Advanced Security for Systems Engineering – Lecture 02: Secure Architectures

Florian Fankhauser, Martin Moutran, Christian Schanes, Christian Brem

esse

Agenda

Introduction

- Software/system architecture: mapping of the problem to software and hardware components
- Main components with connections of them
- Basic, abstract, high level design
- Security architecture
  - Definition of the basic security components
  - Security principles for building the system

Secure Architecture?

(See CERT Secure Coding Standards)

Another Secure Architecture?

(See Bodiam Castle, Chuck Andolino, CC BY-SA 2.0)

Complexity in Real-World IT

(See Curphey M, Arawo R. Web application security assessment tools)

Basis for Designing Secure Systems

- Defining the appropriate security requirements
- in a suited threat model and
- ensuring well-implemented security-mechanisms.
- Project as a whole needs to be considered
  - Software
  - Network
  - Hardware
  - Physical Security

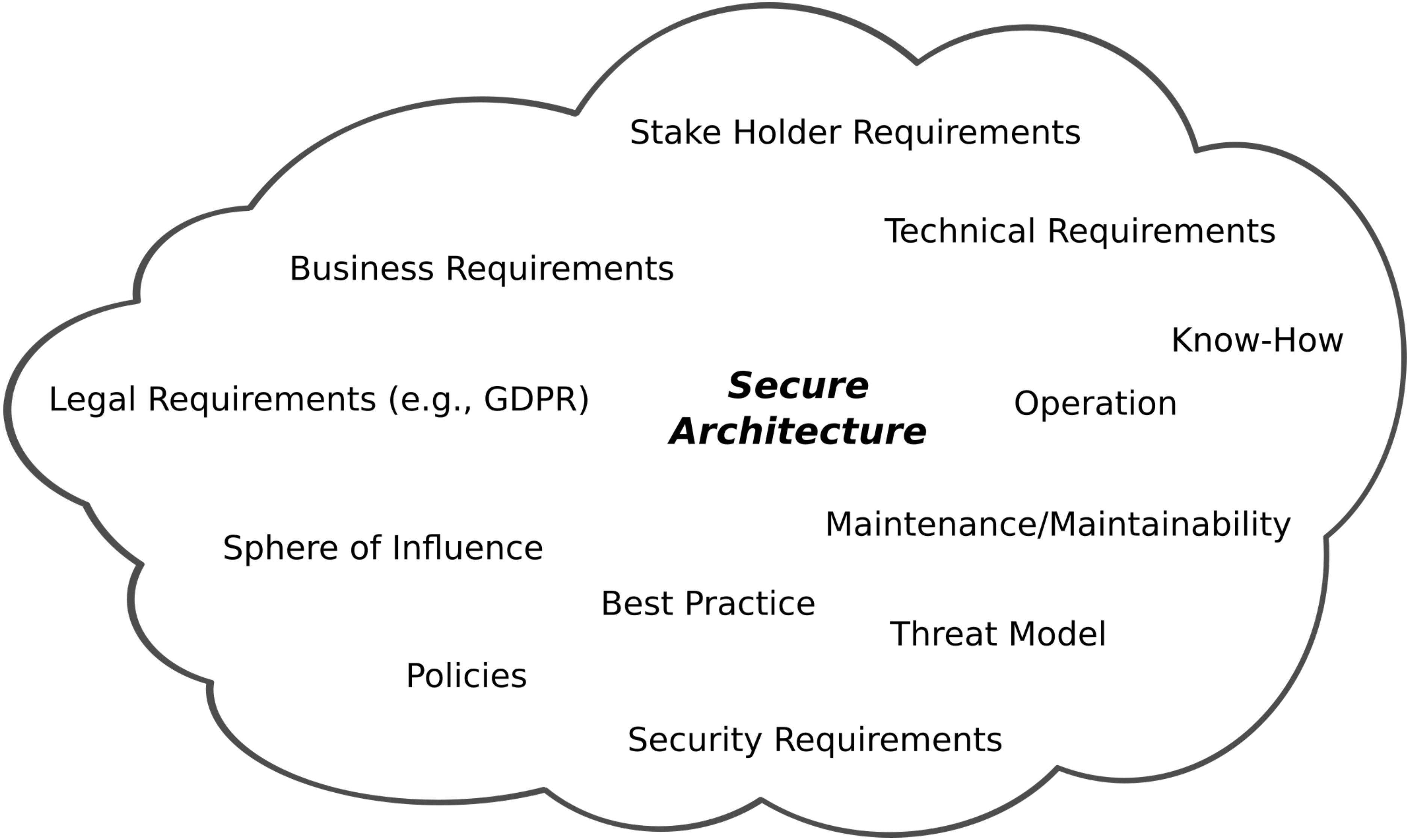
Broad Goals for a Security Architecture

- Detect attacks
- Resist attacks
- React to attacks
- Recover from attacks

(See Bass, Clements and Kazman, 2012)

