

phpGACL

Generic Access Control List

Mike Benoit <ipso@snappymail.ca>
James Russell <james-phpgacl@ps2-pro.com>
Karsten Dambekalns <k.dambekalns@fishfarm.de>

Copyright © 2002,2003, Mike Benoit
Copyright © 2003, James Russell
Copyright © 2003, Karsten Dambekalns

Document Version: 42

Last Updated: **5/20/03 - 18:55:08**

Table of Contents

Advanced usage	21
Using the ACL admin utility	22
API Reference	23
ACL	23
add_acl()	23
edit_acl()	23
del_acl()	24
Groups	24
get_group_id()	24
get_group_parent_id()	24
add_group()	24
get_group_objects()	

edit_object_section()	30
del_object_section()	30
FAQ	31

About

What is it?

Introduction

Understanding Access Control

Han is captain of the Millennium Falcon and Chewie is his second officer. They've taken on board some passengers: Luke, Obi-wan, R2D2 and C3PO. Han needs to define access restrictions for various rooms of the ship: The cockpit, Lounge, Engines and the external Guns.

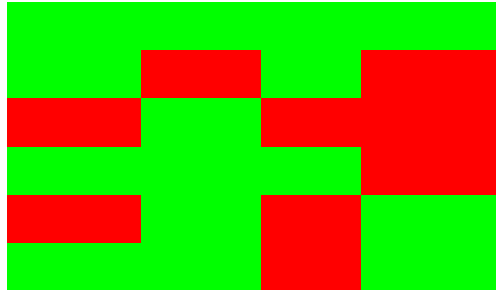
Han says: "Me and Chewie should have access to everywhere, but after a particularly messy hyperdrive repair, I forbid Chewie from going near the Engine Room ever again. Passengers are confined to the Passenger's Lounge."

Let's assume from now on that access is Boolean. That is, the result of looking up a person's access to a room is either ALLOW or DENY. There is no middle ground.



and lengthy adjustment to the matrix, and it' s a difficult task to verify that the final matrix is correct.

- It' s hard to summarize or visualize. The above example is fairly simple to summarize



Millennium Falcon Passengers

- └Crew [ALLOW: ALL]
 - └Han
 - └Chewie
- └Passengers [ALLOW: Lounge]

Another example of fine-grain control happens when the Empire attacks; Han needs to let Luke man the guns, and let R2D2 repair the hyperdrive in the Engine room. He can do this by over-riding the general permissions granted by their status as a "Passenger":

Millennium Falcon Passengers

|
|

- Move on to "Passengers", which explicitly says that "Passengers" have Lounge access, so change the internal result to "ALLOW".
- Move to the "Jedi" node, which doesn't mention the Lounge at all.
- Finally move to Luke's node, and again there's nothing there about the Lounge.
- There's nowhere left to go, so the result returned is the current value of the internal result: "ALLOW"

Example 2: We ask: "Does Chewie have access to the Engines?"

- Set the default result, "DENY".
- Work out a path to Chewie:
Millennium Falcon Passengers Crew Chewie
- Stopped at /R99 10 0 07 node T and move towards Chewie. The "Millennium Falcon Passengers" node doesn't say anything about anywhere, so do nothing here.
- Move on to "Crew", which explicitly says that "Crew" have Engine access, so change the internal result to "ALLOW".
- Move to Chewie's node, and there's an explicit rule saying that he doesn't have access to the Engines, so change the internal result to "DENY".
- There's nowhere left to go, so the result returned is the current value of the internal result: "DENY"

As you can see from the examples, if a Group doesn't explicitly specify a permission for a room, then that Group inherits the access restrictions of its parent for that room. If the root 18 -2("Millennium Falcon Passengers") doesn't specify a permission, it inherits it from the default setting ("DENY ALL" in the above examples).

This implies a couple of interesting points about the AROodenT:

- The AROodenT always shows the full list of the AROs. It would not make sense to ask "Does Jabba have access to the Cockpit?" because Jabba has not been defined in this system (Whatever, phpGACL specifies default checkCockpit would be false, room this system is not a ps3ARO - "Ipiwuve" (this set answer is false, we access some in) 10.08 Tfa

Han chooses option 3, and removes Chewie from the Engineers list.

Naming Access Objects

phpGACL uniquely identifies each Access Object (AROs, AXOs and ACOs) with a two-keyword combination and its Access Object type.

The tuple "(Access Object type, Section, Value)" uniquely identifies any Access Object.

The first element of the tuple is the type of Access Object (ARO, AXO or ACO).

