

MAVA – Modular Security for Distributed Applications

Peer Rexroth and Alex Wiesmaier
Department of Cryptography and Computeralgebra
Technische Universität Darmstadt
Hochschulstr. 10
D-64283 Darmstadt, Germany

July 22, 2007

Abstract

This paper introduces the MAVA system, a framework for securing distributed applications.

MAVA is comprised of two components: a security component and a proxy component. The security component has the purpose of securing distributed applications, whereas the proxy component enables an easy integration of the MAVA system into existing applications. However, the modules which are introduced in this paper are not the only possible implementation solution and can be replaced by other self-developed modules.

The security modules can be extended by sub configurations. Sub configurations are special modules that can be used to enhance the security of security modules. The MAVA system provides the sub configurations Challenge, CAPTCHA and Shredding. An implementation of other sub configurations is furthermore possible.

The proxy component is set up between the client and the application. It analyzes the traffic between client and server and is able to act accordingly this way. The proxy component consists of the modules CommandHandler and Scrambler. It is important to mention that these modules, again, are not the only possible implementations, but can be extended or even replaced by various other modules.

1 Introduction

First of all, we want to mention two important sources. The first source is 'IT Sicherheit' by Claudia Eckert [Cla06]. This book introduces many attacks which affect distributed systems as well. It also contributes to the general understanding of authentication. The second source is 'Outflanking and Securely Using the PIN/TAN-System' by Wiesmaier et al [WKL05]. This paper especially goes into the matter of PIN-TAN-based systems. These systems are important distributed applications which we use for presenting the amenity of our MAVA system.

In order to understand the functions and the purpose of the MAVA system, as presented in this paper, one firstly needs to define the terminology used.

MAVA is a system for securing distributed applications modularly. In the following we shortly describe what a distributed application is and list various application areas

where distributed applications can be used. Subsequently, we identify the problems of distributed applications and describe methods which can be employed to resolve these issues.

A distributed application is an application made up of distinct components which are usually physically located on different computer systems, connected by a network. Nowadays, many examples for distributed applications can be found. Some of the most popular distributed applications are: Online Transaction Processing (OLTP) or Groupware applications, such as Lotus Notes. Online reservations in cinemas are an example of OLTP. This type of system has a central system and machines which are connected to the central system from different geographically locations. Groupware applications help people work together collectively, while located remotely from each other. More information about groupware can be found at [SV97].

OLTP and groupware applications have in common that they are based on the wide spread client/server architecture. The client/server architecture is a network architecture with the goal of distributing applications to numerous computers. Some of these computers are clients, sending requests, whilst one computer usually acts as the server, handles these requests and, if necessary, returns a result. Online banking is an important distributed application, used in this paper as an example for a general distributed application. We have chosen online banking due to its widespread usage, and most people will readily understand the need for high security.

In principle, the security problems which affect online banking apply to other distributed applications as well [Hoy02]. When dealing with security, two terms are essential: Authentication and Authorization. Gharadaghy [Gha07] discusses these terms in detail in the paper 'Authentication/Authorization'. The definition offered by Gharadaghy can be summarized as follows: Authentication is the act of establishing or confirming something or someone as is as they say they are. Authentication of a person often consists of verifying their identity. In computer security, authentication is the process of attempting to verify the identity of the sender of a communication such as a request to log in.

The next term is authorization, which is closely related to authentication. Authorization is the process of verifying that a known person is legitimated to perform a certain operation. Therefore authentication must precede authorization. Authorization can not occur without authentication. If one considers many types of distributed applications, it becomes apparent that authentication and authorization are nowadays often only handled superficially. In the case of some online banking variations, and indeed in almost all other distributed applications, the first step in using the application requires the user to enter their id and password. For most distributed applications this is all that is required for the complete authentication of the user. From thereon the user is authorized to use the application without any constraints. For very sensitive transactions, for example transferring money within online banking, additional methods to authorize the user are used. An example of this is the Transaction Authentication Number (TAN) [Inf02].