Influences for a Secure Architecture

Examples of Influences for a Secure Architecture



Some Details of Influences

- Which security properties shall the system (or specific parts of the system) guarantee? E.g., confidentiality/anonymity,...
- Multiple requirements from different sources/stakeholders
- Are specific standards/regulations necessary?
- Is a certification necessary? Which one? (e.g., Common Criteria, PCI)
- Which requirements conflict? What to do about it?
- What to do if requirements change? Is there a process to recognize that?
- → IT Security in Large IT Infrastructures SS2023

Principles for Secure Architectures

- Security requirements analysis
- Protection requirements analysis
- Create barricades around to be protected artifacts using different mechanisms
- The more important a resource is, the more security measures
- Security architecture

Threat Model

- What are the assumed capabilities of the attacker? E.g.,
  - Active vs. passive attacker
  - Physical access vs. remote access only
- Which components of the system are
  - trusted,
  - partly trusted (e.g., distributed trust, honest-but-curious),
  - untrusted?

Hardware Security Architecture

- Devices
  - PCB design and layout
  - Secure CPU, MCU:
    - RAM encryption
    - Tamper detection
    - Secure keystore
    - Secure boot
    - ...
- Data center
  - HSM (Hardware Security Modules)
  - KMS (Key Management System)

Secure Design Principles

- Least Privilege
- Separation of Duties
- Fail Secure
- Economy of Mechanisms
- Complete Mediation
- Least Common Mechanisms
- Psychological Acceptability
- Leveraging Existing Components

(See Jerome H. Saltzer und Michael D. Schroeder, 1975)

Least Privilege – Design Techniques

- Modular Architecture:
  - Split entire system into smaller subunits
  - Each subunit is discrete with unitary functionality (cohesive)
  - Each subunit is designed to perform a set of logical operations
  - Each subunit is rather independent to others (coupling)
- Use virtualization
- Divide system into different network zones with defined interfaces between the zones, e.g., DMZ
- Best practice suggests: Designed software modules are highly cohesive and loosely coupled

Separation of Duties – Design Techniques

- When designed and implemented correctly, damage by a person or resource is reduced
- Separation of duties should be found in application features (i.e., role-based access control) and software development life-cycle (i.e., deny access for developers on production systems)
- Example when dealing with cryptographic keys:
  - Splitting of cryptographic keys, e.g., key ceremony for exchange of cryptographic keys
- Best practice suggests to implement separation of duties with auditing

Fail Secure – Design Techniques

- Define secure state to get into after errors
- Do not allow exceptions to go unhandled
- Do not allow any exceptions to reach the GUI
- Check, if error handlers are called frequently
  (i.e. there might be a security vulnerability)

Economy of Mechanisms – Design Techniques

- Avoid unnecessary functionality and unnecessary security mechanisms
- Strive for simplicity
  - Keep security mechanisms simple
  - Implementation should not be partial $\rightarrow$ otherwise security issues
  - Model the data in a simple way (results in simpler validation routines)
- Strive for operational ease of use (e.g., SSO)

Complete Mediation – Design Techniques

- Identify code paths that access privileged and sensitive resources and secure them
- Avoid duplicate code for input validation
- Be aware of social engineering attacks and security unaware users
- Keep in mind users that don't know how to use the software

Least Common Mechanisms – Design Techniques

- Design should isolate code (functions) by user roles → limits exposure of sensitive data
- Example:
  - Instead of sharing a function between superusers and nonsuperusers consisting of different code paths for each party, implement two separate functions to serve the different roles

Psychological Acceptability – Design Techniques

- Applications and especially security protection mechanisms should
  - be easy to use
  - not affect accessibility
  - be transparent to the user
- Users should not be burdened by security mechanisms
- Example:
  - Password policy that requires min. 16 characters for passwords may enforce users to write down their passwords and decrease overall security

Leveraging Existing Components – Design Techniques

- Promotes the reusability of existing components
- Tier architecture is advisable
  - Software functionality can be separated into presentation, business and data access tiers
  - Different presentation layers can be implemented
- Reuse tested and well known concepts and patterns for the architecture
- Reuse components if you know them! However, be careful about the security of existing components you don't know

