T U M

I N S T I T U T  F Ü R  I N F O R M A T I K

# Introducing Security Mechanisms after Initial Development: the RAC Case Study

David Bettencourt da Cruz, Bernhard Rumpe, Guido Wimmel

TUM-I0306
Juni 03

T E C H N I S C H E  U N I V E R S I T Ä T  M Ü N C H E N

# Introducing Security Mechanisms after Initial Development: the RAC Case Study

David Bettencourt da Cruz,
Bernhard Rumpe,
Guido Wimmel

Software & Systems Engineering,
Technische Universität München
85748 Munich/Garching,
Germany

This technical report describes an incremental method that allows to add security mechanisms to an existing, but insecure system, such as a prototype or a legacy system. The incremental method is presented and as a showcase its application is demonstrated at the example of a Web-based information system.

# 1 Introduction

Security is an extremely important issue in the development of distributed systems. This applies in particular to Web-based systems which communicate over an open network. Failures of security mechanisms may cause very high damage with financial and legal implications. Security concerns, both on the part of enterprises and consumers, are one of the major reasons why new technologies such as E-commerce or E-government are used very reluctantly.

Developing security-critical systems is very difficult. Security is a complex non-functional requirement affecting all parts of a system at all levels of detail. To secure a system, merely adding mechanisms such as cryptography in some places is not sufficient. Whether a system is secure depends crucially on the complex interplay of its components and the assumptions about its environment. A single weakness can  compromise the security of the entire system.

Therefore, to make it possible to assess a system's security, highly sophisticated collections of evaluation criteria have been developed, like the ITSEC security evaluation criteria [1] or their recent successor, the Common Criteria (CC) [2]. For example, for a system that is to be certified according to the CC, a comprehensive security analysis must be presented and increasingly strict requirements (depending on the chosen evaluation assurance level (EAL)) are put on the development process and its documentation. However, the CC do not give any guidance on how to fulfill these requirements during the development process.

Furthermore, many systems are developed initially without security in mind. Reasons for that are that they were designed for a secure environment such as a local network, that existing legacy systems are to be adapted, or because they were first developed as a functional prototype. Retrofitting security into an existing system is generally believed to be extremely hard to achieve, and it is in effect often advised against doing so at all. In this article we report on the experiences of a Java project where exactly this retrofitting was done after developing initial prototypes.
The RAC system is an Internet information system based on the "push" principle: information is presented to the user on a client application ("pushlet") and updated when necessary, without the user having to explicitly check for such updates. The server regularly or on demand contacts the client for updates. The RAC system was initially developed as a prototype without security functionality as its focus was targeting to be production companies' internal information systems.

In this paper, we describe a method to carry out a security analysis of an existing system and to introduce appropriate mechanisms to achieve high trustworthiness. Our method is demonstrated at the example of the RAC system. It is based on a combination of an evolutionary approach and a method suggested in [3]. We comment on experiences and difficulties in adding security to an existing system, in particular in the context of Web-based Java applications.

The paper is structured as follows. In Section 2, we introduce the RAC system. In Section 3, our method is presented. Sections 4 gives an overview over the threat analysis and determination of security requirements for the RAC system that resulted from the presented method, and Section 5 describes the implementation of the corresponding security functionality into the system. We conclude in Section 6 with a discussion of our experiences.

**Related Work**

The consideration of additional or changed requirements within the lifetime of a system is one of the main aims of iterative processes, such as Boehm's Spiral Model [7]. Few works are available on the integration of security aspects into the development process. In [3], Eckert suggests a top-down approach, which we used as a basis for our work. A mapping of ITSEC security requirements to development activities in the German V-Model 97 is given in [8]. [5] describes a lifecycle process based on the evaluation assurance requirements of the Common Criteria, at the example of a payment system. These processes are mainly tailored to the development of new systems. Security aspects of distributed Java applications are covered in detail in [9], but methodical guidance is missing there.

