Student Research Paper

Administration of Access Rights in Web Applications

Stefan Worm
stefan.worm@s2000.tu-chemnitz.de

August 18, 2005

Supervisor: Dipl. Inf. Chris Hübsch
Abstract

This work deals with the problem to find and rate a solution how to administrate access rights in web based applications that are flexible and offer a fine-grained allocation of rights. In particular the program phpGACL is analyzed and integrated into an example application to prove the feasibility of this system in principle.

German/Deutsch:

Diese Arbeit beschäftigt sich mit der Lösungsfindung und -bewertung des Problems, Zugriffsrechte webbasierter Anwendungen flexibel zu administrieren und eine möglichst feinkörnige Rechtevergabe zu erlauben. Insbesondere das Programm phpGACL wird analysiert und in eine Beispielanwendung integriert um die prinzipielle Realisierbarkeit des System zu überprüfen.

The original (PDF) version of this document can be found here:
http://archiv.tu-chemnitz.de/pub/2005/0143

Worm, Stefan:
Administration of Access Rights in Web Applications
Student Research Paper, Chemnitz
University of Technology, 2005.
# Contents

1 Task of the Student Research Paper .................................................. 1

2 Introduction ......................................................................................... 1
   2.1 Discussion of the Problem ............................................................. 1
   2.2 Authentication .............................................................................. 2
   2.3 Authorization, ACCESS CONTROL ............................................... 3
   2.4 Point of Authentication ................................................................. 4
   2.5 Point of Authorization .................................................................. 4

3 Discussion of several Approaches ....................................................... 5
   3.1 Typical Self-Made Solutions ......................................................... 5
      3.1.1 A very simple way to authorize ............................................. 5
      3.1.2 An enhanced way with additional database tables .................. 6
      3.1.3 An enhanced way with session IDs ......................................... 7
   3.2 .htaccess with integrated front end ............................................... 8
   3.3 Choosing an Approach .................................................................. 10

4 phpGACL — PHP Generic Access Control List .................................... 10
   4.1 Presentation of the phpGACL System ............................................ 10
   4.2 Similar Applications to phpGACL ................................................ 14

5 Integration of phpGACL ..................................................................... 15
   5.1 The T:EaS System ...................................................................... 15
   5.2 What was done to integrate phpGACL in the T:EaS System? ......... 15
   5.3 What else could be done in the future? ......................................... 19

6 Conclusion and Outlook ..................................................................... 21

A Source Code ..................................................................................... 22
   A.1 Exemplary API Operations for phpGACL .................................... 22
   A.2 Example for Customization of a Website ...................................... 23

B Screenshots and Explanation of phpGACL Interface ......................... 25

Abbreviations and Acronyms ................................................................ 39

References .............................................................................................. 40

Index ..................................................................................................... 42
List of Figures

1 ARO Group Admin ............................................. 27
2 Assign Group - ARO (deanship) ................................ 28
3 Assign Group - ARO (student assistants) ...................... 29
4 Assign Group - ARO (professors and staff) .................. 30
5 AXO Group Admin ............................................... 31
6 Assign Group - AXO (courses) ................................ 32
7 Edit Objects ..................................................... 33
8 ACL Administration Page ....................................... 35
9 ACL List, part 1 .................................................. 36
10 ACL List, part 2 .................................................. 37

Listings

1 a basic .htaccess file ............................................ 8
2 includes for phpGACL .......................................... 11
3 use of the acl_check() function ............................... 12
4 parameters of the acl_check() function ....................... 13
5 includes for acl_check() ....................................... 16
6 instantiation of a new gacl() object ......................... 16
7 functional part without acl_check() ......................... 16
8 functional part with acl_check() ................------------ 16
9 API functions .................................................. 23
10 use of acl_check() ............................................. 24

List of Tables

1 Rights of the T:EaS/phpGACL System ......................... 18
1 Task of the Student Research Paper

For web based applications it is often necessary to enable only a subset of the whole functionality of the application depending on the user who currently communicates with the system.

One basic approach to achieve this is to use the access rules of the web server. In complex applications with many different roles and frequently alternating role assignments and many functions, this way of configuring is too inflexible. Hence, it is necessary to find alternative solutions for this problem that are easier in configuration and offer a more fine-grained granting of rights.

The integration in an exemplary implementation shall prove the principal usability.

All the produced software and documentation is to be published under a public license.

2 Introduction

2.1 Discussion of the Problem

In almost every application that is designed for working with a group of users it is necessary to have a rights management system.

In most cases, there is a user who has more rights than others and users with fewer rights. Therefore, it is useful to implement a functionality to control these rights.

The most important things, which have to be paid attention to, are authentication and authorization. Authentication means to find out if someone really is the person he or she pretends to be. In addition, authorization describes the process of granting special rights to the person. This will be described more detailed later in this text.

In this paper, the focus will be directed on a web based application. Especially the needs of the so-called L:AuS system. This is the German abbreviation for "Teaching: Education and Study" (T:EaS) [CR05]. It is used for the registration for courses and for giving credits after passing exams and tests. It is similar to the Stellar™ Course Management System1 of the Massachusetts Institute of Technology2 (MIT) but with less functionality at the moment.

The L:AuS system already exists. It is in operation and it is working without trouble. Nevertheless, there are still some problems:

• There are two groups of users with advanced rights. But if you are in the situation that you want to give a single person only some of the advanced rights this is impossible. You only have the option to add the user to one of the groups that would grant him all the rights of this group. Therefore, there is no option between ‘all rights’ and ‘no rights’. This is not very functional.

1http://stellar.mit.edu/
2http://www.mit.edu/
• Due to the uncomfortable rights management in combination with a web server based authorization system, maintenance is very complex and can be done only by a technical administrator although a person of the management staff could also be able to realise most of the work.

• The usability of the system is not the best in general. The complete rework in that point is not subject of this paper. However, it can contribute a few suggestions for example for personalised webpages, which makes it clearer for the user to navigate within the application.

The most important problem to work on is to find a possibility how to grant or deny rights to single users, so that they can do exactly the job they want to carry out within the system — not more and not less.

For example that a student assistant can get some uncritical rights like view for only the one course he or she is a tutor for — not this right for all courses and not more rights for the specific course than necessary. The allocation and maintenance of this system should be as easy and powerful as possible. Furthermore the rights should be as fine-grained as possible so that you can define if a user is allowed to view or change every single detail regarding the course — not only "global" rights like 'edit all courses'.

2.2 Authentication

When designing an application with rights management, you have to pay attention on the authentication of the user. In this student research paper the aspects of a web application will be followed. This means the requirements of an application that is used by customers through the World Wide Web (WWW) with a web browser will be analysed. In this special case, it is driven by PHP scripts [php].

Authentication of a user means to know "Who?" wants to gain access. Nowadays the most common way to manage this is with the help of a ‘username’ and ‘password’ pair. This is called ‘proof of knowledge’. There are two other techniques: If you have a key, identification card or other physical devices than you have authentication by ‘proof of possession’. The third way is ‘proof of property’, which means that biometric characteristics are used, e.g. finger prints, retinal images or handwritten signatures. [Opp96, p. 17]

Let us assume there is the user Alice Murphy who has the username amu. She has chosen a password, which is only known by her. If she wants to use the application, she only has to type in her username and password. Now the application will accept her login if the application knows the username and if the password is the right one that belongs to this username. With a high chance, the application can be sure that Alice Murphy is logged in now. One says she is authenticated. Nevertheless, why is it only a high chance and not sure? You can only falsify a thing with cirtitude but you can never verify a thing with certitude. To be sure would mean that you can say with 100 per cent certainty that only Alice herself knows her password. However, you can never know if Alice did not give it to other people, that someone watched her typing it in or if
there were other circumstances that result in a compromised password, e.g. an unsafe password that can be guessed easily.

## 2.3 Authorization, ACCESS CONTROL

The authorization describes the rights that are granted to a previously authenticated user. It says, "who may do what to what" [For94, p. 15]. That means that a person (’who’) has or has not the right (first ’what’) to do something (second ’what’). For example, Alice has the right to edit the news section or John has the right to read the details of project XYZ.

