Secure Software Engineering and Embedded Systems

Jan Jürjens

Competence Center for IT Security
Software & Systems Engineering
TU Munich, Germany

juerjens@in.tum.de
http://www.jurjens.de/jan

Personal Introduction + History

Me: Leading the Competence Center for IT-Security at Software & Systems Engineering, TU Munich

- Extensive collaboration with industry (BMW, HypoVereinsbank, T-Systems, Deutsche Bank, Siemens, …)
- PhD in Computer Science from Oxford Univ., Masters in Mathematics from Bremen Univ.
- Numerous publications incl. 1 book on the subject

This tutorial: part of series of 30 tutorials on secure software engineering. Continuously improved (please fill in feedback forms).

A Need for Security

Society and economies rely on computer networks for communication, finance, energy distribution, transportation...

Attacks threaten economical and physical integrity of people and organizations.

Interconnected systems can be attacked anonymously and from a safe distance.

Networked computers need to be secure.

Secure Systems Development

High quality development of security-critical systems difficult.

Many systems developed, deployed, used that do not satisfy security requirements, sometimes with spectacular attacks.

Example: NSA hackers break into U.S. Department of Defense computers.

Causes I

Designing secure systems correctly is difficult. Even experts may fail:

- Needham-Schroeder protocol (1978)
- Attacks found 1981 (Denning, Sacco), 1995 (Lowe)

Designers often lack background in security.

Security as an afterthought.

Little feedback from customers.

Causes II

„Blind“ use of mechanisms:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).

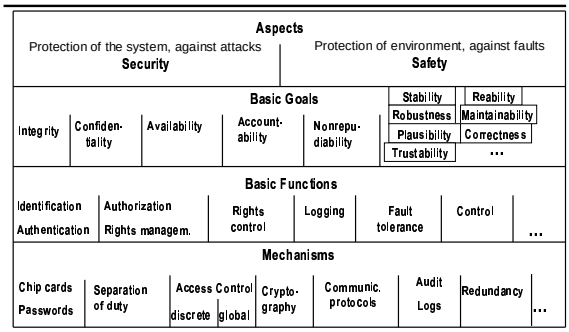
Quality vs. Cost

Correctness in conflict with cost.

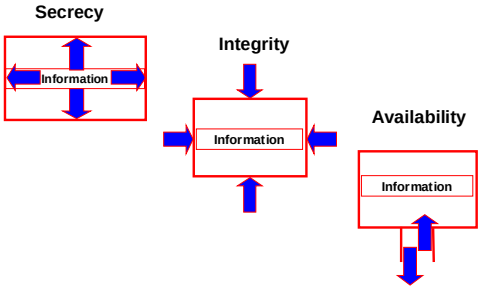
Thorough methods of system design not used if too expensive.

Security Requirements

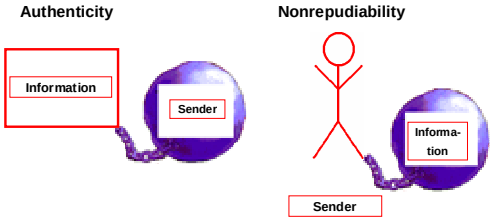
[Wec03]



