Aufgabe 1

1.1

NULL DACL Gewährt allen Nutzern uneingeschränkten Zugriff

1.2

Empty DACL wurde ordnungsgemäß angelegt und bedeutet, dass Niemandem der Zugriff auf das Objekt gestattet ist

1.3

ACEs werden Sequenziell verarbeitet. Dabei befinden sich „vorne“ alle Expliziten (d.h. direkt zum Objekt gehörende) Deny-ACEs, gefolgt von den expliziten Allow-ACESs. Anschließend werden geerbte ACEs (auch wieder Deny vor Allow) verarbeitet. Dabei kommen zuerst die Parent ACEs, dann Grandparent ACEs usw.

1.4

Zuerst sollten Deny ACEs gespeichert werden, da diese nicht mehr greifen, wenn zuerst der Zugriff erlaubt wird.

1.5

Take Ownership – Ein Nutzer kann den Besitz des Objekts übernehmen und so die DACLs umgehen

Backup and Restore – Ein Nutzer kann Kopien von Objekten erstellen auf die die DACLs nicht mehr wirken

Aufgabe 2

2.1

* User SID
* Group SID
* Restriction SID list
* Privilege SID list

2.2

Gruppen Zugehörigkeit wird beim Login zugewiesen

2.3

Wenn ein Nutzer durch z.B seine Gruppe Zugriff auf ein Objekt hätte, der Nutzer aber eine Ausnahme darstellt (Z.B. wird ein neuer Mitarbeiter der Gruppe der Mitarbeiter zugewisen, soll aber während der Probezeit noch nicht vollen Zugriff haben) kann dies durch die Restriktion abgebildet werden.

2.4

Wenn ein Prozess z.B Daten in einem Verzeichnis Analysieren d.h. lesen soll, man aber verhindern will, dass dieser die Daten verändert (schreibrecht verweigern).

Aufgabe 3

3.2

Geschützt werden unter anderem Nutzerdaten, Kursinhalte und Systemkonfigurationen (und damit die Verfügbarkeit von Moodle)

Nutzerdaten werden durch Passwörter Geschützt

/\*\*  
 \* Returns true if the username and password work and false if they are  
 \* wrong or don't exist.  
 \*  
 \* @param string $username The username (without system magic quotes)  
 \* @param string $password The password (without system magic quotes)  
 \*  
 \* @return bool Authentication success or failure.  
 \*/  
function user\_login($username, $password) {

Zudem werden Secrets verwendet um die Identität festzustellen.

else if ($user->secret === $confirmsecret) {

Hashverfahren werden verwendet um das Passwort geheim zu halten:

// This will also update the stored hash to the latest algorithm  
// if the existing hash is using an out-of-date algorithm (or the  
// legacy md5 algorithm).  
update\_internal\_user\_password($user, $plainslashedpassword);

Außerdem gibt es eine Klasse die Festlegt, welche Informationen im Öffentlichen Profil angezeigt werden: (user/profile.php

/\*\*  
 \* Public Profile -- a user's public profile page  
 \*  
 \* - each user can currently have their own page (cloned from system and then customised)  
 \* - users can add any blocks they want  
 \* - the administrators can define a default site public profile for users who have  
 \* not created their own public profile  
 \*  
 \* This script implements the user's view of the public profile, and allows editing  
 \* of the public profile.

Die Klasse lib/encryption.php wird vielen Stelle nverwendet um Daten zu ver- und entschlüsseln:

/\*\*  
 \* Class used to encrypt or decrypt data.  
 \*  
 \* @package core  
 \* @copyright 2020 The Open University  
 \* @license http://www.gnu.org/copyleft/gpl.html GNU GPL v3 or later  
 \*/

Ein Authentification Plugin (auth/db/auth.php) schränkt den Zugriff auf die Datenbank ein:

/\*\*  
 \* Authentication Plugin: External Database Authentication  
 \*  
 \* Checks against an external database.  
 \*  
 \* @package auth\_db  
 \* @author Martin Dougiamas  
 \* @license http://www.gnu.org/copyleft/gpl.html GNU Public License  
 \*/