This only makes sense if an authentication took place before. If there did not, every person could say, "I am Alice." although he is not and he would get all the rights of her. Then the access control would not work.

For the rights, which you give to a person, it is necessary to define rules. Usually you can do this in three different ways:

**rules based on identity:** It is defined that a single person or a group of people has or does not have various rights. It works as described in the example above.

**rules based on roles:** First, you create a role. Then you combine this role with a set of permissions and in the last step, you assign the role to people. Let us assume we have an online newspaper with editors, team leaders and technical staff. Now you define a role for an editor by giving the role a set of rights, e.g. writing articles and editing them. You do the same for the other roles. Then you assign a person to a role that fit best to his kind of work. You can also assign multiple roles to a person, e.g. if he works part time as team leader and technician. In this case, Alice as an editor can do all things for the online newspaper, which are necessary for her as an editor, because she has all the rights she needs.

The advantage of this system is that you combine the rights with roles and not with individuals directly. With this additional abstraction layer, you are able to manage the rights independently from the individuals and that gives you more flexibility. It is also possible to assign multiple roles to one person wherewith the authorization information is defined by the set union of the roles.

**multi-level rules:** They are used mostly in high security environments like the military. The idea behind this is to classify information based on sensitivity by using levels. If a person wants to have information, the person’s clearance level must be equal or higher than the level of the information. The user in subgroups do have only a subset of rights. ”They are most commonly used for protecting information from unauthorized disclosure” [For94, p. 154].

Stefan Worm
2.4 Point of Authentication

A fact that also has to be discussed is at which point the authentication takes place. There are several possibilities where this can happen[For94, p. 59].

- You authenticate to your local end system (e.g. smart card reader). This sends the authentication information (if you are successfully authenticated or not) to a remote system (e.g. online banking website). This means that the remote system has to trust your local system, which is done by an authentication of the local end system itself to the remote system.

- You do the authentication to the remote system. For example, you type in your password on your local end system but this only sends the information to the remote system that does the authentication itself.

- A third way to do this is to use an authentication service like Microsoft Passport, Liberty Alliance or Sibboleth [Weg04].

Now you authenticate to the service as if you would do to a remote system (see above).

You will get a certificate or ticket with which you can authenticate on the remote system. This will work only if the remote system and the authentication service trust each other. Although it is more complex than the other two possibilities, it brings many advantages like single sign-on (SSO) solutions, the possibility to authenticate on not trustable remote systems and many more.

Nowadays the most common way is to authenticate to a remote system. This can be for example a remote application like a web server.

2.5 Point of Authorization

We also have to think about where to perform the authorization. A widely used possibility is to do this on the web server level. At this point, you only have the possibility to control rights for directories or files. For example the popular web server Apache3 can do this among different other methods with .htaccess4 files.

Let us assume you have a section of your website that contains special functions (e.g. billing) that should only be accessed by privileged users. The only way to realise this with the help of web server controlled rights is to separate the functions and write them in different files or directories. For these you have to set the necessary rights. Only authorized users have access to these protected files or directories now and only they can can execute the functions therein. Note that the single functions inside a file are not protected by any system. Only the access to the file or directory that contains the special functions is protected by web server access control.

3http://httpd.Apache.org/
4http://httpd.Apache.org/docs/sections.html
You can do this for every needed function. If you have a small website with only a few functions this is a common way as it is quite simple.

On bigger websites this way becomes very confusing. It is also not very efficient for example if you have a single webpage on which you have a small section that has to be protected. You have two main possibilities. The first is to separate this function from the rest of the website. The second is to create two almost similar websites. One with the function that has to be protected and one without. Both ways are not very smart, because you subordinate decisions regarding the website’s architecture (for example maintainability or usability). There are better ways to realise this.

One way is to displace the authorization directly in the application. If you operate all the details of authorization within the application, you are more flexible in managing the rights on the website. For example, you can specify a function only executable for website administrators. This is usually done with the help of an authorization database. You can use a MySQL database (DB) which holds all the necessary authorization information. The maintenance can be done by yourself or you use a tool which helps you with this (e.g. phpGACL). With this solution, you have full control over granting or denying specific rights on a section of your application. You do not have to care about the limitations of the web server based solution.

3 Discussion of several Approaches

3.1 Typical Self-Made Solutions

The authentication and authorization of users is a basic function in web applications. Every application that has at least one section that should not be available for every user in the Internet needs such a system. Because the need of this is so essential, almost every designer of a bigger website has probably implemented a version of his own. Moreover, that is why there exist a big number of solutions how to do this.

Fortunately, most of them are very similar. Regarding the way to store the authentication and authorization data, websites that are more complex mostly use a database for this. The above-mentioned database MySQL is widely used. It is also imaginable to use files for storing the data. However, this is only useful if you have a very small website or one that does not have a database available.

The used programming language is not as important because nearly each one offers more or less comfortable functions to integrate a database in the application. Widely used in the web environment are Java, PHP, Python and Perl – Ruby, C#, C/C++ as well as others are not used that often.

3.1.1 A very simple way to authorize

In the following section, a general solution how to achieve the authorization in web applications by using a database is going to be introduced [Phr04]. Let us assume that we have a website with at least one section for which special rights are necessary, that
the user is already authenticated, and his username is available by a variable of the web server. Furthermore, every specific section has an identifier. This could be a name like administration area or a number like 23.

In your database you have to create a table where every entry is a tuple: username, section ID (identifier). In this table, you write down all the username/section pairs that you need on your website. Like an entry in the table: amu/administration-section means that amu (= Alice Murphy) has access to the administration-section.

Now, at the beginning of the section in the web application that needs to be protect, you have to perform a conditional statement. In this (for example) "if-statement" you check the database if the currently identified user has the right to access this section. If yes you execute the TRUE branch of the statement and the protected function, but if not you go to the FALSE/else branch and show an error message for example.

3.1.2 An enhanced way with additional database tables

The example explained above works. Most of the applications in that area follow this basic scheme. Nevertheless, in real live this is often too simple. There are a couple of possibilities how to improve the flexibility, security and maintainability.

- For example the organisation of users in groups will help a lot not to have large amounts of single entries for every user combined with every section. To do this you would create another table. This would consist of the columns username and group ID as the foreign key. Additionally you would have to change the already existing table with the columns username/section ID to the new table group ID/section ID.

- To enable the possibility for a more detailed rights management you can define many rights like read, write, delete, etc. instead of only "allowed/ not allowed". In order to realise this it is only necessary to add the column rights to the table with the already existing columns group ID and section ID. The modified table can be used to make a new kind of request. Now on your website you can ask the question: "Does the group (ID) have the necessary rights for this section (ID)?" If the condition is true and the user is member of the group than access will be granted.

In the previous section, it was implied that the user is already authenticated. You can do this if for example you use the web server authentication described in section 2.5. However, sometimes it can be also useful, if you do it on your own with the help of an extra table. For this the table needs the two columns username and password [WT03, ch. 14]. The password will be stored usually MD5-encrypted, because if a person would get unauthorized access to the database he or she would know all passwords if they where stored unencrypted [Cos02, ch. 7]. But due to the encryption he does not, because "there is no way to read back the original" [LW02, ch. 9.3]. Another point is that the password is also protected against the view of persons that have usually authorized access to the database (e.g. administrators).
With this extra table for authentication in cooperation with the tables for authorization, it is possible to secure parts of the web application by performing a database check at the beginning the protected section if there is a valid entry for the requested part of the webpage. However, it is not very comfortable for the user because he has to re-type his username/password every time he surfs on the website and wants to access a protected part of it, except if the web browser memorises the the values and sends the authentication data to the server without asking the user again for it. Especially with the BASIC version of the authentication, because username and password are sent un-encrypted over the Web which makes it easy for a malign observer to get the password [LL02, ch. 5.1].

