Secure Information over Internet through PostgreSQL Database Systems with Java Client

Tiberiu Stef, Virgil Dobrota
Technical University of Cluj-Napoca
Department of Communications
26 Baritiu Street, 3400 Cluj-Napoca, Romania
Tel: +40-64-191689, 195699/208
Fax: +40-64-191689
E-mail: {Tiberiu.Stef, Virgil.Dobrota}@com.utcluj.ro

Abstract
This paper is an overview on the security techniques employed in developing an Internet application. The discussion is based on the security flaws encountered and depicts some solutions to solve these problems. The topics analyzed herein are closely related to an application consisting of a Java applet used as interface between a user and a remote PostgreSQL database.

Keywords: Authentication, Internet, PostgreSQL, Security, TCP/IP

1. Introduction

The need for information arises more and more nowadays with additional requirements such as: fast local and remote access, reliability and integrity of data retrieved, security of connection, safety of information management systems and many others. Within the frame of these requirements, we will present the problems that occur, and some possible solutions to setup a safe database access system, as well as to create a Java applet as a client to access this database. The discussion will be limited to the analysis of techniques and applications available in the domain of free, non-commercial software for Linux operating system.

The choice for this task was the world's most popular free database ISO SQL *(Structured Quiery Language),* ANSI SQL/92 and ANSI SQL/89 compliant RDBMS *(Relational Database Management System)*: the *PostgreSQL*, also known as *pgsql.* We have chosen this system because it provides most of the functionality needed in the case of maintaining small amount of information (names, grades, comments, homework solutions together with submission dates, etc.) related to students attending one course. Since this DBMS, including its sources to customize for own needs, is available for Unix-derived operating systems, the host will run Linux.

2. Description of the client application

The client application presented in this paper is a Java applet, accessed from a Web page. This solution was chosen because it is a platform independent one. As an example, the applet can be embedded in a distance education course to provide interaction or can be used for local use allowing students to submit their answers and comments to course-related questions. Based on JDBC driver for the Postgres database management system, we have created a Java-based interface between the user and the SQL grammar, to simplify the interaction with the database

and to add an additional security layer to the database command interpreter. The interface will pop up from an applet after the user requesting the connection to the database succeeds to authenticate himself as a legitimate database user.

The advantages to this solution are the following:

- Connection to the database is *transparent* to the user. The applet from the web page will have coded the location of the database, thus the user is only required to authenticate himself and it will be logged in. In the case of multiple databases access, the solution can be implemented by creating different instances of the same applet. Each of them includes the destination database address and an attached web page with information about it.

- The database user *does not have to know the SQL syntax*. It is provided with a graphical interface that helps to access, to retrieve and to modify the data within the tables. Also this eliminates the possible attacks to PostgreSQL interpreter by not allowing the commands to be passed directly to the interpreter. The only commands that can be executed are those which have a corresponding item on the graphical interface. Furthermore, to avoid problems caused by the users' parameters, the inputs were filtered before processing. The user can read from the database the tasks generated by the teacher and the previous answers, or can add a new answer to a selected question.