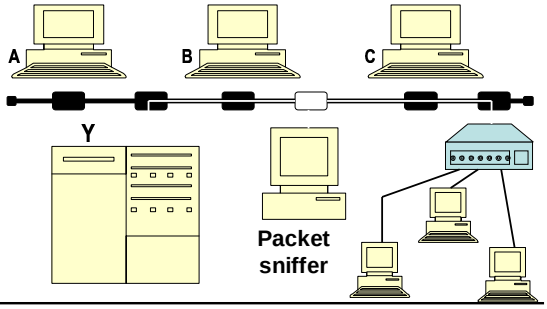
Basic Security Requirements I



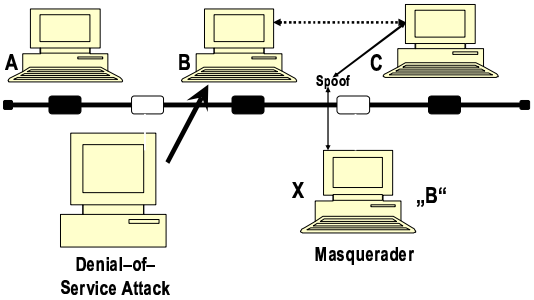
Basic Security Requirements II



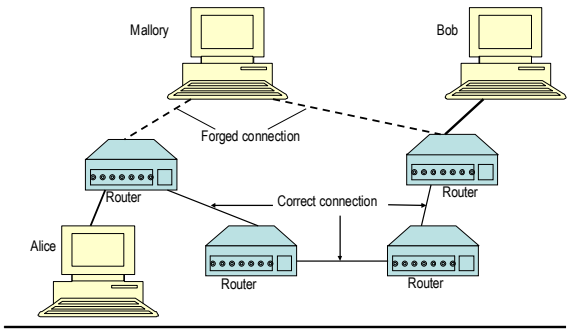
Internet Attacks: Eavesdropping



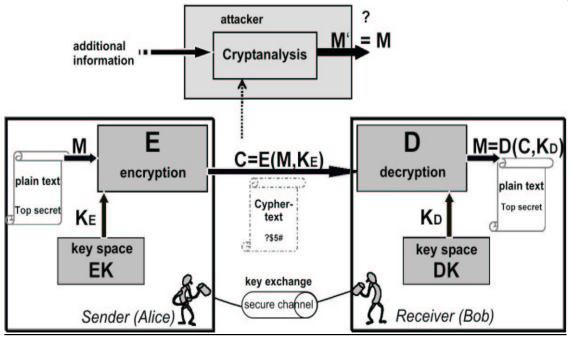
Internet II: Masquerading (Spoofing)



Internet III: „Man-in-the-Middle“



Defense: Cryptography



Cryptographic Algorithms

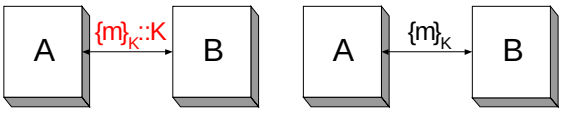
Symmetric:

- Digital Encryption Standard (DES), 3DES
- Advanced Encryption Standard (AES): Ryndael 2001

Asymmetric:

- RSA (Rivest/Shamir/Adleman): relies on integer factorization
- ElGamal: relies on diskrete logarithm
- Diffie-Hellman: Generate key shared between two parties

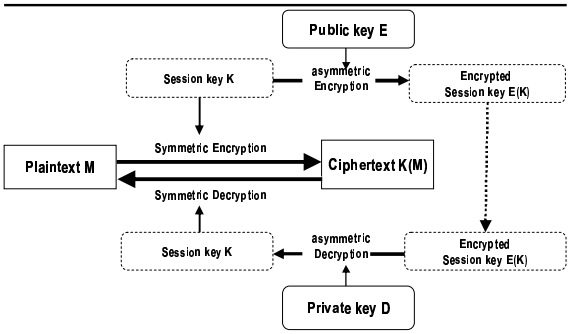
Encryption vs. Secrecy



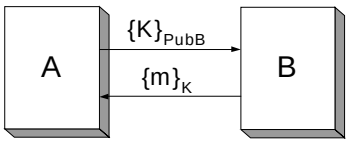
Against eavesdropper:

- Secrecy of $\{m\}_K::K$ not preserved
- Secrecy of $\{m\}_K$ preserved

Hybrid Encryption Schemes

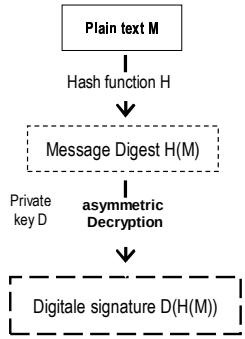


Hybrid Schemes vs. Secrecy

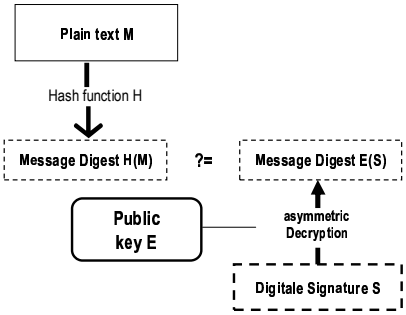


- Secrecy of *m* not preserved against an attacker who can delete and insert messages.
- Secrecy of *m* preserved against an attacker who can eavesdrop, but not alter the link.

Creation of digital signatures



Verification of digital Signatures



Cryptographic Protocols

Need to be able to securely determine identity of communication partners. Threats:

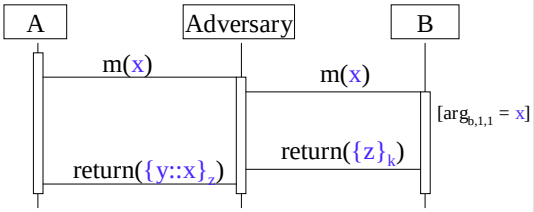
- forge identifications
- replay old identifications

Need to manage keys, perform electronic transactions, ...