### 3.1.3 An enhanced way with session IDs

To avoid this disadvantage you can use the principle of *sessions*. With session management you are able to do the authentication only once on the website. In this step you store the information that a person is authenticated successfully in a session ID. And later on, if you need the authorization information of the person you can use the session ID for this. With the session ID, you also have the authentication information with which you can find out if the person has the necessary authorization for a specific part of the website.

The advantage that the user in this scenario has to authenticate only once is realised by the storage of the session ID on the client side. This can be done by the use of a cookie, uniform resource locator (URL) parameter or hidden element in Hypertext Markup Language (HTML) forms [Sta01, p. 138]. On the server side the session is usually stored in a file or database. The last option with the database is a suitable one for the above-explained situation where we already have tables for the authentication and authorization data.

In combination for example with cookies it will work that way: The first time on the website, where the knowledge of the identity of the user is necessary, you do the authentication. If it is successful you generate a session ID which will be stored in an extra database table together with the username on the server side.

Then you send the session ID back to the user as a cookie that will be stored on his or her computer. Now every time the user wants to access a secured part of the website he or she sends the cookie with the session ID to the server. The server checks if it has an entry with the session ID in the database table. If so, it looks up which username is linked to the ID. Then the web server uses the same tables for checking the authorization as explained in 3.1.2. The session ID itself is only temporary, because of performance and security reasons — due to "risk of someone guessing session IDs and hijacking an old unused session" [LW02, ch. 8.3.7]. It will be deleted after a fixed period of time, usually after some minutes.

In fact, the session offers you the functionality of a website with a state. Usually a website is stateless. If you surf from one page of the website to another page the web server does not know that this action is connected. For the web server the download of one page or another is the same. Now with the help of sessions it can associate the
two or more requests of pages on a website. This feature is not only helpful for storing authentication data — it is also very useful for shopping carts on a shopping website or for the tracking of users to find out what, in which way, for how long, etc. users visit a website. In fact the state of executed code on the web server, respectively the assignment of variables on it for the user, are stored so that the server can continue at a well defined point by courtesy of the session ID.

3.2 .htaccess with integrated front end

As described in section 2.5 the .htaccess system is for authorization on the web server level. But it can also be used for authentication if necessary. In the following section, it will be described a possibility how to use the .htaccess system much easier than you would usually do.

First of all a short introduction regarding the .htaccess file format is given [LL02, ch. 5.15]. On a web server without any precautions for protecting files or directories, all of them below the root directory of a website are accessible for every person over the internet. In most cases this is not intended. Therefore the .htaccess system offers a possibility to deny access to a number of files or directories. This can be done in many different ways. A very basic possibility to protect a directory is to put a file named .htaccess in it with a content similar to this [TC05]:

```
AuthName "Only for members"
AuthType Basic
AuthUserFile /www_root/directoryXYZ/.htpasswd
Require valid-user
```

Listing 1: a basic .htaccess file

"**AuthName** gives the name of the realm in which the users’ names and passwords are valid." [LL02, ch. 5.2] A realm is a "unique name given to each protected area on a server, whether it be a single document or an entire server." [LLC05]

**AuthType** Basic is the simplest method. In more complex situations the use of other possibilities is useful. For example one with encryption of the password.

**AuthUserFile** stores all valid user/password combinations in a file named .htpasswd. The passwords are encrypted. The file has to be generated in advance and needs to contain all users and their passwords that shall pass the .htaccess protection.

**Require** The most important directive you have in this line. It determines what is necessary exactly to pass the .htaccess protection. In this case the argument valid-user accepts any user that is in the .htpasswd password file.

With a basic scheme like this or the use of some of the numerous extensions for this you can determine for every file and directory if and how it shall be accessible.
If you use the .htaccess system in practice, you will notice that it has some disadvantages. Especially if you have a group of people on your website who are responsible for the management of authorization related issues. The main problem is that a person who wants to edit the .htaccess file needs direct access to it on the server. Usually you login to your server via Secure Shell (SSH) [BBS05] or you use the File Transfer Protocol (FTP) [PR85]. For this you need at least one login on the server. Often you only have one login on the server that grants full rights. Especially if your website is hosted on a simple server, the functionality of more than either SSH or FTP login is not provided.

There is the additional problem that it is very insecure if every person who just needs to manage the authorization data has full read/write access to all the files on your website. Also apart from that fact a primal negative thing is that you need advanced skills for this kind of administration. You need to know how the .htaccess configuration really works. This is not for novices. There are many things that can be done wrong for example it could damage the whole website or it opens dangerous security holes on the website.

But you can also implement a web front end for all work that have to be done regarding the .htaccess file. With this option, you have the advantage that you can provide only necessary functions on your web front end but no improper or harmful ones. You can also ensure that only things can be done which make sense. A web front end is an addition to a graphical user interface (GUI) that makes the functionality, e.g. editing of files, available for the user over the internet. It is an interface between a person and a computer program that uses a web browser. That means that only a web browser is necessary — no other software. It can be realised by PHP scripts, Common Gateway Interface (CGI) scripts, etc.

The .htaccess solution compared with the PHP/database solution has the advantages that it is not so complex and you can also use it if you do not have a database available — only the possibility for execution of own scripts have to be allowed. It is also sufficient for a structure of rights on the website that is not so complicated. But if not, this solution becomes tricky very fast. You have to care about lots of files and directories and sooner or later it becomes very confusing.

Unfortunately, there are also some negative aspects of the .htaccess solution with the web front end. If you edit the .htaccess file via the front end with the help of scripts, you have to grant writing access to it. This is problematic in terms of the server’s security. Because if a script has the right to write on the web server’s file system, a user with bad intentions or a faulty script can corrupt the data on the web server. That means the script for the web front end has to be programmed very carefully. Thus on websites where a high level of security is necessary, the use of the .htaccess system is not recommended.

Finally, the possibility to do the management of the authorization with the help of a web front end for the .htaccess files is a practical solution for small websites with low security requirements to provide an easy to use, fast to learn and fault-avoiding system for users with average knowledge. For example, a realisation of this can be found in the .htadmin lite program [Nap04].

Stefan Worm
3.3 Choosing an Approach

We have the following situation regarding to the authentication:

If we use the web server for authentication, we can access its user database. For the T: EaS system we can use a server of the IT group of the Chemnitz University of Technology (TU). This server holds all the data for our target group (students and employees of the TU Chemnitz). Therefore, we do not have to care about authentication in the web application (T: EaS), because in this special case the web server on which the application runs offers this service for us. The authentication database is maintained by the IT group. If the user gets access to the T: EaS system it can be assumed that he is authenticated correctly — otherwise the user would not get access to just the homepage of the T: EaS system.

It is also possible to use an seperate database to handle the authentication yourself. This is a commonly used approach but this would involve more work, which has to be done (creating and maintaining a user database). If you do not have an existing user database, with which you can check the authentication data, it would be the better solution to implement the authentication in your application. Then the authentication and authorization could be done by one application, but in this special case the usage of the given database is the best solution, because we have an existing database and so we can use the approach to split the areas where authentication and authorization take place.

The approaches with the different databases in section 2.1 do not fit the requirements because they are not powerful enough, too inflexible or you have to perform the authorization stringendly with them. The .htaccess approch in section 3.2 is not fine-grained enough.

Because of that, it is necessary that we look for a piece of software (application, function library) for this job.

Regarding to this requirements the software phpGACL was chosen — it will be explained in detail in section 4.

4 phpGACL — PHP Generic Access Control List

4.1 Presentation of the phpGACL System

In this section, the realisation of authorization with the help of a function library named phpGACL (generic access control lists with PHP)[Ben] in version 3.3.4 (released 08-December-2004) will be described. It is a SourceForge project that consists of a set of PHP functions that offer the possibility to integrate a very fine-grained rights management into one’s website. This system is a very flexible and complex one that is rather for bigger website than for a website with only a few pages.