Open Design – Design Techniques

- Avoid security by obscurity
  - No hard coding of sensitive information in source code or binaries (e.g., cryptographic keys, passwords, connection strings)
  - Use hidden form fields in web application carefully, i.e., modified client
  - Hidden URLs for secret documents without authentication, i.e., google hacking
- Use open and proven crypto systems

Defense in Depth – Overview

- Defense in Depth (layered defense) results from layering
  - Security controls = countermeasure to avoid / minimize security risk and
  - Risk mitigation safeguards into software design
- Should give an organization time to detect and respond to an attack
- Goal is that software doesn't get totally compromised, because of single security breach
- Example: it's not good to rely on a firewall only for an internal-use-only application
- More important assets should have more security layers
- Can be reactive (e.g., detect malicious activities and block them) or preventive (e.g., awareness training, security patches)

Defense in Depth – Shortcomings

- Can add complexity to the software system
- Contradicts to the principle of simple design
- Therefore, might introduce new risks

Trade-off between IT Security Goals

- IT Security Goals
  - Confidentiality
  - Integrity
  - Availability
  - Authenticity
- Security Goals may conflict with each other, when designing software architecture, e.g.,
  - Confidentiality vs. availability
    (e.g., Encrypted data not recoverable, if key lost)
  - Availability vs. authenticity
    (e.g., Slow hashing algorithms within authentication process)

Even More Software Architecture Aspects: Cryptography

- Cryptography
  - Where?
  - When?
  - Recovery of encrypted data when private key is lost?
  - Availability/processing speed
- → A Catalog of Security Architecture Weaknesses (Santos, Tarrit and Mirakhorli)