Use cryptographic protocols: Exchange of messages for distributing session keys, authenticating principals etc.

Notoriously hard to get right.

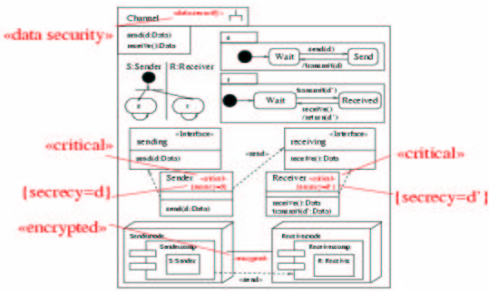
Protocol: Attack Scenario



Adversary knowledge: k^{-1}, y, x, $\{z\}_k$, z

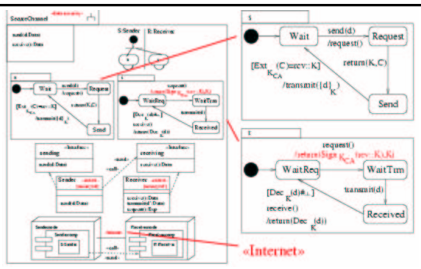
- $\forall e, k. \mathcal{D}ec_{k^{-1}}(\{e\}_k) = e$

Using Protocols: Secure Channel



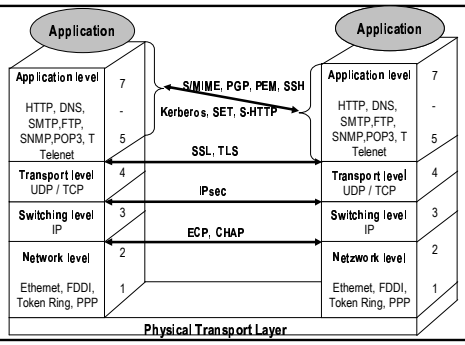
To keep *d* secret, must be sent encrypted.

Secure Channel Pattern: Solution



Exchange certificate and send encrypted data over Internet.

Encryption and Protocol Layers



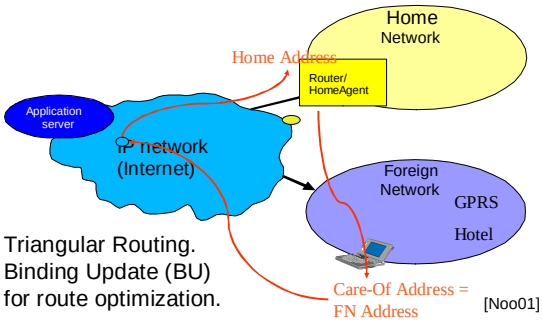
Mobile Systems: Mobile IP

Mobile computers (laptops).
Wireless networks (GPRS, UMTS,…).
Move from one link to another without changing IP address.

- Don't restart communication (VoIP).
- IP-v6: provides "everything" with IP address.

Solution: home address vs. care-of address.

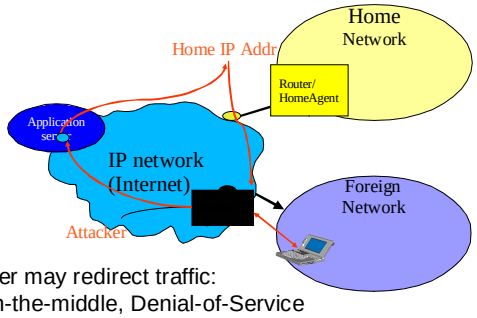
Mobile IP Scenario



Triangular Routing.
Binding Update (BU) for route optimization.

[Noo01]

Security Issues

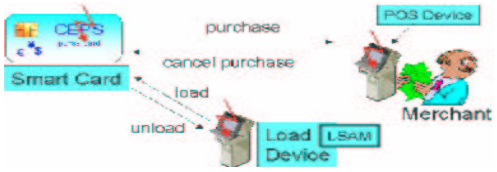


Attacker may redirect traffic:
Man-in-the-middle, Denial-of-Service

Routing-based authentication

Foreign network sends secret key to home agent through (hopefully) secure route.
Home agents forwards key to mobile node through secure tunnel.
Mobile node uses key for authenticating binding update to foreign network.
Goal of Mobile IP: as secure as wired network.

Embedded Application Security: CEPS



Common Electronic Purse Specifications (CEPS): Global electronic purse standard (90% of market).
Smart card contains account balance. Chip performs cryptographic operations securing the transactions.
Securer than credit cards (transaction-bound authorization).

