Masaryk University
Faculty of Informatics

Adding external authentication support to a web project

Bachelor Thesis

Richard Kalinec

Brno, Spring 2017

Masaryk University
Faculty of Informatics

Adding external authentication support to a web project

Bachelor Thesis

Richard Kalinec

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Richard Kalinec

Advisor: prof. RNDr. Václav Matyáš, M.Sc., Ph.D.

Acknowledgement

I would like to thank my advisor prof. RNDr. Václav Matyáš, M.Sc., Ph.D. for support, feedback and guidance especially regarding formal and language aspects throughout writing of my thesis. I would also like to thank my consultant Jan Pazdziora, Ph.D for support, feedback and guidance mainly as regards technical and contentual aspects throughout writing of my thesis.

Abstract

The aim of this thesis is to extend Drupal such that it will support optional external authentication, in particular using mod_auth_gssapi and mod_intercept_form_submit, as well as retrieving additional information about the user from an external identity provider using mod_lookup_identity. Moreover, this functionality will be implemented on the basis of an established mechanism of passing the authentication results to web applications, so there is a viable possibility that it will be usable by multiple external authentication mechanisms. The thesis also provides a description and explanation of the related issues, relevant technologies and their use and configuration, Drupal, as well as the state before the extension, the extension itself, and the resultant state and its implications.

Keywords

external authentication, Drupal, module, extension, login form, external IdP, `REMOTE_USER(_*)`, additional information about the user, mapping of external groups to roles in Drupal

Contents

1 Problems to solve and goals **1**

2 Technologies used **5**
2.1 *mod_auth_kerb/mod_auth_gssapi* 5
2.2 *mod_authnz_pam* 7
2.3 *mod_intercept_form_submit* 8
2.4 *mod_lookup_identity* 9
2.5 *mod_auth_mellon* 10

3 Drupal **11**
3.1 *Hooks* 13
3.2 *User and access management* 14
3.2.1 Users 14
3.2.2 Roles and permissions 16
3.2.3 Implementation and data model 17

4 Module to extend – Webserver authentication **19**
4.1 *Choice of the module* 19
4.2 *Functionality of the module before the extension* 20

5 Extension of the module **25**
5.1 *Patch 1: Do not log the current user out when* `REMOTE_USER` *is empty by default* 25
5.2 *Patch 2: Match existing users also when creating accounts for unknown users is disabled* 26
5.3 *Patch 3: GSSAPI authentication is now optional and login form is re-enabled during the module's uninstallation* 27
5.4 *Patch 4: Retrieve additional information about the user and update it at every login* 29

6 Results **31**
6.1 *Functionality of the module after the extension* 31
6.2 *Security considerations* 32

7 Conclusion **35**

Bibliography **37**

A PHP **49**

B Environment variables **51**

C Drupal – selected additional information **53**
C.1 *Drupal system requirements* 53
C.2 *Drupal User module file structure* 54
C.3 *Drupal identity management schema in detail* 55

D Structure of the Webserver authentication module **57**

E Functionality of the original Webserver authentication module in pictures **65**

F The source codes of the patches making up the extension of the module **69**
F.1 *The first patch* 69
F.2 *The second patch* 71
F.3 *The third patch* 72
F.4 *The fourth patch* 75

G Installation and configuration of prerequisites **81**
G.1 *Drupal installation and basic configuration* 81
G.2 *Kerberos – mod_auth_gssapi* 82
G.3 *HBAC for Kerberos – mod_authnz_pam* 87
G.4 *External identities for the login form – mod_intercept_form_submit* 91
G.5 *Storing external users in internal databases – mod_lookup_identity* 93
G.6 *SAML – mod_auth_mellon* 96

H Functionality of the extended Webserver authentication module in pictures **97**

1 Problems to solve and goals

Currently, those web applications that need to authenticate their users mostly have the login form, into which users have to enter their login and password – these are subsequently validated against some internal user database. If these credentials match, a session record is created and a cookie set, which the browser will send with each subsequent request to the application (HTTP cookies are currently specified by RFC 6265 [1]). The information about the authenticated user must be stored in the session record or on the application server. The application can then show data and grant access to services available only to the authenticated user.

The problems to solve have two dimensions:

1. the use of central identity management (IdM) and single sign-on as opposed to the IdM mechanisms integrated to the web application and managing users locally and

2. the authentication by a username and password as opposed to more sophisticated external authentication mechanisms.

On the positive side, application's own IdM with authentication based on a username and password can be implemented and operated independently from external authentication providers – services that supply some kind of authentication information. These include, for example, OpenID/OpenID Connect, which is an open standard and decentralized authentication protocol [2, 3] or FreeIPA (IPA stands for Identity, Policy, Audit) – a central IdM server that integrates various open-source technologies [4], some of which are relevant for this thesis. It should also be noted that this way the risk of identity theft affecting users' accounts at multiple services at once is lower, provided that users have unique credentials for each service. In the end, if the applications do IdM on their own, it can be tailored to best fit their needs and provide optimal user experience. However, this approach to IdM has multiple drawbacks especially when it comes to larger organizations and more complex systems. First of all, it requires users to remember and submit their credentials for each service and, on the other hand, the service provider must store these credentials and

keep them secure. This becomes a serious burden, for example, in large companies where every user has to log into his workstation and then additionally authenticate to various services that the company uses or operates. Moreover, many users use more or less weak passwords and/or reuse them across multiple services [5, 6]. A recent study of Verizon RISK team has found out that 76 percent of successful data breaches were caused by a weak password usage [7]. Even though these wrongdoings can be levelled out by educating users, it can hardly be significantly changed by one single organization (particularly reaching external users might be complicated) and despite all efforts, this problem will probably persist to some extent.

To simplify identity and access management in larger organizations and enterprises, user identities are usually centrally managed. Many programming languages and frameworks provide libraries or modules for at least some ways of centrally managed authentication [8]. Setups in these environments are often very complex, therefore it can be beneficial to offload the authentication support to a front-end independent from the application code and framework or programming language [8]. One of the ways of achieving this is using Apache modules that provide authentication, access management and information services. On top of that, if additional improvements are needed, System Security Services Daemon (SSSD) can be used as an offline support for the system that offers caching, better data storage and extended user information retrieval [8, 9]. It is primarily aimed at the operating system level, but it can also be used for IdM in web applications [8].

As we have come through problems to solve, we can proceed to the goal of this thesis, which is to implement support for consuming Apache module authentication results by Drupal [10], a popular content management system. Applying the least alterations possible, when Drupal will be able to work with the authentication results from the front-end web server, it will be sufficient to configure that web server to add access control to Kerberos authentication [11], federated authentication via Security Assertion Markup Language (SAML) [12] or use central IdM server like FreeIPA to authenticate either using Kerberos, or by redirecting login and password pair submitted by the user from Drupal's native login form to the FreeIPA server.

Furthermore, the support for reading additional information about the user enables the application to provide more functionality and better user experience. If the group membership information is available to Drupal, the acquired user information can serve as a base for its authorization mechanisms.

Another advantage of this concept is more convenience for users, as they can be logged in automatically even to multiple services (Single Sign-On). Last but not least, Drupal can retain its existing authentication and authorization functionality, as this option presents just an additional possibility. Although not a subject of this thesis, these authentication options can be even combined into two-factor authentication, which provides extra security [6]. According to data from Telesign, 68 percent of people want companies to offer an additional layer of internet security and 86 percent of people who use two-factor authentication feel their accounts are more secure [6]. Despite these declarations, most users do not use it even when available [6]. Most users generally prefer usability and convenience over security or do not care about security at all [13]. Therefore providing support for the IdM technologies described and implemented in this thesis as a comfortable optional alternative to the authentication by a username and password for the particular application will probably better fit the preferences of most users than offering it as a mandatory extension supplementing application's existing user management mechanisms.

Security considerations are discussed in *Section 6.2*.

2 Technologies used

PHP is described in *Appendix A*.

The Apache HTTP Server [20] (colloquially called Apache) is the most used web server software [27–29]. It is a free and open-source project of the Apache Software Foundation, developed collaboratively [20, 27, 30]. It supports many operating systems including Microsoft Windows and various Unix-like systems [20, 27]. Apache is highly configurable and provides a wide variety of functions that can be extended by additional third-party modules [27, 31]. As the Apache modules concerned in this thesis (described below) are not a part of the standard Apache HTTP Server distribution [32], they must be installed manually. The necessary configuration of these modules is described in *Appendix G*.

2.1 mod_auth_kerb/mod_auth_gssapi

The Apache modules mod_auth_kerb [33] and mod_auth_gssapi [34] enable Apache to use Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) [35] and Kerberos [11, 36, 37]. Module mod_auth_gssapi is a modernized successor to the aging and unmaintained mod_auth_kerb [33, 34, 38, 39]. mod_auth_gssapi uses only Generic Security Service Application Program Interface (GSSAPI) [40–42] calls, aims to be as agnostic of the actual mechanism used as possible and supports additional options to the benefit of authentication as well as data transmission, especially in terms of security [34, 38, 43, 44]. Therefore it is being increasingly used in various projects and operating systems instead of mod_auth_kerb [39, 43, 45, 46].

Kerberos authentication in web applications uses SPNEGO/Negotiate, which is one of HTTP Authentication schemes [47–49]. Therefore it shares the use of some mechanisms with HTTP Basic Authentication [50] and HTTP Digest Authentication [51]: firstly, HTTP headers and states used to perform the web authentication [47], and secondly, the mechanism used to pass the authentication results to the web application [52].

The following HTTP headers and states are used during Negotiate authentication process, as described in RFC 4559 (partially paraphrased, some details omitted) [47]:

- The `WWW-Authenticate` response header
  - If a server receives a request for an access-protected object, and if an acceptable Authorization header has not been sent, the server responds with a `401 Unauthorized` status code, and a

    `WWW-Authenticate: Negotiate` *`authentication data`*

    header. The `Negotiate` parameter denotes the authentication scheme used; authentication data are not present in the initial response from the server.
  - A status code `200 OK` status response can also carry a `WWW-Authenticate` response header containing the final leg of the authentication. In this case, the authentication data will be present. Before using the contents of the response, the client should verify that the security context is complete – if it is not, an appropriate action, based on the authentication status, should be taken. For example, the authentication could have failed on the final leg if mutual authentication was requested and the server was not able to prove its identity. It is not always possible to mutually authenticate the server before the HTTP operation. POST methods are in this category.
- The `Authorization` request header
  - If the response received from the server contains a `WWW-Authenticate` header, the client is expected to retry the HTTP request, passing a HTTP `Authorization` header line in the form

    `Authorization: Negotiate` *`authentication data`*.

    Any returned code other than a success 2xx code represents an authentication error. If a 401 containing a `WWW-Authenticate` header with `Negotiate` and authentication data is returned from the server, it is a continuation of the authentication request. A client may initiate a connection to the

server with an `Authorization` header containing an initial token for the server. This form will bypass the initial 401 error from the server when the client knows that the server will accept the Negotiate HTTP authentication type.

The authentication data exchanged use data formats defined for use with the GSSAPI [40] and, in particular, the formats set for the SPNEGO [35] and Kerberos [42] mechanisms for GSSAPI.

Just as it is done after authentication using the login form for a username and password, it is also an efficient manner for Kerberos authentication in web applications to set a cookie with a newly created session record to avoid Negotiate on each HTTP request [52], what would have a significantly negative impact on performance. The overall authentication process is described in the article on *adelton.com* in the section about configuring Kerberos/GSSAPI/Negotiate HTTP authentication [53]. A more complex example of web authentication using Kerberos can be found at Google Search Appliance Help [54].

In order for Kerberos authentication to be optional, it is necessary to display the login form, enable the user to enter his credentials and authenticate him using the application's standard form-based method [53]. The ways to achieve this are described in (*Section G.2*).

The login of the authenticated user is passed to the web application most often by using the `REMOTE_USER` environment variable. Environment variables and their use are explained in *Appendix B*. As HTTP Basic Authentication, which also used `REMOTE_USER`, was used heavily in the past, some web applications may still support it [52]. In any case, the application needs to retrieve and trust the value of `REMOTE_USER` [8]. Many applications will also need to have records in their internal databases even for all externally authenticated users, for foreign keys to work [8]. Therefore if the application retrieves `REMOTE_USER` set to a value it does not have in the database, it can create a user record on the fly [8].

2.2 mod_authnz_pam

The Apache module mod_authnz_pam [59] provides PAM [60, 61] authorization, supplementing authentication done by other modules [59]. It can be also used as a full Basic Authentication provider [59].

Any PAM service with any PAM stack configuration for authentication and account management groups can be used; however, the primary intended use is in connection with SSSD and pam_sss.so [59].

Authorization on the Apache level ensures that a user logging in to the concerned service will be required to not only have an appropriate service ticket, but also to have allowed access by a host-based access control (HBAC) rule on a FreeIPA server [8]. Otherwise, the authentication will fail even with the valid Kerberos ticket [8]. This way it is possible to grant access only to a particular set or group of users, as well as prevent pollution of the application's database – in the case that it stores a user record for every successful external authentication – with users who have no access rights in the application [8].

2.3 mod_intercept_form_submit

In many cases neither Kerberos nor any other authentication method requiring additional setup on the client's side (e. g. certificates) can be used, for practical reasons [8]. The Apache module mod_intercept_form_submit [62] offers another possibility in the case the organization uses FreeIPA as its central user management: connecting organization's existing applications to the FreeIPA authentication service, while retaining the application-specific look and feel of their login forms [8]. Again, it is sufficient for the application to work with `REMOTE_USER` [8].

The module intercepts the submission of application's login forms, retrieves the login and password from the POST HTTP request and runs the PAM authentication with those credentials (it calls a function of mod_authnz_pam) [62]. Afterwards [62]:

- If the authentication passes, it sets `REMOTE_USER`; in the case `REMOTE_USER` is already set (presumably by some previous module), no authentication takes place.
- If the authentication fails, a string describing the error is stored to the `EXTERNAL_AUTH_ERROR` environment variable.

If `REMOTE_USER` is set, the application is expected to trust its value and skip validating the credentials on its own [62].

The module's homepage at *adelton.com* [62] provides a detailed example of communication and actions when this module is used.

2.4 mod_lookup_identity

When a user is created and stored by a web application, the web application has full control over what fields (attributes) it will require to be present – without them creating the user record will not be allowed [63]. However, when the user authentication happens against an external IdP, asking users for their email address and name that they already have correctly filled in the central server might not be ideal [63]. It would not only slow user's work down, it could also lead to inconsistencies, and in some enterprises, only dedicated departments can modify the personnel information in the central identity store [63].

The Apache module mod_lookup_identity [64] allows lookup of a user based on a certificate and retrieval of the list of the group names the user belongs to as well as any custom attributes and stores them in notes or environment variables [64]. To achieve this, it uses D-Bus [65] calls of SSSD (provided by the sssd-dbus package) to communicate with SSSD ifp service which in turn retrieves the requested data from the FreeIPA server [8, 64].

For this thesis, the retrieval of additional data about the user is the relevant part of this module's functionality. Web applications most often need information like user's email address, full name, phone number and others for better user experience and user's group membership for authorization [52]. After retrieval, these attributes are mapped to the environment variables [8], such as `REMOTE_USER_EMAIL`, `REMOTE_USER_FULLANME`, `REMOTE_USER_GROUPS` and/or `REMOTE_USER_GROUP_N`, `REMOTE_USER_GROUP_1`, `REMOTE_USER_GROUP_2...`– the environment variables like these are proposed and described at *freeipa.org* [66]. Then the application can use these environment variables to enhance its functionality; external group membership information can be internally mapped to its internal role and access control handling, thereby automatically granting the externally authenticated user appropriate privileges [8]. Using this mechanism, web application developers do not need to understand all possible identity providers (IdPs) [8]. Likewise, web applications do not need to implement all possible authentication mechanisms and can instead rely on the specialized Apache modules to do it and work with the aforementioned environment variables [8, 66].

The web applications can use these attributes from the central IdP to have the same set of information as is the case with their internal users, typically storing the attributes into their internal user databases when saving records for externally authenticated users [63]. Last but not least, even in the case of externally authenticating the user that already has a record in the application's database, it is desirable to always compare the current information about the user in the central IdP with the information in the application's database, and if it changed, update it [8].

2.5 mod_auth_mellon

The Apache module mod_auth_mellon [67] facilitates the authentication of a user against a SAML [12, 68] 2.0 IdP and grants him access to directories depending on attributes received from the IdP [67]. A practical and concrete example of SSO using SAML is at Google Developers [69].

When the web application is modified to make use of `REMOTE_USER` and `REMOTE_USER_*` as described in the previous sections, then it is sufficient to enable a particular external authentication mechanism by configuring the front-end server [8]. This also holds for SAML using mod_auth_mellon: its version 0.11.0 and newer can be configured to store attributes into the environment variables exactly like mod_lookup_identity does [8].

3 Drupal

Drupal [10] is a free and open-source content management software [70] written in PHP. It is collaboratively developed by the Drupal community with support of the Drupal Association [70, 71]. Being the third most popular content management system (CMS) [72], as of April 2017, Drupal market share amounts to 4.7 percent, thus it runs 2.2 percent of websites [73]. These include for example Twitter, Pinterest, weather.com, Taboola, independent.co.uk, europa.eu and nist.gov [73]. The usage data further shows that Drupal is used by websites with relatively higher traffic rate than other most popular CMS [73]. Last but not least, the version 8 of Drupal released in November 2015 [74] still only accounts to 3.6 percent of websites that use Drupal, while the version 7 is used by over 80 percent of Drupal-powered websites [73]. The remainder uses even older versions, mainly the version 6, that are no longer supported [73, 74]. Therefore this chapter is written with Drupal 7 in mind, but not Drupal 8. As explained in *Section 4.1*, as of April 2017, it is still more worthwhile to extend the stable release of the Webserver authentication module for Drupal 7 than its development release for Drupal 8 released when work on the extension was already underway.