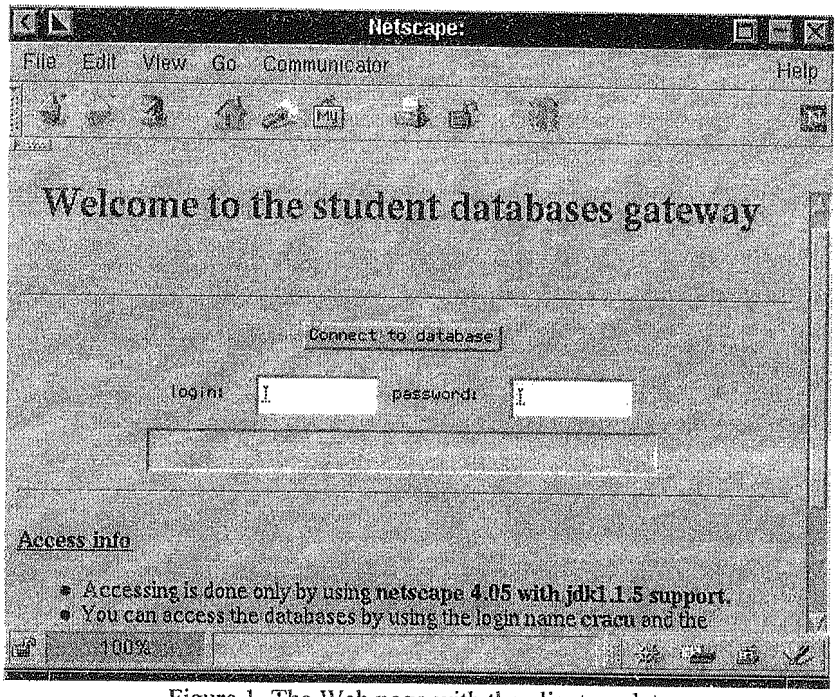

Figure 1. The Web page with the client applet

The drawbacks of this solution are summarized now.

1. Since the request for connection to the database comes from an applet within a web page, the present security restrictions impose that the web page containing the applet should come from the same host where the database management system is located. This means *that we must run a Web server on the same machine*, thus leading to additional security measures to be taken in order to secure the Web server. The solution to this problem is to use *signed applets* which are allowed to connect anywhere in the net, not just to the Web server's host.

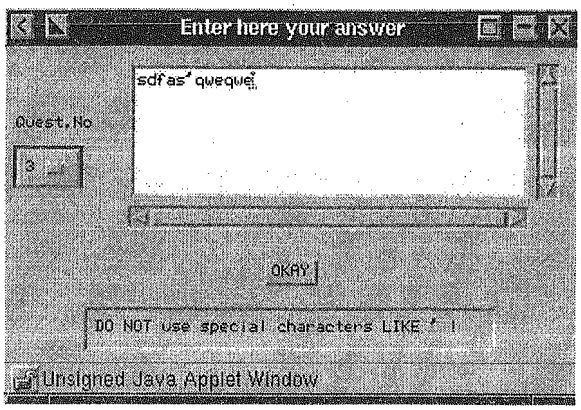

Figure 2. Dialog box for answers

We did not implement this solution because the signing tool employed produced signatures recognized at the present only by HotJava browser from Sun, and we wanted to keep the application browser-independent. The alternative solution is to purchase from Verisign a digital ID and use it to sign the applet that connects to the database. Since we wanted a free solution, we have chosen not to employ signed applets and to take care of this additional attack target.

2. In the present form, the communication between the applet client and the database is *unencrypted*, thus subject to password and data sniffing, to data altering, eaves dropping, and all the other attacks possible inside a network. This can be avoided by using one encrypted communication form through the use of encrypted sockets (end-to-end encryption) or application level encryption. We will discuss this later, at the TCP/IP section. An idea is to use SSL *(Secure Sockets Layer)* or an encryption algorithm for the link, and this is easy because encryption classes are available in Java, but this means to change parts from the database backend module and for the moment we wanted to remain compatible to the distributed version of database system. Although it is the best solution available, the SSL classes (available at the Technical University from Gratz, Austria, at [4]) were not used because the Postgres database does not support SSL yet.

3. One thing that could not be overlooked is the way *authentication* of the user is done. This is determined by the configuration of the database server and will be discussed later (see Postgres database system security section). This issue depends on how many of the existent authentication techniques have been implemented in the JDBC driver. The available techniques will be presented in the following paragraphs. Currently the Kerberos-based authentication is not available in the JDBC driver.