The TAN operation reveals the next problem of distributed applications. At the beginning of online banking it was common to demand a simple TAN only, which was one of the TANs that had been sent to the user, usually by post. Over the years more and more criminals tried to trick the user and obtain their TANs [Lam05]. Only recently have some banks changed to the indicated transaction authentication number (iTAN) operation to hamper the use of stolen TANs. Within the iTAN operation the TANs are enumerated and the system demands a particular, randomly selected TAN. iTAN has been the victim of several successful attacks [Ver05].

The problem, which can be exemplarily seen by the TAN operation, is that threats develop with time. Therefore it is imperative for distributed applications to grow with the threats and always be one step ahead of attackers and scammers.

For many applications this would require fundamental changes to the existing system with corresponding high costs.

In the following we present how the MAVA system attempts to resolve the aforementioned problems, before we explore the way it interacts with existing applications and the fundamental advantages - modularity and flexibility – in the next section.

2 The MAVA System

As a basic principle, distributed applications operate on two different layers [Hub80]. The first layer acts as presentation, while the second layer enables the user to input the data. Both mechanisms need to be secured by appropriate security precautions.

The MAVA System can be applied to secure distributed applications that communicate with different types of protocols. It is attached as a component that can easily be integrated into existing applications. Furthermore, it is possible to realize the belated integration of the security features with minimal modification to the existing application.

This is done by incorporating a proxy server, instead of integrating the component itself into the application. All data is buffered and not forwarded until the configured security mechanisms complete successfully.

A variety of protocol types can generally be secured this way. A common protocol used by distributed applications is the Hypertext Transfer Protocol (HTTP). The protocol is specified in RFC1945 [TBL96] and RFC2068 [RF97].

Securing a HTTP-based web-application is a possible application area of the MAVA System. HTTP-based web applications are platform independent.

The example of the implementation of the proxy server demonstrates the protection of a web application. Figure 1 illustrates the communication between a client and a web server.

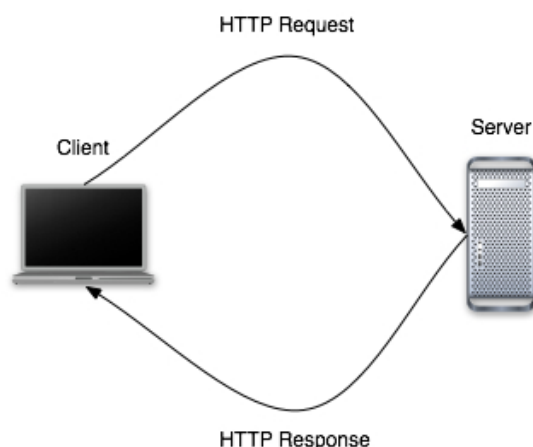


Figure 1: HTTP Request/Response

When considering HTTP it is important to recognize that the client always initiates the connection, requests data via an HTTP-request and receives a response from the server via an HTTP-response. The server responds by providing a form in case more data input from the client is required. The user then completes the form which leads to the data being sent to the server as an HTTP-request. Finally, the client receives the requested data as response in the same way as a standard information request.

HTTP-based protocols are liable to a variety of attacks, especially skimming and data manipulation [Hoy02]. Extensions and mechanisms like the SSL [T. 99] protocol have been developed to solve such problems.

However, time has shown that new attacks are being permanently developed and formerly unknown security holes are found, creating the necessity to constantly develop new counter measures. A belated adjustment to increase the security can incur a large expense when the new security mechanisms have to be implemented directly into the distributed applications. Additionally, a vast amount of distributed applications are very complex so that improvements can only be implemented very slowly.

The MAVA System provides security mechanisms to enhance the security between communication partners in the face of authentication, privacy, integrity and obligation without the need for a drastic modification of the existing system. Furthermore, every security mechanism is addressed by a consistent interface so that the mechanisms can be considered as single modules. These modules can be added to or removed from the web application without any major modification.