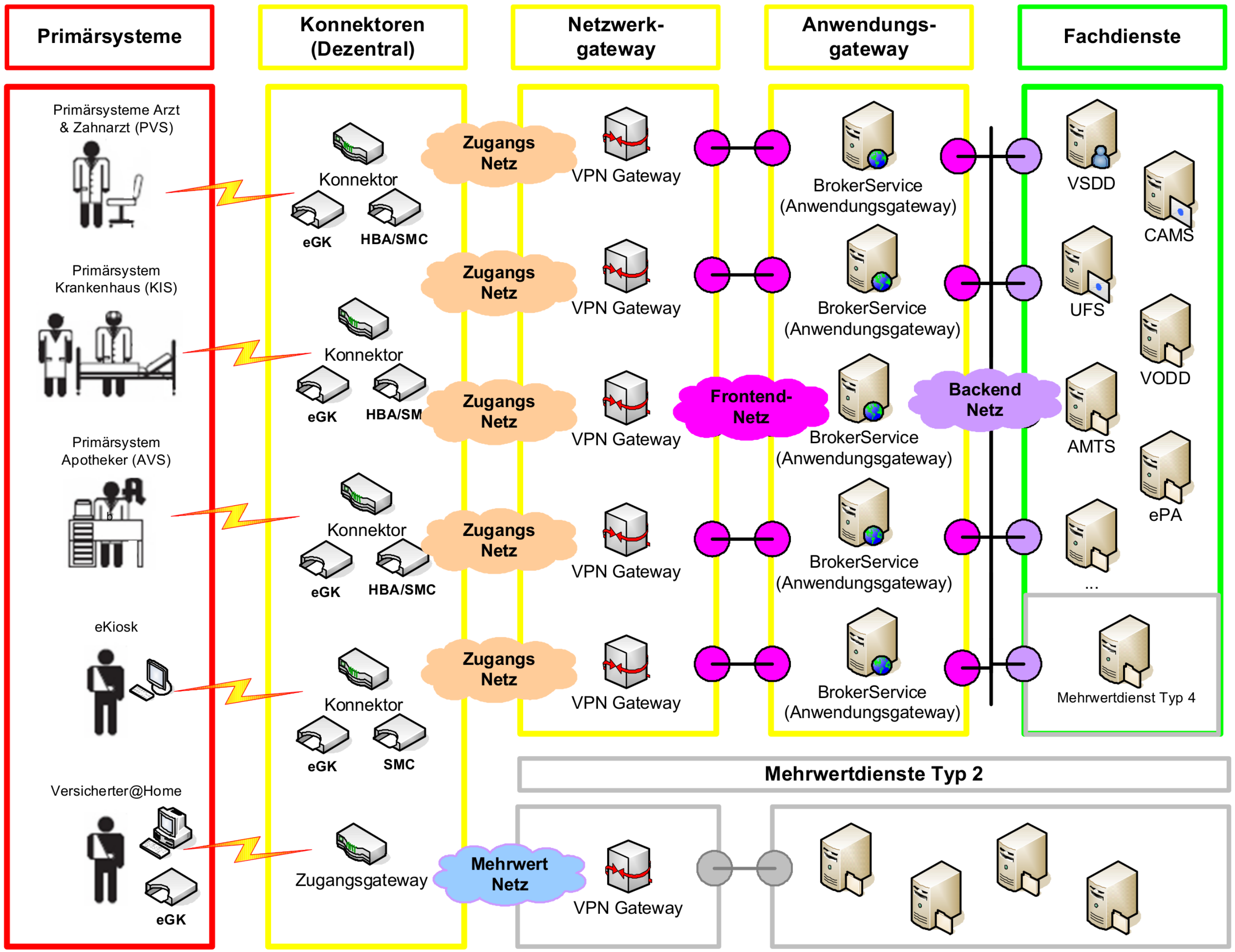
Network Security Architecture

- Connection of external sites/remote work
- Firewalls, VPN, Honeypots, DMZ, TLS, ...
- Adversarial Model in a Networking Setting
  - Protection against tampering: integrity protection
  - Protection against Replay and Re-order attacks: protocol must use, e.g., a message counter in the authenticated data, or an authenticated nonce value with the message that must be different for each new message
  - Protocol Downgrade Attack
    - TLS versions supported by server
    - HSTS

Example of Failed Security Architecture

(See The Washington Post)

Example of Large Security Architecture



(See Gesamtarchitektur Release Version 1.5.0, gematik)

Outlook: Guest Lecture about Zero Trust Architectures

- Guest lecture on November 25 finalized
- CSO of gematik
- *Implementation of Zero Trust in Complex IT Infrastructures (Current Status of Implementation within the German Health Telematics Infrastructure)*

References and Bibliography 1/5

- Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, 1975
- Matt Bishop. *Computer Security: Art and Science*. Pearson Education, Inc, 2003. ISBN 0-201-44099-7
- Ross Anderson. *Security Engineering. A Guide to Building Dependable Distributed Systems*. Wiley Publishing, Inc., 2 edition, 2008. ISBN 978-0-470-06852-6. `https://www.cl.cam.ac.uk/~rja14/book.html`
- Open Security Architecture

References and Bibliography 2/5

- Joshua J. Pauli and Dianxiang Xu. Misuse case-based design and analysis of secure software architecture. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 2, pages 398–403 Vol. 2, April 2005a. doi: 10.1109/ITCC.2005.199
- Mark Curphey and Rudolph Arawo. Web application security assessment tools. *Security & Privacy, IEEE*, 4(4):32–41, 2006. ISSN 1540-7993. doi: 10.1109/MSP.2006.108
- Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012. ISBN 0321815734, 9780321815736

References and Bibliography 3/5

- Joshua J. Pauli and Dianxiang Xu. Misuse case-based design and analysis of secure software architecture. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 2, pages 398–403 Vol. 2, April 2005b. doi: 10.1109/ITCC.2005.199
- Koen Yskout, Riccardo Scandariato, Bart De Win, and Wouter Joosen. Transforming security requirements into architecture. In *2008 Third International Conference on Availability, Reliability and Security*, pages 1421–1428, March 2008. doi: 10.1109/ARES.2008.47

References and Bibliography 4/5

- Eduardo B. Fernandez, Mihai Fonoage, VanHilst Michael, and Mirela Marta. The secure three-tier architecture pattern. In *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 555–560, March 2008. doi: 10.1109/CISIS.2008.51
- Joanna C. S. Santos, Katy Tarrit, and Mehdi Mirakhorli. A catalog of security architecture weaknesses. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 220–223, April 2017. doi: 10.1109/ICSAW.2017.25
- Gunnar Peterson. Don't trust. and verify: A security architecture stack for the cloud. *IEEE Security Privacy*, 8(5):83–86, September 2010. ISSN 1540-7993. doi: 10.1109/MSP.2010.149

References and Bibliography 5/5

- Gilberto Pedraza-Garcia, Hernan Astudillo, and Dario Correal. A methodological approach to apply security tactics in software architecture design. In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages 1–8, June 2014. doi: 10.1109/ColComCon.2014.6860432

Summary

- Solid architecture is the basis of secure systems
- IT security goals may conflict with another
- Due to business needs decide what design principles to use
- Balancing of different design principles is recommended
- “Complexity is the worst enemy of security” (Schneier)

esse

Thank you!

https://establishing-security.at/