3. Security issues of the proposed system

The most exciting part of creating an Internet application (from our point of view) is how ho set it up to be secure. Some problems encountered in the case of a database which provides access to the web come mostly from the following:

3.1. Internet and TCP/IP

The most used attacks on a network service based on TCP/IP are crafted by techniques such as:

- Network sniffers, software tools that can read the information passing through the physical cable connected to the attacker's computer. They rely on the possibility to rewrite the network card's software driver in a manner that it will capture the desired information.
- Spoofing attacks target the termination of an active connection by the means of sending prematurely IP packets containing reset signals.
- Connection hijacking is used to seize control of existing interactive sessions (such as telnet) by reading the sequence numbers from the IP packets and generating the next response packet faster than the legitimate communication peer.
- Data spoofing, a technique used to insert data into an ongoing communication between two other hosts. This is proven to be an effective way to compromise a communication protocol or data transmitted between parties. In the case of a determined attacker, the results coming from the database could be faked.

All types described above are currently possible due to the TCP/IP stack implementation. They will be no longer a problem when IPv6 with authentication and encryption support will be employed in the Internet. Other security problems are generated by the simplistic implementation of the network services. For example, many UNIX network servers rely on IP address or host names to authenticate incoming connections. This approach is fundamentally flawed, as neither the IP protocol nor DNS *(Domain Name Server)* were designed to be resistant to attack. Thus many reports of successful IP spoofing and DNS compromise have been frequently issued. Also a large number of services, on which authentication of users and hosts rely, are proven to have their security flaws.

The solutions to the most of the problems are:

- Use of encryption to protect the data. This has the benefit of making the data unusable for the eavesdropper and also alterations in data will be instantly noticed.
- Avoid using passwords and host-based authentication. Instead we should rely on tokens, one-time passwords and cryptographically secure connections (SSL, Authentication provided by Kerberos).
- Some of the network programs perform a reverse lookup thus making more difficult the attacks.

SSL is a handshake protocol that provides security and privacy over Internet. It supports client and server authentication, maintains the security and integrity of the transmission channel by using encryption, authentication and message authentication codes. This protocol is a good choice and is not supported by the Postgres database yet (delivered as a patch with a very reduced functionality) and it is not implemented currently on the Java. However, it is available in external Java packages, such as the one we tested from Technical University of Gratz. The SSL technique can be used instead of a secure Web server to hold the pages that contain the client applet. It requires a signed certificate to be accepted by the communicating parties.

This analysis leads to the conclusion that in the present situation, due to commercial policies and non-standardization of the technologies employed in the Internet, the solutions are either to

continue to use the connection unencrypted, either to implement our own encryption system at the client side, to preserve the platform independent characteristics.

3.2. Java

Java is portable across many platforms, is powerful, contains graphical interface elements, can be embedded within Web pages and get executed locally by most modern browsers. The applet described above has been tested on both Windows with Netscape and Internet Explorer and on Linux with Netscape browsers supporting Java version 1.1.3 or above [3].

One of the applet's tasks is to provide the interface to the potential user so that he could login. In case the security manager of the web browser would permit a malicious applet to be loaded, the unsuspecting user would type his password and this would be captured/sent to the attacker. This is only one case of attack and many more can be crafted by exploiting weaknesses of both Java and of the host operating system. We will limit our discussion in this field because we consider that Java is not very close to a final standardized form.

3.3. Postgres database system security

The Postgres database system security is based on *host access control*. This means that the security restrictions when accessing the information within the tables rely on the ability of a client to prove that he logs to the database from a network which is granted access and that he can authenticate himself according to an entry in the "Host-based access control" file *(pg_hba.conf)*. Every client that wants to access a table within the database must be covered by one of the entries in *pg_hba.conf* [2]. Otherwise, all attempted connections from that client will be rejected with a "User authentication failed" error message. Connections from clients can be made using UNIX domain sockets or Internet domain sockets (i.e. TCP/IP). Connections for Unix Domain sockets are meant for the case of local (console) access. Since this is not the case of out application, we will present it briefly having in mind the possibility of a local attack. Connections made using UNIX domain sockets are controlled using records of the following format:

```
local <database> <authentication method>
```

where `<database>` specifies the database that this record applies to. The value *all* specifies that it applies to all databases. `<authentication method>` specifies the method a user must use to authenticate themselves when connecting to that database using UNIX domain sockets. Note that in this case the authentication method does not imply great risks as long as the password is kept secret. A threat from a user logged into an account on the machine running the database backend will be discusses in the Operating system weaknesses section. Connections made by Internet domain sockets are controlled using records of the following format:

```
host <database> <TCP/IP address> <TCP/IP mask>  <authentication method>
```