[^5]: http://SourceForge.net/
But how does it work? First of all you need PHP support on your web server as well as a database like MySQL. But compared to the solution in section 3.1.2 it is not necessary to create the structure of tables in the database yourself, because this is done by phpGACL. The only thing the webmaster has to do is to design the characteristically structure of the rights hierarchy. That means the he or she has to think about how to organize the rights in a useful way, it has to be paid attention to these and other questions like: 'Do I organize the users in groups?', 'How many groups will I need?', 'What kinds of rights do I need?', 'In what way shall they be granted to a group or a single user?', etc. If this is done, you must input the structure that you have preconceived. This can be done by the administration front end that is included in the phpGACL function set. There you have to input everything manually — every user, every group, every right and every assignment of rights to users, groups and other objects. This is practical for a small amount of users or if you want to change only a few things. Unfortunately, the usability of the administration interface is not very good, some initial training is necessary. But if you use phpGACL for a large amount of data (many objects have to be entered) or you want to customize it to your own ideas you are able to use the application programming interface (API). With this you have the functionality to handle all the necessary data in the system. Now you are also able to integrate this authorization system into your website. For example if you have a section on your website where you perform all actions which have to be done if you add a new user to the website, you can simply insert a few lines of code for the integration of the user in the phpGACL system via the API.

If you want to use the functionality of the phpGACL authorization system on a webpage, you only have to do two things.

The first is to integrate the phpGACL include file on your webpage (1) and create a new object of the phpGACL class (2).

```php
(1) include('phpgac1/gacl.class.php');
(2) $newobj = new gacl();
```

Listing 2: includes for phpGACL

The second is to use the acl_check() function in the section of your page where you need the authorization. This can be the entire webpage or only part of it. You can also use it on different parts of a webpage with different authorization conditions. This is one aspect of the flexibility named at the beginning of this section. The simplest way to use the function is in connection with an if statement, so that you get an expression like this one for example:
if ($newobj -> acl_check(RIGHT, USER [, optional_OBJECT ] )){
    /* authorization is successful: 
    you can do what you wanted to do */
} else {
    // 1. do nothing or 
    // 2. display an error message or 
    // 3. perform alternative actions
}

Listing 3: use of the acl_check() function

In the code example above the acl_check() function checks if the logged in USER has the necessary RIGHT on the OBJECT — the test upon the OBJECT is optional. A comprehensive explanation of the function’s parameters and their meaning is given below.

If the statement is true, this branch will be executed wherein the developer of the webpage can write the desired code for part of the application that has to be protected.

If it is false there are three possibilities that could be performed. The first is that this branch does not contain any code and as a consequence of it no code will be executed. This is suitable for example if the user does not have access to an unessential part of the page, e.g. an additional picture where the absence of it will not be serious. The second is that an error message is displayed. For example in the situation where the user wants to gain access to a section of the webpage which is certainly not predetermined for his or her eyes. In addition, the third possibility is, that an alternative content of the secured section will be displayed. This is practical for example if a chart with all details should be shown to a person with the necessary rights and for all other persons the chart should be shown in a modified way, e.g. depersonalized or with fewer details.

If you use this strategy you are able to personalise the website entirely for a user in a very easy way. For example to use the acl_check() with the possibilities explained above for small parts of a webpage, for single objects like buttons, pictures or web links which are displayed or not depending on the user’s rights or for any other content on your whole website. Even the design can be changed depending on the user or the user’s group.

With phpGACL it is not only possible to organize the users in groups, it is also feasible to create a structure of administrators. With this you can have administrators for different sections and/or with a different level of rights. It is no longer necessary to grant full rights to subarea administrators. Now you can also allocate the duties to different people without having the fear that they can do things which they are not allowed to do.

The phpGACL system does not have its own authentication module because it does not have the focus on that as there are a large amount of applications which are designed especially for this and doing its job well. Therefore it is not necessary to reinvent the wheel. Without doubt, authentication is necessary to know if a user really is the one he or she pretends to be. But how does the system know which user is on the other side of
the internet connection? It uses the authentication data that were passed on to it by other applications. For example, you can use those of the web server, an external service, a third party application or your own one. If you use e.g. the web server ones and PHP, you are able to access the user’s name via \$_SERVER[ ‘PHP_AUTH_USER’ ]\(^6\). Now you have the user’s identifier (name) and the information that this is valid, because of the verification by the authentication process, you can work with that in the phpGACL system.

The last issue that has to be discussed is how the data structures for the authorization in phpGACL are organised. The system consists of the three elements "Access Control Objects" (ACO), "Access Request Objects" (ARO) and "Access eXtension Objects" (AXO).

To describe in detail [BRD04]:

ACO Access Control Objects are the actions that are requested. For example these can be so called 'rights' like read, write or add.

ARO The Access Request Object is the thing which requests access, e.g. to a secured section of the webpage. In the easiest way this can be a user, e.g. Alice Murphy (username = amu).

AXO The third object, the Access eXtension Object is the data structure on which the access needs to be controlled — it is optional, contrary to ACO and ARO which are required. For most reasons the first two are enough to model a structure for authentication. But for the realisation of the authorization management for the T:EaS system a third dimension is necessary. Then, an AXO is for example a specific project or department like project: new invention or department: security issues.

At this point, you can do the following check operation, provided that all the necessary entries regarding the ACO, ARO and AXO were done in advance.

```
 acl_check ( 
 'action' , 'edit' , /* ACO*/
 'user' , 'amu' , /* ARO*/
 'project' , 'new_invention' ) /* AXO*/
```

Listing 4: parameters of the acl_check() function

Now you are able to integrate the phpGACL solution in your web project to manage the authentication at every point of your website where it is necessary. Compared to the .htaccess solution from section 3.2 no further security risks have yet appeared — there is the same security level like any other PHP script with database access.

\(^6\)http://de2.php.net/features.http-auth
4.2 Similar Applications to phpGACL

At this point of the paper some solutions are presented that have similarities with the program phpGACL. It was not possible to find programs written in PHP that achieve almost the same like phpGACL — no authorization is needed and authentication that is very flexible and comfortable. Nevertheless, if an application fulfills the two following conditions it could be used likewise phpGACL for the extension of the T:EaS system.

- It should be possible to handle the authentication yourself by an external component or the program uses your existing, external database for that. The program does not force using the usernames and passwords in its own data structures, which is important especially when the username/password is already stored in specific database.

- It should be possible to store a minimum of three combinations of objects in the programs database. For example a username, a right and a object (e.g. user, read, course no. 23). It is not enough to have for example only two combinations like user and right (e.g. user, read courses).

The program LiveUser [W+05] accomplishes those two conditions as far as they can be evaluated by checking the online documentation only. It can use external databases for authentication but it is not possible to skip this step. As a consequence you have to do some configuration work. Another negative aspect is that PEAR\(^7\), a framework and distribution system for reusable PHP components, has to be installed on the server. Apart from that LiveUser seems to be prepared to meet the requirements as an authentication system.

Because of the big effort to analyse this program and other alternatives in detail they could not tested in operation. The selection of the programs does not mean that there maybe others that are better than the mentioned ones.

Among many others, the following programs / function libraries exist, but they do not fulfill the basic requirements listed above (1), there are other problems like the fact that it is only possible to secure entire files or directories (2), that any further development has been stopped (3), there is no documentation (4) or they a too complex and authentication is only by-product (5).

- PHPSecurePages (1) http://www.phpsecurepages.com/

- patUser (1) http://www.php-tools.de/ > patUser

- PHP-Auth (2) http://sourceforge.net/projects/auth/

- FACL (3)(4) http://www.regfish.com/persons/amallek/php/facl/

- pphplib (5) http://sourceforge.net/projects/pphplib/

---

\(^7\)http://pear.php.net/
As you can see there are no applications or function libraries that are real alternatives for phpGACL. This is surprising, because usually there is more than one implementation of a specific problem, especially if it is as common as this one and a solution for it is needed very often (cf. section 3.1). On SourceForge you can find several projects that tried to do the same as phpGACL and the descriptions of the projects sound good but either they were never completed (phortify) or never even started (mr_Authorize, PAAS or Personna).

5 Integration of phpGACL

The following sections are about the T:EaS system itself, how it works, how the phpGACL system was integrated and how they work together, as well as additional implementation issues which are necessary to make it perfect.