Drupal tries to balance simplicity and flexibility – as Drupal's documentation puts it: *'The average content management system (CMS) is like a toy truck—specific assumptions have been made about how it will be used, and these assumptions are difficult to override. Content management frameworks, on the other hand, are like the raw materials needed to make any toy—no assumptions have been made about how they'll be used, and the builder needs expert technical knowledge in order to make anything at all.'* [75]. For this purpose, Drupal uses modular approach to site building: *'Unlike other CMSs, Drupal isn't a prefabricated toy truck, but rather a collection of wheels, windshields, axles, frames, etc., that a toy maker can easily connect together. With Drupal, a maker could create a toy truck, but she or he could just as easily create a toy airplane, submarine, or robot. For this reason, Drupal may be described as both a content management system and a content management framework—one unified system that strives to have the strengths of both, without their deficiencies.'* [75]. This means that for creating a particular kind of site, the right modules need to be com-

bined [75]. As Drupal's modules operate using a standard mechanism and interface with a common underlying system, it is possible to build customized and more complex features [75]. However, this requires learning how Drupal components, mechanisms and interfaces work [75].

Drupal processes information in five main separate layers [75]:

1. Data – the basic input for the system [75]. Most content types are stored as nodes, which are sets of related bits of information [75]. This encompasses structured information pertaining to the content (such as the body text, the title, the author link, the creation date etc.) as well as metadata controlling when the node will be displayed (for example the taxonomy or the publishing status) [75]. Comments can be enabled for any node type (and thus any content type) [75].

2. Modules – collections of functions that extend Drupal and enhance its functionality [71]. They are either part of the Drupal core or they are available from the repository of contributed Drupal modules [75–77], which contains over 35600 items as of 7th October 2016 [78]. However, anyone can create his own module [71]. As Drupal tries to balance innovation and stability as well as maintain thorough control over core's code quality, only essential functions are included in the core [71]. The contributed modules build on the core functionality and add various other functions [75]. Changing the core files to make changes to a site or application by its developer is strongly discouraged, as this would pose a grave obstacle for further updates – including the security ones – to the Drupal core and maintenance of the site [71, 79]. Therefore any changes or additions to the site's functionality should be done using modules (which use the hooks system, as explained in *Section 3.1*) and themes [71].

3. Blocks and menus – blocks display particular content, often providing the output from a module [75]. They can be configured to output in various ways, as well as to show only on some defined pages or only for some defined users [75]. Besides, they can be placed in various 'Regions' in the site's template (theme) layout [75]. Menus facilitate navigation, defining content for each

defined menu path (which are defined by relative URLs) [75]. Therefore they are a core element of Drupal providing links to all the pages created in Drupal [75]. Together with views and taxonomy (tagging of content – albeit tags are stored in nodes), the menu system creates an information architecture [75].

4. User permissions – permissions are defined for roles and users are assigned to these roles in order to grant them the defined permissions [75]. These settings determine what various kinds of users are allowed to do and see [75].

5. Template – more often called a theme, or also a 'skin' [75]. This top layer determines the look and feel of a Drupal site [71]. It usually consists mainly of XHTML and CSS intermixed with some PHP variables that define appropriate locations for Drupal-generated content [75]. Moreover, a theme can include INFO files and pictures (JPG, GIF or PNG, for instance) [71]. It contains elements such as a site's header, icons, a block layout as well as a collection of functions that can be used to override standard functions in the modules to provide complete control over modules' markup generation at output time [71, 75]. Templates can also be assigned on-the-fly based on user permissions [75]. As with modules, anybody can create his own themes [71].

Drupal aims for easy creation of versatile, structured content and connecting powerful integration tools [71]. It is able to operate multiple sites and offers deep multilingualism support [71].

Drupal also offers various distributions – downloadable packages that consist of the core and a set of contributed modules, themes, and configurations [71]. They combine features and functions tailored for a specific kind of site [71]. Distributions are usually built by advanced Drupal contributors [71].

System requirements for Drupal are described in *Section C.1*.

3.1 Hooks

As modules are actually no different from regular PHP files, and they execute within the context of the site, they can use all the functions

and access all the variables and structures of the Drupal core [80]. This enables the Drupal core to enhance its functionality by calling at specific places certain functions defined in modules [80]. The places where modules' code can be executed are called 'hooks' and they are defined by a fixed interface [80]. The hooks provided by Drupal 7 application programming interface (API) are listed at *api.drupal.org* [81]. They can add new functionality, as well as override or modify the core's functionality [80]. Modules can also define their own hooks – most such modules provide documentation about them [80]. When a point where a hook occurs is reached, Drupal seeks contributions from all enabled modules by scanning all modules codes for functions that have the name `module_name_hook_name` [80]. So, for example, the block module's help hook is called `block_help` and the node module's help hook is called `node_help` [80]. A hook may provide parameters – in the aforementioned example developers can use the `hook_help` parameters `$path` and `$arg` to determine what page or pages will display the module's documentation [80]. A hook can also be viewed as an event listener – an event triggers an action [80]. The event in Drupal, for example deleting a node, would trigger the hook `hook_delete`, what results in running functions in all the enabled modules that implement it [80]. The `hook_delete` implementation can, for instance, decrease the node count by 1 [80]. A comprehensive example in a wider context is available at *alanstorm.com* [82].

3.2 User and access management

3.2.1 Users

If not cited otherwise, this subsection draws from the article in the Understanding Drupal guide at *drupal.org* [83].

Every visitor to a Drupal site, whether logged in with his account or visiting the site anonymously, is considered a user by Drupal. Every user also has a numeric user ID special to the user type.

There are following user types in Drupal:

1. Computer user – the person installing Drupal must have access to the computer where the installation runs. Sometimes only one person is associated with an account, especially when installing

Drupal on one's own computer – in such a case the user is also most likely the 'administrator' user role. On the other hand, if Drupal is being installed on a remote server, accounts at some services (such as a web hosting) may be shared by several people.

2. Database user – as a Drupal installation requires a database to run and databases usually have user and access management mechanisms as well, a database user with full privileges is created in the database software and then the credentials of this user are given to the Drupal installation in order to grant Drupal full control over the database. Therefore this account is not a person.

3. Master administrator – it has the ID 1 (one) and it is the primary administrator user account usually created immediately after the installation of a new Drupal site. Therefore it is also known as the 'maintenance' or 'super-user' account. It is unique to the site and it has no permissions limitations. This account should not be associated with a particular person, but rather with one or more people who are responsible for keeping software up-to-date on the site. Last but not least, creating site content with this account should be avoided. Not only it can present increased risks [84], but also if the responsibility for site maintenance needs to change to other person and the present maintainer created content, which still needs to be associated with him/her, with this account, that content needs to be assigned to a new user account. Changing a content attribution from one account to another is not a trivial operation [84]. It is better to create that second account immediately after installing the site.

4. Logged in – these users are assigned a user ID 2 and higher when they register for the website. Every registered user should be associated with an individual person. A user name and email address is associated with all such users. Registered users can be assigned to roles, which are given fine-grained permissions, as described in the following subsection. The content they create is asociated with their account [85].

5. Anonymous – users who visit the website, but do not login. They all share the user ID 0 (zero).

On top of this, modules can associate other information with users, e. g. the core Profile module allows to define user profile fields to be associated with each user.

3.2.2 Roles and permissions

Permissions on Drupal sites are typically administered by creating roles [86]. Instead of assigning permissions individually to each user (what can be done with the contributed module User Permissions [87]), permissions are assigned to roles and roles are assigned to users [86]. Any user can have multiple roles at any time [86]. This approach gives administrators greater control over permissions and makes their management easier; they can be easily divided into multiple roles and users can be assigned complex permissions [86].

Drupal 7 automatically creates 3 roles during a site installation: anonymous user, authenticated user (a logged in user with no special role assignments) and administrator (a logged in user who will be assigned site administrator permissions) [83, 85, 88]. Other commonly created roles are, e. g., user administrator, site contributor, editor, author, moderator, member etc. [83, 84, 86, 88, 89] Roles can be created as needed, as well as renamed or removed (if there are no users attached, what should be checked fist) later [86].

The permissions that can be set for roles include creating new content, editing own/any content and deleting own/any content – all of these separately for various types of content [86, 89]. Adding new modules often adds extra permissions options, therefore in such cases permissions settings should be reviewed for appropriate access for all roles [84]. This should also be done whenever a module is enabled, even if it is merely turned off and on, as in such a case permissions for that module are unassigned to all roles [90]. The anonymous user role usually should have the most limited access to the site of all roles, while authenticated users could be granted more permissions, such as the ability to create some types of content [88]. Users matching certain criteria may get even more permissions, and the most trusted users can be given special privileges [88]. It is important to remember

that permissions given to authenticated users are also automatically given to custom created roles, but permissions given to anonymous users are not automatically given to the authenticated users role or custom created roles [88]. Therefore when adding any permissions for anonymous users, equivalent permissions should almost always be added for authenticated users to avoid odd site behaviour [88]. Another notable fact is that users who are permitted to administer users are able to grant themselves additional permissions [84].

3.2.3 Implementation and data model

User and access management in Drupal is handled by the core module User [85]. It allows users to register, log in and log out and facilitates all the functionality described in the two previous subsections [85]. Users can use their own name or handle and they can fine-tune their personal configuration settings using their individual *my account* page [85].

In Drupal 7, registered users need to authenticate by supplying their username and password, or alternately an OpenID login [85]. The local username and password, which is hashed using the SHA-512 cryptographic algorithm (SHA stands for Secure Hash Algorithm), are stored in the database [91]. When a password is entered into the login form, it is also hashed with SHA-512 and compared with the hash stored in the database [91]. If the hashes match, the username and password entered into the login form are correct [91].

Once an authenticated session is started, the user will not have to re-authenticate until that session is over [91]. To keep track of individual sessions, Drupal relies on PHP session support [91]. A visitor accessing a Drupal site is assigned a unique ID, the so-called session ID, which is stored in a cookie [85, 91]. For security reasons, the cookie does not contain personal information but acts as a key to retrieving the information stored on the site server [85, 91]. When the visitor accesses the site again, Drupal will check whether a specific session ID has been sent with the request. If this is the case, the prior saved environment is recreated [91].

The module consists of 7 program code files, 4 template files and 5 other files [85], which are concretely described in *Section C.2*.

The User module also uses 5 database tables defined by the function `user_schema` in the file `user.install` [85, 92] (described in detail in *Section C.3*):

1. `authmap` – stores the distributed authentication mapping,
2. `role_permission` – stores the permissions assigned to user roles,
3. `role` – stores user roles,
4. `users` – stores user data,
5. `users_roles` – maps users to roles.

4 Module to extend – Webserver authentication

4.1 Choice of the module

There are two contributed modules for Drupal that address the relevant functionality: External HTTP authentication [93] and Webserver authentication [94].

The External HTTP authentication module enables single sign-on to Drupal using HTTP Basic Authentication provided that the site is already protected with HTTP Authentication – this module will not protect it by itself [93]. The module is intended to be used for running an intranet site fully protected with HTTP Basic Authentication, or an administrators site that is separate from one's public Drupal site [93]. The functionality of the module is very limited – it does not support logging users out, managing additional information about users nor creating a database record for a new externally authenticated user. Therefore user accounts in Drupal have to be managed manually, as it works only for existing accounts [93]. What is more, it supports only Drupal 6 [93], which is already EOL (end-of-life) [95]. Its source code is only 49 lines long and it consists of:

1. `http_auth_ext_init()` – the implementation of `hook_init()`, which verifies that no user (including an external one) is logged in, and if not, it calls the `http_auth_ext_login()` function, to which it passes `REMOTE_USER`,

2. `http_auth_ext_login($remote_user)` – checks if the user exists in the `users` table without checking the password (trusting `REMOTE_USER` from Apache) and if he exists, it performs very basic tasks associated with logging him in.

These functions do not provide a sufficient basis for implementing the desired functionality. While lacking support for Drupal 7, this module is completely obsolete with regard to Drupal 8 – most importantly, it builds on the implementation of `hook_init()`, which is removed in Drupal 8. Upgrading this module (possibly even to a Drupal 7 version) would therefore practically mean writing a new module from scratch.

On 25th of September 2016, it was marked unsupported and obsolete on *drupal.org*.

The Webserver authentication module delegates authentication to the external web server such that Drupal honours the web server's authentication, users do not have to maintain separate passwords for Drupal and terminated users automatically become inaccessible in Drupal [94]. It aims to be useful when using HTTP Authentication, NTLM or .htaccess authentication and particularly useful when an easy way to implement single sign-on for Microsoft Windows networks is desired [94]. It can also be helpful for running an LDAP (Lightweight Directory Access Protocol) or Microsoft AD (Active Directory) server, or for the use of Unix authentication [94]. The Webserver authentication module is actively maintained and although there are no immediate plans to implement new features, it is open for further development [94]. The module has been developed by several developers that overall performed tens of commits to its source code since October 2003 [94]. For all these reasons as well as much richer functionality, this is the module of choice for this thesis. The stable release of this module for Drupal 7 is fully functional, while a development version for Drupal 8 was released in January 2017 – however, as of April 2017, it is only semi-functional and not ready for use in production environments [94]. In consideration of broad changes and increased complexity of codebases and their structure in Drupal 8 and the new development version of this module, as well as still relatively little usage of Drupal 8, as described in *Chapter 3*, it is still more worthwhile to extend the version for Drupal 7. The structure of the module as well as the files and functions of which it consists are described in *Appendix D*.

4.2 Functionality of the module before the extension

This section assumes that FreeIPA (and in the case of using SAML, Ipsilon) and all relevant Apache modules are installed and configured as described in *Appendix G*.

Let us have a Drupal-powered test site (*Figures E.1, E.2 and E.3*) containing a basic page with the PHP code that dumps the username and email address from the `user` object – which contains the information about the current user – as well as the user's email from

REMOTE_USER_EMAIL and group membership from REMOTE_USER_GROUP_*, if the user is externally authenticated. However, the last two pieces of information only inform about the current state on the IdP, but not about what is stored in the Drupal database. At the beginning, there are two users on the page: master, the main administrator, and localuser, an ordinary authenticated user with some other roles.

Besides, let us have six users on the FreeIPA server, the IdP used in this test setup, each being a member of a different group. All of these FreeIPA groups will be mapped to roles on the Drupal site after the extension. All of the FreeIPA users have all the attributes relevant for the test page filled in. The FreeIPA users master and localuser will match the existing Drupal users with the same usernames, therefore this users will have a choice what authentication method to use.

With all described above set up, the Webserver authentication module can be installed. It is better for the user to authenticate against FreeIPA as master using the kinit master command beforehand. The files of the module must be moved to the /etc/drupal7/all/modules/contrib/webserver_auth directory and have appropriate permissions and SELinux labels set. Drupal will immediately detect the module automatically, therefore it can be enabled by the administrator of the test page in the admin/modules menu. Upon saving the configuration with the module enabled, the administrator will be immediately logged out and redirected to the 'Acess denied' page. After returning to the home page as an anonymous user, it can be seen that the Login block is not displayed, and therefore the login form is not available any more. Despite GSSAPI authentication being configured as described in *Section G.2* and successfully passing (as can be verified using the klist command), the user remains anonymous on the test page even after refreshing the page multiple times (*Figure E.4*).

The reason is how two of the module's configuration variables are set by default and how they are used in the code. The webserver_auth_match_existing configuration variable is set to TRUE by default – this determines that external users are automatically matched with existing local users (with a record in the users table), and if there is a match with a username equal to the current REMOTE_USER, a record in the authmap table is created for this user using this module so that this user can authenticate also this way from now on. This behaviour is correctly intended by design to prevent the aforementioned

situation to occur in these cases. However, the check for a value of this configuration variable appears after the check for a value of the `webserver_auth_create_user` configuration variable, which is FALSE by default, what triggers logging an error message and exiting the method without ever reaching the check for a `webserver_auth_match_existing` value. `webserver_auth_create_user` controls whether completely new accounts, and therefore user records in both the `users` and `authmap` tables, should be created on the page for unknown external users. It is possible to manually create records in the `authmap` table for users – the advantage of that approach is flexibility, as external users can be mapped to Drupal users with different usernames. On the other hand, the disadvantage is demandingness and, for administrators not understanding the internals of Drupal and technologies used at least to some extent, inconvenience. It is therefore easier and sufficient for this test setup (at this point) to bypass this problem by a manual insertion of the `webserver_auth_create_user` configuration variable set to TRUE into the database.

With this workaround, the test page creates a record in the `authmap` table and successfully authenticates the user (*Figure E.5*). Thus after overcoming this difficulty that generally complicates the deployment of the module, the module is already basically capable of externally authenticating users using `REMOTE_USER`. As can be seen on *Figure E.5*, the page now also dumps email and group membership from `REMOTE_USER_*` for an externally authenticated user. However, the email address, which is dumped from the `user` object and previously loaded from the database, is still the same as before despite the email address on the FreeIPA server being different. There is also no straightforward way how to use group membership to determine the roles that the user belongs to and store this information into the `users_roles` table other than manually inferring roles relevant for the Drupal site and modifying the roles of the user according to that. In general, the module does not support retrieving information about the user from `REMOTE_USER_*`, not to mention using it to update the information in the database.