## 2 The Web-based Information System RAC

The RAC system is an experimental prototype serving a variety of issues. On the one hand it is an innovative concept that allows to study issues of component based software, distribution, distributed development, independent deployment and security. On the other hand, RAC was designed to address and improve some of the main disadvantages of common browser-based information retrieval systems. The RAC system is designed to serve the following purposes:

1. It is an information retrieval system that updates its information automatically.

2. The information presented is up to date. It may change in seconds only, such as in stock information systems.

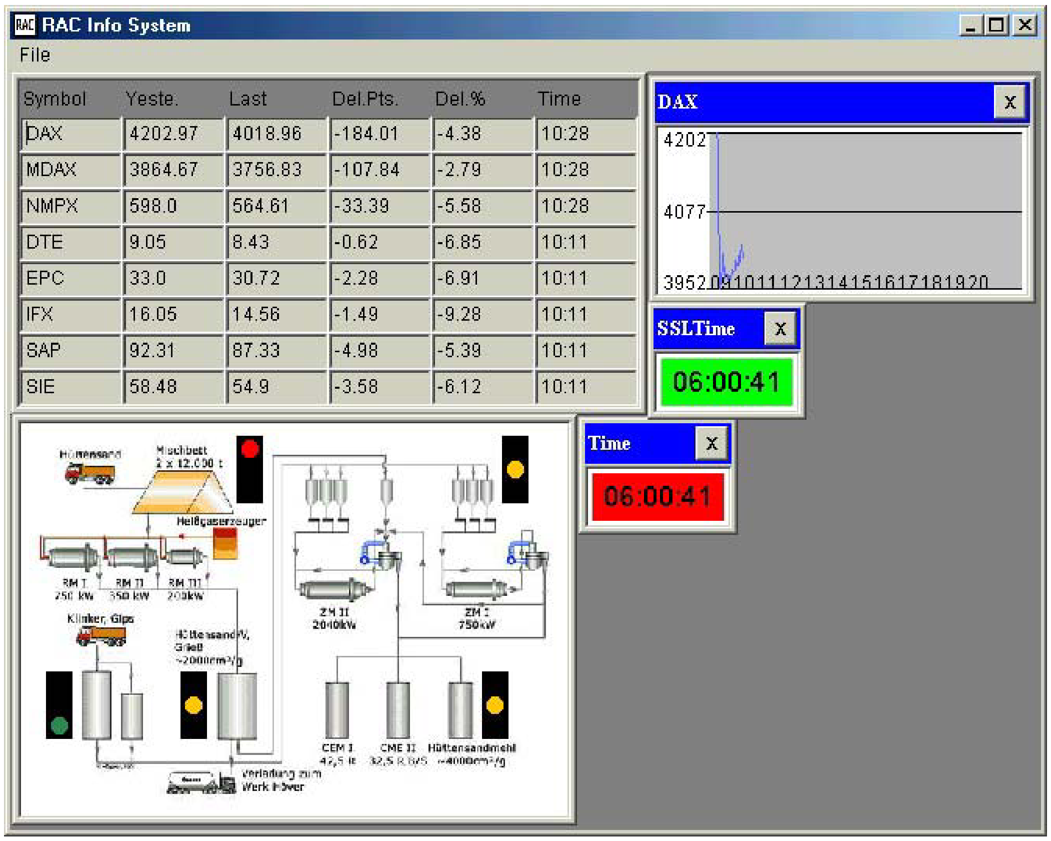
3. The system thus has an efficient communication mechanism, which includes that it can be used over low-bandwidth communication lines (telephone) as well as high-bandwidth ones.

4. The RAC client offers a multi-view, in the sense that it shows several different pieces of information at the same time. This implies that the display concentrates on information presentation without overhead, such as banners.

5. The RAC system is configurable: so-called "pushlet" components can be downloaded and added to the client's view dynamically.

6. The pushlets can be combined from different sources in the net.

Figure 1 shows a typical layout of the RAC client with several pushlets being active while there is still space to add more pushlets.



| Symbol | Yeste. | Last | Del.Pts. | Del.% | Time |
|---|---|---|---|---|---|
| DAX | 4202.97 | 4018.96 | -184.01 | -4.38 | 10:28 |
| MDAX | 3864.67 | 3756.83 | -107.84 | -2.79 | 10:28 |
| NMPX | 598.0 | 564.61 | -33.39 | -5.58 | 10:28 |
| DTE | 9.05 | 8.43 | -0.62 | -6.85 | 10:11 |
| EPC | 33.0 | 30.72 | -2.28 | -6.91 | 10:11 |
| IFX | 16.05 | 14.56 | -1.49 | -9.28 | 10:11 |
| SAP | 92.31 | 87.33 | -4.98 | -5.39 | 10:11 |
| SIE | 58.48 | 54.9 | -3.58 | -6.12 | 10:11 |

**Figure 1.** RAC Client with Pushlets

The RAC system addresses a number of disadvantages identified by traditional Web-based information systems:

1. If the users wanted accurate and up to date information in traditional Web-based information systems, they would have to constantly reload the Web page, unless the page itself used a polling mechanism forcing it to reload periodically. This is particularly annoying, as it generates a lot of traffic. Furthermore, such a page is created by the server and thus prevents multi-view.