5.1 The T:EaS System

A short introduction to the T:EaS system and a description of the problem was made in section 2.1. In addition to that, the following facts are also important. This management system for the organisation of all relevant details regarding courses is designed as one platform for students, professors, scientific assistants and the staff of the deanship.

It works that way: A professor or a scientific assistant creates a new course. He or she enters all necessary data like date, subject, room and maybe requirements for it. The deanship reviews the course record and approves it if everything is ok. After this the students can register for the course, choose a project team and look for the results of tests and end of term exams.

From the technical view the T:EaS system itself, without the phpGACL, is realised with PHP scripts, a MySQL database and .htaccess protected directories. It is one directory for every user group which is protected by different .htaccess files. In this directory it is stored the PHP code for the functions that every user of a group is allowed to execute. The problem is that the .htaccess files have to be managed separately by one person and that the PHP code which is necessary for every group has to be put in every directory which makes the maintenance and bug fixing difficult (see section 2.1).

5.2 What was done to integrate phpGACL in the T:EaS System?

First of all, the usual procedure of installation and configuration of phpGACL was uncomplicated. It does not have to be paid attention to anything special, so it is not

---

8 http://sourceforge.net/
9 http://sourceforge.net/projects/phortify/
10 http://sourceforge.net/projects/mrauthorize/
11 http://sourceforge.net/projects/paas/
12 http://sourceforge.net/projects/personna/

Stefan Worm
necessary to elaborate this matter. This can be looked up in the manual [BRD04, p. 18]. The phpGACL installation script creates the necessary database tables into the existing database of the application, but it can also use an extra database for this.

After this you have to include the functionality of phpGACL in your application with the declaration of the gacl.class.php and gacl_admin.inc.php files.

```php
include('phpgacl/gacl.class.php');
include('phpgacl/admin/gacl_admin.inc.php');
```

Listing 5: includes for acl_check()

The best place for this is the global include file of the application so that you do not have to do the include in every file. Although there is one thing you have to do in every file of the application. It is the instantiation of a new object for the phpGACL operations, for example with the name $gacl1:

```php
$gacl1 = new gacl();
```

Listing 6: instantiation of a new gacl() object

The T: EaS system consists of many IF/ELSE clauses that contain one functional part of the application each. Therefore you have to add one gacl_check() at the beginning of every IF clause to secure the parts that you want.

For example the part that contains the code for deleting a course has to be modified from the original version

```php
} elseif ($_GET['cmd']=="del") {
...
} elseif ...
```

Listing 7: functional part without acl_check()

to the version with the phpGACL

```php
} elseif ($_GET['cmd']=="del" &&
$gacl1->acl_check('action','del',
'user', $SERVER['PHP_AUTH_USER'],
'lv', 'lvid'. $_GET[lvid]) ) {
...
} elseif ...
```

Listing 8: functional part with acl_check()

that checks if the action delete (del) is allowed to be performed by the user (that is authenticated by the web server, 'PHP_AUTH_USER') on the course (lv) with the ID $_GET[lvid].

Stefan Worm
5.2 What was done to integrate phpGACL in the T:EaS System?

The API functions

With these quite simple modifications, most of the work is already done to add the phpGACL functionality to the T:EaS system. Although, there are some other parts that have to be modified as well. Mostly, parts like the rights management of single users that was added, especially to make the T:EaS system more flexible. At this point it is necessary to implement several functions for that part by using the API to improve the usability for T:EaS users.

The use of the API is a little complicated. The automatically generated documentation of the phpGACL API functions leaves several questions unanswered. Nevertheless the API functions are necessary e.g. for the usability of the T:EaS system in tandem with phpGACL. For example, if you add a user and you want to give him or her some rights, you have to do this with a number of extra steps via the phpGACL administration interface or you can do this with one easy step integrated in the T:EaS’ interface, realised with the help of the API functions. This is also good for the integrity of the data, because it cannot happen that you create a user and don’t give him or her not even one right. If you try to do this the application will warn you and at the end it will not create the user. An example for a set of API operations for creating a course and assign rights for it is available in section A.1.

Another thing that was done is to bring together similar parts of the source code from different files, so that you have to edit only one section of the code if you want to change something.

User groups

The users of the system are organised in four different groups with specific rights.

The group with the fewest rights is students. Every user of the system who has been authenticated successfully has at least the rights of this group. Users can sign in and out for courses, can look up their results in tests and exams, as well as sign in for specific topics of the course or the user can join a special interest group within the course. In the T:EaS system this part of the application does not have to be modified with phpGACL functions, because no functions have be protected there and the authentication via the web server ensures that all actions of a user can be assigned to a specific person. Nothing more is necessary, hence no modifications have to be made in this section of the application.

The second important group is professors and staff. They have the right to create a course and if they do so they get all rights regarding to this course. It is also possible for them to add other tutors and assign rights to them regarding the course. The creator can assign only a few rights or all possible rights to a person. This can be another professor or staff or a student assistant.

The third group is student assistants. They have the rights that were given to them by the creator of a course. This could be all the rights, except the right to create a new course (only professors and staff can do this), so he or she can manage a course on his or her own, or only a few rights with which he or she can just support the performance...
of the course. An overview of the rights that can be assigned to a person can be seen in table 1.

The last group in the system is *deanship*. This group has only a few members. They have all rights within the system. They can do whatever they need to, for example changing the details of courses or the assignment of tutors, as well as the tutor’s rights. But the most important task of the *deanship* is to approve the courses. If a course was created successfully and all necessary details were entered, it is not possible for the student to register for it right away. First, the *deanship* has to review if everything is correct. If so, the course becomes approved and the students can register for it. The members of this group can also add and remove users from the *professors and staff* group.

<table>
<thead>
<tr>
<th>rights</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>approve</td>
<td>to approve a course</td>
</tr>
<tr>
<td>admin</td>
<td>to add/delete a tutor of a course, modify the rights of a course’s tutor</td>
</tr>
<tr>
<td></td>
<td>(also modifying own rights)</td>
</tr>
<tr>
<td>view</td>
<td>to view the main facts of the course</td>
</tr>
<tr>
<td>edit</td>
<td>to edit the main facts of the course</td>
</tr>
<tr>
<td>del</td>
<td>to delete a course</td>
</tr>
<tr>
<td>copy</td>
<td>to copy an existing course to create a new one which is identical</td>
</tr>
<tr>
<td>create</td>
<td>to create a new course</td>
</tr>
<tr>
<td>export</td>
<td>to export the data set of an existing course as an extensible markup</td>
</tr>
<tr>
<td></td>
<td>language (XML) file</td>
</tr>
<tr>
<td>import*</td>
<td>to import the data set of a course</td>
</tr>
<tr>
<td>matrix</td>
<td>to issue credits for successful passing of intermediate exams and the final</td>
</tr>
<tr>
<td></td>
<td>course exam</td>
</tr>
<tr>
<td>achievements</td>
<td>to create and edit intermediate exams which have to be passed during the course</td>
</tr>
<tr>
<td>topics</td>
<td>to create and edit the topics and the members of the study groups regarding the course; set deadlines</td>
</tr>
</tbody>
</table>

*a this is not yet implemented in the T: EaS system

Table 1: Rights of the T:EaS/phpGACL System

**Effects of the Installation**

Besides the general advantages of the T:EaS and phpGACL system and its combination, there are several other effects. One is that it is necessary to have a chief administrator, because the administration interface of the phpGACL system cannot be protected by itself. In the T:EaS system, an extra directory was created that is protected by a .htaccess file. Hence, only users that are registered in the .htaccess file have access to
the web interface of phpGACL. This is not a problem, because all daily routine tasks can be done with the T:EaS system. That means that granting and denying of rights and other things can be done without the phpGACL interface. Nevertheless, the chief administrators have the right to modify all data sets in the phpGACL system, which means that they can add/delete users, their rights, the courses, the user groups, etc.

The chief administrator has to look up the interface very rarely. He or she needs to use it only for changing members of the deanship group or for fixing possible problems.