This is the limitation of the current functionality of the module – there are only two ways to reach any automation of retrieving, processing and saving the information from `REMOTE_USER_*` environment variables into the database:

1. An appropriate PHP code including database queries can be put into the `webserver_auth_insert` configuration variable, which will be automatically executed by the `webserver_auth_user_insert()` hook implementation. However, this is called only upon a user account creation, so the information about the user will not be updated in the case of their change on the FreeIPA server.

2. The REMOTE_USER_* environment variables can be accessed and their values or inferred information (as described in the previous paragraph) saved to the database by the page code. This way it is also possible to update the information in the database.

Especially the second option is highly impractical at best, while the first may not be usable at all due to impossibility to update the information in the database.

The module's settings are available at the `admin/config/people/webserver_auth` menu path – this configuration page can be navigated to either by going to the 'Modules' menu and clicking on the 'Configure' button in the row for the Webserver authentication module (*Figure E.6*), or by going to the 'Configuration' menu and clicking on the 'Webserver authentication' link in the 'People' section – this logically mirrors the menu path of the module's configuration page (*Figure E.7*). The configuration page of the module features three settings directly and nine settings in the 'Advanced settings' section, which can be expanded by clicking on it (*Figure E.8* – shows the default settings). In the test setup, the first setting controlling the `webserver_auth_create_user` configuration variable was enabled before using the original version of the module, as described above, and it remained enabled throughout all the testing of the implementation of the module extension. The advanced settings were left intact except for the login and logout URLs, which were later changed to contain addresses of the login and logout pages, respectively, as it was necessary in the case of performing GSSAPI authentication only on the login page, as described in *Section G.2*. This enables the login and logout links, respectively; for their maximal reliability, it is optimal to fill in their absolute URLs – in the test setup it was `/drupal7/user/loginGSSAPI` for the former and `/drupal7/logoutMessage` for the latter.

5 Extension of the module

5.1 Patch 1: Do not log the current user out when `REMOTE_USER` is empty by default

The Webserver authentication module in its original version logs the current user out not only when a value in `REMOTE_USER` changes to other non-empty value, what is desirable, but also when `REMOTE_USER` is empty. This means that the only way how to prevent unwanted logging out of the user if he visits a location on the Drupal site for which GSSAPI authentication is not configured, is configuring GSSAPI authentication (for `REMOTE_USER` to be set) for the whole Drupal site. That has potentially serious drawbacks and it does not even fully work in some setups and configurations, although this configuration is one of the possible options considering other factors, as described and explained in *Section G.2*.

Therefore when using the Webserver authentication module with mod_auth_gssapi, it should not log the current user out if `REMOTE_USER` is empty, but on the other hand, when using those ways of authentication that it was designed for so far, it is probably desirable to preserve the current behaviour. As a solution, the first patch adds a new Boolean configuration variable, `webserver_auth_logout_empty_remote_user`, that determines whether the current user will be logged out if `REMOTE_USER` is not set.

In the `_webserver_auth_route()` function, two changes are made with adequate modifications and additions to the comments:

1. Instead of logging the current user out if `$authname` (the variable with a value retrieved from `REMOTE_USER`) is empty (*Listing F.1*), there is now another condition for logout: the newly added configuration variable must be TRUE (*Listing F.2*).

2. The function will now exit not only when `$authname` is equal to the `webserver_authname` session variable (*Listing F.3*), but also when this value is empty, as this means that the module is not configured to logout the user in this case (*Listing F.4*).

The new configuration variable is controlled by a new setting on the configuration page – the 'Logout the current user if the authname is empty'

check box (*Listing F.5*). Finally, the new configuration variable will be deleted during the module's uninstallation, in the `webserver_auth_uninstall()` function (*Listing F.6*).

The patch with a description of the issue has been submitted as a feature request for review on *drupal.org* [96].

5.2 Patch 2: Match existing users also when creating accounts for unknown users is disabled

As already explained in *Section 4.2*, the original version of Webserver authentication effectively requires manual insertion of the `webserver_auth_create_user` configuration variable set to TRUE into the `variable` table to function. The reason is that the module conditions matching existing users, which is enabled by default, by creating completely new accounts for unknown users being enabled, but that is disabled by default. As a result, the administrator is logged out immediately after enabling the module and locked out just as all other users, as the login form is permanently disabled (this problem as well as its solution is described in the following section). If the first patch (described in the previous section) has been applied beforehand, the user will stay logged in after enabling the module and have a chance to change the problematic setting of the module, but if he does not do so before he logs out (most administrators may be unaware of this problem), he runs into the aforementioned situation as well. Even after applying the workaround mentioned above, this problem still poses a considerable limitation.

As a remedy, the second patch moves the check for `webserver_auth_create_user` from before checking for `webserver_auth_match_existing` and potential matching an existing user (*Listing F.7*) to after these actions and right before possible creation of a completely new account for the user (*Listing F.8*). This makes these two settings independent from each other and make `webserver_auth_match_existing` being set to TRUE by default effective from the beginning, what in turn prevents a lockout of existing users after enabling the module.

The patch with a description of the issue has been submitted for review on *drupal.org* [97]. As the current behaviour definitely runs against the intended behaviour of the module, as the comment on the

lines 180-181 in the `webserver_auth.module` states: `'match_existing must be TRUE initially so admin users aren't logged out when they enable the module'`, this issue is the only part of the extension that has been submitted as a bug report.

5.3 Patch 3: GSSAPI authentication is now optional and login form is re-enabled during the module's uninstallation

After enabling the Webserver authentication module in its original version, the login form is permanently hidden and the `user/login` task transformed into a 'Log in' link for external authentication on a specified URL (the `user/logout` task is also modified). The only way how to re-enable it is to uninstall the module and manually change the values of `status` and `region` attributes in the `block` table in the Login block's row(s) such that it will be enabled again and available in a specified region of the page. This effectively makes GSSAPI authentication mandatory when using this module, what may be sometimes desirable, but at other times not. What is more, this makes using mod_auth_gssapi side by side with mod_intercept_form_submit for authentication impossible. Last but not least, re-enabling the login form after uninstallation of the module may be problematic to perform for administrators not understanding the internals of Drupal and technologies used at least to some extent. Until the Login block is enabled, users have to go to `user/login`, manually modifying the URL in the address bar to reach the login form.

To sort out these problems, the third patch adds the `webserver_auth_optional_external_auth` Boolean configuration variable, which will be FALSE by default - this will preserve the current behaviour. However, setting it to TRUE will ensure that GSSAPI authentication is optional and users can also choose to authenticate using local authentication of a Drupal site, or mod_intercept_form_submit. Re-enabling the login form upon the module uninstallation is also done automatically.

Instead of only disabling the Login block (*Listing F.9*), the `webserver_auth_block_info_alter()` function now disables or enables it depending of the value of the new configuration variable (*Listing F.10*).

Transformation and modification of the `user/login` and `user/logout` tasks will now occur only if the new configuration variable is FALSE (*Listing F.11*), and a new tasks `user/external_login` and `user/external_logout` are created for use by this module, if the new configuration variable is TRUE (*Listing F.12*). Accordingly, the `webserver_auth_menu_link_alter()` function will now set the weight of the link for the external login depending of the menu path at which it is set, what directly results from the value of the new configuration variable (*Listings F.13 and F.14*). The new configuration variable will also be controlled by a new setting on the configuration page – the 'Preserve the ability to log in using local username + password and login form' check box (*Listing F.15*). The new configuration variable will be deleted during the module uninstallation, in the `webserver_auth_uninstall()` function (*Listing F.16*). Last but not least, the login form must be re-enabled during the uninstallation of the module for the Drupal site to remain conveniently usable – this is possible only by editing the database. For each theme, two attributes of the row representing the Login block of the User module must be changed: `status` must be set to `1` and `region` must be set to a name of some block – let it be the default `sidebar_first` block (*Listing F.17*).

There is one complication when changing the new setting: as the `webserver_auth_block_info_alter()` function is called only when rendering a page from scratch, Drupal's caches must be cleared for disabling/enabling the login form to take effect. Similarly, `webserver_auth_menu_alter()` is called only when rebuilding menus. Both of these tasks can be performed easily with the Devel module [98], but there are several other ways how to perform them, caches can even be cleared in the menu at `admin/config/development/performance` by clicking the 'Clear all caches' button. This is described in detail and discussed on *drupal.org* at multiple forums [99–102].

The patch with a description of the issue has been submitted as a feature request for review on *drupal.org* [103].

5.4 Patch 4: Retrieve additional information about the user and update it at every login

Before the extension, the Webserver authentication module is not capable of retrieving any additional information about an externally authenticated user. Possibilities in this matter are outlined in *Section 2.4*. The extension of the module reflects the fact that out of the information relevant for this thesis that can be stored on the IdP and for which the environment variables are proposed [66], the `users` table in the Drupal database has a column only for the email (not counting the deprecated `data` column), as described in *Appendix C*, and group membership information can be processed to assign or remove roles for the user in the `users_roles` table.

The fourth patch therefore adds two new functions – `_webserver_auth_update_remote_user_info()` and `_webserver_auth_update_remote_user_roles()` – that handle retrieval and processing of the information about the user immediately after every login. Additionally, two new configuration variables control whether this is done: `webserver_auth_email_from_idp` for email and `webserver_auth_ext_groups_roles_map` for group membership. The former is a Boolean, which determines if the user's email address will be updated from `REMOTE_USER_EMAIL`. The latter is a string – if it is empty, roles are managed locally (the current behaviour), but if it is non-empty, its value is used to map external groups from `REMOTE_USER_GROUP_*` to roles in Drupal. The format of the mapping must be in the form `external_group:role;external_group:role...` – that means pairs of external groups and Drupal roles internally separated by colons and mutually separated by semicolons. Note that external groups can also be retrieved from `REMOTE_USER_GROUPS`, if it is set, but this would be less convenient and reliable, as this environment variable is a string containing all groups that the user is a member of, separated by a delimiter specified in the mod_lookup_identity configuration. Therefore the delimiter in the configuration matching the delimiter used in the code when processing this string would be required (it might be desirable to create a new configuration variable for this purpose); this processing would in turn pose an extra load. These disadvantages can be simply avoided by requiring `REMOTE_USER_GROUP_*` to be set and

using only these environment variables, as together they are contentually equivalent to REMOTE_USER_GROUPS, but are easier to process and there is no risk of a delimiter mismatch.

To ensure not only the initial retrieval of the information, but also its potential subsequent updates, right after every login, the _webserver_auth_attempt_login() function will call the new _webserver_auth_update_remote_user_info() function (*Listing F.18*). This function will retrieve the user's email from the REMOTE_USER_EMAIL environment variable, if webserver_auth_email_from_idp is TRUE and REMOTE_USER_EMAIL is set. In the case that it is different from the one in the users table, the function will update the user's record in the table with this value (*Listing F.19*). Before exiting, when webserver_auth_ext_groups_roles_map is non-empty, it will call _webserver_auth_update_remote_user_roles(). That function first parses the mapping in webserver_auth_ext_groups_roles_map, assuming its proper format and validity. It subsequently loads the roles that are currently assigned to the user in the users_roles table (role names are loaded from the role table). Then it continues by retrieving the group names from the REMOTE_USER_GROUP_* environment variables, using the aforementioned mapping to determine the corresponding Drupal role names for these external groups and filter out irrelevant external groups (which are not in the mapping), and verifying if the mapped role names actually represent roles on the Drupal site, thus making a list of the roles externally assigned to the user. With this information available, this function removes records of the roles that are no longer assigned to the user on the IdP from the users_roles table and adds records of the roles that are newly assigned to the user on the IdP to the users_roles table (*Listing F.20*). The email field in the user account modification form is disabled by the webserver_auth_form_user_profile_form_alter() function, if webserver_auth_email_from_idp is TRUE (*Listing F.21*). The new configuration variables will be controlled by two new settings on the configuration page – the 'Retrieve users' email from REMOTE_USER_EMAIL variable' check box and the 'External groups to site's roles mapping' text area, and they will be deleted during uninstallation of the module, in the webserver_auth_uninstall() function (*Listing F.22*).

The patch with a description of the issue has been submitted as a feature request for review on *drupal.org* [104].

6 Results

6.1 Functionality of the module after the extension

After the extension of the module described in *Chapter 5*, the module can be enabled without having to manually edit the database or being locked out. Furthermore, it is now possible to configure mod_auth_gssapi only for a subset of locations or a specific location on a Drupal site, ideally a GSSAPI login page. In that case, the login and (optionally) logout URLs for the Webserver authentication module must be configured on its configuration page. As a result, a user is not authenticated automatically, but he can authenticate when he wants to by clicking on the 'Log in' link (*Figure H.1*). He is also able to log out (*Figure H.2*). The Drupal site is thus relieved the load and network traffic resulting from redundantly repeated GSSAPI authentication. However, it is still possible to configure mod_auth_gssapi for the whole Drupal site (*Figure H.3*). If a user uses one of the authentication methods that should work in the way as before the extension, he can enable a new option – 'Logout the current user if the authname is empty'.

If a user wants to turn on the ability to log in using the login form, both at `user/login` and on all the pages (not requiring administrative or other special permissions) on the Drupal site in the Login block, without giving up the option of GSSAPI authentication, he can enable another new option – 'Preserve the ability to log in using local username + password and login form'. This will make the login form available while preserving the possibility of external authentication by clicking on the 'Log in externally' link (*Figure H.4*). A user can therefore login either using mod_auth_gssapi, or locally – using the login form to authenticate against the database of the Drupal site, or using mod_intercept_form_submit – using the login form to authenticate against an external IdP (*Figures H.5* and *H.6*). There will now be two ways to log out: in a standard way, by clicking on the 'Log out' link, or with a redirection to the aforementioned configured logout URL – however, the 'Log out externally' link pointing to `user/external_logout` must be created for this way of logout to be conveniently reachable.

The module is now also capable of retrieving the user's email and group membership – this can be enabled by two more new options, 'Retrieve users' email from REMOTE_USER_EMAIL variable' and (by setting a mapping of external groups to roles on the Drupal site) 'External groups to site's roles mapping'. Immediately after every login, this information is retrieved, processed and if it changed, updated (*Figures H.7* and *H.8*). If retrieving the email from an external IdP is enabled, the email field in the user account modification form is disabled (*Figure H.9*). Overall, the module now offers four more settings on its configuration page, as can be seen on *Figure H.10*.

6.2 Security considerations

When using mod_intercept_form_submit, the only difference from local authentication is that the user's credentials are sent not only to the server running the Drupal site, but mod_intercept_form_submit in turn sends them to a third-party IdP, which informs it about the authentication result. Therefore credentials are sent two times over the network instead of one, and there are two servers that get credentials instead of one. This way of authentication therefore poses greater security risks, as there are more opportunities to steal credentials.

mod_auth_gssapi can be used to authenticate using Kerberos, which is a more complex authentication method. A user still enters his password, but only once and it is not sent over the network. Kerberos also has a protection against replay attacks (an intruder steals an authentication packet from the network and forwards that packet to a service or application as if the intruder was the user who originally sent the packet) [107]. Still, especially on Windows operating systems, it is susceptible to various types of attacks, including [108–112]:

1. finding passwords in SYSVOL and exploiting Group Policy Preferences – the simplest method, no special tool is needed, all the attacker has to do is opening Windows explorer and searching the domain SYSVOL DFS share for XML files that are likely to contain credentials,

2. Pass-the-Hash/Pass-the-Credential – grabbing the password hash and using it to access a resource,

3. OverPass-the-Hash/Pass-the-Key – using the password hash to get a Kerberos ticket,
4. Pass-the-Ticket – grabbing the Kerberos ticket(s) and using it to access a resource,
5. Golden Ticket – forged Kerberos TGT authentication ticket,
6. Silver Ticket – forged Kerberos TGS (ticket-granting service) service ticket,
7. Kerberos TGS service ticket offline cracking using Kerberoast,
8. Forged PAC (Privilege Account Certificate) – exploiting the unpatched MS14-068 Kerberos vulnerability on Domain Controllers,
9. Diamond PAC – blended attack type using elements of the Golden Ticket and the MS14-068 forged PAC,
10. SPN (Service Principal Name) Scanning – finding services by requesting SPNs of a specific SPN class/type,
11. Skeleton Key (In-memory Malware) – malware 'patches' the LSASS (Local Security Authority Subsystem Service) authentication process in-memory on Domain Controllers to enable a second, valid 'skeleton key' password, which can be used to authenticate any domain account,
12. 'The Credential Theft Shuffle' – compromising a single workstation, escalating privileges and dumping credentials, which are in turn used to laterally move to other workstations and repeating this cycle over and over again, what usually quickly results in stealing Domain Admin credentials,

These types of attacks as well as more general assessments are described in the cited sources. There are sophisticated tools capable of performing multiple of these types of attacks, such as Mimikatz [113]. It is also possible to abuse Kerberos in Windows Domain Authentication to bypass local Windows authentication and defeat full disk encryption [114]. The risks posed by these attacks can be greatly reduced by following good security practices, regularly updating at

least operating systems and critical software, periodically changing passwords and other measures [108, 115, 116]. Some of the aforementioned attacks have been addressed by patches from Microsoft [108, 109, 113]. Other platforms and software needs caution and regular maintenance as well – a list of vulnerabilities in SSSD can be seen on *securitytracker.com* [117].