2. Since http is a stateless protocol, session information could only be stored by using cookies or encoding session-ids in the http address. Both were considered to be unsatisfying solutions.
3. Most of the information coming from the Web is overloaded with irrelevant information like ads or unnecessarily detailed information. If a user wanted to

follow information from several different pages he would constantly have to switch between browsers, because all the information would never fit on the screen.

The RAC-server is designed to push information to its clients whenever new information is available as shown in Figure 2. This ensures up to date information and reduces traffic considerably.

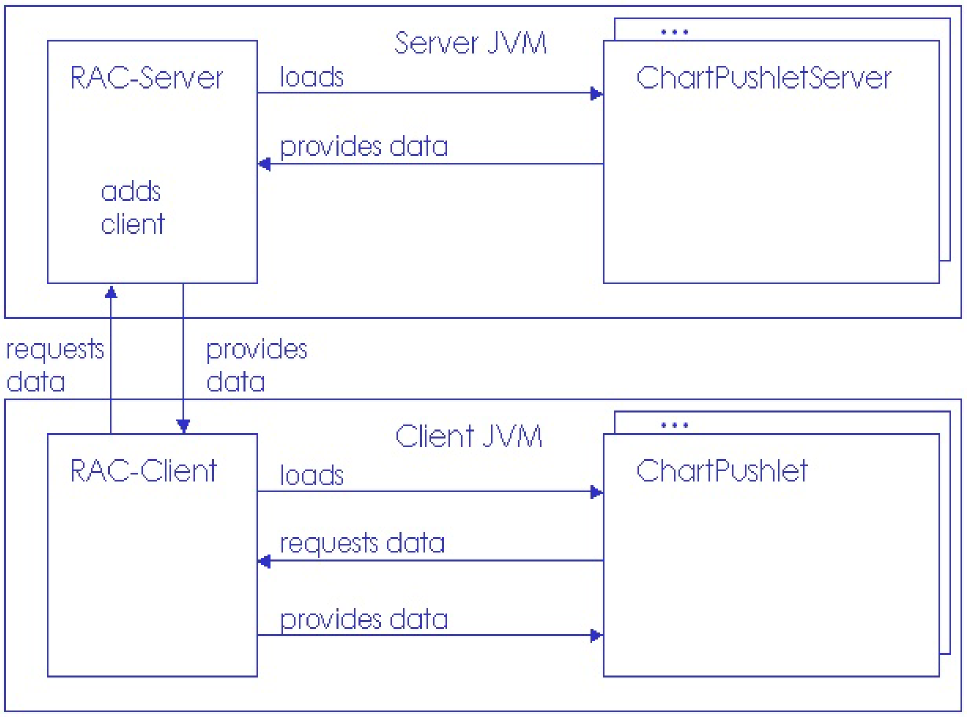


**Figure 2.** The Overall RAC Architecture

The RAC-client basically consists of an empty window and the ability to dynamically load pushlet components from local filesystems and through the Web. A pushlet started by the user presents its information in a small window within the main window. All of the control information (reload, register, unregister, …) is handled uniformly within the main system. On the one hand this allows pushlets to become very small components that can easily be loaded even through low bandwidth communication lines. On the other hand the uniform presentation allows a condensed presentation of valuable information and suppression of clutter. The pushlet presentation can be resized, dynamically added and removed and its parameters can be changed.

One client may connect to several servers. Servers provide the information shown by the pushlet and the pushlet code itself. When a server is started, it loads its available PushletServer-objects. The RAC-client then has to register its interest at the server in one or several information categories. As soon as the client is registered, it will automatically be provided with all the data that the pushlet server of interest generates, whenever new or changed information is available. The client then passes the data to the corresponding pushlet, which is responsible for its graphical representation. Since the connection between the registered client and the server is kept alive, there is no difficulty in tracing the session and storing session information.