There are two ways to integrate the MAVA System, illustrated in Figure 2 and Figure 3 to secure web applications. The web application (1) can be considered as a blackbox in this process, primarily considered in terms of its input and output characteristics. The web server (2) provides the application server's (3) functions via the HTTP protocol. Different protocols could however be used at this point in a similar way. An arbitrarily distributed application could be secured by changing the web server and if necessary the proxy component.

The application has access to a data store (4) storing user- and application-based data. The security component requires an additional data store which can be set up on the same or on a different database server without functional consequences.

At the beginning, the client (5) establishes the connection with the proxy, pictured in Figure 2. The proxy then contacts the application and the security system (7) and communicates the obtained data or security requests to the client.

The proxy server appears as a web application to the client, which consequently has no knowledge of what can be found behind the proxy.

The advantage of this is that the security system can be integrated into an already existing application more easily, the reason being that the proxy server reacts to commands within the data transferred by the application. The security component does therefore not need to be integrated directly. Moreover, protocol layer based security functions can be implemented at this point. The renaming of variable names into random strings can be viewed as an example for this, resulting in potential hampering attackers being able to draw conclusions from analyzing the data stream.

The second possibility is illustrated in Figure 3. The client (5) has direct access to the web application (1) which again has direct access to the security component (7). The

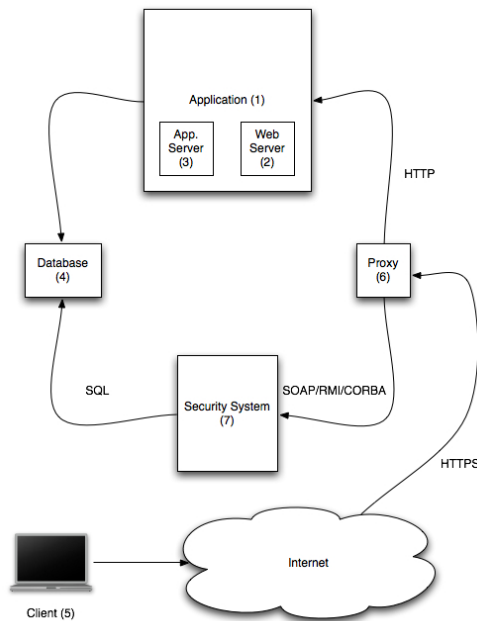


Figure 2: Access via proxy: Client → Proxy → Application

provided security functions are straight addressed from the application. At this stage, various types of protocols can be used to integrate the security component. This alternative, however, requires a modification to the existing web application.

The security system is made up of two stand-alone components: the proxy and the security component. The security component is responsible for the security functions and has the potential of also being used independently from the proxy component. The proxy component eases the integration of the security component into an existing application by analyzing the data between the communication partners and by inter-connecting the defined security mechanisms of the security component. We take now a closer look at the two components and their functionality.

3 The security component

The security component is made up of different parts. Whereas user and session management are handled in fixed classes, every security mechanism is realized by implementing the SecurityModule interface. This interface provides a high degree of flexibility because it is possible to develop new security modules or remove existing modules any time, without modifying the application. The input of user data can be done in different ways, as merely an implementation of the LoginModule interface is necessary. The LoginModule is responsible for data collection.

In our example, the single modules return forms which are made up of HTML fragments which can either be included directly into the HTML documents or be transformed for the secured application. Other data structures for data propagation are also imaginable. One possibility is the XML data structure, which can be implemented in-

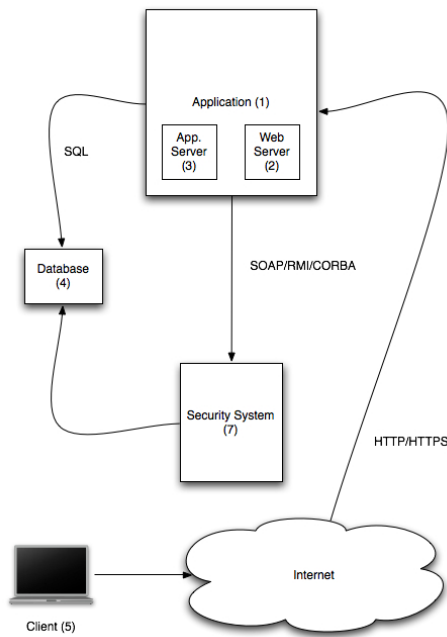


Figure 3: Direct access: Client → Application

dividually for the respective secured application.