7 Conclusion

In *Chapter 1*, this thesis analysed and described the problems surrounding the conventional way of authentication in more complex environments and outlined goals for their solution. In *Chapter 2*, it described the technologies relevant for this thesis and explained their use and in *Chapter 3*, it provided basic information about Drupal, the project to extend, followed by a more detailed description of its relevant aspects.

As the Drupal core is restricted to the most commonly used functionality, it was appropriate to extend one of its contributed modules or create a new one. The Webserver authentication module proved to be a suitable option, as explained in *Section 4.1*. Its original version had severe limitations with regards to the functionality of the Apache modules mod_auth_gssapi, mod_intercept_form_submit and mod_lookup_identity, as described and demonstrated in *Section 4.2*. With the extension of its version for Drupal 7 that I implemented, and which is described in *Chapter 5*, the (still) most used version of Drupal has gained full support for external authentication using mod_auth_gssapi, which facilitates Kerberos authentication. It also made this way of authentication optional, what enables its use side-by-side not only with conventional authentication, but also with authentication using mod_intercept_form_submit. On top of that, the Webserver authentication module is now capable of retrieving additional information about the user from mod_lookup_identity and directly (email) or indirectly (external groups -> roles on the Drupal site) storing it to the Drupal database.

I implemented this external authentication functionality on the basis of the universal mechanism of the Apache `REMOTE_USER(_*)` environment variables that can be used to store results of any authentication mechanism. Therefore Drupal has gained support not only for authentication using the aforementioned Apache modules, but potentially for any other authentication mechanism that supports storing its output to the `REMOTE_USER(_*)` environment variables.

The configuration of the aforementioned Apache modules is thoroughly described and explained in *Appendix G*, while a basic hint is

given for the configuration of mod_auth_mellon, which facilitates authentication using SAML.

In the end, the thesis summarizes and demonstrates the new functionality of the Webserver authentication module in *Section 6.1* and analyses the security of authentication using mod_intercept_form_submit or mod_auth_gssapi and compares it to the security of conventional authentication in *Section 6.2*.

I tested and debugged the extension in the test setup consisting of three virtual machines running Fedora. I plan to actively participate in the (likely long-lasting) review and discussion about the extension of the Webserver authentication module on `drupal.org` and possibly adjust the extension in line with the needs and findings of users. To pursue porting the extension to the module's version for Drupal 8, I will work with its maintainer.

Bibliography

[1] Internet Engineering Task Force. *HTTP State Management Mechanism*. Mar. 31, 2016. URL: https://tools.ietf.org/html/rfc6265 (visited on 04/08/2016).

[2] OpenID Foundation. *Specifications & Developer Information*. 2016. URL: http://openid.net/developers/specs/ (visited on 09/13/2016).

[3] OpenID Foundation. *OpenID Connect FAQ and Q&As*. 2016. URL: http://openid.net/connect/faq/ (visited on 09/13/2016).

[4] [Contributors to FreeIPA]. *FreeIPA*. Sept. 1, 2016. URL: http://www.freeipa.org (visited on 09/13/2016).

[5] Robert Lemos. *Poor Passwords, Poor Security*. 2015. URL: http://us.norton.com/yoursecurityresource/detail.jsp?aid=password_security (visited on 04/09/2016).

[6] Kimberlee Morrison. *People Are Still the Weak Link in the Password System*. June 3, 2015. URL: http://www.adweek.com/socialtimes/people-are-still-the-weak-link-in-the-password-system-infographic/621180 (visited on 04/12/2016).

[7] Noah Gamer. *Secure entry with strong passwords: Improving authentication credentials*. Oct. 29, 2015. URL: http://blog.trendmicro.com/secure-entry-with-strong-passwords-improving-authentication-credentials/ (visited on 04/12/2016).

[8] [Contributors to FreeIPA]. *Web App Authentication*. Aug. 23, 2016. URL: http://www.freeipa.org/page/Web_App_Authentication (visited on 09/14/2016).

[9] [Contributors to SSSD]. *SSSD - System Security Services Daemon*. Dec. 15, 2015. URL: https://fedorahosted.org/sssd/ (visited on 04/01/2016).

[10] [Drupal community with support of the Drupal Association]. *Drupal – Open Source CMS*. Apr. 2, 2016. URL: https://www.drupal.org/ (visited on 04/02/2016).

[11] Massachusetts Institute of Technology. *MIT Kerberos Documentation (1.14.3)*. July 21, 2016. URL: http://web.mit.edu/kerberos/krb5-latest/doc/ (visited on 09/13/2016).

[12] OASIS Security Services Technical Committee. *FrontPage – SAML Wiki*. Oct. 26, 2015. URL: https://wiki.oasis-open.org/security/FrontPage (visited on 09/12/2016).

[13] Kimberlee Morrison. *Online Security Is for Nerds*. Aug. 6, 2014. URL: http://www.adweek.com/socialtimes/online-security-nerds/202480 (visited on 04/14/2016).

[14] The PHP Group. *PHP Manual*. June 20, 2016. URL: http://php.net/manual/en/ (visited on 06/20/2016).

[15] The PHP Group. *Preface*. June 20, 2016. URL: http://php.net/manual/en/preface.php (visited on 06/20/2016).

[16] The PHP Group. *What is PHP?* June 20, 2016. URL: http://php.net/manual/en/intro-whatis.php (visited on 06/20/2016).

[17] [Contributors to Wikipedia]. *PHP*. June 18, 2016. URL: https://en.wikipedia.org/wiki/PHP (visited on 06/20/2016).

[18] The PHP Group. *What can PHP do?* June 21, 2016. URL: http://php.net/manual/en/intro-whatcando.php (visited on 06/21/2016).

[19] Refsnes Data. *PHP 5 Introduction*. June 21, 2016. URL: http://www.w3schools.com/php/php_intro.asp (visited on 06/21/2016).

[20] The Apache Software Foundation. *Apache HTTP Server Project*. Apr. 11, 2016. URL: http://httpd.apache.org/ (visited on 06/24/2016).

[21] The PHP Group. *What do I need?* June 22, 2016. URL: http://php.net/manual/en/tutorial.requirements.php (visited on 06/22/2016).

[22] Network Working Group. *The Common Gateway Interface (CGI) Version 1.1*. Oct. 2004. URL: https://tools.ietf.org/html/rfc3875 (visited on 06/22/2016).

[23] Paul Krill. *Why developers love and hate PHP*. Dec. 1, 2014. URL: http://www.infoworld.com/article/2852329/php/reasons-for-developers-to-love-hate-php.html (visited on 06/28/2016).

[24] Q-Success. *Usage statistics and market share of PHP for websites*. Aug. 3, 2016. URL: https://w3techs.com/technologies/details/pl-php/all/all (visited on 08/03/2016).

[25] Jolie O'Dell. *8 Experts Break Down the Pros and Cons of Coding With PHP*. Nov. 19, 2010. URL: http://mashable.com/2010/11/19/pros-cons-php/ (visited on 06/28/2016).

[26] Adam Englander. *What are the pros and cons of using PHP?* Nov. 24, 2015. URL: https://www.quora.com/What-are-the-pros-and-cons-of-using-PHP (visited on 06/28/2016).

[27] [Contributors to Wikipedia]. *Apache HTTP Server*. July 18, 2016. URL: https://en.wikipedia.org/wiki/Apache_HTTP_Server (visited on 07/31/2016).

[28] Netcraft. *July 2016 Web Server Survey*. July 19, 2016. URL: http://news.netcraft.com/archives/2016/07/19/july-2016-web-server-survey.html (visited on 07/31/2016).

[29] Q-Success. *Usage of web servers for websites*. July 31, 2016. URL: https://w3techs.com/technologies/overview/web_server/all (visited on 07/31/2016).

[30] The Apache Software Foundation. *About the Apache HTTP Server Project*. Feb. 20, 2016. URL: https://httpd.apache.org/ABOUT_APACHE.html (visited on 08/01/2016).

[31] The Apache Software Foundation. *FAQ – Httpd Wiki*. Dec. 16, 2015. URL: http://wiki.apache.org/httpd/FAQ (visited on 08/01/2016).

[32] The Apache Software Foundation. *Module Index – Apache HTTP Server Version 2.4*. May 21, 2016. URL: https://httpd.apache.org/docs/2.4/mod/ (visited on 08/01/2016).

[33] Daniel Kouril. *Kerberos Module For Apache*. May 9, 2013. URL: https://sourceforge.net/projects/modauthkerb/ (visited on 08/02/2016).

[34] [Simo Sorce]. *GSSAPI Negotiate module for Apache*. July 22, 2016. URL: https://github.com/modauthgssapi/mod_auth_gssapi (visited on 08/03/2016).

[35] Network Working Group. *The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism*. Oct. 2005. URL: https://tools.ietf.org/html/rfc4178 (visited on 08/01/2016).

[36] Network Working Group. *The Kerberos Network Authentication Service (V5)*. July 2005. URL: https://tools.ietf.org/html/rfc4120 (visited on 08/03/2016).

[37] [Contributors to Wikipedia]. *Kerberos (protocol)*. Aug. 3, 2016. URL: https://en.wikipedia.org/wiki/Kerberos_(protocol) (visited on 08/06/2016).

[38] Simo Sorce. *Announcing mod_auth_gssapi*. Aug. 14, 2014. URL: http://comp.protocols.kerberos.narkive.com/lCmYdi3K/announcing-mod-auth-gssapi (visited on 08/06/2016).

[39] Simo Sorce. *PSA: heads-up replacing mod_auth_kerb with mod_auth_gssapi*. Oct. 29, 2015. URL: https://lists.fedoraproject.org/pipermail/devel/2015-October/216204.html (visited on 08/06/2016).

[40] Network Working Group. *Generic Security Service Application Program Interface Version 2, Update 1*. Jan. 2000. URL: https://tools.ietf.org/html/rfc2743 (visited on 08/08/2016).

[41] [Contributors to Wikipedia]. *Generic Security Services Application Program Interface*. Oct. 7, 2015. URL: https://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface (visited on 08/08/2016).

[42] Network Working Group. *The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2*. July 2005. URL: https://tools.ietf.org/html/rfc4121 (visited on 08/08/2016).

[43] [Simo Sorce]. *[RFE] Switch apache to use mod_auth_gssapi*. Mar. 30, 2015. URL: https://fedorahosted.org/freeipa/ticket/4190 (visited on 08/07/2016).

[44] [Simo Sorce]. *[RFE] Use GSS-Proxy for the HTTP service*. June 15, 2016. URL: https://fedorahosted.org/freeipa/ticket/4189 (visited on 08/07/2016).

[45] [RedHat]. *new packages: mod_auth_gssapi*. Nov. 19, 2015. URL: https://rhn.redhat.com/errata/RHEA-2015-2242.html (visited on 08/08/2016).

[46] [Simo Sorce]. *Switch Negotiate auth from mod_auth_kerb to mod_auth_gssapi*. Apr. 17, 2015. URL: https://fedorahosted.org/ipsilon/ticket/89 (visited on 08/08/2016).

[47] Network Working Group. *SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*. June 2006. URL: https://tools.ietf.org/html/rfc4559 (visited on 08/29/2016).

[48] Internet Engineering Task Force (IETF). *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. June 2014. URL: https://tools.ietf.org/html/rfc7235 (visited on 08/10/2016).

[49] Internet Engineering Task Force (IETF). *Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry*. Mar. 10, 2016. URL: http://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml (visited on 08/27/2016).

[50] Internet Engineering Task Force (IETF). *The 'Basic' HTTP Authentication Scheme*. Sept. 2015. URL: https://tools.ietf.org/html/rfc7617 (visited on 08/28/2016).

[51] Internet Engineering Task Force (IETF). *HTTP Digest Access Authentication*. Sept. 2015. URL: https://tools.ietf.org/html/rfc7616 (visited on 08/28/2016).

[52] Jan Pazdziora. *External Identity and Authentication Providers For Apache HTTP Server*. Nov. 17, 2014. URL: https://www.adelton.com/docs/idm/apache-external-idm-auth (visited on 08/29/2016).

[53] Jan Pazdziora. *Optional Kerberos Authentication*. May 22, 2015. URL: https://www.adelton.com/docs/apache/optional-kerberos-authentication (visited on 09/01/2016).

[54] [Google]. *How The Google Search Appliance uses Kerberos to Authenticate Users and to Authorize Users to See Content*. Dec. 2008. URL: https://support.google.com/gsa/answer/6055202 (visited on 09/08/2016).

[55] [Contributors to Wikipedia]. *Environment variable*. June 30, 2016. URL: https://en.wikipedia.org/wiki/Environment_variable (visited on 09/03/2016).

[56] The Apache Software Foundation. *Environment Variables in Apache*. Apr. 30, 2016. URL: https://httpd.apache.org/docs/current/env.html (visited on 09/03/2016).

[57] The PHP Group. *HTTP authentication with PHP*. Sept. 6, 2016. URL: http://php.net/manual/en/features.http-auth.php (visited on 09/06/2016).

[58] The PHP Group. *$_SERVER*. Sept. 6, 2016. URL: http://php.net/manual/en/reserved.variables.server.php (visited on 09/06/2016).

[59] Jan Pazdziora. *Apache module mod_authnz_pam*. 2016. URL: https://www.adelton.com/apache/mod_authnz_pam/ (visited on 09/09/2016).

[60] Red Hat. *42.4. Pluggable Authentication Modules (PAM)*. Mar. 17, 2007. URL: https://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-pam.html (visited on 09/09/2016).

[61] [The NetBSD Foundation]. *Chapter 18. Pluggable Authentication Modules (PAM)*. May 28, 2016. URL: https://www.netbsd.org/docs/guide/en/chap-pam.html (visited on 09/09/2016).

[62] Jan Pazdziora. *mod_intercept_form_submit*. 2016. URL: https://www.adelton.com/apache/mod_intercept_form_submit/ (visited on 09/10/2016).

[63] [Contributors to FreeIPA]. *Web App Authentication/Example setup*. Aug. 23, 2016. URL: http://www.freeipa.org/page/Web_App_Authentication/Example_setup (visited on 12/03/2016).

[64] Jan Pazdziora. *mod_lookup_identity*. 2016. URL: https://www.adelton.com/apache/mod_lookup_identity/ (visited on 09/11/2016).

[65] [Contributors to freedesktop.org]. *dbus*. Jan. 14, 2016. URL: https://www.freedesktop.org/wiki/Software/dbus/ (visited on 09/11/2016).

[66] [Contributors to FreeIPA]. *Environment Variables: Proposed Additional Variables*. Aug. 23, 2016. URL: http://www.freeipa.org/page/Environment_Variables#Proposed_Additional_Variables (visited on 09/11/2016).

[67] [Olav Morken]. *An Apache module with a simple SAML 2.0 service provider*. Aug. 26, 2016. URL: https://github.com/UNINETT/mod_auth_mellon (visited on 09/12/2016).

[68] [Contributors to Wikipedia]. *Security Assertion Markup Language*. Aug. 28, 2016. URL: https://en.wikipedia.org/wiki/Security_Assertion_Markup_Language (visited on 09/12/2016).

[69] [Google]. *SAML Single Sign-On (SSO) Service for Google Apps*. Jan. 6, 2015. URL: https://developers.google.com/google-apps/sso/saml_reference_implementation (visited on 09/12/2016).

[70] [Drupal community with support of the Drupal Association]. *About*. Sept. 24, 2016. URL: https://www.drupal.org/about (visited on 09/24/2016).

[71] [Drupal community with support of the Drupal Association]. *FAQ*. Sept. 29, 2016. URL: https://www.drupal.org/faq (visited on 09/29/2016).

[72] Q-Success. *Usage of content management systems for websites*. Oct. 4, 2016. URL: https://w3techs.com/technologies/overview/content_management/all (visited on 10/04/2016).

[73] Q-Success. *Usage statistics and market share of Drupal for websites*. Oct. 4, 2016. URL: https://w3techs.com/technologies/details/cm-drupal/all/all (visited on 10/04/2016).

[74] [Jess (xjm)]. *Drupal 8.0.0 released*. Nov. 19, 2015. URL: https://www.drupal.org/blog/drupal-800-released (visited on 10/04/2016).

[75] [Drupal community with support of the Drupal Association]. *Overview*. July 28, 2016. URL: https://www.drupal.org/docs/7/understanding-drupal/overview (visited on 10/06/2016).

[76] [Drupal community with support of the Drupal Association]. *Core modules*. Jan. 26, 2014. URL: https://www.drupal.org/node/1283408 (visited on 10/07/2016).

[77] [Drupal community with support of the Drupal Association]. *Contributed modules*. Feb. 8, 2016. URL: https://www.drupal.org/node/340271 (visited on 10/07/2016).

[78] [Drupal community with support of the Drupal Association]. *Download & Extend: Modules*. Oct. 7, 2016. URL: https://www.drupal.org/project/project_module (visited on 10/07/2016).

[79] [Drupal community with support of the Drupal Association]. *Never hack core*. Aug. 17, 2016. URL: https://www.drupal.org/docs/7/site-building-best-practices/never-hack-core (visited on 10/07/2016).

[80] [Drupal community with support of the Drupal Association]. *Understanding the hook system for Drupal modules*. Sept. 15, 2016. URL: https://www.drupal.org/node/292 (visited on 10/10/2016).

[81] [Drupal community with support of the Drupal Association]. *Hooks*. Oct. 10, 2016. URL: https://api.drupal.org/api/drupal/includes!module.inc/group/hooks/7.x (visited on 10/10/2016).