As we can see, the access control is based on IP addresses, which is a bad idea. The authentication methods supported by the Postgres database system are used in other database management systems and in different kinds of network-service server software. We will describe briefly what is available in the 6.3 version of Postgres and stress out what is appropriate to be used in a secure Internet application.

3.3.1. Authentication methods for both UNIX and TCP/IP domain sockets

1. *trust* - the connection is allowed unconditionally, and *reject* - the connection is rejected unconditionally. Both can be used with hosts and/or networks for which we have already decided whether they can or cannot connect. Of course it is the default case in setting up every information providing service and if we set it by the book, we will have to choose between the following:

    - *Default allow* - this is the case when we are very generous with the rest of the world or we trust that the service we offer cannot be forced to execute dangerous actions on the physical host or cannot be used to damage other services on different hosts which might lead to break-ins or DoS *(Denial of Service)*. In this case the configuration files only restrict access to potentially dangerous services from potentially dangerous networks or hosts. As an example are the common web or ftp servers, but they are not covered by this paper.

    - *Default deny* - this is the case of almost all other services within a local network, where an administrator usually permits access only to local (registered) users.

2. *password* - the client is asked to provide the user's password. This is sent in clear and compared against the password held in the *pg_user* table. If the passwords match, the connection is allowed. This option is somehow depreciated because it looks like an ordinary logging and it is insecure, due to eavesdropping and sniffing attacks. Note also that the password authentication is by definition weak and should be replaced with one-time password systems.

3. *crypt* - the client is asked to provide the users's password. This is sent encrypted (using *crypt*) and it is compared against the password held in the *pg_user* table. If the passwords match, the connection is allowed. This method was used in our application by sniffing on the connection and reading the *crypt()* -encrypted passwords. It would stop a lazy attacker to try a break-in because it takes some supplementary effort for a brute force attack. At the present time it is available the EFF's hardware DES Cracker, which can break finding the 56-bit key in approx. three hours. For testing purposes we have made a password cracker using the *crypt()* function and a word dictionary. However this method makes more difficult sniffing, spoofing and hijacking but still allows eavesdropping.

3.3.2. Authentication methods for TCP/IP domain sockets (Internet) only

1. *krb4* - Kerberos V4 is used to authenticate the user. *krb5* - Kerberos V5 is used to authenticate the user. The *Kerberos* authentication system is based on DES and usually protects sensitive information such as passwords. When a user logs in to a workstation running Kerberos, the user is issued a ticket from the Kerberos Server. The user's ticket can only be decrypted with the user's password; it contains information necessary to obtain additional tickets. This way the user's password does not travel through the network. All the information in the Kerberos tickets is encrypted before it is sent over the network, the information is not susceptible to eavesdropping. Normally, Kerberos is used solely for authentication. If eavesdropping is an ongoing concern, the information between the workstation and the service can be encrypted using a key previously exchanged by the two parties [1]. This technique seems to solve a lot of our previously mentioned problems, but

there is a little inconvenience: the JDBC driver does not implement yet the Kerberos authentication service. That's why temporary we have dropped this solution.