Single system properties can be administrated in a configuration file, which is also read by the security modules, while session and user management are accomplished in the database. The following illustrates the procedure of the security mechanism integration by means of the HTTP protocol example. In principle three security levels are distinguished:

1. Publicly accessible sites without access protection. For this case no security mechanisms are necessary.
2. The access to simple secured pages or transactions takes place after a successful login of an authorized and authenticated user. This is mostly employed for simple data, such as credit balances in an online banking system.
3. Single pages or transactions can be secured by additional security mechanisms. These are used to, for example, securing input data for credit transfers.

It is possible to define other security levels and integrate these into the system. The real security mechanisms are provided by single reloadable modules and can be activated by the user separately, loading after the input of a user id and engaging with the authorization mechanism at four entry points.

- LOGIN: Requests verifying the authenticity after the user id was entered.
- PRETRANS: Requests handled before the transaction form is shown.
- POSTTRANS: Requests executed before and after showing the transaction form.

- LOGOUT: Requests necessary for a correct termination of the application, in order to ensure that transactions entered during the session are executed after a successful logout at the latest, depending on the situation

By using a configuration front-end, it is possible to choose from a list of available security modules desired by the user. It is furthermore possible to adjust the sub configurations of every module. Every single module generates the necessary data for authorization and stores it in individual files, which then need to be given to the user, in order to ensure the essential security. Some modules require external hardware which has to be provided pre-configured as well.

Table 1 illustrates a hypothetical protocol run for the online banking example. The first column describes the online banking progress. Column two and four describe the actions of the client and the bank. The third column describes the requests and responses sent between the client and the bank.

	Client		Bank
1		→ Request: Online banking	
2		← Post: Modul data	
3		→ Browser: www.bank.de	
4		← Server: Login mask	
5		→ Browser: User authentication	Verification, load security modules
6	If necessary, verification	← LOGIN entry point	
7		→ Browser: Requested authorization data	Verification, if necessary return to point 6 for other modules, which use LOGIN as entry point
8		← Server: User area	
9		→ Browser: User account balance?	
10		← Server: User account balance	
11		→ Browser: Credit transfer	
12	If necessary, verification	← Server: PRETRANS entry point	
13		→ Browser: Requested authorization data	Verification, if necessary return to point 12 for other modules, which use PRETRANS as entry point
14		← Server: Credit transfer mask	
15		→ Browser: Credit transfer data	Intermediate storage
16	If necessary, verification	← Server: POSTTRANS entry point	

17		→ Browser: Requested authorization data	Verification, if necessary return to point 16 for other modules, which use POST-TRANS as entry point
18		← Server: Credit transfer OK, (ACK)	Commit credit transfer
19		→ Browser: Logout	
20	If necessary, verification	← Server: LOGOUT entry point	
21		→ Browser: Requested authorization data	Verification, if necessary return to point 20 for other modules, which use LOGOUT as entry point
22		← Server: Logout OK, (ACK)	

Table 1: Hypothetical protocol run: Online Banking

3.1 Sub configuration

Sub configurations are special modules which can be combined with other security modules to enhance the protection provided. A very high degree of security is possible by combining several sub configurations. In principle, all sub configurations can be used at the same time.

The most familiar sub configuration is Challenge. Today, Challenge is often used in combination with PIN/TAN based systems. In the following we present the sub configurations Challenge, CAPTCHA and Shredding. The development and implementation of further sub configurations is possible. The integration of these via the MAVASystem is as easy as the integration of other security modules.

3.1.1 Challenge

Challenge operation occurs when a server generates a challenge which has to be solved by the user for a successful authentication/authorization. For this purpose, the server can use a scheme to bind the authentication/authorization data to the current transaction and vice-versa.