[82] Alan Storm. *Drupal Module Hooks*. Feb. 17, 2011. URL: http://alanstorm.com/drupal_module_hooks (visited on 10/10/2016).

[83] [Drupal community with support of the Drupal Association]. *Users, permissions, and roles*. Sept. 22, 2016. URL: https://www.drupal.org/docs/7/understanding-drupal/users-permissions-and-roles (visited on 10/10/2016).

[84] [Drupal community with support of the Drupal Association]. *Accounts and roles*. Nov. 9, 2009. URL: https://www.drupal.org/node/22284 (visited on 10/13/2016).

[85] [Drupal community with support of the Drupal Association]. *User: access and management settings*. Oct. 30, 2015. URL: https://www.drupal.org/documentation/modules/user (visited on 10/14/2016).

[86] [Drupal community with support of the Drupal Association]. *User Roles*. Aug. 17, 2016. URL: https://www.drupal.org/docs/7/managing-users/user-roles (visited on 10/12/2016).

[87] [Drupal community with support of the Drupal Association]. *User Roles*. Aug. 17, 2016. URL: https://www.drupal.org/docs/7/managing-users/user-roles (visited on 10/12/2016).

[88] [Drupal community with support of the Drupal Association]. *Managing access control with permissions and user roles*. Aug. 17, 2016. URL: https://www.drupal.org/docs/7/managing-users/managing-access-control-with-permissions-and-user-roles (visited on 10/12/2016).

[89] [Drupal community with support of the Drupal Association]. *Users, roles and permissions*. Oct. 6, 2016. URL: https://www.drupal.org/node/120614 (visited on 10/14/2016).

[90] [Drupal community with support of the Drupal Association]. *Adjusting permissions after adding modules*. May 7, 2005. URL: https://www.drupal.org/node/22279 (visited on 10/14/2016).

[91] [Drupal community with support of the Drupal Association]. *User authentication*. June 22, 2012. URL: https://www.drupal.org/node/22276 (visited on 10/15/2016).

[92] [Drupal community with support of the Drupal Association]. *function user_schema*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.install/function/user_schema/7.x (visited on 10/15/2016).

[93] [Vesa Palmu]. *External HTTP authentication*. Sept. 25, 2016. URL: https://www.drupal.org/project/http_auth_ext (visited on 11/15/2016).

[94] [Contributors to Webserver authentication]. *Webserver authentication*. Jan. 26, 2017. URL: https://www.drupal.org/project/webserver_auth (visited on 04/10/2017).

[95] [Michael Hess]. *Drupal 6 end-of-life announcement*. Nov. 9, 2015. URL: https://www.drupal.org/drupal-6-eol (visited on 11/15/2016).

[96] [Richard Kalinec]. *When used with mod_auth_gssapi, the module may not be usable and requires Negotiate at every request*. Apr. 18, 2017. URL: https://www.drupal.org/node/2859664 (visited on 04/20/2017).

[97] [Richard Kalinec]. *Existing users are never matched unless creating new users is enabled*. Apr. 21, 2017. URL: https://www.drupal.org/node/2859747 (visited on 04/21/2017).

[98] [Contributors to Devel]. *Devel*. Oct. 1, 2016. URL: https://www.drupal.org/project/devel (visited on 04/24/2017).

[99] [Drupal community with support of the Drupal Association]. *Clearing or rebuilding Drupal's cache*. Dec. 29, 2016. URL: https://www.drupal.org/docs/7/administering-drupal-7-site/clearing-or-rebuilding-drupals-cache (visited on 04/24/2017).

[100] [Contributors to Drupal Answers]. *How do I call the menu_rebuild function in Drupal 7?* Jan. 27, 2016. URL: https://drupal.stackexchange.com/questions/58618/how-do-i-call-the-menu-rebuild-function-in-drupal-7/ (visited on 04/24/2017).

[101] [Contributors to Drupal forum]. *hook_menu_alter is not being called*. July 4, 2014. URL: https://www.drupal.org/node/1054402 (visited on 04/24/2017).

[102] [Contributors to Stack Overflow]. *hook_menu_alter or hook_menu_link_alter?* June 18, 2014. URL: http://stackoverflow.com/questions/18442622/hook-menu-alter-or-hook-menu-link-alter (visited on 04/24/2017).

[103] [Richard Kalinec]. *External authentication is mandatory, login form is not available anymore*. Apr. 24, 2017. URL: https://www.drupal.org/node/2859975 (visited on 04/24/2017).

[104] [Richard Kalinec]. *Additional information about the user cannot be retrieved from the external IdP*. Apr. 26, 2017. URL: https://www.drupal.org/node/2865538 (visited on 04/26/2017).

[105] [Contributors to Fedora Project Wiki]. *How to install Drupal*. Oct. 29, 2015. URL: https://fedoraproject.org/wiki/How_to_install_Drupal (visited on 12/22/2016).

[106] [Drupal community with support of the Drupal Association]. *Enable clean URLs*. Apr. 11, 2017. URL: https://www.drupal.

org/docs/7/configuring-clean-urls/enable-clean-urls (visited on 04/13/2017).

[107] Jan De Clercq. *Understanding How Kerberos Authentication Protects Against Replay Attacks*. Nov. 14, 2013. URL: http://windowsitpro.com/active-directory/understanding-how-kerberos-authentication-protects-against-replay-attacks (visited on 05/07/2017).

[108] DFIR blog. *Protecting Windows Networks – Kerberos Attacks*. Dec. 19, 2015. URL: https://dfir-blog.com/2015/12/13/protecting-windows-networks-kerberos-attacks/ (visited on 05/07/2017).

[109] Sean Metcalf. *Attack Methods for Gaining Domain Admin Rights in Active Directory*. Jan. 1, 2016. URL: https://adsecurity.org/?p=2362 (visited on 05/07/2017).

[110] Sean Metcalf. *Cracking Kerberos TGS Tickets Using Kerberoast – Exploiting Kerberos to Compromise the Active Directory Domain*. Dec. 31, 2015. URL: https://adsecurity.org/?p=2293 (visited on 05/07/2017).

[111] Matan Hart. *Kerberos Attacks: What You Need to Know*. May 23, 2016. URL: https://www.cyberark.com/blog/kerberos-attacks-what-you-need-to-know/ (visited on 05/07/2017).

[112] Skip Duckwall and Benjamin Delpy. *Abusing Kerberos*. Aug. 8, 2014. URL: https://www.blackhat.com/docs/us-14/materials/us-14-Duckwall-Abusing-Microsoft-Kerberos-Sorry-You-Guys-Don%27t-Get-It-wp.pdf (visited on 05/08/2017).

[113] Sean Metcalf. *Mimikatz and Active Directory Kerberos Attacks*. Nov. 22, 2014. URL: https://adsecurity.org/?p=556 (visited on 05/07/2017).

[114] Ian Haken. *Bypassing Local Windows Authentication to Defeat Full Disk Encryption*. Nov. 12, 2015. URL: https://www.blackhat.com/docs/eu-15/materials/eu-15-Haken-Bypassing-Local-Windows-Authentication-To-Defeat-Full-Disk-Encryption-wp.pdf (visited on 05/07/2017).

[115] [Contributors to the discussion on Reddit]. *[Reddit -> Information Security News Subreddit -> Protecting Windows Networks – Kerberos Attacks]*. Dec. 22, 2015. URL: https://www.reddit.com/r/netsec/comments/3wnvm6/protecting_windows_networks_kerberos_attacks/ (visited on 05/07/2017).

[116] Sean Metcalf. *Kerberos & KRBTGT: Active Directory's Domain Kerberos Service Account*. Nov. 10, 2014. URL: https://adsecurity.org/?p=483 (visited on 05/07/2017).

[117] SecurityGlobal.net LLC. *View Topics > Target > System Security Services Daemon (SSSD)*. Nov. 20, 2015. URL: http://securitytracker.com/archives/target/10630.html (visited on 05/09/2017).

[118] [Drupal community with support of the Drupal Association]. *Web server*. Aug. 18, 2016. URL: https://www.drupal.org/docs/7/system-requirements/web-server (visited on 10/09/2016).

[119] [Drupal community with support of the Drupal Association]. *What is PDO?* Aug. 4, 2016. URL: https://www.drupal.org/docs/7/system-requirements/what-is-pdo (visited on 10/09/2016).

[120] [Drupal community with support of the Drupal Association]. *PHP*. Oct. 9, 2016. URL: https://www.drupal.org/docs/7/system-requirements/php (visited on 10/09/2016).

[121] [Drupal community with support of the Drupal Association]. *Database server*. Aug. 4, 2016. URL: https://www.drupal.org/docs/7/system-requirements/database-server (visited on 10/09/2016).

[122] [Drupal community with support of the Drupal Association]. *Browser requirements*. Aug. 4, 2016. URL: https://www.drupal.org/docs/7/system-requirements/browser-requirements (visited on 10/09/2016).

[123] [Drupal community with support of the Drupal Association]. *user.admin.inc*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.admin.inc/7.x (visited on 10/15/2016).

[124] [Drupal community with support of the Drupal Association]. *user.api.php*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.api.php/7.x (visited on 10/15/2016).

[125] [Drupal community with support of the Drupal Association]. *user.install*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.install/7.x (visited on 10/15/2016).

[126] [Drupal community with support of the Drupal Association]. *user.module*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.module/7.x (visited on 10/15/2016).

[127] [Drupal community with support of the Drupal Association]. *user_form_test.module*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!tests!user_form_test.module/7.x (visited on 10/15/2016).

[128] [Drupal community with support of the Drupal Association]. *user.pages.inc*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.pages.inc/7.x (visited on 10/15/2016).

[129] [Drupal community with support of the Drupal Association]. *user.tokens.inc*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user.tokens.inc/7.x (visited on 10/15/2016).

[130] [Drupal community with support of the Drupal Association]. *user-picture.tpl.php*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user-picture.tpl.php/7.x (visited on 10/15/2016).

[131] [Drupal community with support of the Drupal Association]. *user-profile.tpl.php*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user-profile.tpl.php/7.x (visited on 10/15/2016).

[132] [Drupal community with support of the Drupal Association]. *user-profile-category.tpl.php*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user-profile-category.tpl.php/7.x (visited on 10/15/2016).

[133] [Drupal community with support of the Drupal Association]. *user-profile-item.tpl.php*. Oct. 15, 2016. URL: https://api.drupal.org/api/drupal/modules!user!user-profile-item.tpl.php/7.x (visited on 10/15/2016).

[134] [Contributors to FreeIPA]. *Howto/HBAC and allow all*. Aug. 23, 2016. URL: http://www.freeipa.org/page/Howto/HBAC_and_allow_all (visited on 12/03/2016).

A PHP

PHP [14] is an open-source general-purpose scripting language that can be embedded into HTML using start and end processing instructions `<?php` and `?>` [15, 16] or it can be used in combination with various web template systems, web content management systems and web frameworks [17]. Its syntax borrows from C, Java and Perl [15].

This language is primarily designed for web development and its main goal is to allow web developers to write dynamically generated web pages quickly [15]. However, it can also be used to work with files on the server, collect form data, send and receive cookies, work with a wide range of databases, control user-access, encrypt data, interact with other services using various means and protocols, process text and output HTML or several other types of files (text, XHTML, XML, PDF, images, Flash movies) [18, 19]. The language also supports many extensions that provide additional functionality, for example parsing and accessing XML documents [18].

PHP is focused on server-side scripting. This requires a web server, such as Apache [20], PHP and a database, such as MySQL, to be installed in the case of developing locally [21]. If using a remote server, the developer has to make sure that it has activated support for PHP [21]. The code is usually interpreted and executed on the server by a PHP parser, which can be implemented either as a server module or a CGI (Common Gateway Interface [22]) executable [16–18]. Files containing PHP code most often have `.php` extension [21]. The web server subsequently combines the output of processed PHP code with the generated web page [17]. In the end, the client receives the resulting HTML file that is displayed by a web browser, therefore he does not know what the underlying code was [16]. It is even possible to configure the web server to process all HTML files with PHP, eliminating the ability to recognize if there is PHP code in the original file, or not [16]. Nevertheless, PHP scripts can be run from the command line using only the PHP parser [18]. This alternative is suitable mainly for scripts that are regularly executed by the operating system's task scheduler or for simple text processing tasks [18]. It is also possible to write cross-platform desktop applications with a graphical user

interface using the extension of PHP – PHP-GTK [18]. However, PHP is usually not a preferable alternative for such use [18].

PHP can be used on almost every operating system, including all major ones [17, 18]. It also supports the majority of web servers [17, 18]. PHP is the most popular web development language that is installed on the vast majority of servers [23]. According to *w3techs.com*, as of August 2016, it is used by over 80 percent of websites [24]. For example, it powers Facebook, Twitter, Pinterest, VKontakte, Baidu and web content management systems including WordPress, Drupal, Joomla, MediaWiki and Moodle [17, 24]. The language is praised for a low barrier of entry and a rich offer of features for professional programmers, as well as its documentation, availability of online resources and a supportive community [16, 23, 25, 26]. Furthermore, it provides developers flexibility and the choice between procedural programming, object-oriented programming or a combination of both [18, 25]. On the other hand, PHP is criticised for inconsistency and lack of focus [23, 25]. What is more, its easiness to learn and popularity resulted in many PHP developers producing code with bad practices and security vulnerabilities [23, 25, 26]. Last but not least, as PHP is an interpreted language, its performance may suffer [26]. This is addressed by the Zend Engine, which is the original, only complete and most widely used PHP implementation that also serves as the de facto reference implementation, because PHP still does not have a written formal specification, although work on its creation is underway since 2014 [17]. The standard PHP parser, powered by the Zend Engine, is a free software [17]. PHP and its implementations continue to evolve – new features are added, while some features enabling bad practices and opening security vulnerabilities are gradually removed from the language [17, 23].

B Environment variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer [55]. They are a part of the environment in which a process runs [55]. Running processes can query them for example to determine in which folder to search for a particular category of data [55]. Environment variables are usually referenced by putting special symbols in front of or around the variable name [55].

There are two kinds of environment variables that affect Apache, as described on *httpd.apache.org* (partially paraphrased) [56]:

- Environment variables controlled by the underlying operating system. These are set before the server starts. They can be used in expansions in configuration files, and can optionally be passed to the web applications.

- Environment variables stored and manipulated in an internal Apache structure. These variables can be used to control various operations such as logging or access control. They also serve as a mechanism to communicate with external programs including the web applications.

The `REMOTE_USER` environment variable belongs to the latter category. In general, this is another mechanism shared with HTTP Basic and Digest Authentication [52].

In PHP, however, there are special environment variables for Basic Authentication – `PHP_AUTH_USER` (the username) and `PHP_AUTH_PW` (the password), as well as for Digest Authentication – `PHP_AUTH_DIGEST` (the Authorization header) [57, 58]. However, `REMOTE_USER` can be also used to identify an externally authenticated user in PHP; `REDIRECT_REMOTE_USER` does the same if the request had been internally redirected [57, 58]. These variables are stored in the `$_SERVER` array [57, 58]. For example, in the module to extend (described in *Chapter 4*), they are accessed like this (*Listing A.1*):

```
function _webserver_auth_retrieve_remote_user() {
  // Get the login name from the $_SERVER variable.
  $authname = '';
  if (isset($_SERVER ['REDIRECT_REMOTE_USER'])) {
    $authname = $_SERVER ['REDIRECT_REMOTE_USER'];
  }
  elseif (isset($_SERVER ['REMOTE_USER'])) {
    $authname = $_SERVER ['REMOTE_USER'];
  }
  elseif (isset($_SERVER ['PHP_AUTH_USER'])) {
    $authname = $_SERVER ['PHP_AUTH_USER'];
  }
  return $authname;
}
```

Listing B.1: Retrieving REMOTE_USER & co.

C Drupal – selected additional information

C.1 Drupal system requirements

- Web server – Apache is most commonly used and best tested regarding Drupal's operation, but other web servers with PHP support are also supported [118]. Some security features are provided only for Apache and, to a lesser extent, to Internet Information Services (IIS) – these should be recreated when using any other web server [118]. The libraries for PHP Data Objects (PDO) need to be installed and enabled as well [119].

- PHP – Drupal is designed to work with the distribution of PHP from *php.net* [120]. Best effort is made to make it work with other PHP distributions, but proper functioning is not guaranteed [120]. Whatever distribution is used, some configuration and extensions (at least for certain functions) are needed [120].

- Database server – MySQL or an equivalent such as MariaDB or Percona Server is recommended [121]. However, PostgreSQL is also supported, as well as SQLite since Drupal 7 [121]. Additional modules can be installed to support Microsoft SQL Server, Oracle, and MongoDB [121].

- Browser – websites built using only the Drupal core are fully functional in all modern browsers that support CSS and JavaScript [122]. However, varying levels of compliance with internet standards may produce minor differences in the appearance across browsers [122]. The Drupal core can be also used with browsers that do not support JavaScript and/or CSS, but this will lead to alterations in the functionality and/or the appearance, respectively [122]. Some contributed modules and themes may not be compatible with all browsers or require JavaScript to work at all [122].

As for all these requirements, only versions from a particular milestone onward are supported. This milestone may differ for every major version of Drupal [118, 120–122].

C.2 Drupal User module file structure

The module's program code consists of the following files [85]:

1. `user.admin.inc` [123] – the administration page callback file for the User module,
2. `user.api.php` [124] – the hooks provided by the User module,
3. `user.install` [125] – the installation, update and uninstallation functions for the User module,
4. `user.module` [126] – the main file of the User module, which enables the user registration and login system,
5. `user_form_test.module` [127] – a dummy module implementing a form to test user password validation,
6. `user.pages.inc` [128] – the user page callback file for the User module,
7. `user.tokens.inc` [129] – builds placeholder replacement tokens for user-related data.