## 3 Method for Introducing Security Features

Early in the development, it was decided that the RAC system is to be developed in increments (actually using an Extreme Programming like approach [11,12] that was inspired from a survey [16] indicating a number of enhancements). Also based on our experiences in building similar systems [13,14,15] it was decided to start with several increments to build an efficient feasibility prototype without any security mechanisms. Instead, any security considerations should be retrofitted into the existing system in a later increment. We were well aware that this might make it necessary to refactor parts of the code. The rest of the section sketches the main steps of the iteration that add security to the existing insecure system, which will then be illustrated in detail in the following sections.

The method used for adding security to the RAC system was based on existing methodologies for developing new security-critical systems ([2], [3], [5]). It consists of the following steps:

- threat analysis,
- threat classification,
- finding countermeasures,
- countermeasure classification,
- combining classification results.

During the *threat analysis* each and every possible threat to the system has to be documented (threats are situations or events that may lead to unauthorized access, destruction, disclosure or modification of data, or to denial of service). Obviously, only threats that have been identified at this stage can later on be considered to be countered. Therefore, it is important that the threat analysis is as complete as possible. Hence it is crucial to use a systematic approach to identify the threats. In case of the RAC-System, threat trees [10] were used. The root of a threat tree consists of all possible threats to a system. Its successor nodes correspond to more fine-grained threat classes (which together make up all possible threats), and the leafs consist of single threats or very small related groups of threats. There is some degree of freedom in how the threats are decomposed, as long as no threats are lost during the process. In a state-driven program, for example, the tree can first be divided into the different states the program might reach. Each of those states can then be further analyzed. In a distributed environment, a refinement into subtrees depending on the physical distribution (e.g. threats against system components and threats against communication links) can be appropriate.

After having completed the search for the threats, the resulting *threats are classified* (1) by an estimate of the potential *damage* caused if the threat can be realized, and (2) by an estimate of the *effort* it would take an attacker to realize the threat. To allow for a systematic risk analysis, damage and effort are measured in a quantitative metrics, depending on the use of the program. In programs with a commercial use, money is mostly a good scale to choose. Other appropriate metrics are necessary time, hardware or knowledge which can in turn again be represented by money.

The next step to take is to *find countermeasures* against each of the identified threats independently of how they have been classified. Fortunately, quite a number of standard security techniques, patterns and concepts [3,4] that provide countermeasures against most of the major threats already exist. At this step the countermeasures found against the threats do not yet have to be worked out in great detail. Providing a basic idea is usually enough to be able to classify them in the next step.

The *classification of the countermeasures* is carried out in a way similar to the way used in threat analysis. For this classification, the effort needed to realize a countermeasure is estimated. Again the classification is based on a metrics and thus allows comparison. Ideally the metrics used is the same as the one used for threat classification, since that would ease the following combination of all classifications.

Finally the results from these *classifications are combined* to identify which threats should be dealt with. This largely depends on the available budget and the level of security that must be reached. This procedure will result in the functional security requirements, which then can be used as if they were traditional requirements and implemented. As a specialty, we found that these so called “requirements” usually go deep into design activities and thus combine analysis and design phase. In this respect, we deviate from the strictly sequential approach suggested in [3]. Postponing the decision which threats should be countered to the point after the assessment of corresponding countermeasures adds a little overhead. But it is easier in our case as there is already an implemented system available and it leads to a more effective selection of security measures within the given time/budget.

## 4 Threat Analysis and Security Requirements

Since RAC is a rather classic client-server system, the threat tree was first divided into the components Client, Server and Internet. The threats that affect the server are divided into three categories (see Figure 3). The categories are threats originating from RAC itself, threats originating from the pushlet servers, and threats that originate from outside the RAC system.

