorithm is the key. Everything else must be secure even if made public.
- It helps the systems’ architectural security if the Kerckhoffs’ Principle is applied to the overall design of the system as well. Usually, in architectural risk analysis (session 4), if someone makes an argument that “the attacker wouldn’t know how we do things”, this should be an illegal argument, and it should be countered as a violation of the Kerckhoffs’ Principle. Any security control should be effective even if attacker would know how it worked. Otherwise you are relying on obfuscation - see the discussion about it below.
- One of the major benefits of following Kerckhoffs’ Principle on all architectural decisions is that it keeps the architectural design “intellectually honest”. It makes bad security decisions and so-called security debt (a type of quality debt) visible, forcing the “right thing” to be made, and things not being swept under the carpet. This makes it a kind of a meta-pattern that actually triggers other security activities.
- Often, the most typical step to apply Kerckhoffs’ Principle in a design is to identify the location of encryption or authentication key material, and ensure that this key material really needs to be known for the cryptographic operation to take place successfully. Once you have this logic verified (i.e., you are sure that without a key, an attacker cannot do something), you can apply an extra security boundary that wraps the key material. (For this, see the third security pattern.)
- **Obfuscation** as a security measure
  - An “anti-pattern” to Kerckhoffs’ Principle is obfuscation. It means that something is made very difficult to understand, or reverse-engineer, and some level of security is obtained.
  - In most cases, obfuscation should not be used as a security control. This doesn’t mean you should freely tell everyone how you have done security engineering, but the point is that the system should stay secure even if you did.
  - You are likely to meet obfuscation in cases where you have client-side code in a high-level language where, for some reason, someone has decided that it
shouldn’t be readable - for example, JavaScript on a web UI. In most cases, the question should be that exactly what is the security target and threat model. In the worst case, you will find actual access credentials hidden in the obfuscated code.

- Another place where obfuscation is often seen is in cryptographic algorithms where a DRM (Digital Requirements (or Restrictions) Management) key is being used. There are obfuscated decryption algorithms where the decryption key does not need to reside in device memory all at one time during decryption, and obfuscation is used to frustrate memory dump attacks.

- In most cases that I’ve seen, discussion about obfuscation just means that the security risk analysis is trying to address an issue that would require a large architectural change, but this change is not possible due to business reasons.

- Reasons why obfuscation is not suitable as the only security practice:
  - Obfuscated code is hard to maintain. Code that has been obfuscated is by definition not readable, and security engineering practices that are based on code analysis get more difficult.
  - Good obfuscation is surprisingly hard to do. If you do it manually, then you are potentially depending on certain persons who are able to do it (creating a personnel dependency). In many cases, naive obfuscation can be completely bypassed because the code can be attacked through an alternate channel; e.g., wait until the system is in a state where, say, an encryption key is in memory, and then dump the process memory. Whatever obfuscation happened before this becomes irrelevant.
  - You are trusting your people with the “keys” (information of how to compromise your system) but you have no real way of revoking this information if these people turn out to be untrustworthy. With proper engineering, you could just revoke those persons’ access.

- There are, however, three valid reasons why obfuscation can be a part of a security control. It just needs to be understood what benefits it offers and what restrictions it has.
  - Obfuscation can act as insurance, if the obfuscation tool / service provider gives you guarantees that it won’t be reversed - essentially, they agree to pay you if it gets reversed. Some commercial obfuscation tool providers may do this. They should be prepared to cover the actual loss expectancy due to a breach. (If they don’t, avoid them - this means they won’t trust their own product for your specific business case.)
  - Obfuscation can be an additional filter for attackers, meaning that you only have to deal with the best ones (those that can either reverse, or bypass your obfuscation), or an attack might be only working against a part of your customer base (if you use different obfuscation for different customers). This won’t make your system more secure in absolute terms, though. It just could mean that the really good attackers won’t even bother, having more profitable targets elsewhere.
Obfuscation can act as time based security for systems that are to be fielded for a limited time and become irrelevant after a short time. For an attacker to figure out the obfuscation layer may be long enough that the system will be obsolete. However, beware of the tendency of information systems staying around for much longer than originally planned!