Furthermore, the module contains the template files, which are default themes implementations [85]:

1. `user-picture.tpl.php` [130] – presents a picture configured for the user's account.
2. `user-profile.tpl.php` [131] – presents all user profile data. This template is used when viewing a registered member's profile page, e.g. *example.com/user/user_ID*.
3. `user-profile-category.tpl.php` [132] – presents profile categories (groups of profile items).
4. `user-profile-item.tpl.php` [133] – presents profile items (values from user account profile fields or modules). This template is used to loop through and render each field configured for the user's account. It can also be data from modules. The output is grouped by categories.

The remaining files making up this module are `user.info`, `user.css`, `user-rtl.css`, `user.js` and `user.permissions.js`.

C.3 Drupal identity management schema in detail

1. `authmap` – stores the distributed authentication mapping, contains the following fields:
   (a) `aid` – a unique authmap ID, the primary key,
   (b) `uid` – the user's uid, a foreign key connected to the `uid` field of the `users` table,
   (c) `authname` – a unique authentication name,
   (d) `module` – the module controlling the authentication.
2. `role_permission` – stores the permissions assigned to user roles, contains the following fields:
   (a) `rid` – a foreign key connected to the `rid` field of the `role` table,
   (b) `permission` – a single permission granted to the role identified by `rid`,
   (c) `module` – the module declaring the permission.
3. `role` – stores user roles, contains the following fields:
   (a) `rid` – a unique role ID, the primary key,
   (b) `name` – a unique role name,
   (c) `weight` – the weight of the role in listings and the user interface.
4. `users` – stores user data, contains the following fields:
   (a) `uid` – a unique user ID, the primary key,
   (b) `name` – a unique user name,
   (c) `pass` – the hash of the user's password,
   (d) `mail` – the user's email address,
   (e) `theme` – the user's default theme,
   (f) `signature` – the user's signature,

(g) `signature_format` – the format of the signature, a foreign key connected to the `format` filed of the `filter_format` table,

(h) `created` – the timestamp for when the user was created,

(i) `access` – the timestamp for previous time user accessed the site,

(j) `login` – the timestamp for user's last login,

(k) `status` – indicates whether the user is active (1) or blocked (0),

(l) `timezone` – the user's time zone,

(m) `language` – the user's default language,

(n) `picture` – the `fid` of the user's picture, a foreign key connected to the `fid` field of the `file_managed` table,

(o) `init` – the email address used for initial account creation,

(p) `data` – a serialized array of name value pairs that are related to the user; use of this field is discouraged and it will likely disappear in a future version of Drupal.

5. `users_roles` – maps users to roles, contains the following fields:

   (a) `uid` – the `uid` for the user, the primary key and a foreign key connected to the `uid` field of the `users` table,

   (b) `rid` – the `rid` for the role, the primary key and a foreign key connected to the `rid` field of the `role` table.

The `uid = 0` row in the `users` table is assigned for anonymous users [83]. This row must exist, otherwise the site will have severe problems [83].

D Structure of the Webserver authentication module

In this thesis, the version 7.x-1.6 of the Webserver authentication module is described and built upon. Its shortened name, which is used in the program code, is `webserver_auth`. The module consists of 4 files:

- `license.txt` – GPLv2 (GNU General Public License Version 2),
- `readme.txt` – provides a brief description of the module's functionality as well as installation and configuration instructions,
- `webserver_auth.info` – contains information about the module for use by Drupal,
- `webserver_auth.install` – installation file for the module,
- `webserver_auth.module` – the main source code of the module.

The installation file is 25 lines long and it contains one function – `webserver_auth_uninstall()`, which implements `hook_uninstall()`. It only deletes the module's configuration variables from the database during an uninstallation of the module. The main source code is 557 lines long and it consists of 22 functions, with 11 of them being the implementations of core hooks. The names of all the functions start with '`webserver_auth_`', the names of the functions that do not implement any hook are additionally prefixed with an underscore (_) for distinction. These functions perform the following tasks:

1. `webserver_auth_menu()`
   - Implements `hook_menu()`.
   - It adds the module's settings page to the configuration by configuring the `drupal_get_form()` function to display the module's settings form at the `admin/config/people/webserver_auth` menu path using the `_webserver_auth_settings()` function.
2. `webserver_auth_menu_get_item_alter(&$router_item, $path, $original_map)`

- Implements `hook_menu_get_item_alter()`.
- It checks if the login has already been checked using the `drupal_static(__FUNCTION__)` variable, and if not, it calls the `_webserver_auth_route()` function.

3. `webserver_auth_boot()`

   - Implements `hook_boot()`.
   - It checks if a cached page is being served to an authenticated user, and if this is the case, forces the start of a session and reloads the page. As for interactions with other components of the module, it calls the `_webserver_auth_retrieve_remote_user()` and `_webserver_auth_route()` functions, and it uses the `webserver_auth` cookie to check if the verification of the current user was attempted recently.

4. `_webserver_auth_logout()`

   - Logs the current user out and ends the current session.

5. `_webserver_auth_route()`

   - Validates that the remote user name from the environment variables and the one from the current session are synchronized, and if not, ends the current session and attempts a fresh login for the remote user. It calls the `_webserver_auth_retrieve_remote_user()`, `_webserver_auth_logout()`, `_webserver_auth_attempt_login()` and `webserver_auth_user_presave()` functions (the last of them being a hook implementation) and it indirectly invokes the `webserver_auth_webserver_auth_authname_alter()` hook implementation. As for variables, it uses the `webserver_authname` session variable to keep track of what user is logged in with the current session, the `webserver_auth_create_user` configuration variable to decide if it should create accounts for unknown users and the `webserver_auth_match_existing` configuration variable to decide whether to create missing records for this module in the `authmap` table for existing users. The latter configuration

variable must be initially TRUE, so as to administrators are not logged out when they enable the module.

6. `_webserver_auth_retrieve_remote_user()`

   - Returns the remote user name or an empty string.
   - It retrieves the remote user name from the `REDIRECT_REMOTE_USER`, `REMOTE_USER` or `PHP_AUTH_USER` environment variable. If the user is logged in to the web server, one of these environment variables will contain the remote user name.

7. `_webserver_auth_attempt_login($preparedauthname)`

   - Returns a Boolean denoting whether the user is found and logged in or not.
   - It attempts to log the user specified by the parameter in. It uses the `webserver_auth_skip_check` configuration variable to decide whether to check only the username in the `users` table or whether to check the module name and username in the `authmap` table to map the username to the Drupal user. If the user has been found and is being logged in, it sets the `webserver_authname` session variable for comparison in the `_webserver_auth_route()` function.

8. `webserver_auth_user_presave(&$edit, $account, $category)`

   - Implements `hook_user_presave()`.
   - If `$edit['mail']` is not set and if the module has an email domain configured by the `webserver_auth_email_domain` configuration variable, it creates an email address for the user by concatenating his name in `$edit['name']` and domain in `webserver_auth_email_domain` and stores it into `$edit['mail']` – it removes any `@` from domain to prevent double `@` in the email address and then replaces any spaces in the resulting email address with dots.

9. `webserver_auth_user_insert(&$edit, $account, $category)`

- Implements `hook_user_insert()`.
- If the `webserver_auth_insert` configuration variable contains some code, it executes it. Then, if the `webserver_auth_add_all_new` configuration variable is TRUE, it adds all new users created using any means to the `authmap` table.

10. `_webserver_auth_settings($form, &$form_state)`

    - Defines the module's administration settings form. This is where the configuration variables are set. For the case that they were not set before, and therefore they do not have records in the database, all the ones of the Boolean type have a default value set to TRUE or FALSE, while all the ones of the string type contain an empty string as a default value. If they are set, they are stored into the database.

11. `webserver_auth_help($path, $arg)`

    - Implements `hook_help()`.
    - It configures the help text for the path `admin/help#webserver_auth`.

12. `webserver_auth_form_user_profile_form_alter(&$form, &$form_state, $form_id)`

    - Implements `hook_form_FORM_ID_alter()`, `FORM_ID = user_profile_form`.
    - It disables the ability to change the username and/or password if the `webserver_auth_disallow_username_change` and/or `webserver_auth_disallow_pw_change` configuration variables are TRUE, respectively.

13. `webserver_auth_webserver_auth_authname_alter(&$username)`

    - Implements `hook_webserver_auth_authname_alter()`.
    - Strips the prefix (the entire part of the username up to the last backslash (\) in it) and/or suffix (the entire part of the

username starting from the first at (`@`) in it) from the username if the `webserver_auth_strip_prefix` and/or `webserver_auth_strip_domain` configuration variables are TRUE, respectively.

14. `webserver_auth_block_info_alter(&$blocks, $theme, $code_blocks)`

    - Implements `hook_block_info_alter()`.
    - It disables the login block.

15. `_webserver_auth_user_page()`

    - The user page callback. If a user is authenticated, it falls back to the User module's page for authenticated users. If a user is anonymous, it redirects him to the homepage.

16. `_webserver_auth_redirect($config_name)`

    - The redirect callback.

17. `_webserver_auth_login_access()`

    - The login access callback. If a user is anonymous, it returns a Boolean denoting whether the `webserver_auth_login_url` configuration variable contains something. If a user is logged in, it returns FALSE.

18. `_webserver_auth_login_page()`

    - The login page callback. It redirects to the URL configured by the `webserver_auth_login_url` configuration variable using the `_webserver_auth_redirect()` function.

19. `_webserver_auth_logout_access()`

    - The logout access callback. If a user is logged in, it returns a Boolean denoting whether the `webserver_auth_logout_url` configuration variable contains something. If a user is anonymous, it returns FALSE.

20. `_webserver_auth_logout_page()`

- The logout callback. It calls the `_webserver_auth_logout()` function and subsequently redirects to the URL configured by the `webserver_auth_logout_url` configuration variable using the `_webserver_auth_redirect()` function.

21. `webserver_auth_menu_alter(&$items)`

    - Implements `hook_menu_alter()`.
    - It disables the `user/register` and `user/password` tasks. Afterwards, it sets redirecting the user page for anonymous users by setting the page callback of the user page to point to the `_webserver_auth_user_page()` function. In the end, it customizes login and logout by setting the login/logout page/access callbacks to the corresponding of the previous four functions and transforming the `user/login` task to an item of the User menu, what practically makes it a 'Log in' link.

22. `webserver_auth_menu_link_alter(&$link)`

    - Implements `hook_menu_link_alter()`.
    - It forces the 'Log in' link to appear on the top-level of the login menu by default (unless it has been customized, for example).

Figure D.1 on the page *63* comprehensively visualizes the structure of the module. The meaning of its elements is as follows:

- orange rectangles denote hook implementations,
- cyan rectangles denote functions that do not implement any hook,
- grey rectangles denote configuration variables of the module that are administered by the `_webserver_auth_settings()` function,
- green rectangle denotes a `drupal_static(__FUNCTION__)` variable,
- pink rectangle denotes a session variable,

- red rectangle denotes a cookie.

The arrows between the elements mean:

- black arrow denotes a function call,
- blue arrow denotes using a variable or cookie,
- arrow with a full triangle (▼) ending denotes that function call/variable usage is always performed
- arrow with a plain 'V' ending denotes that function call/variable usage is not always performed

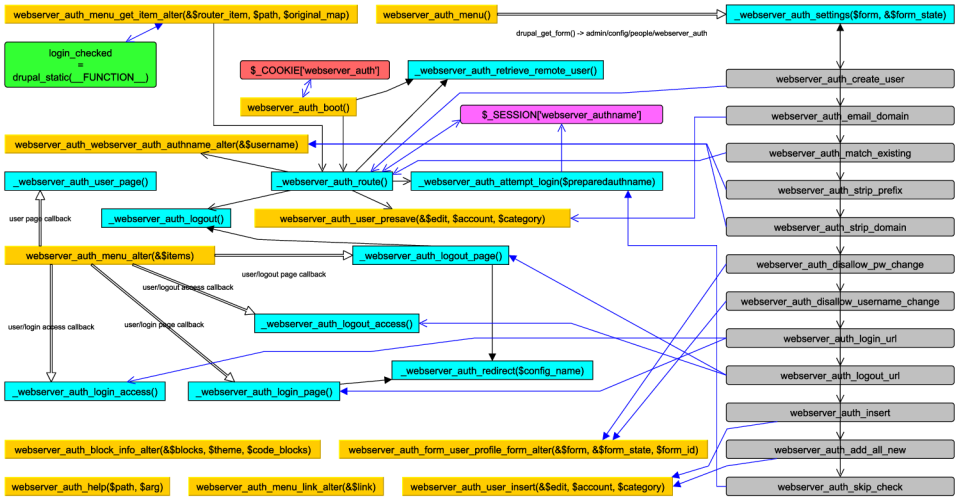

Figure D.1: Diagram visualizing the structure of the module.

E Functionality of the original Webserver authentication module in pictures

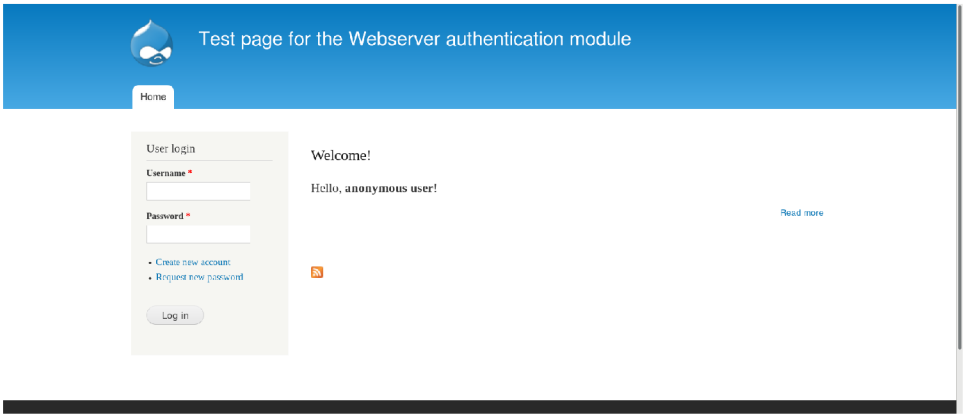

Figure E.1: The test page viewed by an anonymous user

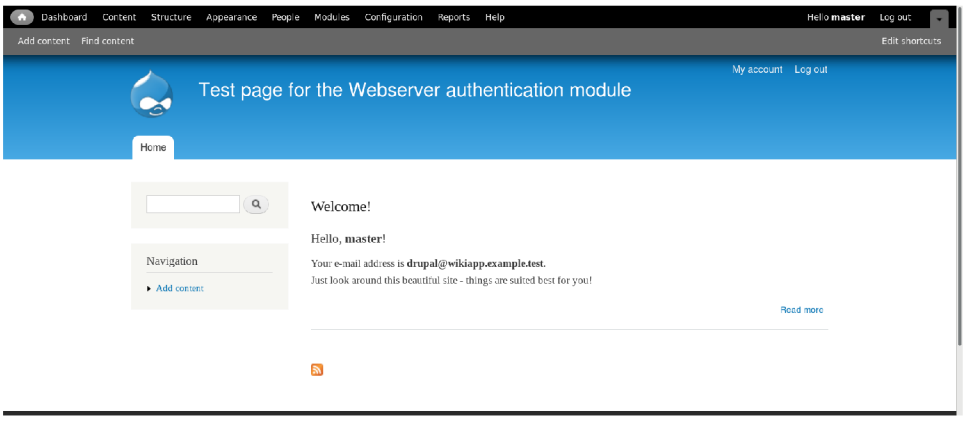

Figure E.2: The test page viewed by `master`

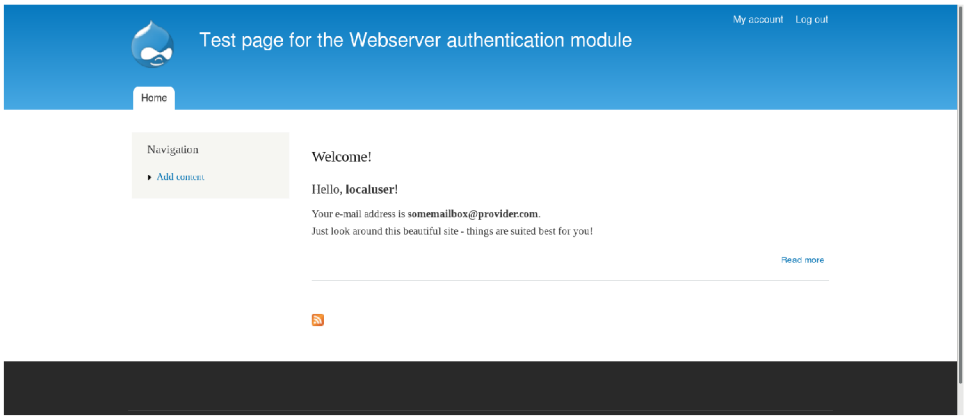

Figure E.3: The test page viewed by `localuser`

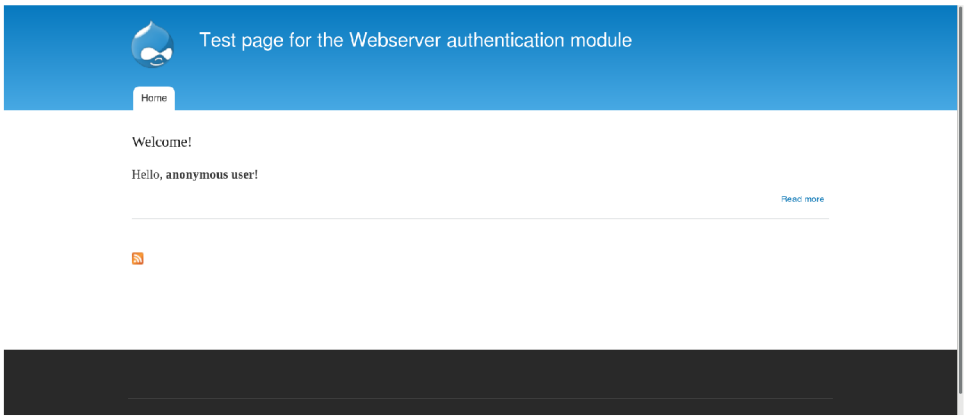

Figure E.4: The test page viewed by externally authenticated `master` *after the module has been enabled*

