TOWARDS SECURITY AT ALL STAGES OF A SYSTEM'S LIFE CYCLE

M. Schumacher[1], R. Ackermann[2], R. Steinmetz[1,2,3]
Darmstadt University of Technology, Department of Computer Science
Wilhelminenstr. 7, 64283 Darmstadt, Germany
[1] Department of Computer Science - ITO
[2] Department of Electrical Engineering and Information Technology - KOM
[3] German National Research Center for Information Technology - IPSI
Markus.Schumacher@ITO.tu-darmstadt.de,
{Ralf.Ackermann,Ralf.Steinmetz}@KOM.tu-darmstadt.de

Abstract: *Recent experience has shown, that interconnected systems are vulnerable to attacks, if security questions are not met appropriately. In this paper we present selected reasons for the current dissatisfying security level of distributed systems and present selected approaches of making systems secure. We describe our concept for a systematic way of understanding security weaknesses and elaborating efficient solutions.*

KEYWORDS:* *Security, Software Engineering, System and Network Vulnerabilities

1 Introduction

Much attention has recently been devoted to security issues and it has become apparent that a high security level should be a fundamental prerequisite for digital market places of the future. The recent occurence of the *I Love You virus* [**?**] or the *Distributed Denial-of-Service Attacks* [**?**] attacks against famous web sites in beginning of 2000 showed, that we will still need quite some time to reach a security standard of IT systems alike the standard already usual in other fields.

One reason is, that - especially in distributed environments - it is very difficult to make a software sytem secure, as there are many different components and mechanisms involved. In addition, trust relationships change frequently, which makes an analysis of all security requirements very hard [**?**], [**?**].

Another finding is that the software industry does not seem to learn from past errors as even well-known security problems such as buffer-overflows continue to appear over and over again [**?**], [**?**], [**?**], [**?**].

Though not basically related to security, the y2k problem demonstrated, that it is not an impossible mission to cope with known problems in advance. As preventive measures was taken in advance that time the lesson had been learned and major damage could be prevented.

This document is organized as follows: Section **??** discusses major reasons for the current dissatisfying security level of distributed systems. Section **??** presents a selection of security approaches at different stages of the life cycle of a software system. Section **??** introduces our concept for the systematic analysis of software errors and the determination of appropriate solutions. Section **??** outlines the related work in the field of the analysis of software vulnerabilities. Finally, section **??** presents the conclusions, summarizes our findings, and discusses future directions.

2 Thinking about Security Weaknesses

In the following we present selected reasons for the current dissatisfying security level of distributed systems. Based on the author's experience those reasons do not originate from a limited number of technical problems only.

2.1 Complexity

The problem of complexity in distributed systems is described best with a quotation of Bruce Schneier [**?**]: *Complexity is the worst enemy of security. Secure systems should be cut to the bone and made as simple as possible. There is no substitute for simplicity. Unfortunately, simplicity goes against everything our digital future stands for.* In fact today's IT systems have properties, that make the consideration of security difficult such as heterogeneity, dynamics, and lack of transparency [**?**].

2.2 Innovation Cycles

An ever increasing number of new features and new products hits the market and innovation cycles become shorter. Unfortuneately security is often - if at all - only seen as an add-on in contrast to other frequently demanded features of IT systems such as performance, useability, and reliability. Furthermore it is very difficult to *retrofit security in an application* [**?**] due to time consuming modifications of the design, rewritings of code, and enhancements of testing procedures. Thus, systems are often shipped with „quick-and-dirty" patches or no security at all.

2.3 Incomplete or Wrong Assumptions

As stated in [**?**] „*assumptions that programmers make regarding the environment in which their application will excecute* [...] *frequently do not hold in the excecution of the program*". This is mainly because the assumptions are incomplete or simply wrong and (partially) explains flaws like race conditions and buffer overflows.

2.4 Know-How Transfer

Making a system secure in a convinient time requires a high amount of expert knowledge. In the follwoing we list some non-technical aspects, that prevent know-how transfer in the field of security.

- *Monetary Aspects*: security sells and many people buy it. Many consultants offer seminars, workshops or professional security scans. As their know-how is a monetary value, chances are good to assume, that they are giving away „only pieces of the whole truth". Additionally, non-disclosure agreements might prevent them from passing available information to the public.