Another result of the integration of the two applications is that the database table, in which the assignments of courses and tutors are stored, no longer needs to be misused for rights management of users and courses.

5.3 What else could be done in the future?

The combination of phpGACL and T:EaS as a system for easy and reliable support of course management is not perfect yet. Putting aside the fact that a piece of software is never perfect — there are always things that could be done to improve it, even if it seems to be finished.

Although, the application is ready so far and you can use it productively, there are a few things that could be implemented in addition. These are not really necessary but they would enhance it in a useful way.

**Init script** Generally, the first thing that could be added is an initialization (init) script that brings the phpGACL system in a well defined state, so it is faster (not every single step has to be done by hand) and less fault-prone to set up the installation.

However this is not essential because the script would have to be adapted to the needs of the phpGACL system. It is not possible to create an init script that is general enough to be used otherwise. Furthermore, the integration of the two applications is done. If you develop further the T:EaS system you have to do it in consideration of phpGACL in the future. If you do not care and you want to integrate phpGACL in a separately new developed version, the init script would not fit in any case. Therefore, an init script is useful, but the expensive development, because of the difficulty to manage API functions, does not justify its benefits at the moment.

**Overview** A good thing regarding the ease of use is the possibility to check one’s rights also at the web interface. Not only for administration but also for a person itself, so that the user can check for example in which course he or she is registered as a tutor and what kind of rights he or she has. The result can be presented clearly on one webpage so that the user does not need to check all courses to find out what rights he or she has regarding them.

**Hard coded Problems** Another fact is that names are hard coded in the sources of the PHP files. That affects primarily the names of the groups deanship and professors_and_staff, as well as the group lvgroup in which all courses are included. Fur-
thermore also the names of the rights (view, create, del, ...) that were used with the `gacl_check()`-function are hard coded. So if you want to change one of the names, you have to replace it in every file it occurs. Storing these names in the global include file would solve the problem only at first sight. It would be easy to change it at one single point but that would be only a partial success. Because of comprehensibility of the source code you would name the variables for the group names similar to them. If you need to change the names of the groups one day, which is the basis of the problem, the source code would become very hard to understand because the names of the variables would not fit the meanings of the groups anymore. To avoid this you would have to change the names of the variables — as a consequence of that you move in a circle.

That shows a problem as a matter of principle. It is not possible to formulate a usable approach that keeps T:EaS and phpGACL disunited in an easy way. The integration of phpGACL in the system that should be enhanced by that is a very significant step that cannot be revoked easily. If you decide to use the phpGACL system in an application, the code for it will become an essential part of it. It is soonest comparable to a function library that normally cannot be removed easily either.

The only option is to add an intermediate layer\(^{13}\). With this you have the possibility to solve the above-mentioned problems, but it is usually very complex and another point where mistakes could be made that would endanger the reliability of the system. If you add the additional layer in advance, you have more work with the integration of phpGACL and T:EaS. But if you create the layer at the time when you need it (e.g. use of another authorization system), it will not be significantly less work than changing the whole system altogether.

**Export Courses** The T:EaS system offers a possibility to export the data of a course in a XML file. That apparently does not include information for the rights management of phpGACL. If you import\(^{14}\) a course it would be without any rights management data. In a future version of T:EaS this aspect of exporting could be implemented. However, you can manage the problem by feigning that the imported course is one which was created just now, which means that you add the person that imports the course as a tutor and assign all necessary rights to him or her. After this the person can add more tutors if necessary. This way is feasible, because you do not use the export function for a real backup of all data. For this, the chief administrator have to backup the database and the scripts with the source code separately.

The export function is mostly used in the case that you want to give the course in a year or two again. Then, the tutors will have changed most likely and the rights information of the "old" course are not valid anymore.

---

\(^{13}\) An old maxim in computer science states, "There is no problem that cannot be solved by adding another layer of indirection."

\(^{14}\) Import: this function is not yet implemented in the T:EaS system
6 Conclusion and Outlook

This work deals with the problem to find and rate a solution how to administrate access rights in web based applications that are flexible and offer a fine-grained allocation of rights.

It is possible to find an application / function library based on PHP that offers these features. The system with the name phpGACL is a very flexible one that can be customized to the needs of an existing application quite well. It is suitable for a large amount of users and other authorization objects that have to be administrated. This system primarily has been created for the treatment of authorization data, it does not have an authentication unit included. However, this is not necessary because it shall be used in combination with T:EaS, a system for managing university courses and its students, which is responsible for the authentication.

This paper also shows the attempt to improve the access management of the web server itself, but at the end it has been exposed that the adding of a web front end for the .htaccess configuration does not achieve the desired performance. Also the widely used approach to implement one’s own version of an authorization and parts of an authentication system does not bring more benefits than the phpGACL system apart from a lot of work. Only through the API functions of phpGACL it is possible to build a very user-friendly webpage that you can adapt to your demands.

Finally the phpGACL solution can also be recommended for websites with forum systems that have traditionally a lot of users with many different roles and many content for which a specific structure of access rights is necessary. Besides that it could also be used for community websites, e.g. business platforms or dating agencies where every single user can decide which information he or she wants to enable for specific other users. This would not be a problem for phpGACL even if the websites would implement new features with new demands, because it can be extended very well.
This section is about examples of the use of phpGACL in connection with the T:EaS system. The source code listed below is almost the same like in the real T:EaS system. It was modified a little because of didactic reasons mostly regarding variable names and the names of self defined functions.

A.1 Exemplary API Operations for phpGACL

In this subsection you can see the source code listing for creating a course and assigning all rights for this course to the tutor that has created it. The explanation of the API functions is a part of the documentation [Ben04] (the beta version of the new API is also available in an online version [Ben05]).

This code fragment is from the file index.php that is in the folder mitarbeiter (German for: professors and staff) of the T:EaS application.

Some introductory comments in advance:

- **variables**
  - $id11 integer: internal identifier that was assigned by phpGACL to the course
  - $gr_id integer: number of the group "lvgroup"
  - $succ boolean: its value indicates if the execution of the function was successful or not
  - $right1 array: list of rights
  - $obj_id integer: identifier assigned by phpGACL to the user
  - $acl_no integer: number that was assigned by phpGACL for this ACL
  - $iid integer: the real number of the course that was assigned by the database of the T:EaS system
  - $_SERVER[‘PHP_AUTH_USER’] string: name of the authenticated user

- **$gacl_api functions**
  - $add_object inserts a new object
  - $get_group_id finds out the group_id of a given name
  - $add_group_object assigns an object (e.g. course, user, etc.) to a group
  - $add_acl creates a new ACL
A.2 Example for Customization of a Website

In the listing below you can see a possibility to customize the presentation of a webpage for users with different rights. Depending on the user’s rights he or she gets presented the entire website or only part of it.
This code fragment is also from the file index.php that is in the folder mitarbeiter (German for: professors_and_staff) of the T:EaS application as in section A.1.

```php
if ($gacll->acl_check('action','approve','user',
$_SERVER['PHP_AUTH_USER'])) {
    h2("Overview of all courses: ");
    list_LV_edit(mysql_query("select * from lv 
        order by lvid desc"));
} else {
    h2("Overview of courses you are a tutor for: ");
    list_LV_edit(mysql_query("select * from lv, tutor 
        where tutor.nkz='$_SERVER[PHP_AUTH_USER]' 
        and lv.lvid=tutor.lvid order 
        by lv.lvid desc"));
}
if ($gacll->acl_check('action','create','user',
$_SERVER['PHP_AUTH_USER'])) {
    printlink($_SERVER['SCRIPT_NAME'],"cmd=new","Create a new course.");
}
```

**Listing 10:** use of acl_check()

1: checks if the currently authenticated user has the right approve
2: prints a header by means of the self-defined function h2()
3: prints the result of the mysql_query() which selects all courses from the database in descending order by means of the self-defined function list_LV_edit()
4: same as line 2:
5: similar to line 3: but this time only the courses for which the currently authenticated user is a tutor are printed out and not every single course
6: checks if the currently authenticated user has the right create
7: prints a link to a webpage where you can create new courses by means of the self-defined function printlink()

This short code example above shows how you can present different content on a webpage depending on the user’s rights. If the user has the right approve which have normally only members of the group deanship then he or she can see a list of all courses that are stored in the database. However, if the user does not have this right, he or she can see only the courses for which they are a tutor. If he or she is not a tutor at all for one single course then nothing will be displayed.