The server can use arbitrary information to determine a challenge. Examples of this are: Current date and time, last used secret, user transferred data, random numbers and current session ID. The challenge is transferred to the user without a clue as to the answer.

A simple implementation of Challenge is often used by banks in combination with the PIN/TAN operation. This method is known as 'iTAN' (indicated TAN). Indeed this method represents only a very special and simple case of the Challenge operation.

3.1.2 Shredding

Secrets are not entered in their entirety within the Shredding operation, as the server only chooses some parts of information.

The server can use a scheme, binding relevant parts of the secret to the current transaction and potentially vice-versa. Any kind of data can be used for determining the secret parts. Examples of this are: Current date and time, last used secret, user-transferred data, random numbers and current session ID. The server informs the user of the needed parts, without providing a hint on the secret's real value. One possibility of doing this is to request selected positions of the secret. Here, it is not necessary to challenge the positions of the parts in the correct order.

3.1.3 CAPTCHA

Inputs and requests of transaction data are not realized directly by the use of the CAPTCHA operation but via the CAPTCHA-based (Completely Automated Public Turing test to tell Computers and Humans Apart) operation. It is possible, for example, to realize the PIN input with the graphic password scheme, presented by Hong et al [DHM04].

It is also reasonable to combine CAPTCHA with one of the sub configurations mentioned before (Challenge, Shredding) to avoid that the order or the index number are acquired and analyzed automatically.

3.2 Security Modules

The following security modules are implementations of security mechanisms. They represent the essential functions of the security system. Further security modules, sub configurations and standard operations (for example PIN/TAN) can be implemented, if required, and integrated into the system. In principle, all security modules can be used at the same time and all sub configurations can be combined with any security modules.

It follows a description of all security modules implemented in the MAVA system. In each case all possible entry points and reasonable sub configurations are mentioned. Entry points are described in Section 3, sub configurations in Section 3.1.

3.2.1 PIN

The PIN operation, after the login with the user ID, consists of a singular arranged PIN for acknowledgement which is requested from the user for the purpose of authorization.

Implementation as security module

Entry point(s):	LOGIN
Sub configuration(s):	SHREDDING, CAPTCHA

3.2.2 LIN

A usual PIN secures the transaction at the PIN-TAN operation. This PIN does not lose its validity. A potential attacker is therefore able to achieve durable and complete authorization data by sniffing ID and PIN once.

In contrast to the simple PIN-TAN operation the authorization is not realized using a durable valid PIN but using a LIN (LogIN Number), which is valid only once. The user has several LINS which lose their validity after they have been used.

Implementation as security module

Entry point(s):	LOGIN
Sub configuration(s):	CHALLENGE, SHREDDING, CAPTCHA

3.2.3 TAN

After having entered the transaction data, a single TransAction Number (TAN) is transmitted to authorize the transfer.

Implementation as security module

Entry point(s):	POSTTRANS
Sub configuration(s):	CHALLENGE, SHREDDING, CAPTCHA

3.2.4 TSC-TCN

The TSN-TCN operation uses a Transaction Start Number (TSN)- Transaction Commit Number (TCN)-pair for authorization. The user transmits the TSN before every transaction and the respective TCN of the same pair after the transaction. The server checks correctness, completeness and validity of the TSN and TCN, essentially their authenticity as a pair. Only if the validation is verified, transaction is granted.

Implementation as security module

Entry point(s):	PRETRANS, POSTTRANS
Sub configuration(s):	CHALLENGE, SHREDDING, CAPTCHA

3.2.5 TSG- und TSGASYN-Method

This security module usually requires an additional mobile hardware device. In the TSG- TSGASYN operation Transaction SiGnatures (TSG) are generated, depending on the current transaction and the mobile hardware token used. The token does not need connection to the computer, as it works completely autonomous. A token can be a special device, specifically constructed for this purpose, or even a cell phone or a palmtop, equipped with the necessary software.

The token is personalized which means that it is configured for an individual user and secured against unauthorized use via, for example, a PIN. Different tokens compute different TSGs. This can be achieved by using various secret keys. The employed algorithm may be a standard encryption, signature or hash algorithm.