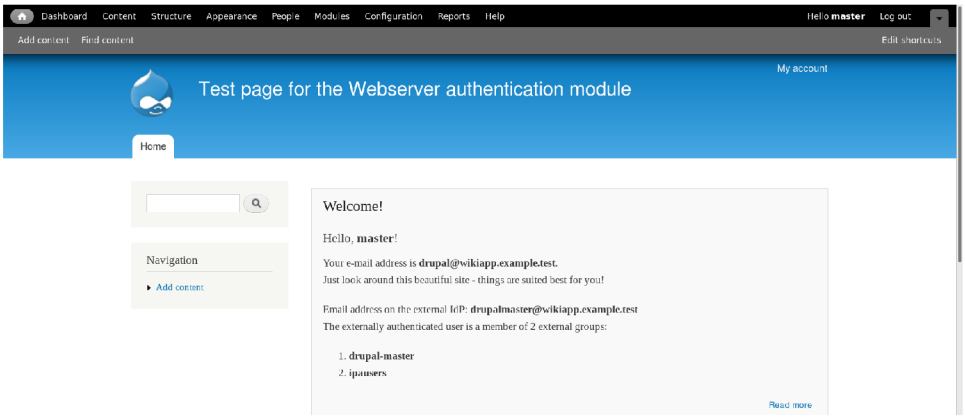

Figure E.5: The test page viewed by externally authenticated `master` *after the module has been enabled and the database manually edited*

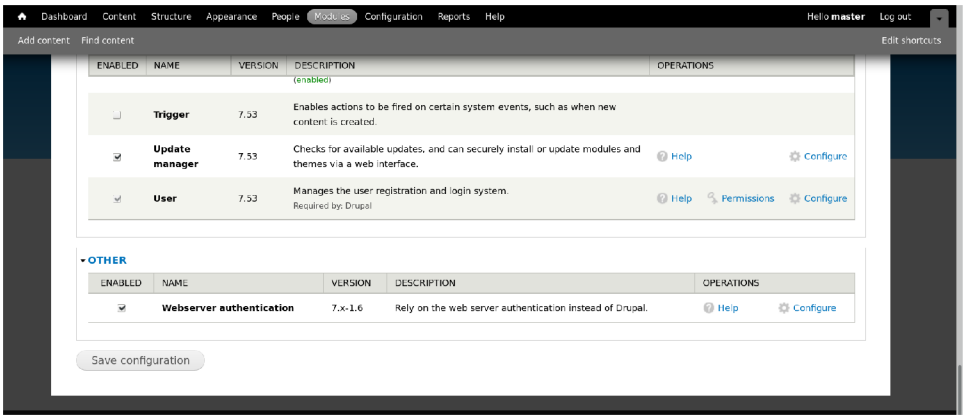

Figure E.6: The first way of navigating to the module's configuration page

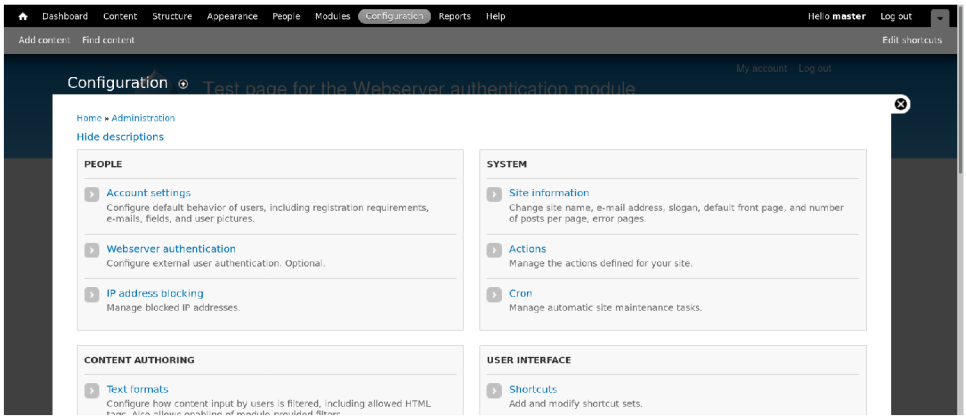

Figure E.7: The second way of navigating to the module's configuration page

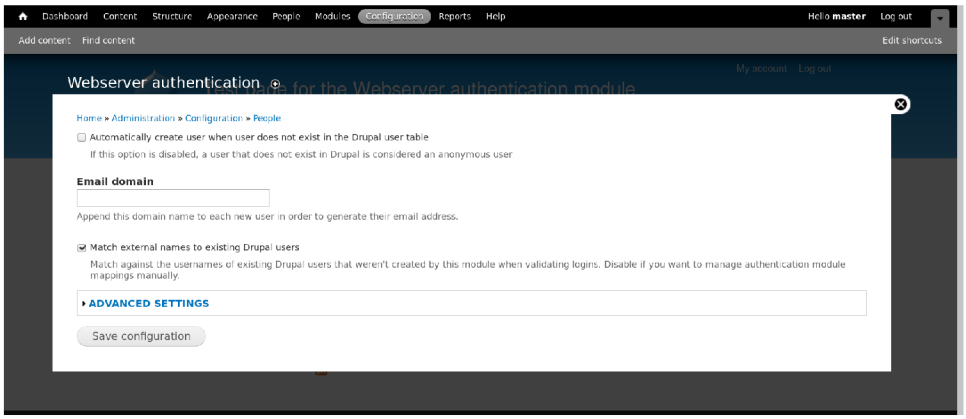

Figure E.8: The module's configuration page with 'Advances settings' section collapsed

F The source codes of the patches making up the extension of the module

F.1 The first patch

```
if (! $authname) {
  // Empty authname is anonymous: close any current
   session.
  if (user_is_logged_in()) {
    _webserver_auth_logout();
  }
```

Listing F.1: Before: Logging the user out when REMOTE_USER is empty

```
/* Empty authname is anonymous: if configured so, close
   any current session. This is FALSE by default,
   as it may ultimately prevent mod_auth_gssapi
   authentication from working, or at least it makes it
   practically impossible to configure it only for a
   login page, what effectively means a requirement
   for HTTP Negotiate at every request. */
if (variable_get('webserver_auth_logout_empty_remote_user
   ', FALSE) && ! $authname) {
  if (user_is_logged_in()) {
    _webserver_auth_logout();
  }
```

Listing F.2: After: Added a condition that the new configuration variable must be TRUE

```
/* Check if the authname matches the current session.
   If it does, we're done. */
if (user_is_logged_in()) {

  /* Validate session authname.  We don't match against
   uid as authmap may
     link to a user with uid different than authname. */
  if (isset($_SESSION['webserver_authname'])
&& ($_SESSION['webserver_authname'] == $authname)) {
    return;
  }
```

Listing F.3: Before: When the `_webserver_auth_route()` function exits

```
/* Check if the authname matches the current session. If
   it does, we're done.
   Empty authname here means that the module is not
   configured to logout the user
   in this case, and therefore we're done. However, in
   the case it is nonempty
   and different from the current session, we want to log
    the current user out
   and attempt to log in the user represented by the new
   authname. */
if (user_is_logged_in()) {

  /* Validate session authname.  We don't match against
   uid as authmap may
     link to a user with uid different than authname. */
  if (! $authname || isset($_SESSION['webserver_authname'
   ])
&& ($_SESSION['webserver_authname'] == $authname)) {
    return;
  }
```

Listing F.4: After: Added a condition that `$authname` is empty

```
'webserver_auth_logout_empty_remote_user' => array(
  '#type' => 'checkbox',
  '#title' => t('Logout the current user if the authname
   is empty'),
  '#default_value' => variable_get('
   webserver_auth_logout_empty_remote_user', FALSE),
  '#description' => t("Logout the current user if the
   remote user name retrieved from the environment
   variables is empty. This must be disabled for
   mod_auth_gssapi to work."
. "  Enable if you are not going to use mod_auth_gssapi,
   but other authentication methods supported by this
   module."),
),
```

Listing F.5: The newly created item in the settings form (`_webserver_auth_settings()`)

```
variable_del('webserver_auth_logout_empty_remote_user');
```

Listing F.6: Deletion of the new configuration variable during the module's uninstallation

F.2 The second patch

```
if (_webserver_auth_attempt_login($authname)) {
  return;
}

// Check if we should create accounts for unknown users
if (! variable_get('webserver_auth_create_user', FALSE))
   {
  watchdog('webserver_auth', 'user @authname does not
   exist in the database or is not included in the
   authmap table for webserver_auth, and the create user
   option is disabled.', array('@authname' => $authname))
   ;
  return;
}

// First check if just authmap is missing
$account = user_load_by_name($authname);
if ($account) {
```

Listing F.7: Before: Check whether to create users before potential setting authmaps

```
// add missing authmap
  user_set_authmaps($account, array("
   authname_webserver_auth" => $authname));

  // now try login again
  if (! _webserver_auth_attempt_login($authname)) {
    watchdog('webserver_auth', 'user @authname blocked.',
    array('@authname' => $authname), WATCHDOG_INFO);
  }
  return;
}

// Check if we should create accounts for unknown users
if (! variable_get('webserver_auth_create_user', FALSE))
   {
  watchdog('webserver_auth', 'user @authname does not
   exist in the database or is not included in the
   authmap table for webserver_auth, and the create user
   option is disabled.', array('@authname' => $authname))
   ;
  return;
}
```

```
/* We didn't find the user so we create an account for
   them.
   Note, hook_user_presave() is called during this
   process.
   That's where we modify the user account to add email
   etc. */
try {
  user_external_login_register($authname, 'webserver_auth
   ');
  $_SESSION['webserver_authname'] = $authname;
}
```

Listing F.8: After: Check whether to create users after potential setting authmaps

F.3 The third patch

```
function webserver_auth_block_info_alter(&$blocks, $theme
   , $code_blocks) {
  // Disable the login block.
  $blocks['user']['login']['status'] = 0;
}
```

Listing F.9: Before: Disabling the Login block

```
function webserver_auth_block_info_alter(&$blocks, $theme
   , $code_blocks) {
  if (!variable_get('
   webserver_auth_optional_external_auth', FALSE)) {
    // Disable the login block.
    $blocks['user']['login']['status'] = 0;
  }
  else {
    // Enable the login block and specify default region
   for it.
    $blocks['user']['login']['status'] = 1;
    $blocks['user']['login']['region'] = 'sidebar_first';
  }
}
```

Listing F.10: After: Disabling or enabling the Login block depending of the new configuration variable

```
function webserver_auth_menu_alter(&$items) {
  /* Disable register and password tasks */
  $items['user/register']['access callback'] = FALSE;
```

```
  $items['user/password']['access callback'] = FALSE;
  /* Redirect user page if anonymous */
  $items['user']['page callback'] = '
    _webserver_auth_user_page';
  /* Customize login/logout */
  $items['user/login']['page callback'] = '
    _webserver_auth_login_page';
  $items['user/login']['type'] = MENU_NORMAL_ITEM;
  $items['user/login']['menu_name'] = 'user-menu';
  $items['user/login']['access callback'] = '
    _webserver_auth_login_access';
  $items['user/logout']['page callback'] = '
    _webserver_auth_logout_page';
  $items['user/logout']['access callback'] = '
    _webserver_auth_logout_access';
}
```

Listing F.11: Before: The original `webserver_auth_menu_alter()` function

```
function webserver_auth_menu_alter(&$items) {
  /* If external auth is configured as mandatory: */
  if (!variable_get('
    webserver_auth_optional_external_auth', FALSE)) {
      // ...EXISTING CODE OF THE METHOD...
  }
  /* If external auth is configured as optional: */
  else {
      /* Create new links for external login/logout */
      $items['user/external_login']['title'] = 'Log in
    externally';
      $items['user/external_login']['page callback'] = '
    _webserver_auth_login_page';
      $items['user/external_login']['type'] =
    MENU_NORMAL_ITEM;
      $items['user/external_login']['menu_name'] = 'user-
    menu';
      $items['user/external_login']['access callback'] = '
    _webserver_auth_login_access';
      $items['user/external_logout']['title'] = 'Log out
    externally';
      $items['user/external_logout']['page callback'] = '
    _webserver_auth_logout_page';
      $items['user/external_logout']['access callback'] = '
    _webserver_auth_logout_access';
      $items['user/external_logout']['weight'] = 5;
```

```
    $items['user/external_logout']['menu_name'] = 'user-
    menu';
  }
}
```

Listing F.12: After: The extended `webserver_auth_menu_alter()` function

```
function webserver_auth_menu_link_alter(&$link) {
  // Force the Login link to appear on the top-level of '
    user-menu' menu by
  // default (i.e., unless it has been customized).
  if ($link['link_path'] == 'user/login' && $link['module
    '] == 'system' && empty($link['customized'])) {
    $link['plid'] = 0;
  }
}
```

Listing F.13: Before: The original `webserver_auth_menu_link_alter()` function

```
function webserver_auth_menu_link_alter(&$link) {
  // Force the Login link to appear on the top-level of '
    user-menu' menu by
  // default (i.e., unless it has been customized).
  if (!variable_get('
    webserver_auth_optional_external_auth', FALSE)) {
    if ($link['link_path'] == 'user/login' && $link['
    module'] == 'system' && empty($link['customized'])) {
      $link['plid'] = 0;
    }
  }
  else {
    if ($link['link_path'] == 'user/external_login' &&
    $link['module'] == 'system' && empty($link['customized
    '])) {
      $link['plid'] = 0;
    }
  }
}
```

Listing F.14: After: The extended `webserver_auth_menu_link_alter()` function

```
'webserver_auth_optional_external_auth' => array(
  '#type' => 'checkbox',
  '#title' => t('Preserve the ability to log in using
    local username + password and login form'),
```

```
  '#default_value' => variable_get('
   webserver_auth_optional_external_auth', FALSE),
  '#description' => t("With this option disabled (default
   ), users will be able to log in only by using one of
   supported external authentication methods and the
   login form will be disabled."
. " When this option is enabled, the login form is re-
   enabled, users can log in also by using their username
    and password and external authentication happens
   after clicking on 'Log in externally'"
. " button. IMPORTANT: in order to this setting having
   real effect, webserver_auth_login_url must be nonempty
   !"),
),
```

Listing F.15: The new item in the settings form (`_webserver_auth_settings()`)

```
variable_del('webserver_auth_optional_external_auth');
```

Listing F.16: Deletion of the new configuration variable during the module's uninstallation

```
  /* Re-enable login form as it may have been disabled */
  $query = db_update('block')
    ->fields(array(
      'status' => 1,
      'region' => 'sidebar_first',
    ))
    ->condition('module', 'user', '=')
    ->condition('delta', 'login', '=')
    ->execute();
}
```

Listing F.17: Re-enabling the login form during the module's uninstallation

F.4 The fourth patch

```
  // We found the user so just log them in.
  $form_state['uid'] = (int)$uid;
  user_login_submit(array(), $form_state);
  // Save authname in session to compare in
   webserver_auth_route()
  $_SESSION['webserver_authname'] = $preparedauthname;

  _webserver_auth_update_remote_user_info($uid);
```

```
  return TRUE;
}
```

Listing F.18: Calling the `_webserver_auth_update_remote_user_info()` function right after login

```
/**
 * Retrieves the information about the remote user (e-
   mail and/or, using the
 * _webserver_auth_update_remote_user_roles() function,
   group membership
 * information) and if it has changed from the state in
   the Drupal's database,
 * it updates it in the database.
 *
 * @param $uid
 *   UID of the remote user that has just been
   authenticated and for which the
 *   information is being retrieved.
 */
function _webserver_auth_update_remote_user_info($uid) {
  if (variable_get('webserver_auth_email_from_idp', FALSE
   )) {
    if (isset($_SERVER['REMOTE_USER_EMAIL'])) {
      // If the user's email in REMOTE_USER_EMAIL has
   changed, update it in the database
      $query = db_select('users', 'u')
        ->groupBy('u.uid')
        ->fields('u', array('mail'))
        ->condition('u.uid', $uid, '=');
      $result = $query->execute();
      $mail = $result->fetchField();

      if ($_SERVER['REMOTE_USER_EMAIL'] != $mail) {
        $query = db_update('users')
          ->fields(array(
            'mail' => $_SERVER['REMOTE_USER_EMAIL'],
          ))
          ->condition('uid', $uid, '=')
          ->execute();
      }
    }
  }

  /* If mapping of external groups to roles is set,
   retrieve group membership
```

```
    information from REMOTE_USER_GROUP_* and update it
   in the database */
  if (variable_get('webserver_auth_ext_groups_roles_map',
    '') != '') {
    _webserver_auth_update_remote_user_roles($uid);
  }
}
```