Load protocol

Unlinked, cash-based load transaction (on-line).

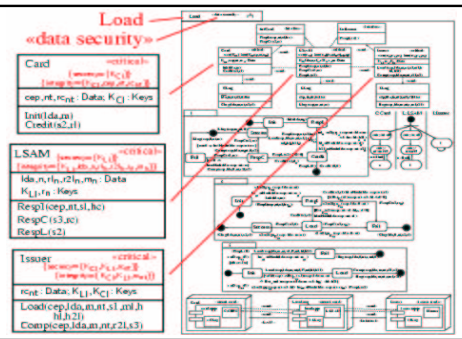
Load value on card using cash at load device.

Load device contains Load Security Application Module (LSAM): secure data processing and storage.

Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

Load protocol



Security Threat Model

Card, LSAM, issuer security module assumed tamper-resistant.

Intercept communication links, replace components.

Possible attack motivations:

- **Cardholder**: charge without pay
- **Load acquirer**: keep cardholder's money
- **Card issuer**: demand money from load acquirer

May coincide or collude.

Audit security

No direct communication between card and cardholder. Manipulate load device display.

Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception). For example:

Load acquirer security: Load acquirer has to pay *m* to card issuer only if load acquirer has received *m* from cardholder.

Load acquirer security: details

Suppose card issuer *I* possesses $ml_n = Sign_{rn}(cep::nt::lda::m_n::s1::hc_{nt}::hl_n::h2l_n)$ and card *C* possesses rl_n, where $hl_n = Hash(lda::cep::nt::rl_n)$.

Then after execution either of following hold:

- $Llog(cep,lda,m_n,nt)$ has been sent to I:LLog (so load acquirer L has received and retains m_n in cash) or
- $Llog(cep, lda, 0, nt)$ has been sent to I : LLog (so L returns m_n to cardholder) and L has received rc_{nt} with $hc_{nt}=Hash(lda::cep::nt::rc_{nt})$ (negating ml_n).

"ml_n provides guarantee that load acquirer owes transaction amount to card issuer" (CEPS)

Flaw

L does not provide load acquirer security against adversaries of type insider.

Automatically detected using UML tools.

Modification: use asymmetric key in ml_n, include signature certifying hc_{nt}.

Automatically verified this version wrt. above conditions.

Code-based attack: Buffer Overflow

Common security vulnerability.

Values written into fixed length buffer and partially written outside buffer's boundaries.

Facilitated by use of vulnerable library routines (Unix: e.g. `gets` and `strcpy`)

- execute arbitrary code with superuser rights
- often in connection with stack smashing.

C, C++, and assembly offer no protection.

The Stack

Portion of address space of a process.

Provides storage for local variables.

Scratch-pad area when process needs temporary storage.

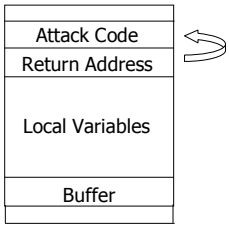
"Bookkeeping" information for function call:

- Parameters that won't fit in registers
- Saved values of registers
- Address from which function was called.

Stack Smashing

Attacker provides input string containing executable binary code.

The buffer overflow lets return address in stack frame for currently active function point to attack code.



On function return: control transferred to attack code, not returned to calling routine.

Stack Smashing Example

`line`: 512-byte array allocated on stack

`gets()` provided with more than 512 bytes: still puts data on stack

```
main(argc, argv)
{
 char line[512];
 ...
 gets(line);
 ...
```

By choice of data in `line`, can divert flow of execution to special instruction sequence calling `execv()` to replace running image with a shell.

Stack Smashing and Privileges

Often used to attack programs running as root or binaries installed SetUID root:

SetUID permissions in UNIX grant user privilege to run programs or scripts as another user.

Programs that are SetUID root may be executable by underprivileged user, but run with unrestricted system access.

Attacks may allow unprivileged user to acquire root privileges with one exploit.

Preventing Stack Smashing in C

Validate all input. Perform bounds checking on all arrays. Let programs execute at lowest necessary privilege level.

Avoid using functions that do not check bounds (`strcpy()`, `strcat()`, `sprintf()`, `gets()`, …).

Use safer alternative functions (`strncpy()`, `strncat()`, `snprintf()`, `fgets()`, …).

Use libraries and tools that can prevent buffer overflow vulnerabilities.

Preventing Stack Smashing: Tools

StackGuard: gcc patch enabling executed code to detect change in return address (and fail safely). Moderate performance penalty. No modification in source code necessary.