- *Lack of Experience*: unfortunately the common developer is no security expert. Usually only a selected circle of people really understands, what security means and how it can be deployed into systems.

- *Political Issues*: in some cases individual states also intentionally try to avoid a (too) high security level. As an example, the British intelligence service has originated a weakening of the GSM encryption mechanism [**?**].

2.5 Findings

As long as no suitable means of implementing secure systems are available, security remains a time and money consuming software feature. That leads to the omnipresent penetrate-and-patch approach we notice today. In a shipped product vulnerabilties will be eliminated only after they are accidentally discovered, in many cases after a successful attack. The analysis of the current situation mainly shows a passive and reactive approach instead of the attempt to prevent errors in advance.

3 System Life Cycle and Security Approaches

Ideally, security should be considered at all stages of the software engineering process. In the following we present selected approaches of making systems secure. The stages of the system life cycle are subset that is derived from [**?**].

3.1 Design: Pattern Approaches

As described in [**?**] *a pattern is a recurrent solution to a specific problem in a context* and should help novices to act as (security) experts. For experts it can be seen as a common vocabulary for (security) problems. As written in [**?**], it allows the members of the pattern community to identify, name and discuss both problems and solutions more efficiently.

In the field of pattern languages we find security related contributions too. A pattern language for cryptographic software is introduced in [**?**]. It focuses on the main objectives of information security, i.e. confidentiality, integrity, authentication and proof of origin. The authors realized that *cryptograhy is becoming a default feature* *in many applications* and destilled the essential design concepts for cryptographic software components.

On a higher level of granularity [**?**] identifies patterns forsecurity enabled applications. In contrast to [**?**], those do not focus on cryptography but on a framework for building secure applications. It can be thought of as a set of functional blocks, e.g. a single access point or a secure access layer, which should be best practice in secure applications.

3.2 Implementation: Guidelines and Source Code Analysis

Security guidelines, checklists or programming conventions can improve security during the development and testing of software. As an example we see FAQs such as [**?**] or checklists like [**?**] that provide guidance in secure programming.

Based on the research of software assurance for security a method for the security analysis of C and C++ source code has been developed [**?**]. The tool allows to check for known vulnerabilities in security critical software packages. Other approaches are to replace libraries with secure implementations or to provide runtime checks of security critical library calls.

3.3 Operation: Security Analysis, Infrastructure and Safeguards

Tools for security analysis such as [**?**] and [**?**] can be used for detecting know vulnerabilties. Typically they can detect errors in the configuration or the presence of faulty pieces of software. We consider these tools to have both a preventive and a reactive nature - they can be used before a system becomes operational and to monitor a certain security level.

In a similar way, we classify components of the security infrastructure such as Intrusion Detection Sytems and Firewalls. They are used to enforce a defined security level and help to protect from known threats. Additionally they may emit notifications on the occurance of unusual situations.

Standard security safeguards can be found in reference models like the IT Baseline Protection Manual [**?**] or the Site Security Handbook [**?**].

3.4 Findings

So far there seem to be single solutions for particular problems, but an isolated approaches does not solve the security problem. It is necessary to understand, that security aspects must be considered during all phases of software engineering, especially preventive measures in the earlier phases would improve the security level of distributed systems significantly.

4 An Integrated Approach to Software Security

In the following we describe our concept for a systematic way of understanding security weaknesses and elaborating efficient solutions. Our approach is clarified in figure 1 and is based on the concept of a closed feedback loop. The top-level components and interfaces that we have already identified and partly implemented are introduced in the following.

Figure 1: OUR Approach - a Feedback-Loop: Observe, Understand, (Re)act

4.1 Interface A: Analysis and Utilization

A highly structured *Vulnerability Database*[1] (VDB) is the most important prerequisite for the systematic analysis of security problems, which will help to render both existing and new systems more secure. As we have described in [**?**] appropriate data mining procedures help to identify and improve patterns that are in turn used to engineer new or to improve existing systems. Our main objectives are described as follows:

- **Assessment** of the system's hazard: Through information on comparable compromised systems vulnerabilities can be indicated and counter-measures can be recommended. For completion the force of expression of the assessment can be improved by providing test procedures for individual vulnerabilities.
- **Prognosis** on how likely it is that vulnerabilities occur and on the category of vulnerability to be expected for new software components not yet registered.
- **Avoidance** of known faulty design patterns with future software projects: Through analysing the vulnerabilities found the faulty design patterns behind are identified. Building up on this, the corrected design pattern can be developed and made available.