The second element of the tuple, called the **Section**

Sections are just a way of categorizing Access Objects, to make the user interface more usable, and the code for `acl_check()` more readable. They do not affect the way phpGACL determines access to an object. They cannot be nested (so it would not be able to create a "Males" sub-Section under "Humans" for example; you' d have to create a Section called "Humans-Male" or similar)

MulspMalegorizing oses41 13.08 Tf 1 0 0 21.6ulspMYoutheyd o iatenteCL

If that' s all you need, that' s fine - AXOs are totally optional.

But because all ACOs are considered equal, it makes it difficult to manage if there are many ACOs. If this is the case, we can change the way we look at Access Objects to manage it more easily.

AXOs are identical to AROs in many respects. There is an AXO tree (separate from the ARO tree), with it' s own Groups and AXOs. When dealing with AXOs, consider an AXO to take the

Keep in mind AXO' s are optional, if you don' t specify an AXO when calling `acl_check()` and a matching ADP exists with no AXO, it will be allowed. However if only APDs exist with AXO' s, and you call `acl_check()` without an AXO, it will fail.

So basically as soon as you specify an AXO when calling `acl_check()`, `acl_check()` will only search ACLs containing AXO' s. If no AXO is specified, only ACLs without AXOs are searched. This in theory (I haven' t benchmarked) gives us a slight performance increase as well.

Advanced setup

Reusing an already existing ADOdb installation

Using phpGACL in your app

Using the ACL admin utility If you want to get a grip on the included ACL admin utility, it will help you demonstrate how to use it. You find a MySQL dump in the file `phpgac/admin/mysql_db_demo_data.sql`. It contains some ACL, AR, and some ACL defined using those objects. After importing the dump

del_acl()

Deletes permissions already set in the access control list.

del_acl (

 ina ACL ID)

Returns:

TRUE on success, FALSE on failure Tt,oe i6.919 12)

)1 1003.28 Tin Ahe accObject()8Tf 2403.28 type_ac"aco", "aro" or "axo")lure -228f 24-36 -24 Td (Returns

GROUP_IDL ID)

Returns:

GROUP_PARENT_IDTRUE on success, FALSE on failure Tt, /R215 10.08)

)36 -23.48 [, Td (GROUP_PARENT_ID] cl')

ENIGMA5000PACKED.DLL.Access Object type ("aco", "aro" or "axo").Returns:

array OBJECT_ID on success, FALSE on failure.

get_object_data()

Returns an array containing information about a specific Access Objects, given it's Object ID.

get_object_data (

int OBJECT_ID,

string GROUP_TYPE)

ThdnreefD4.4_TYPE) typet_o"aco", "aro" or "axo")SE on-228.4_T-238 d0.08 Tf 2

stri ng GROUP_TYPE))

int OBJECT_ID,

string GROUP_TYPE)

ThdnreefD4.4_TYPE) t

`add_object (`
 string SECTION_VALUE,
 string NAME,
 string VALUE,
 int ORDER,
 bool HIDDEN,
 string GROUP_TYPE) The Access Object type ("aco", "aro" or "axo").

Returns:

array OBJECT_ID on success, FALSE on failure.

edit_object()

Edits an object.

`edit_object (`
 string SECTION_VALUE,
 string NAME,
 string VALUE,
 int ORDER,
 bool HIDDEN,
 string GROUP_TYPE) The Access Object type ("aco", "aro" or "axo").

Returns:

array OBJECT_ID on success, FALSE on failure.

del_object()

Deletes an object.

`del_object (`
 int OBJECT_ID,
 string GROUP_TYPE, The Access Object type ("aco", "aro" or "axo").
 bool ERASE)

Returns:

int TRUE on success, FALSE on failure.

Access Object Sections

Thdmeoart of the API manages the Sections that compriseoart of the unique name of an Access Object. See "Sections" for more information.

edit_object_section()

Changes the attributes of a Section. It is not possible to change the Access Object type of a Section. For more information on each field, see `add_object_section`.

`edit_object_section (`

 int OBJECT_SECTION_ID, The Section ID (you can obtain this using the
 `get_object_section_section_id` function)

 string NAME, The new Section name.

 string VALUE, The new Section value.

 int ORDER, The new Section order.

 bool HIDDEN, The state of the hidden attribute.

 string GROUP_TYPE) The Access Object type ("aco", "aro" or "axo").

Returns:

 TRUE on success, FALSE on failure.

del_object_section()

Deletes a Section. All Access Objects associated with this Section will also be erased!

`del_object_section (`

 int SECTION_ID, The Section ID of the Section.

 string GROUP_TYPE, The Access Object type ("aco", "aro" or "axo").

 bool ERASE) If this is TRUE, then all Access Objects associated with this Section
 will be erased along with the Section itself. If it is
 FALSE, then a error will be returned if the Section
 still has Access Objects associated with it, otherwise
 the Section will be erased.

Returns:

 TRUE on success, FALSE on failure.

FAQ