- Canary Words: Write random „canary“ word between local variables and return address. Jump back only if intact.
- MemGuard: Make memory page with return address „read-only“. Emulate write‘s to variables on same memory page. Securer but slower.

Tools II

- Linux kernel patch: stack non-executable. Non performance penalty but buffer-overflow attacks still possible.
- gcc patch: array bounds checking. Securer than StackGuard but significant performance penalty.
- Where possible, use type-safe languages (Java).

Software Engineering & Security

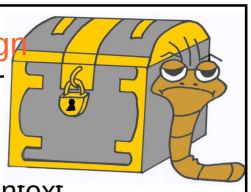
„Penetrate-and-patch“ (aka „banana strategy):

- insecure
- disruptive

Traditional formal methods: expensive.

- training people
- constructing formal specifications.

Goal: Security by Design



Consider security

- from early on
- within development context
- taking an expansive view
- in a seamless way.

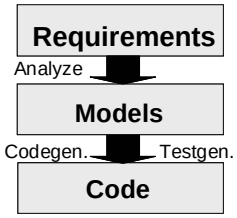
Secure design by model analysis.

Secure implementation by test generation.

Model-based Security Engineering

Combined strategy:

- Analyze models automatically against security requirements
- Generate code from models where reasonable
- Write code and generate test-sequences otherwise.



Increase quality with bounded time-to-market and cost.

Using UML

UML: unprecedented opportunity for high-quality critical systems development feasible in industrial context:

- De-facto standard in industrial modeling: large number of developers trained in UML.
- Relatively precisely defined (given the user community).
- Many tools in development (also for analysis, testing, simulation, transformation).

UMLsec: Goals

Extensions for secure systems development.

- evaluate UML specifications for weaknesses in design
- encapsulate established rules of prudent secure engineering as checklist
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- make certification cost-effective

The UMLsec Profile

Recurring security requirements, adversary scenarios, concepts offered as stereotypes with tags on component-level.

Use associated constraints to evaluate specifications and indicate possible weaknesses.

Ensures that UML specification provides desired level of security requirements.

Link to code via test-sequence generation.

Further Applications

Multi-layer security protocol for web application of German bank

SAP access control configurations

Biometric authentication system of German telecommunication company

Automobile emergency application of German car manufacturer

Electronic signature architecture of German insurance company

Test-generation: Conformance testing

Classical approach in model-based test-generation (much literature).

Can be superfluous when using code-generation [except to check your code-generator, once and for all].

Works independently of criticality requirements.

Conformance testing: Problems

Complete test-coverage usually infeasible. Need to somehow select test-cases.

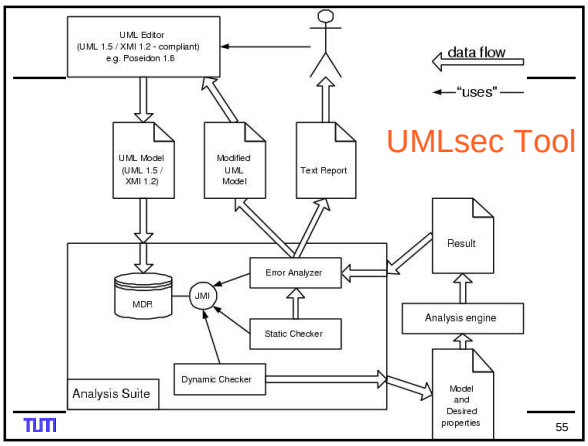
Can only test code against what is contained in behavioral model. Usually, model more abstract than code. So may have „blind spots“ in the code.

For both reasons, may miss critical test-cases.

Criticality testing

Strategies:

- Ensure test-case selection from behavioral models does not miss critical cases: Select according to information on criticality („internal“ criticality testing).
- Test code against possible environment interaction generated from external parts of the model (e.g. deployment diagram with information on physical environment).



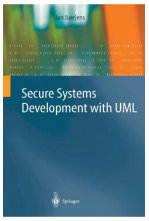
Some resources

Book: Jan Jürjens, Secure Systems Development with UML, Springer-Verlag, 2004

Tutorials: Sept.: SAFECOMP (Potsdam), ASE (Linz), NODe (Erfurt).

Summer School Lecture: FOSAD (Bertinoro, Italy, Sept.)

Workshop: CSDUML@UML04



More information (papers, slides, tool etc.):
http://www4.in.tum.de/~juerjens/csdumltut
(user Participant, password lwasthere)

Finally

We are always interested in industrial challenges for our tools, methods, and ideas to solve practical problems.
More info: http://www4.in.tum.de/~secse

Contact me here or via Internet.

Thanks for your attention !