Security

Software security is a pervasive cross-cutting design concern of grave importance to modern software systems. Security concerns are not just about bad actors but also include unlucky users, untrained users, or malicious insiders. While there are obvious monetary costs associated with security (for instance through money being stolen directly), personnel costs, customer loss, and legal violations also have important indirect costs. Additionally, reputation, privacy, and customer satisfaction can all be influenced by the security of a system. This reading provides only the briefest of glimpses into the field of security.

> Security is ultimately concerned with managing risk whereby the risks are are balanced against the costs associated with securing the system.

Security comprises a huge space including physical security (e.g., key loggers), system security (e.g., denial of service), network security (e.g., man-in-the-middle), and human security (e.g., social engineering). Assets represent the information or systems that is being secured. The subjects of the system are the people who use, administer, and may try to infiltrate or undermine the system. Policies are required to understand which subjects are allowed to access which assets. Threats represent the reason we need policies (e.g., the existence of unauthorized subjects accessing the wrong assets through software vulnerabilities).

The DREAD model provides one way to think about security risks:

```
risk =
damage(1..10) +
reproducibility(1..10) +
#users(1..10) +
discoverability(1..10)
```

Four high-level requirements pervade most software security discussions:

- ***Confidentiality***: Preventing unauthorized parties from accessing (or even knowing about) secured data.
- ***Integrity***: Ensuring that only authorized parties can manipulate data, and only in an approved manner.
- ***Availability***: Resources should always be accessible by authorized users on appropriate occasions.
- ***Accountability***: It should be possible to know how subjects interact with sensitive systems and data.

In a security context, *assets* are the systems and data being secured. *Subjects* are the actors, both legitimate and otherwise, who are interacting with the system and data. The *policies* are the rules associated with the system; these are key for determining which subjects are permitted to view, interact with, or modify which assets. The *threats* to the system are the violations of the policies that can be employed to break the security of the system.

There are four high-level steps involved in securing systems:

1. ***Understanding***: Before we can secure a system we need to understand the system and its context.

2. ***Analysis***: The longest phase involves taking our understanding of the system and building meaningful threat models to understand how and why a subject would want to violate the system's security policies.

3. ***Mitigation***: Once this understanding has been built, the work of actually securing the system begins. At its simplest, this involves reducing the amount of system surface available to unauthorized parties while maintaining authorized user access.

4. ***Validation***: Finally, testing and security reviews are crucial to check the mitigation steps have been successful.

There are a variety of kinds (personas) of unauthorized user, all of whom may have different motivations for compromising a system.

- Accidental attackers.
- Disgruntled employees.
- Script kiddies.
- Curious attackers.
- Automated malware.
- Motivated attackers.
- Hacktivists.
- Cyber terrorists.
- State actors.

Each of these personas will have different motivations, goals, tools, and experience available to them. In particular, approaches further down the list will have the expertise and resources to exploit much more challenging system vulnerabilities. The STRIDE.aspx) threat model was developed at Microsoft to characterize the kinds of vectors these personas may employ to gain access to the system.

- ***S***poofing: pretend to be another subject.
- ***T***ampering: modifying data.
- ***R***epudiation: denying performing a task.
- ***I***nformation disclosure: access private data.

- ***D***enial of service: halt service.
- ***E***levation of privilege: gaining unauthorized rights.

Security principles

Several security principles have been developed to help mitigate some of these vectors (this is an incredibly incomplete list):

- ***Defence in depth***: A small violation should not easily lead to a big one. This means that security should not only be implemented at the perimeter of the system and internal controls should still be present. Varying internal security measures can further help avoiding failure propagation. As with most security countermeasures this negatively impacts system complexity and can decrease system usability (as is true for many of these principles).

- ***Least privilege***: Coarse-grained privileges allow both malicious and accidental access to unauthorized data. Implementing roles and access control lists are one of the most common ways of avoiding these exploits, although this only really works once the subjects, assets, and policies for the system has been carefully specified.

- ***Separation of duties***: Sensitive or critical actions in a secure system should not be left to a single user. This applies to both authorized (to avoid accidents) and unauthorized parties (to avoid broad exploits). This kind of separation is often apparent in accounting systems whereby actions must be approved by second or third parties to ensure compliance and correctness.

- ***Strong authentication***: Making it hard for unauthorized users to gain access to the system can be enhanced using strong passwords or some kind two-factor (or multi-factor) authentication. In two-factor authentication, users must both know something (like their password) and have something (like a phone, chip-with-pin card, or number-generating fob). This makes it so if only one part of the mechanism was lost (for instance the password written on a post-it note) the system could not be directly breached.

- ***Non-repudiation***: System users must be held accountable. Non-repudiation means auditing system behaviour to track how the system is being used. Auditing is an essential tool for tracking the extent to which unauthorized parties have accessed the system and what they have done. It is also useful for ensuring that authorized parties know their usage of the system is being traced. One downside of this approach is that audit logs themselves can contain sensitive information (or point to it) and must themselves be carefully secured.

Mediation strategies

After a system has been modelled and its threats understood, we can start to design mitigation strategies. Rather than step through an abstract list of strategies, here we will discuss more concretely the kinds of steps Google undertakes. These are covered in a great white paper if you want additional detail.

Google's security strategy revolves around providing a secure infrastructure for all of their services to use. These range from low-level hardware support to high-level operational support. Their mitigation strategies can be split into five key levels:

- Hardware infrastructure: In-house designs for server and network equipment to reduce risk of unknown hardware running in secure facilities.

- Service deployment: All services communicate using encrypted links. Services can only interact with other services with which they are authorized. Services can also only read/write data they are authorized to access. Services run in independent hypervisors although some critical services are also run on completely independent machines.
- User identity: All Google employees use multi-factor authentication internally. Employees are also granted restricted fine-grained access to they services and data they are allowed to access.
- Secure data storage: All data is encrypted with rotating key encryption. Data remanence (e.g., deletion) is also carefully managed, although there are challenges given the distributed nature of the data.
- Internet communication: All front-end services go through the GoogleFrontEnd (GFE) that ensures that perfect forward security is correctly applied and that all certificates are correctly configured. The GFE also has the ability to mitigate DoS attacks by throttling or balancing connections.
- Operational security: Engineers practice safe coding practices (such as security code reviews, using static and dynamic analyses, and employing security-aware frameworks and libraries). Google also works with the OSS projects that its tech stack relies on to find vulnerabilities in code they are using that could compromise their own services.

At their core, these mitigation strategies are a great example of defence in depth (multiple layers of countermeasures that does not solely rely on border security), least privilege (employees are strictly limited in the services and data they can access, as are the services themselves from one another), strong authentication (universal 2FA authentication that not only grants access to systems but is passed along with individual service calls), and non-repudiation (extensive logging and unusual activity detection).

References

- OWASP
- Web application discussion.
- Secure by design Eoin Woods GOTO slides.
- OWASP threat modelling wiki page.