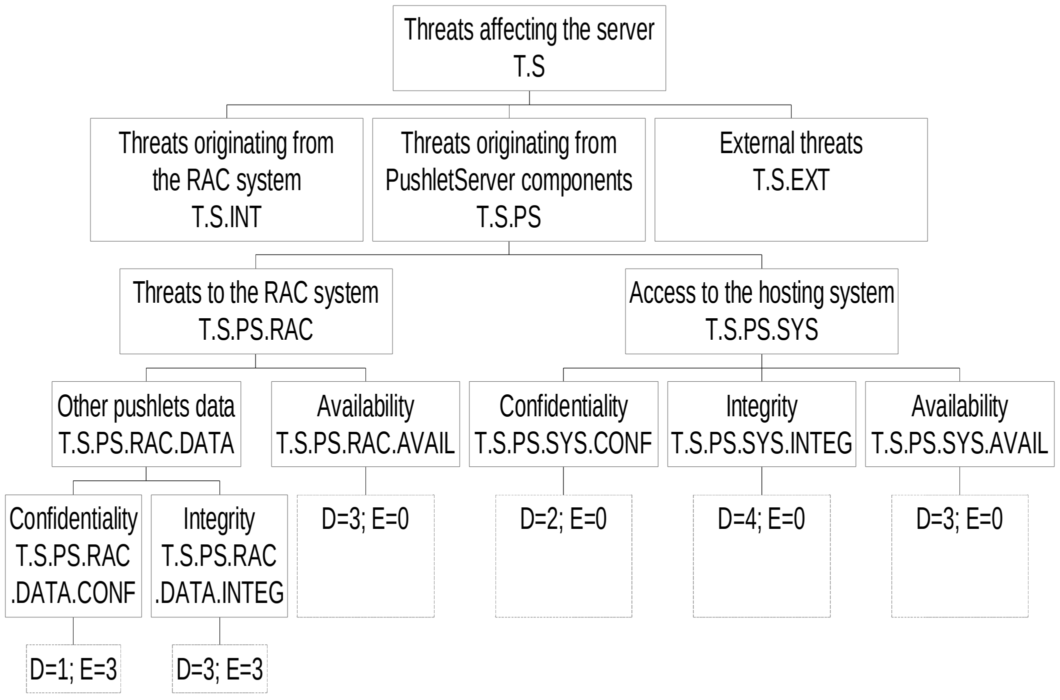

Threats affecting the server T.S
Threats originating from the RAC system T.S.INT
Threats originating from PushletServer components T.S.PS
External threats T.S.EXT
Threats to the RAC system T.S.PS.RAC
Access to the hosting system T.S.PS.SYS
Other pushlets data T.S.PS.RAC.DATA
Availability T.S.PS.RAC.AVAIL
Confidentiality T.S.PS.SYS.CONF
Integrity T.S.PS.SYS.INTEG
Availability T.S.PS.SYS.AVAIL
Confidentiality T.S.PS.RAC .DATA.CONF
Integrity T.S.PS.RAC .DATA.INTEG
D=3; E=0
D=2; E=0
D=4; E=0
D=3; E=0
D=1; E=3
D=3; E=3

**Figure 3.** Server Threat Tree with Threat Classification

Many threats originating from outside the RAC-system are similar as in other Web-based systems and can be countered by well-known security patterns such as the use of a firewall. However, threats coming from the pushlet servers may be a serious problem, since pushlet servers are dynamically loaded and can contain third-party code. For sake of space, we therefore focus on the threats that originate from the PushletServer objects (T.S.PS). These threats were further divided into threats to the hosting system (T.S.PS.SYS) and threats to the RAC-system itself (T.S.PS.RAC).

The threats affecting the RAC-system are either threats to the confidentiality and integrity of the data belonging to other pushlets (T.S.PS.RAC.DATA.CONF, .INTEG) or to the availability of the whole system (T.S.PS.RAC.AVAIL). The confidentiality (CONF), availability (AVAIL) and integrity (INTEG) of the hosting system could also be corrupted by the PushletServer objects.

Table 1 shows how the effort for an attacker to realize a threat is classified. Accordingly, Table 2 classifies the damage caused by the threats by extra costs caused by the damage. Both tables are grouped into six classes {0..5}.

Since the RAC system was initially developed neglecting any security issues, the effort an attacker needs to accomplish threats T.S.PS.SYS and T.S.PS.RAC.AVAIL is nearly zero. Nevertheless the effort for the threats affecting the confidentiality and integrity of the other pushlets' data (T.S.PS.RAC.DATA.CONF, .INTEG) is not zero, because Java does not allow direct access to memory and therefore protects the data of other pushlets (for detailed information on Java security, see [9]).