The user enters the relevant parts of his transaction into the token after committing to a special secured transaction. In the case of a credit transfer, for example, the target account number and the amount may be entered. Following this, the token computes the TSG to be used from the information supplied. The more transaction specific the data used is, the more unique the TSG will be. The token can pull up other parameters, for example the current date, time or the amount of TSG computations done. The user specific secret information (a key, for example) is involved in the computation. It is not possible to compute the TSG without the knowledge of all used parameters. The longer a TSG is, the harder it is for an attacker to guess. The user transmits this TSG to the server.

In the symmetric case, the server knows all parameters and can also compute the TSG,

using the transferred transaction request (e.g. credit transfer). Hence, the server can compare the computed TSG and the transmitted TSG and is subsequently able to verify whether the request for transaction is authorized, as only the user and the server are able to compute a correct TSG.

In the asymmetric case the server cannot compute the TSG. But the server can verify that the TSG meets the transmitted user data (e.g. a money transfer). Thereby the server can confirm that exactly this transaction was desired, because only the user could compute the correct TSG.

Implementation as security module

Entry point(s):	POSTTRANS
Sub configuration(s):	none

3.2.6 TAC- und TACASYN-Method

Using a simple PIN-TAN based system, the user receives insufficient information about his transaction to prove it to a third party. In the case of a TAC- (Transaction ACnowledge) or TACASYN (Transaction ACnowledge Asynchronous) operation the user obtains information enabling him to check and confirm the correctness of his transaction. This validation can be achieved by authenticating the correctness of the TAC using a mobile hardware token, where the token does not need to be connected to the computer, but works completely autonomous.

As with TSG and TSGASYN, the token can be a specialist device, palmtop or even a cell phone (with adequate software) and uses standard encryption, signature or hash algorithm as the employed algorithm.

The user enters the relevant parts of his transaction into the token after committing to a special secured transaction.

In the case of a credit transfer, for example, the target account number and the amount is entered. Following this, the token computes the TAC to be used from the information supplied. The more transaction specific the data used is, the more unique the TAC will be. Other parameters for the TAC computation are for example the current date, time or the amount of TAC computations done. The server specific secret information (a key, for example) flows into the computation. It is not possible to compute the TAC without the knowledge of all used parameters. The longer a TAC is, the harder it is for an attacker to guess. The server transmits this TAC to the user.

In the symmetric case, the token knows all parameters and can also compute the TAC, using the transferred transaction request (e.g. credit transfer). Hence, the user can compare the computed TAC and the transmitted TAC and is subsequently able to verify whether the server received the correct transaction, as only the user and the server are able to compute the correct TAC.

In the asymmetric case the user cannot compute the TAC by itself. Using the transmitted TAC, the user or a third party can compare the transaction used for computation of this TAC, with the real transmitted transaction. Thereby the user can proof that exactly this transaction was desired, because nothing but the server could compute the correct TAC.

Implementation as security module

Entry point(s):	POSTTRANS
Sub configuration(s):	none

3.2.7 LON-Method

For authorization in the LogOut Number (LON) operation, the user transfers for every transaction a single TAN which then loses its validity. He can deal several transactions. However, the pending actions are not being performed. At the end of the process, the user transfers a LON, which the server checks for its correctness, completeness and the validity. If the LON is correct, all outstanding actions get performed. The LON operation is not mandatory linked to the TAN operation. It is also possible to link pairs of LONs and LONs. In this case, the server checks essentially their authenticity as a pair.

Implementation as security module

Entry point(s):	POSTTRANS
Sub configuration(s):	CHALLENGE, CAPTCHA

4 The proxy component

The proxy component is set up between the client and the real application server. A direct communication between the client and the application server is not possible. The proxy component operates as front-end server and replies to the clients' HTTP requests. During this process the communication is secured via SSL.