Currently we performed a survey [**?**]in order to determine the most acceptable operational properties of a vulnerability database that will be of use for the greatest possible group of people, companies and institutions. The evaluation will reveal whether an existing VDB is sufficient for systematic analysis.

[1] A Vulnerability Database is contains detailed data on vulnerabilities such as possibilities of exploitation, impact on system security, and possible ways to solve the problems caused by the vulnerability.

4.2 Interface B: Transformation and Screening

A uniform data scheme is important for (semi-)automated examinations of data. In order to achieve this, it is important to know the *structure of information*. In general, highly structured information is more suitable for machine-based processing. Besides structure, the storage of information is also important. We distinguish between database or file-based storage systems.

Usually it will be necessary to transcode information into the desired database scheme. Depending on the structure, human interaction will be necessary. With the help of dynamic ontologies [**?**], important catch-words out of the vulnerability descriptions can be used. The characteristics of catchwords are catalogued with the help of logic-based description language in order to achieve a standardized vocabulary for rating and screening of information.

4.3 Interface C: Information Retrieval

In order to gather information efficiently, we work on components for (semi-)automated information retrieval. Currently we have a prototype implementation for the monitoring of mailing-lists, newsgroups, and HTML pages[2]. A converter that allows for queries from other VDBs may be desirable.

Whenever events such as `NewMessage` and `PageModified` occur, the related information is sent to components that implement *interface B*. Form-based interfaces can be used to guide human users by entering information that comes from non-digital sources such as books and articles.

4.4 Interface D: Observation

Observations of security weaknesses of existing systems are reflected in various forums. Continueing our work in [**?**] we elaborated an overview of the *origins of information* that is characterized by the author of a security related contributions. Credibility, actuality, and completeness are important characteristics of an information source, examples are presented in table 1.

Type	Credibility	Actuality	Completeness
Bugtraq Message	high	high	middle
CERT Advisory	high	middle	high
Security Book	high	low	high
Vendor Mailinglist	middle	middle	middle
Hacker Web-Site	low	high	middle

Table 1: Classification of Information Sources

[2] Actually these are Mailing-list archives.

5 Related Work

In [**?**] a unifying definition of software vulnerabilities is given. Beside that, as one of the most important results the author shows, that previous classifications of vulnerabilities were ambiguous. Based on that knowledge the definition of mandatory features that are necessary for the development of classifactions led to a remarkable improvement.
As a formal approach a Vulnerability Database contains detailed data about security weaknesses or vulnerabilities. It stores and documents possible exploits and their impact on system security as well as possible ways to (temporarily or permanently) solve the problems. Additionally it holds metadata further describing the primary content and its structure. Such a vulnerability database forms a good basis for the systematic analysis of software failures.
The evaluation process tremendously benefits from the possibility to combine different information sources. A first step towards such a sharing of information was made with the development of a scheme for unified identifiers of vulnerabilities (Common Enumeration of Vulnerabilities, CVE) [**?**].

6 Summary, Conclusions and Future Directions

The digital future heavily relies on the Internet which has no appropriate security level today. There are strong efforts to change that situation - but the rule that a chain is as weak as its weakest link applies to security as well. I.e. a strong cryptographic protocol designed into a system gets more or less useless, if its implementation comprises buffer overflows or similar security weaknesses. Thus our approach involves a systematic analysis of system components and interfaces, in order to improve the understanding of the security problem and to elaborate comprehensive solutions.
In this paper we have

1. provided some reasons for the worse situation in the field of secure software,
2. pointed out the correlation of security solutions to the stages of a systems's lifecycle,
3. and introduced our concept of a closed feedback-loop for the overall software engenieering process.

The need and suitability of mechanisms and tools for describing and (semi-)automating transitions between the involved components has been shown. Based on the model our actual work concentrates on enhancing the quality of information gathering within the static parts (e.g. by means of the ongoing setup of a publically usable Distributed Vulnerability Database) and on further identifying, describing and implementing the dynamic parts, mechanisms and tools.