**Table 1.** Classification Scheme of Attacking Effort

| Rating | Time needed | Money needed | Skills needed |
|---|---|---|---|
| **0** | < 1 Hour | < 100 EUR | Basic IT skills |
| **1** | < 1 Day | < 500 EUR | Basic knowledge of the RAC system |
| **2** | < 1 Week | < 1.000 EUR | Average knowledge of the RAC system |
| **3** | < 1 Month | < 5.000 EUR | Good IT skills or good knowledge of the RAC system |
| **4** | < 6 Months | < 10.000 EUR | Very good IT skills or very good knowledge of the RAC system |
| **5** | > 6 Months | > 10.000 EUR | Excellent IT skills or excellent knowledge of the RAC system |

**Table 2.** Classification Scheme of Damage

| **Rating** | **Extra costs through damage** |
|---|---|
| **0** | 0 EUR |
| **1** | < 500 EUR |
| **2** | < 1.000 EUR |
| **3** | < 10.000 EUR |
| **4** | < 100.000 EUR |
| **5** | > 100.000 EUR |

Some knowledge of RAC is needed to access that data, thus the effort was given class three. The damages caused by a denial of service attack to both RAC and the hosting system (T.S.PS.RAC.AVAIL, T.S.PS.SYS.AVAIL) are equal, since it is assumed that the hosting system is only used to run RAC. The evaluation of three (critical) is given, because a denial of service-attack can result in high costs. If the integrity of the system (T.S.PS.SYS.INTEG) is attacked, the costs can become very critical (four). If the operating system does not provide sufficient protection it may even be possible to delete a whole database. On the other hand, the confidentiality of the system (T.S.PS.SYS.CONF) is not as costly if compromised, as the system only hosts the RAC program. Of course this value must be adapted if RAC is used to run pushlets which store important data on the server side. Compromising the confidentiality of other pushlets' data (T.S.PS.RAC.DATA.CONF) is regarded as harmless, whereas compromising the integrity (T.S.PS.RAC.DATA.INTEG) may already be critical, since changing pushlet data may result in the pushlet not working anymore or even worse result in the pushlet displaying incorrect information.

As a next step, countermeasures were provided for all identified threats. The classification of the countermeasures is again measured primarily against a money metrics. Two other important metrics were the time needed to implement such a countermeasure and the difference the implementation would make to the performance of RAC. Table 3 shows the values used to classify the countermeasures.

**Table 3.** Classification scheme for countermeasures

| Rating | Costs | Time needed | Difference in performance |
|---|---|---|---|
| **0** | < 100 EUR | < 1 Day | No difference |
| **1** | < 500 EUR | < 2 Days | Almost no difference |
| **2** | < 1.000 EUR | < 1 Week | Small difference |
| **3** | < 10.000 EUR | < 1 Month | Bearable difference |
| **4** | < 100.000 EUR | < 1 Year | Great difference |
| **5** | > 100.000 EUR | > 1 Year | Unbearable difference |

A Java Sandbox is used as countermeasure against the threats T.S.PS.SYS.CONF and T.S.PS.SYS.INTEG to prevent any access to the hosting system. The Java Sandbox is an access control mechanism which is quite easy to introduce as long as the program is kept modular. Through good design this was the case in the RAC system. As RAC was designed in a very modular way, this solution was classified as relatively inexpensive with the value three.

The threats T.S.PS.RAC.AVAIL and T.S.PS.SYS.AVAIL can be countered by using a good scheduling system, which would have to control the access to the system resources. This scheduling system has to be built for the RAC system specifically, which results in high developing costs (value: four). Providing a countermeasure against the threats T.S.PS.RAC.CONF and T.S.PS.RAC.INTEG requires a detailed revision of the access groups (public, protected, package and private), eventually a restructuring of the object hierarchy and possibly an adaptation of RAC objects in the Java Sandbox. The costs for such a solution would again be relatively high compared to the existing resources.

The steps described above for server threats were applied to all of the threats identified for the RAC-system. The result was a table with values of damage, effort and solution effort for each threat to the RAC system. These values were combined to find the security requirements and the priority of these requirements. In the combination of the damage, effort and solution effort classification, a priorization according to the actual project situation (tight iteration schedule, limited resources, but more iterations to come) was used to select a subset of countermeasures for the implementation in that iteration.

# 5 Implementation of Security Mechanisms

The evaluation of the results from all previous classification steps led to the conclusion that for the given circumstances there were three countermeasures to threats that were most important to be implemented given the tight time/budget restrictions:

- To integrate a Java Sandbox into the RAC server, to isolate the PushletServer from the hosting system.

- To integrate a Java Sandbox into the RAC client, to analogously isolate the Pushlets from the client system.