It is important to mention that we name the protocols HTTP and SSL only exemplary. The proxy itself is generic. Other protocols are also supported. By using this configuration the proxy component is able to analyze, save and change all HTTP requests, which are sent by the client and all replies sent by the server. Furthermore, the proxy component can interconnect its own sites.

Basically, two types of modules have to be distinguished, the CommandHandler and the Scrambler. The CommandHandler is the link between the proxy component and the security component. Incoming requests, sent by the client, are forwarded to the server. The CommandHandler intercepts the responses and searches for configured inline commands. An example of this implementation of the CommandHandler interface is the CommandBasic module, which works on the HTML protocol

The Scrambler, on the other hand, provides a security function which makes it much harder to handle the web application using an automated script. It scans the communication and randomizes, if possible, the syntax without changing the semantic. Our example of an implementation of the scrambler interface is the ScramblerBasic module. It performs before and after the CommandHandler, but can also be used without it. Therefore it is possible to integrate the security component directly into the application and to use the proxy component as Scrambler only. Both CommandHandler and Scrambler are implemented as modules and can be exchanged anytime by extended implementations of the interface. They use separate configuration files.

It follows a description of the modules CommandBasic and ScramblerBasic

4.1 CommandBasic

The CommandBasic module offers the possibility of integrating the security component into an existing web application with a minimum of effort. To achieve this it simply requires the definitions of the respective session ID and of the security level (none, static, transaction) within the HTML code. It then forwards the session IDs to the SessionHandler of the security component, which provides the login mask.

Besides, the SessionHandler loads the security settings of the current user and provides the according functionality.

Furthermore, the HTML code, transmitted by the application server, may contain the logout command which itself causes the CommandHandler to load the relevant entry point of the security component. The used inline HTML commands can be modified in a configuration file. The standard properties are:

```
session_regex = <!--\s*SESSION:\s*"([\^"]+)\s*-->
secllevel_regex = <!--\s*SECLEVEL:\s*"
(none|static|transact)\s*-->
command_regex = <!--\s*COMMAND:\s*"(\s*(logout))\s*-->
```

The command session_regex is necessary to identify the user, secllevel_regex defines the syntax for declaring the security level. Command_regex defines how a direct command to the CommandBasic looks like.

Table 2 shows a hypothetical protocol run for online banking with static security.

Static:

	CLIENT	PROXY	BANK
1	→ Request		
2		→ Forward (http/1.0)	
3			← Reply
4		Analyze Reply: - Search for inline-commands - Initialize session in the security component - Save reply	
5		← Login mask of security component	
6	User id		
7		Verification, load security modules	
8		← LOGIN Entry point	
9	→ Authentifikation data		
10		Verification	
11		← Forward stored reply	

Table 2: Hypothetical protocol run for online banking with static security

Table 3 shows a hypothetical protocol run for online banking with transaction security. In this case a successful login is assumed.

Transaction:

	Client	Proxy	Bank
12	→ Request		
13		→ Forward (http/1.0)	
14			← Reply
15		Analyze Reply: Search for inline-commands. Load session in the security component. Save reply	
16		← PRETRANS entry point	
17	→ Authentifikation data		
18		Verification	
19		← Forward intermediate stored reply	
20	→ Input of transaction data		
21		Save data	
22		← POSTTRANS entry point	
23	→ Authentifikation data		
24		Verification	
25		→Forward stored data	
26			← Transaction OK
27		← Forward OK	

Table 3: Hypothetical protocol run for online banking with transaction security

4.2 ScramblerBasic

The Scrambler analyzes the data that has been sent and received by the proxy server. For outgoing replies of the CommandHandler and the application server respectively, the Scrambler constructs internal random strings for address referencing and the form element names. Incoming requests are written into the required format by the CommandHandler and the application server respectively, as a result of the internal administrated tables. In the configuration file it is possible to determine which sites may be forwarded directly and whether the sites that are not configured should be blocked for direct access. Additionally, it is possible to adjust the address references. Forms using the POST-method only give the functionality of random form names.

Figure 4 shows an example of a page which uses no scrambling. It is possible to extract information from the address string. In our example the bank identifier code is determinable.