The program code in line 6 and 7 has the effect that the user can see the link if he or she has the right create, otherwise the link is hidden.

Stefan Worm
The `gacl_check()` functions in line 1 and 6 are without the AXO check for the course, because no specific course can be selected here. Therefore it would not make sense to perform the `gacl_check()` for all courses in the database. It is more efficient to use the assignment of user and course from the database table (cf. 5).

B Screenshots and Explanation of phpGACL Interface

This section is about the administration interface of phpGACL and its initial configuration so that it can be used in combination with the T:EaS system. The screenshot will show interesting parts of the system by means of a typical system that was created for testing. They are modified (shortened) a little for didactic reasons.

The usernames that are stored in the phpGACL system are the login names that are assigned by the IT group of the University to every user, like `amu` for Alice Murphy (cf. section 2.2).

The explanation of the use of the interface in detail is explained below each screenshot picture with the help of examples.

In the following section it will be explained for what you have to pay attention on if you set up the system yourself and what needs to be done to prepare phpGACL for the T:EaS system:

1. You have to be careful with the names of the groups (ARO and AXO), sections (ACO, ARO and AXO) and the names of rights (edit, delete, etc.). Because they are hard coded keywords that are used in the T:EaS system and cannot be changed (see section 5.3, p. 19).

2. First it is necessary to create the rights for the T:EaS system. In this case a ACO Section with the name `action` has to be created first. There you can add the rights by creating an Access Control Object (ACO) for every one of them. You need the rights: `approve, admin, view, edit, del, copy, create, export, import, matrix, achievements` and `topics` (cf. Table 1, p. 18). You can do this on the ACL Admin website of the phpGACL interface (fig. 8).

   You also have to create the object `user` in the ARO Section and `lV` for the courses in the AXO Section. Users or courses do not have to be added at this point. You can do this later by hand (see below) or with the help of the functions in the T:EaS system.

3. Second you have to create three groups with the names `deanship`, `professors_and_staff` and `student_assistants` as ARO objects (fig. 1). The creation of the dummy group `persons` is not necessary but it is helpful for clarity of the structures. The name of the group `persons` can be changed.

Stefan Worm
a) In the group deanship (fig. 2) you have to enter the names of the persons who should be member of it. Usually that are the dean, vice-dean for collegiate affairs, an office clerk and maybe other people that are entrusted with tasks in this field of work. Registering and signing out of members of this group is the only thing that has to be done as regular maintenance work in the phpGACL administration section. All other work besides this initial configuration can be done with the help of the functions in the T:EaS system.

b) In the group professors_and_staff no names have to be entered. This can be done with the help of functions in the T:EaS system. Nevertheless you can go the not recommended way and enter the names of the professors and staff at this point (see fig. 7). If you do this you have to assign all the entered names to this group (fig. 4). That has the advantage that it is a little faster but there is no security check if the usernames are correct.

c) In the group student_assistants no names should be entered, because they were added automatically when a student assistant has become a tutor of a course (fig. 3).

4. The next thing that has to be done is to create the group lvgroup for the courses as an AXO object. Creating a dummy group for example with the name root in advance is also a good idea at this point (fig. 5). From now on all courses will be organised in this group automatically, so it is not necessary to enter something by hand (fig. 6).

By means of the first screenshot (fig. 1) several things were explained that are equal or similar in other parts of the administration interface as well. First, the navigation menu of phpGACL contains the following entries — besides those that are available as a screenshot:

**ACL Test** offers you the possibility to perform the `gacl_check()` function with self chosen parameters. This is good if you want to quickly check if something works or not.

**ACL Debug** performs a two-dimensional (ACO and ARO objects) or three-dimensional (ACO, ARO and AXO objects) check of every combination of an object with another object and shows the result if it is `true` or `false`.

**About** contains among other things the credits of the program.

**Manual** contains an explanation of the way how phpGACL works by the mean of a tutorial with Star Wars® characters. Instructions for the installation of phpGACL are included as well.

**API Guide** is an automatically, from the source code of phpGACL generated, documentation of the API interface.
The ID number of a group is assigned by phpGACL and cannot be changed. The Name of an object is the unique identifier for it. It is the string which phpGACL uses for its internal operations. The Value of an object can be any string you like. It should more clearly describe the Name. Because of simplicity Name and Value are always the same in the examples of the screenshots.

The numbers in the Objects column shows the number of members in the group. In the Functions column following operations can be performed:

**Assign ARO** is for editing the members of the group. You can add or delete users, as well as editing the values of users that are already assigned to the group. You get to one of the sections shown in figures 2, 3 or 4.

**Add Child** offers the possibility to create a subgroup of this group.

**Edit** means that you can edit the Name and the value of the object (group).

**ACLs** show all access control lists that are connected with the object (in this case with the group). It is similar to figure 9.
A form similar to this (fig. 2) is often presented to the administrator in the phpGACL interface. It works the following way: Some self explaining functions regarding a form are usually on top of it (e.g. Edit or Search). If you want to select an object inside a form, you have to left-click on it with the mouse and then you need to click the button with the two right angle brackets (»). The object is now listed in the Selected form. When the selection is complete, you have to press the SUBMIT button to store the data. If you want to deselect an object, use the right mouse button. To select more than one object, use the left mouse button and the CTRL or SHIFT keys.

Underneath the forms you can find a table with the members of the group. If you want to delete one of them you can use the check buttons in the last column and the button REMOVE.
Figure 3: Assign Group - ARO (student assistants)

The figures 3 and 4 are very similar to figure 2 that was explained above.
Figure 4: Assign Group - ARO (professors and staff)
Figure 5: AXO Group Admin

The screenshot of the AXO groups (fig. 5) shows only one group that contains the courses. The two letters LV are the German abbreviation for course. It is the term that is also used in the T:EaS system and has therefore not been changed.

For a detailed explanation of the screenshot please refer to the explanation of fig. 1.
The screenshot (fig. 6) shows the table of courses that are registered for the sample application. The numbers for the courses result of values from the T:EaS system (cf. $iid$, section A.1).

Figure 6: Assign Group - AXO (courses)
Figure 7 shows a form for editing ARO Objects (user). The forms for ACO (action) and AXO (courses) are almost the same. It allows editing the existing objects and adding new ones as explained in the text for figure 1 (p. 27). An interesting, but in the beginning slightly confusing, thing is the column *Order* which value determines the sequence of objects. Depending on the numbers that you have to choose if you enter the objects (usernames, rights) yourself, they appear in ascending order in all forms (e.g. *ACL Admin*, fig. 8) — objects that were created with the help of the API get a number automatically. This is good for keeping the overview and slightly speeding up work. If you assign small numbers to often used objects, they will appear on top of the list in the forms — with this you have fast access to these objects.
The webpage of phpGACL where you can define an ACL yourself (fig. 8) is the most complicated one but with the knowledge of the previous ones it is not so hard to understand. It works that way: You start in the left upper corner. You select an entry of the ACO Section and the Access Control Objects you want. Confirm it by pressing the » button. Then you do the same for the AROs and AXOs. There it is also possible to select a group just separately or in addition to other objects, e.g. if you want to create an ACL that should effect the users A, B and C as well as all users of the group deanship. If you press the SUBMIT button the ACL will be saved.

If necessary you can choose some additional options.

- With Access allow or deny you can choose if your selection of the ACL should be interpreted as if it allow(s) or deny(s) something.

- Usually you create ACLs that allow a certain action or function and that is why the option allow is preselected.

- If you want to abrogate an ACL temporarily you can unselect the Enabled option.

- The ACL Section in the left down corner is to change the classification if the ACL was created by a user or via the API functions by the system. Generally you do not need this configuration. In practice it does not matter if an ACL is a user or system one.