2. *ident* - the *ident* server on the client is used to authenticate the user (RFC 1413). An optional map name may be specified after the *ident* keyword that allows *ident* user names to be mapped onto PostgreSQL user names. This option can be used in the case of a server which runs the identification protocol (auth) on TCP port 113. It is useful to set up such a service when we want to know the name of the user associated with a particular TCP/IP connection. In our case, if a certain user tries to connect to a database, the user's pretended name from the logging dialog box can be verified to be the name of a real user owning the TCP/IP connection. The identification protocol is based on a simple callback scheme. When the server wants to know the real name of a person initiating a TCP/IP connection, it simply opens a connection to the client machine's *identd* daemon and sends a description of the TCP/IP connection in progress; the remote machine sends a human-readable representation of the user who is initiating the connection - usually the username and the user's full name from /etc/passwd file.

This approach to security based on the identification protocol is *not a good choice because it depends on the honesty of the computer at the other end of TCP/IP connection*, in our case the server that runs the postmaster daemon [1]. If somebody has broken into the server computer, a backdoor attack would be to create a user that will permit unconditional access to the database information. Another drawback for this is that the ident service does not exist on Windows - operating system driven machines. These are some reasons why we did not choose it in practice.

After the user has logged in, it is available the built-in layer of protection, which permits to the database administrator or to the owner of the information within the database to set up restrictions for the rest of the users accessing the data. We will not discuss any further because it might be some differences form a DBMS to another, although the concepts appear at almost every database system.

The final conclusion after this section is that it's difficult to put together software pieces coming from different sources. Also the free software, although proven to be very used and useful, will be always in an unpredictable evolution, and when setting up a safe and robust application one must rewrite the existing components to suit our needs. In our case we used the simple DES-encrypted password authentication system, without encryption of the communication channel because the database contains student information (especially course grades) and not some top-secret information. Though we expect attacks from students, and this will be a good way to test the whole application.

3.4. Operating system weaknesses

The discussion in this section depends very much on the operating system and we will point out briefly the potential threats coming from the use of the Linux. The most encountered attacks target the supervisor account. Usually attackers are trying these:

- to get a root shell.
- password cracking
- data integrity alteration and theft
- setting up a backdoor in the system

The reason for such attacks would be to gain local rights on the server running the database backend. This way is much easier to get to the information from the database. All these can be avoided if the system administrator sets up correctly the file permissions and network services.

4. Conclusions

This paper is an overview on the security techniques employed in developing a secure Internet application. We have stressed the existing problems in the today's network applications and we have analyzed the existing non-commercial solutions available in the domain of free software for Linux. The discussion was related to a specific database application and restricted to elements from Postgres, Java and TCP/IP that we were able to modify. Most of the problems arise from the highly insecure nature of the Internet and from the unprofessional programming style and insufficient testing while elaborating the applications. When using GPL-like *(General Public License)* software, the effort employed in creating secure application for the Internet is focused on customizing the existing modules to suit the local requirements, and on creating interfaces between the existing modules. The continuous evolution in the field of network security and the lack of non-commercial cryptographic software have not permitted us to use the best solution to secure the application, and we relied on weaker security measures (like DES-encrypted passwords). The application has been tested against the described attacks and its weaknesses are due mainly to the present insecure form of Internet transport service and to bugs in the operating system and in the database software.

References

[1] Spafford, G. Garfinkel, S. - *Practical UNIX and Internet Security.* Second Edition. O'Reilly & Assoc. Publishing, 1996

[2] *** - *http://www.postgresql.com*

[3] *** - *Java Development Kit 1.1.5 doc*

[4] *** - *http://iaik.tu-gratz.ac.au*

[5] *** - RFC 1413

Híradástechnikai Tudományos Egyesület
Scientific Society for Telecommunications
H-1055 Budapest, Kossuth Lajos tér 6-8.
Tel.: (36-1) 353-1027 Fax: (36-1) 353-0451

EURASIP

ICOMT '98

3rd International Conference on Multimedia Technology and Digital Telecommunication Services

October 28-30, 1998

Budapest, HUNGARY

Organised by

the Scientific Society for Telecommunications

Supported by

National Committee for Technological Development (OMFB)

PROCEEDINGS