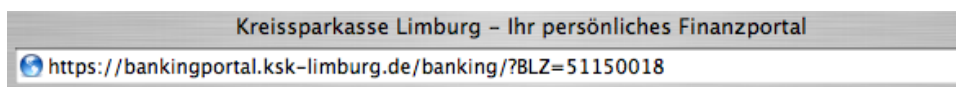


Figure 4: Standard adress string

Figure 5 shows how the same address string could look like with activated scrambling.

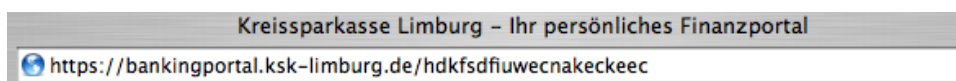


Figure 5: Scrambled adress string

5 Conclusion

The MAVA system gives the user the opportunity to secure existing distributed applications. It provides different security mechanisms which can be integrated easily into the system via the proxy component. However, it does not limit the user to the proxy component. The user can also enable the application to access the security component directly.

The advantage of the MAVA system is that other new security modules can extend it anytime. It is very flexible and merely provides a framework, to integrate other new mechanisms.

The implemented security modules are only examples for enhancing the security of an existing system. They easily outperform most security mechanisms found in up to date systems.

New attacks on distributed applications will certainly be developed in future. The scope for expansion of the MAVA system allows the user to integrate new security mechanisms as required. Those security mechanisms enable the user to increase the security of their application accordingly, with respect to the permanently increasing risks. The MAVA system is a cost-saving and flexible approach for securing distributed systems.

References

- [Cla06] Claudia Eckert. Konzepte - Verfahren - Protokolle. In *IT-Sicherheit*, 2006.
- [DHM04] B. Hawes D. Hong, S. Man and M. Matthews. A Grafical Password Scheme Strongly resistant to Spyware. In *Proceedings of the International Conference on Security and Management (SAM'04)*, pages 94–100, 2004.
- [Gha07] Rojan Gharadaghy. Authentication/Authorization. 2007.
- [Hoy02] Kristin Hoyer. Angriffe auf Anwendungsprotokolle. page 8, 2002. <http://www-rnks.informatik.tu-cottbus.de/de/materials/ws2002psInternet/kristin.pdf>.

- [Hub80] Hubert Zimmermann. OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. In *IEEE Transactions on Communications*, pages 425–432, 1980.
- [Inf02] Informatikzentrum der Sparkassenorganisation GmbH, Gesellschaft für automatisierte Datenverarbeitung eG. Erweiterung PIN/TAN. In *HBCI - Schnittstellenspezifikation*, 2002. <http://www.hbcizka.de/dokumente/aenderungen/HBCI22>.
- [Lam05] Martina Lamping. Phishen im Trüben. 2005.
- [RF97] J. Mogul R. Fielding, G. Gettys. Hypertext Transfer Protocol – HTTP/1.1. 1997. <http://www.ietf.org/rfc/rfc2068.txt>.
- [SV97] Gleb G. Pogodayev Sergey V.Zykov. Object-Based Groupware: Theory, Design and Implementation Issue. 1997. <http://arxiv.org/pdf/cs/0607121>.
- [T. 99] T. Dierks, Certicom and C. Allen. The TLS Protocol. 1999. <http://www.ietf.org/rfc/rfc2246.txt>.
- [TBL96] H.Frystyk T. Berners-Lee, R. Fielding. Hypertext Transfer Protocol – HTTP/1.0. 1996. <http://www.w3.org/Protocols/rfc1945/rfc1945>.
- [Ver05] Heise Verlag. Erfolgreicher Angriff auf iTAN-Verfahren. 2005. <http://www.heise.de/newsticker/meldung/66046>.
- [WKLB05] A. Wiesmaier, E. Karatsiolis, M. Lippert, and J. Buchmann. Outflanking and securely using the pin/tan-system. In *Proceedings of the 2005 International Conference on Security and Management (SAM'05)*, pages 313–319, 2005. <http://arxiv.org/abs/cs/0410025>.