- Normally the acl_check() function returns true or false. If you enter something in the Extended Return Value form, the acl_check() function will return this.

- The Note form is simply for notes regarding this ACL.

The same webpage is used for editing ACLs.
Figure 8: ACL Administration Page
The last two screenshots (fig. 9, 10) show the webpage of the phpGACL administration interface that is for a general overview over all ACLs in the system. If you have a lot of ACLs then it is helpful to use the Filter function. With this you can select for example all ACLs where the user tkus is a tutor if you enter tkus in the column ARO and the row Object.

If you use the Edit link in the Functions column you get to the ACL Admin forms for this ACL with all current options preselected. There you can make changes as explained in the text for figure 8.
<table>
<thead>
<tr>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
<th>action</th>
<th>Groups</th>
<th>LV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>admin</td>
<td></td>
<td></td>
<td>1.</td>
<td>approve</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>edit</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>export</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>import</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>matrix</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>view</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. htor</td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. mid60</td>
<td>ALLOW</td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>User</td>
<td></td>
<td></td>
<td>Return Value:</td>
<td>User</td>
<td></td>
</tr>
</tbody>
</table>

- This ACL is necessary to secure the webpage of the dean's office (no course number is assigned). For this, the existing right APPROVE was used that is assigned only to dean's office members.

<table>
<thead>
<tr>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
<th>action</th>
<th>Groups</th>
<th>LV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>create</td>
<td></td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. profs_and_staff</td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. deanship</td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. mid60</td>
<td>ALLOW</td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>User</td>
<td></td>
<td></td>
<td>Return Value:</td>
<td>User</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>export</td>
<td></td>
<td></td>
<td>1.</td>
<td>copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>abis</td>
<td></td>
<td></td>
<td>1.</td>
<td>create</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>del</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>edit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>import</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>matrix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>topics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>view</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. abis</td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. mid59</td>
<td>DENY</td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>System</td>
<td></td>
<td></td>
<td>Return Value:</td>
<td>System</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
<th>action</th>
<th>user</th>
<th>Groups</th>
<th>LV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>copy</td>
<td></td>
<td></td>
<td>1.</td>
<td>del</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>abis</td>
<td></td>
<td></td>
<td>1.</td>
<td>edit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td></td>
<td></td>
<td></td>
<td>1.</td>
<td>view</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. abis</td>
<td></td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1. mid65</td>
<td>DENY</td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>No</td>
<td>1.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value:</td>
<td>System</td>
<td></td>
<td></td>
<td>Return Value:</td>
<td>User</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: ACL List, part 2
Abbreviations and Acronyms

AC .......... Access Control — page 11
ACL .......... Access Control List — page 22
ACO .......... Access Control Objects — page 13
API .......... Application Programming Interface — page 11
ARO .......... Access Request Objects — page 13
AXO .......... Access eXtension Objects — page 13
CGI .......... Common Gateway Interface — page 9
DB .......... DataBase — page 5
FTP .......... File Transfer Protocol — page 9
GUI .......... Graphical User Interface — page 9
HTML ........ HyperText Markup Language — page 7
ID .......... IDentifier — page 6
IT .......... Information Technology — page 10
L:AU:S ...... (German) Lehre: Ausbildung und Studium, see T:EAS — page 1
MD5 .......... Message Digest algorithm no. 5 [Bis02, ch. 9.4], [Riv92] — page 6
MIT .......... Massachusetts Institute of Technology — page 1
MYSQL ........ open source RDBMS that uses SQL — page 5
PHP .......... PHP Hypertext Preprocessor — page 2
PHPGACL ...... Generic Access Control Lists with PHP — page 5
RDBMS ....... Relational Database Management System — page 38
SQL .......... Structured Query Language — page 38
SSH .......... Secure SHell — page 9
SSO .......... Single Sign-On — page 4
T:EAS ......... Teachings: Education and Study — page 1
TU .......... Technical University — page 10
URL ............ Uniform Resource Locator [BLMM94] – page 7
WWW ........... World Wide Web – page 2
XML ............ eXtensible Markup Language [Vas02] – page 18
References


[Ben04] Mike Benoit: *phpGACL 3.3.4 Developer’s Manual (API documentation)*, 2004. Included in the program’s tarball under the /docs/phpdoc/ directory.
URL http://prdownloads.sourceforge.net/phpgacl/phpgacl-3.3.4.tar.gz [p. 22]

URL http://phpacl.sourceforge.net/phpdoc/ [p. 22]


URL http://dict.tu-chemnitz.de/ [p. 1]


40
References


[TC05] IT Devision of the TU Chemnitz: *Apache: Access Control (AC)*, 2005. This document is in German language. URL [http://www.tu-chemnitz.de/urz/www/access.html](http://www.tu-chemnitz.de/urz/www/access.html) [p. 8]


# Index

## A
- ACL, 22, 23
- ACO, 13, 23
- Apache, 4
  - .htaccess, 4, 8
- API, 11, 17, 21, 22, 26
- application, 1
  - web based, 1
- ARO, 13
- authentication, 2, 3, 7
- authorization, 3, 7
- Authorization Programs
  - FACL, 15
  - LiveUser, 15
  - mr_Authorize, 15
  - PAAS, 15
  - patUser, 15
  - Personna, 15
  - phortify, 15
  - PHP-Auth, 15
  - PHPSecurePages, 15
  - pphpip, 15
- AXO, 13, 23

## B
- branch, 12
- bug fix, 15

## C
- C, 5
- C++, 5
- C#, 5
- CGI, 9
- code execution, 12
- cookie, 7
- course, 1, 15, 19
  - create, 17
  - delete, 16
  - lvgroup, 19
  - register for, 17
  - rights regarding it, 18

## D
- database, 5–7, 10, 20
- directive, 8

## E
- encryption, 6, 8

## F
- file, 7
- foreign key, 6
- form, 7
- FTP, 9

## G
- groups of the T:eaS system
  - deanship, 18
  - professors and staff, 17
  - student assistants, 17
  - students, 17
- hash, *see* md5
- homepage, 10
- HTML, 7

## I
- ID, 6, 7
- identifier, 13
- init script, 19
- IT, 10, 25

## J
- Java, 5

## L
- L:AuS, 1
- Liberty Alliance, 4
- LiveUser, 14
- login, 9
- lvgroup, 19

## M
- maintenance, 2, 5, 15
md5, 6  
Microsoft Passport, 4  
MIT, 1  
MySQL, 5, 11

O  
object, 11  
objects of phpGACL  
ACO, 13  
ARO, 13  
AXO, 13

P  
password, 6  
Pear, 14  
Perl, 5  
PHP, 2, 5, 11, 13, 14, 21  
phpGACL, 10, 14  
phpGACL frontend, 27  
About, 27  
ACL Admin, 27  
ACL Debug, 27  
ACL List, 27  
ACL Test, 27  
API Guide, 27  
ARO Group Admin, 27  
AXO Group Admin, 27  
Manual, 27  
phpGACL objects  
ACO, 13  
ARO, 13  
AXO, 13  
Python, 5

R  
RDBMS, 38  
realm, 8  
rights  
achievements, 18  
approve, 18  
copy, 18  
create, 18  
del, 18  
edit, 18  
export, 18  
import, 18  
matrix, 18  
topics, 18  
view, 18  
roles, 3  
Ruby, 5  
rules, 3  
  based on identity, 3  
  based on roles, 3  
  multi-level, 3

S  
security holes, 9  
session, 7  
  ID, 7  
  management, 7  
Sibboleth, 4  
SourceForge, 10  
SourceForge.net, 15  
SQL, 38  
SSH, 9  
SSO, 4  
Stellar™ Course Management System, 1

T  
T:EaS, 1, 15  
TU Chemnitz, 10  
tuple, 6

U  
URL, 7  
usability, 2, 5, 11, 17  
user-friendly, 21

V  
validation, 13  
verification, 13

W  
web browser, 2, 9  
web front end, 9  
web server, 6  
webmaster, 11  
World Wide Web, 2

Stefan Worm
Index

X
XML, 18, 20