Moodle stellt außerdem einen Mechanismus bereit um Backups zu konfigurieren. (backup/backup.php) Dardurch kann die Verfügbarkeit der Daten auch im Falle eines Datenverlustes gewährleistet werden. (Vorausgesetzt das Backup wird an einem anderen Physischen Ort als die Originaldaten gesichert. Im Falle eines Feuers im Serverraum wäre ein Backup auf einer anderen Festplatte im selben Raum nutzlos)

Fehlerbehandlung spielt auch eine wichtige Rolle für die Verfügbarkeit des Systems. So werden Try-catch Blöcke verwendet, z.B. bei der Nutzererstellung: (auth/db/auth.php)

try {  
 $id = user\_create\_user($user, *false*, *false*); // It is truly a new user.  
 $trace->output(get\_string('auth\_dbinsertuser', 'auth\_db', array('name'=>$user->username, 'id'=>$id)), *1*);  
} catch (moodle\_exception $e) {  
 $trace->output(get\_string('auth\_dbinsertusererror', 'auth\_db', $user->username), *1*);  
 continue;  
}

oder auch einfache if statements um im Fehlerfahl eine Exception zu werfen

Hier wird beispielsweise geprüft, ob die Datenbankabfrage durchgeführt werden konnte

$rs = $authdb->Execute("SELECT \*  
 FROM {$this->config->table}  
 WHERE {$this->config->fielduser} = '".$this->ext\_addslashes($extusername)."' ");  
  
if (!$rs) {  
 throw new \moodle\_exception('auth\_dbcantconnect', 'auth\_db');

(auth/db/auth.php)

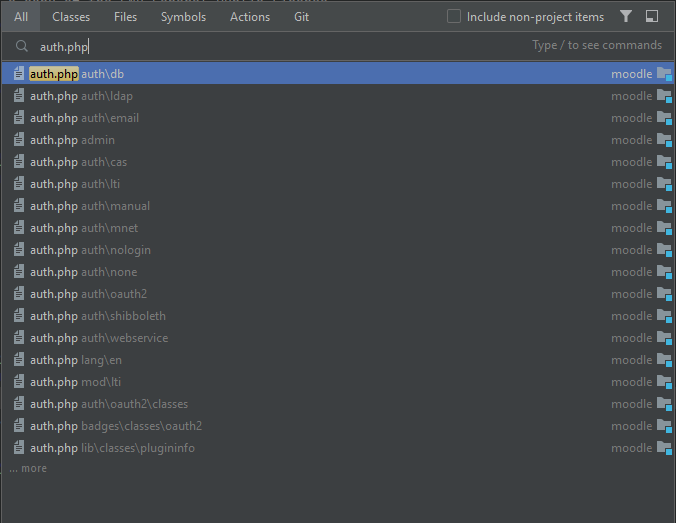
3.3

Die Ordnerstruktur gibt Aufschluss darüber, welche Dateien Interessant sein könnten.

Z.B moodle/auth, moodle/backup, moodle/payment

Wenn man sich zudem überlegt, dass die Nutzer(daten) irgendwie geschützt werden müssen, so lohnt sich auch ein Blick in moddle/user

Zudem kann die Volltextsuche einer DIE einem dabei helfen nach Schlagwörtern oder speziellen Dateien zu suchen. Z.B gibt es die Datei auth.php recht häufig in verschiedenen Modulen



3.4

Für eine Vollständige Inspektion wären vermutlich folgende Module interessant:

moodle/admin

moodle/auth

moodle/availability

moodle/auth

moodle/course

moodle/lib (zumindest individuelle Teile, z.B. db, da hier evtl weitere Sicherheitsmechanismen für Zugriffe auf die Datenbank implementiert sein könnten.)

moodle/login

moodle/payment

moodle/user

Zudem alle Module die eine auth.php Datei enthalten

Da das immer noch ein großer Umfang ist hab ich mich bei meiner Inspektion vor allem auf die Module auth, backup und user konzentriert.