- To introduce a mechanism to secure the communication between the RAC client and the RAC server. The mechanism chosen was to introduce the possibility of using an SSL secured channel for communication.

All three solutions have been added to the RAC system within one iteration. They can easily be configured as needed. The Java Sandboxes were configured following the “need-to-know”-principle [17]. This means that every component is only allowed access to the resources it needs to operate properly. Changes to these restrictions can be made at any time within the policy files that control the Java Sandbox. Whether the communication between the RAC-server and the RAC-client is SSL-secured or not depends on the method call within the pushlet (see Figure 4).

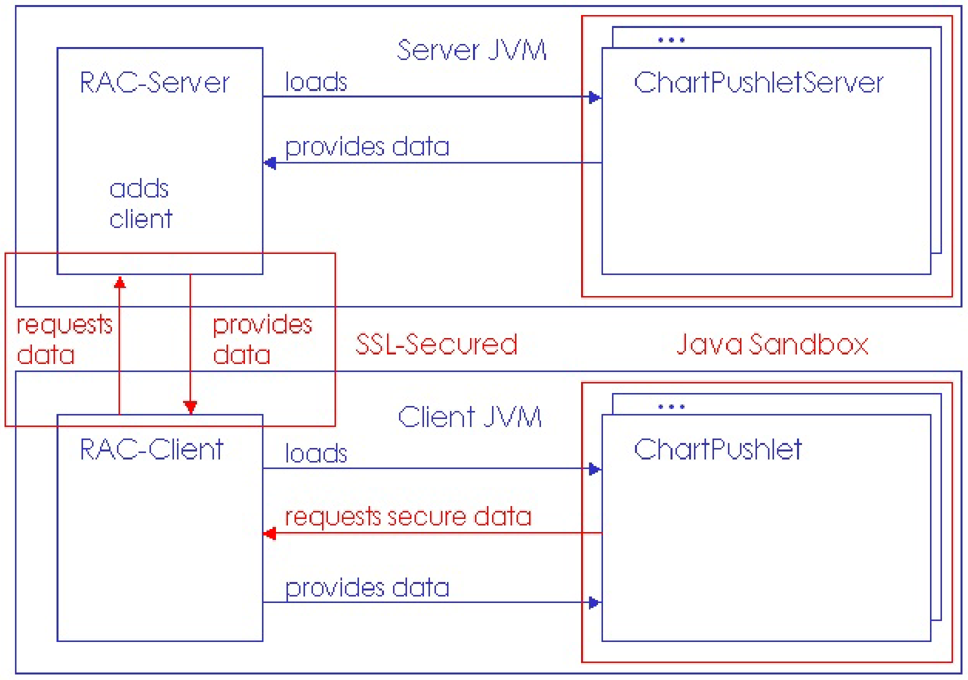


**Figure 4.** Implemented Security Components

The reason for this is that cryptographic details should be hidden from the user, yet not all pushlet data needs to be encrypted. Hence the developer of the pushlet has to decide whether the data being exchanged should be protected from outside access.

Since automated tests were already available from prior iterations, these tests were used as regression tests to certify that the functionality of the program had not been affected by the new security components. Additional tests had the purpose of verifying that the security mechanisms had been implemented properly and actually serve to counter the identified threats. Without going deep into testing details, we want to note that e.g. tests cannot only serve to check whether functionality still works, but also whether certain threats are indeed tackled. E.g. a test pushlet can try to surpass a sandbox, or a communication can be subjected to a man in the middle attack. These kinds of tests greatly improve the confidence in the proper implementation of the measures taken.

## 6 Discussion

In this paper, we described a method to retrofit security into an existing system. We presented the Web-based information system RAC, which was implemented in an incremental way using a Java architecture, without at first considering security aspects. We demonstrated our approach by showing how we integrated security features into the RAC system.

This part of the approach is based on a standard, step-wise process primarily designed for implementing new security-critical systems. It turned out that the initial steps, the threat analysis and classification, could be taken over fairly directly. They were even facilitated considerably by the fact that a running version of the system was already available and there was no uncertainness about its functionality. Available automated tests also greatly improved confidence that the changes made have been correct.