Bootstrapping trust

- One of the reoccurring questions in security engineering is the chicken-and-egg problem: In order to trust something, we'll often have to trust something else. At some point, we have to make a leap of faith, or “take something for granted”.
- The thing we “take for granted” is known as the trust root. You might already be familiar with this from the world of certificates (such as X.509 certificates used in TLS and S/MIME) where there is a trusted party, a Certification Authority, that has a root certificate. The certificates presented by various web sites are tied back to this root, but the root itself is taken for granted - in many cases, it comes bundled with your operating system or browser.
- The key to understand bootstrapping trust is to understand the chain of trust that is being analysed, and where the “buck stops”. Examples:
  - Certificates: A root certificate signs an intermediate certificate; an intermediate certificate signs the server certificate.
  - Secure boot: The BIOS and Trusted Platform Module (TPM) ensure that the operating system that is being booted has not been tampered with; the operating system verifies that the applications have not been tampered with; and the application then verifies that its data has not been tampered with. This way, a machine could (theoretically) be booted to a known state even after compromise, and you only have to replace the layer that was breached.
- Each member of the trust chain performs an action called a measurement. This is usually a cryptographic one-way hash function calculated over the thing whose integrity is being attested. Before the control is given to the next link in the trust chain, the previous link needs to check that the measurement has not changed.
- When you require this sort of trust chaining, there are some major things that need to be kept in mind:
  - The original point of trust that is “taken for granted” may need to be highly secure. Depending on your threat model, you might need a hardware system that has anti-tampering systems in place. In many security systems, this means it would be a smart card or an embedded security system. (For example, ARM has a brand called TrustZone.) Especially in systems that reside physically with an attacker - like payment terminals and mobile phones - you may need to resort to tamper-resisting hardware solutions. This may also require “traditional” physical security measures such as tamper-evidence (if there is someone to see the evidence).
  - Sometimes it is ok for the source of trust to be readable but not writable - like root certificates often are. You just need to know that the roots haven’t changed. However, it does happen that sometimes these roots need to be revoked - it has happened several times that a Certificate Authority has been found to be
breached, and a root certificate needs to be removed. This sort of rekeying of trust roots may prove to be very costly especially in embedded systems or physical products, if you have not planned the update channel in advance. But remember that the update channel needs to be more trustworthy than the trust root that it is updating! Otherwise you have just created a back door.

- Typically, when you move towards a more complex system - as in a secure boot, where you first start with TPM and the BIOS, and end up running a full-fledged server with virtualised OSes - the complexity of code being run in the end means that the system may not be “secure” or “integrity protected”. The code may have vulnerabilities. However, the point of a trust chain is that if you pull the plug (and re-measure everything), the system can be brought back into a known (“trusted”) state. If you have patched the vulnerability and cleaned up, you have a theoretical way of recovering from an attack and to avoid attacker becoming persistent on the system.

- Today, in most systems, you are not doing architectural risk analysis down to the first trust root. You will stop at some layer - typically, in many cases, the OS layer - and assume that the OS is secure and its security functionality actually works. This is for practical purposes often something you have to do. However, let this pattern remind you that you need to explicitly accept this risk. If you are building a system that cannot take that risk, you must still go deeper to find the chicken that laid that specific egg.

Abstracting away problems through adding security boundaries

- You will find some “naturally occurring” security boundaries in your design. At a minimum, you would usually see that a server is run within an OS (although other programs could also run in there) and that your application probably is a process within that OS (so, if you aren’t running on a ‘90s home computer, other processes probably cannot just freely mess around with your memory).

- However, there is a very powerful architectural analysis technique where you will encapsulate security problems by creating new security boundaries.

- This can be used as an optimisation: If you cannot make something secure enough (there’s always that little nagging doubt), you can just wrap that into a security domain, and then you don’t have to worry (that much) about what happens in that security domain. What happens in Vegas, stays in Vegas.

  - Of course, what you have to remember is that the outputs from that security domain you introduced may not be trustworthy, if the attacker compromised that domain.

- The canonical example of this thinking and pattern is from qmail, which is a Mail Transport Agent (that is, a mail server) by Daniel J. Bernstein. qmail web page claims it has not had a security vulnerability since 1997 even though it has an ongoing bug bounty program.

- qmail consists of several processes that are mutually untrusted. If someone would somehow find a vulnerability in a process that processes incoming mail, that would be the extent of the attacker. The rest of the qmail would not be compromised.

- Typically, if you have to parse a data format (such as an incoming protocol), you could make use of the “qmail thinking” by extracting the parsing part - which, as we saw in
Session 1, would be prone to vulnerabilities - into a separate process that would be run with little permissions.

- Typically, you would use various OS and platform provided methods to add these security boundaries, or to *compartmentalise*:
  - On UNIX-type systems, you can have different processes running under different users, and segregate file access with users and groups;
  - again, on POSIX (and, for example, Linux), you have “capabilities” which are further permissions that can be granted to some processes but not others;
  - you can use a Mandatory Access Control system such as AppArmor to enforce which sort of actions executables can do;
  - you can run things inside a Virtual Machine - either a VM that runs an OS, or a bytecode interpreter that has some protection, like a Java VM. The hypervisors can offer further access controls for what the programs running inside them can access;
  - or isolate parts of a computation on a restricted system that can only talk to you through a limited connection. Some systems do key operations on devices that only link over a serial connection. A smart card (e.g., your SIM card) is such a thing;
  - or you can do “air-gapping”, which is a low-tech but very effective way of ensuring that something is integrity protected. Save it on a filesystem on a disk, unmount the file system and put it into a safe.

- You will have to remember the data flow diagram from Session 4, though. Any security domains that are within other security domains will be compromised if the outer domain is compromised. An example: If your virtual machine hypervisor gets compromised, everything within that hypervisor is compromised - even if you would have split the processing into several security domains.

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.