Specifying and assessing countermeasures is more difficult, as the design of the existing system must be taken into account. If the countermeasure against a threat cannot be assigned to a small, modular or easily separable part of the program, it becomes much harder to retrofit. Choosing a “good” architecture (which does not necessarily have to be security specific) can even serve to counter some of the threats without additional implementation effort, as it is the case for modular Java applications with the built-in access control mechanisms and Java’s protection mechanisms against implementation-level attacks such as buffer overflow.

In our approach, the choice of the countermeasures that should be implemented is based on both an assessment of the threats and of the countermeasures. This was only possible with reasonable time and effort because an implementation and automated tests were already available. Based on this information, the most effective security functions given particular time/budget constraints for their implementation can be selected. The existing tests could be re-used to verify that the program’s functionality has not been affected, and additional tests for the security functionality could be added. Finally, it is important that the existing software is well documented. Otherwise the threat analysis becomes hard to carry out.

We believe that Java is to date the most appropriate architecture for information systems where security might become important after some iterations, because of its built-in security features. However, there are also problems. Firstly, the Java

Sandbox does not implement the principle of complete mediation [17]. It usually only checks for correct access when a protected object is created. Further use of the object works independently of the Sandbox, thus enabling attackers to gain access to the object through the program. Secondly, if the program is not strictly modularized, the Sandbox becomes much harder to introduce, and restructuring might become necessary to be able to use the access modifiers (private, protected, public).

If the mentioned points are considered, a certain level of security can be actually retrofitted into an existing software without great overhead.

## Acknowledgements

We would like to thank Roger Rosette, Ansgar Haase, Christopher Lewis, Alexandra Remptke and Martin Winter for helping to develop the RAC system. This work was partially supported by the Bayerisches Staatsministerium für Wissenschaft, Forschung und Kunst and through the Bavarian Habilitation Fellowship and the German Bundesministerium für Bildung und Forschung through the Virtual Software Engineering Competence Center (ViSEK).

## References

[1] ITSEC. Information Technology Security Evaluation Criteria – Harmonised Criteria of France, Germany, the Netherlands, the United Kingom, May 1990. Version 1.

[2] Common Criteria for Information Technology Security Evaluation, Version 2.1. Technical report, 1999. URL: http://www.commoncriteria.org/docs/index.html

[3] C. Eckert. IT-Sicherheit (in German). Oldenbourg Verlag, 2003.

[4] R. Anderson: Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2001.

[5] Monika Vetterling, Guido Wimmel, Alexander Wißpeintner. Secure Systems Development Based on the Common Criteria. 10th International Symposium on the Foundations of Software Engineering (FSE-10), 2002.

[6] IABG. V-Modell 97, 1999. URL: http://www.v-modell.iabg.de/

[7] B. Boehm. A Spiral Model of Software Development and Enhancement. IEEE Computer, vol. 21, #5, May 1988.

[8] IABG. SEC: Using the V-Model and the ITSEC. Part 3 of [6].

[9] Li Gong. Inside Java 2 Platform Security: Architecture, API Design, and Implementation. Addison-Wesley, 1999.

[10] E. G. Amoroso. Fundamentals of Computer Security Technology. Prentice Hall, 1994.

[11] K. Beck. Extreme Programming Explained. Addison-Wesley, 1999.

[12] B. Rumpe. Extreme Programming – Back to Basics? In: Proceedings of Modellierung 2001, 28.-30.3.2001 Bad Lippspringe. G. Engels, A. Oberweis, A. Zündorf (eds.). Lecture Notes in Informatics, Band 1, GI-Edition, Bonn. 2001.

[13] B. Rumpe. Online Auctions (lessons learned from strategic E-Business consulting). In: Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. Idea Group Publishing, Hershey, London, pp. 682-686. 2002.

[14] M. Fontoura, W. Pree, B. Rumpe: The WebShop E-Commerce Framework. International Conference on Internet Computing. June 25th - 28th, Nevada, USA. CSREA Press, 2001.

[15] B. Rumpe, G. Wimmel. A Framework for Realtime Online Auctions. In: Managing Information Technology in a Global Economy - Proceedings of IRMA International Conference, Toronto. Idea Group Publishing, Hershey, London, 2001.

[16] B. Rumpe, A. Schröder. Quantitative Survey on Extreme Programming Projects. In: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002, May 26-30, Alghero, Italy, 2002.

[17] J.H. Saltzer, M.D. Schroeder. The Protection of Information in Computer Systems. Proceedings of the IEEE 63, 9, 1975.