Listing F.19: The new `_webserver_auth_update_remote_user_info()` function

```
/**
 * Retrieves and parses the remote user's group
   membership information, checks
 * if some roles have been newly assigned/unasigned to
   the user using the
 * webserver_auth_ext_groups_roles_map mapping, and if so
   , assigns/unassigns
 * the corresponding roles to the user in the database.
 *
 * @param $uid
 *   UID of the remote user that has just been
   authenticated and for which the
 *   information is being retrieved.
 */
function _webserver_auth_update_remote_user_roles($uid) {
  // divide the string from
   webserver_auth_ext_groups_roles_map to pairs
  $pairs = explode(';', variable_get('
   webserver_auth_ext_groups_roles_map', ''));
  // remove potential duplicate entries
  $pairs = array_unique($pairs);
  $map_ext_groups = array();
  $map_roles = array();
  // process pairs into an associative array
  foreach ($pairs as $pair) {
    $components = explode(':', $pair);
    $map_ext_groups[] = $components[0];
    $map_roles[] = $components[1];
  }

  // load the roles that are currently assigned to the
   user in the database
  $query = db_select('users_roles', 'ur')
    ->condition('ur.uid', $uid, '=')
    ->fields('r', array('name'));
```

```
$query->join('role', 'r', 'ur.rid = r.rid');
$result = $query->execute();
$roles_db = $result->fetchCol('name');

// make a list of the roles externally assigned to the
 user
$num_groups = isset($_SERVER['REMOTE_USER_GROUP_N']) ?
 $_SERVER['REMOTE_USER_GROUP_N'] : 0;
$roles_remote = array();
// for every external group...
for ($i = 1; $i <= $num_groups; $i++) {
  // search for matching externals groups in the
 mapping and retrieve roles that they are mapped to
  $matches_ext_groups = array_intersect($map_ext_groups
 , array($_SERVER["REMOTE_USER_GROUP_$i"]));
  $matches_roles = array_intersect_key($map_roles,
 $matches_ext_groups);

  // for each role mapped by the matching entries...
  foreach ($matches_roles as $match_role) {
    // verify whether the role exists in Drupal
    $result = db_select('role', 'r')
      ->fields('r', array('name'))
      ->condition('r.name', $match_role, '=')
      ->execute();
    if ($result->fetchField() != $match_role) {
      continue;
    }

    // add the role to an array, if it is not already
 there
    if (!in_array($match_role, $roles_remote)) {
      $roles_remote[] = $match_role;
    }
  }
}

/* if some roles are in the database, but are not
 assigned to the user anymore,
   delete the records assigning them to the user in the
  database */
$roles_to_delete = array_diff($roles_db, $roles_remote)
 ;
foreach ($roles_to_delete as $role) {
  $result = db_select('role', 'r')
```

```
      ->fields('r', array('rid'))
      ->condition('r.name', $role, '=')
      ->execute();
    $role_rid = $result->fetchField();
    $query = db_delete('users_roles')
      ->condition('rid', $role_rid, '=')
      ->execute();
  }

  /* if some roles assined to the user, but there is not
   a corresponding record
     in the database, add that record */
  $roles_to_add = array_diff($roles_remote, $roles_db);
  foreach ($roles_to_add as $role) {
    $result = db_select('role', 'r')
      ->fields('r', array('rid'))
      ->condition('r.name', $role, '=')
      ->execute();
    $role_rid = $result->fetchField();
    $query = db_insert('users_roles')
      ->fields(array(
        'uid' => $uid,
        'rid' => $role_rid,
      ))
      ->execute();
  }
}
```

Listing F.20: The new `_webserver_auth_update_remote_user_roles()` function

```
/* If e-mail is retrieved and updated from an external
   identity provider,
   we don't want to let users modify it manually */
if (variable_get('webserver_auth_email_from_idp', FALSE))
    {
  $form['account']['mail']['#disabled'] = TRUE;
  $form['account']['mail']['#description'] = t('This
   value has been set by default from the external
   identity provider and cannot be changed.');
}
```

Listing F.21: Addition to the `webserver_auth_form_user_profile_form_alter` function

```
'webserver_auth_email_from_idp' => array(
```

```
  '#type' => 'checkbox',
  '#title' => t('Retrieve users\' email from
   REMOTE_USER_EMAIL variable'),
  '#default_value' => variable_get('
   webserver_auth_email_from_idp', FALSE),
  '#description' => t("Retrieve users' email from
   REMOTE_USER_EMAIL variables at account creation and
   modification and update it at every login. Disable the
    ability to set it manually"
. " in the account modification form."),
),
'webserver_auth_ext_groups_roles_map' => array(
  '#type' => 'textarea',
  '#title' => t('External groups to site\'s roles mapping
   '),
  '#default_value' => variable_get('
   webserver_auth_ext_groups_roles_map', ''),
  '#description' => t("Using this mapping, groups
   retrieved from REMOTE_USER_GROUP_* are mapped to site'
   s roles. Pairs in the format external_group:role will
   be separated by semicolons. The group"
. " membership information is then updated from current
   REMOTE_USER_GROUP_* values at every login of any user.
    If this variable is empty, it means that retrieving
   of external groups is disabled"
. " and users' groups will be managed locally. CAUTION:
   If you set this mapping, the user will be unassigned
   from all roles that will not be assigned to the user
   by REMOTE_USER_GROUP_* variables"
. " and defined by this mapping."),
),
```

Listing F.22: The new items in the settings form (`_webserver_auth_settings()`)

```
variable_del('webserver_auth_email_from_idp');
variable_del('webserver_auth_ext_groups_roles_map');
```

Listing F.23: Deletion of the new configuration variables during the module's uninstallation

G Installation and configuration of prerequisites

Besides the additions and modifications of the module code as described in the previous chapter, multiple things must be installed and configured properly so that the Webserver authentication module can provide the desired functionality.

First of all, there must be a Drupal site under the organization's control. Besides, it is desirable to have an IdP that is under the organization's control as well. Especially if the organization sets up the IdP on its own, network, hostnames, firewall and DNS (Domain Name System) configuration on all concerned machines needs to be checked and adjusted as needed. That depends on the setup as well as the IdP used. This chapter assumes that FreeIPA IdP is used and all concerned machines run Fedora or possibly other Unix-like operating system. It largely draws from the example setup described in the article on *freeipa.org* [63]; configuration examples as well as other tasks described in this chapter are adjusted to at least partially match those made as a part of the test Drupal page setup used to test the extension of the Webserver authentication module described in *Chapter 5* and demonstrate the functionality of the module before (described in *Section 4.2*) and after the extension (*Section 6.1*). All features and options of the relevant Apache modules are described on their home sites, as cited in *Section 2.2*. To perform some of the tasks below, the user must obtain a TGT from the FreeIPA server for `admin` when configuring and for some ordinary user, let it be `exusr` (Example User), when testing, on a particular machine. This can be done using the command `kinit` *`username`*. Later, before authenticating to a Drupal site, a TGT for a given FreeIPA user mirroring the corresponding Drupal user can be obtained the same way.

G.1 Drupal installation and basic configuration

If the organization does not have a Drupal site under its control yet, it can set up one. The installation and basic configuration of Drupal on Fedora is described in the article on `fedoraproject.org` [105].

It is important to know that pages on the Drupal site are not stored as HTML files, but as data in multiple tables in the database. These data are loaded and processed by handlers upon request for a given location. The main handler is in the `index.php` file, which receives request for location on a Drupal site in the 'q' parameter of the query string. As 'index.php' can be omitted when typing an URL, URLs can look like `example.test/?q=admin/config`, for example. Such URLs may hamper search engines in indexing all the pages of the Drupal site, are less legible and they complicate the configuration of Apache modules (which is described in the following sections). However, the Clean URLs feature can transform these URLs into more convenient and typical form, like `example.test/admin/config`. Therefore if this feature is disabled, it should be enabled. For this to be possible, Apache, especially its settings regarding Drupal, must be properly configured. The instructions for this are in the article on *drupal.org* [106]. A working configuration may vary from setup to setup, and even one particular configuration may work at some time on a particular setup, but not at another time, with software updates, but also other factors possibly playing a role. The configurations of Apache modules described in the following sections assume that Clean URLs are enabled and working properly.

The Drupal site in the test setup runs at `wikiapp.example.test/drupal7/`, therefore the configurations of the Apache modules described in the following sections reflect this. Sites powered by Drupal – especially the production ones – most often run at the root of the whole site, for example `europa.eu`.

G.2 Kerberos – mod_auth_gssapi

For Kerberos authentication to work, these things must be set up:

1. On the FreeIPA server, a TGT for `admin` must be obtained and the access to the server hosting the web application through HTTP must be added:

   ```
   kinit admin
   ```

   ```
   ipa service-add HTTP/wikiapp.example.test
   ```

   Afterwards, add `admin` as the user permitted to retrieve the

keytab for the new service:

```
ipa service-allow-retrieve-keytab HTTP/wikiapp.example.test --users=admin
```

As `admin` cannot have all the attributes filled in, what is undesirable later, create `exusr`:

```
ipa user-add exusr --first=Example --last=User --displayname="Example User" --email=exusr@mailprov.com
--random --phone=123456789
```

2. On the server hosting the web application consuming the authentication results, a TGT for `admin` must be obtained:

```
kinit admin
```

The keytab must be retrieved from the FreIPA server:

```
ipa-getkeytab -s $( awk '/^server/ { print $3 }' /etc/ipa/default.conf ) -k /etc/http.keytab -p HTTP/wikiapp.example.test
```

It is necessary to set Apache as the owner of the keytab file and grant Apache exclusive read and write permissions to it.

Much of the remainder of this section is also based on the information from another article on *adelton.com* [53].

The mod_auth_gssapi module must be installed:

```
sudo dnf install mod_auth_gssapi -y
```

Then, it must be configured in the `/etc/httpd/conf.d/wikiapp_kerb.conf` file as follows, if no PAM access control is desired (*Listing G.1*):

```
LoadModule auth_gssapi_module modules/
   mod_auth_gssapi.so

<LocationMatch ^/drupal7/>
<If "%{QUERY_STRING} !~ /noext=1/">
<Limit GET>
  AuthType GSSAPI
  AuthName "Kerberos Login"
  GssapiCredStore keytab:/etc/http.keytab
  require valid-user
```

```
    ErrorDocument 401 '<html><meta http-equiv="refresh
     " content="0; URL=/drupal7/?noext=1"><body>
     Kerberos authentication did not pass.</body></
     html>'
  </Limit>
  </If>
  </LocationMatch>
```

Listing G.1: mod_auth_gssapi configured for the whole Drupal site

Finally, Apache must be restarted for the changes to take effect:

```
sudo systemctl restart httpd.service
```

The GSSAPI authentication is thus configured to run on `/drupal7/`. The `ErrorDocument` client-side redirect to `/drupal7/?noext=1` facilitates a fallback to the login form in the case that the user has no valid ticket. This configuration may be a preferable choice when mod_intercept_form_submit is also used, as it guarantees that mod_intercept_form_submit can be configured to cover the whole Drupal site (see *Section G.4 for details*), and it is possibly necessary, if mod_lookup_identity is used, due to complexities of processing a page in Drupal (see *Section G.5 for details*).

In general (not particularly in the case of Drupal), fallback to a login form can be a part either of the application code, application/framework mapping, or of the Apache configuration. If the browser cannot obtain a valid ticket, it will just show the content of the location configured by the `ErrorDocument` directive. The directive needs to use the 401 HTTP response status and can either:

- directly redirect the user to `/application/login&noext=1`, or
- point to a static HTML file redirecting to `/application/login&noext=1`, which can be more comfortably filled with more complex content than just a short error message (like instructions for enabling Kerberos or obtaining a TGT) and use a longer timeout for the refresh, or

- directly contain a HTML document that is usually very simple – it is sufficient when it contains only an error message and redirects to `/application/login&noext=1`.

To avoid GSSAPI authentication that would fail again, the redirect must point to the same URL as the URL on which GSSAPI authentication works, but with an additional parameter, for example `noext=1`, as it is above, or to a different URL, like `/application/login2`. However, in the former case, the GSSAPI authentication must be configured to take place only when the URL does not contain the aforementioned additional parameter – this is facilitated by the `<If ...>` clause in the aforementioned configuration. On the other hand, in the latter case, the login form must be accessible also through the different URL used – that requires changing the web application.

Another thing that must be addressed is that the login form is submitted to the URL, on which GSSAPI authentication runs now. In any case, this can be solved by modifying the web application so that the `action` attribute of the `form` element or whatever other property of the mechanism used for displaying of the login form will point and thus drive its submission to the URL not covered by GSSAPI authentication corresponding to the URL configured by the `ErrorDocument` directive.

As this probably requires modification of the web application itself, which might not be feasible, it is preferable to solve this by a different approach, relying only on the Apache configuration. However, such approach requires the URL for the fallback login form to be the same as the one for GSSAPI authentication, but with the additional parameter, as described above. In this case, the GSSAPI authentication is configured to perform only when accessing the configured URL using GET HTTP method by the `<Limit GET>` clause. As the first access to the login page usually uses GET HTTP method, while the login form is typically submitted using POST HTTP method, this will produce the desired functionality. This is therefore the arrangement on which the extension of the module described in *Chapter 5* relies.

If everything is configured properly, after a short message

Kerberos authentication did not pass.

the login form will be displayed from `/application/login&noext=1` and the users can log in as usual, with their login and password.

The drawbacks of the configuration described above are that:

- it necessitates performing GSSAPI authentication, and therefore Negotiate, upon every request to the Drupal site, whatever page on it is requested, what results in a potentially considerable waste of system resources, network bandwidth and time,
- it is impossible to log out in a way other than closing the browser or destroying the TGT of the user by the `kinit destroy -A` command and subsequently logging out,
- consequently, if some user is authenticated using mod_auth_gssapi and other user wants to authenticate without destroying the TGT of the former user, he can use the login form – either for local authentication or for mod_intercept_form_submit authentication – only after manually appending `?noext=1` to the URL in the address bar of the browser.

On the other hand, this is the only configuration with which mod_auth_gssapi is practically usable with the Webserver authentication module in the state before the extension. However, applying the first patch, described in *Section 5.1*, opens new possibilities described below.

The aforementioned problems can be softened by configuring GSSAPI authentication to run only on the home page – this is done by one-character alteration to the configuration on *Listing G.1*:

```
<LocationMatch ^/drupal7/$>
```

This will limit (repeated) GSSAPI authentication to the home page and enable logging out, although only with `webserver_auth_logout_url` set, as the user cannot avoid re-authentication on the home page otherwise. However, this is for a trade-off in the case that GSSAPI authentication is either mandatory, or the

authentication method of choice on the part of a user. To log in this way when the user comes to some other location on the Drupal site than the home page – probably most often some article or page linked from some other website, the user must go to the home page to authenticate, and then, if he authenticated to access some features or content on the previous location, return or navigate back to it.

It is also possible to restrict GSSAPI authentication to a special login page only (*Listing G.2*):

```
<LocationMatch ^/drupal7/user/loginGSSAPI$>
  AuthType GSSAPI
  AuthName "Kerberos Login"
  GssapiCredStore keytab:/etc/http.keytab
  require valid-user
  ErrorDocument 401 '<html><meta http-equiv="refresh
   " content="0; URL=/drupal7/"><body>Kerberos
   authentication did not pass.</body></html>'
</LocationMatch>
```

Listing G.2: mod_auth_gssapi configured only for a login page

This configuration will minimise the load caused by GSSAPI authentication and allow seamless choice of the authentication method, login as well as logout when visiting the Drupal site at any location. On the other hand, this possibly requires mod_intercept_form_submit to be configured only for a specific location, most suitably the standard login form at `/drupal7/user/login` and it completely bars mod_lookup_identity from working properly, as mentioned above and further explained in *Section G.4* and *Section G.5*, respectively.

G.3 HBAC for Kerberos – mod_authnz_pam

If PAM access control is used, the following steps need to be undertaken (most information regarding this are taken from the separate article on *freeipa.org* [134]):

1. On the FreeIPA server, HBAC rules need to be configured.

   To start actively using PAM authentication, the `allow_all` HBAC

rule must be disabled, but it cannot be done straight away, as such step would lock out users out of their systems. It is needed to preserve the existing behaviour for existing hosts and users.

Therefore create a new host group for existing hosts – for example, `allow_all_hosts`:

```
ipa hostgroup-add --desc 'Host group which will have
allow_all_users_services HBAC enabled.' allow_all_
hosts
```

Make all existing machines members of the `allow_all_hosts` group:

```
ipa host-find --raw --pkey-only --sizelimit=0 | awk
'$1 == "fqdn:"  print "--hosts=" $2 ' | xargs -n100
ipa hostgroup-add-member allow_all_hosts
```

Make the `allow_all_hosts` the target host group for the new HBAC rule, for example `allow_all_users_services`, which will grant access to everyone on these machines:

```
ipa hbacrule-add allow_all_users_services --usercat=
all --servicecat=all --desc='Allow access to hosts in
group allow_all_hosts to anybody from anywhere.'
```

```
ipa hbacrule-add-host allow_all_users_services
--hostgroups=allow_all_hosts
```

Then the `allow_all` rule can be disabled:

```
ipa hbacrule-disable allow_all
```

From now on, for access to work like it used to do before, any new host needs to be added to the `allow_all_hosts` host group using the `ipa hostgroup-add-member` command or a similar mechanism. Alternatively, it is also possible to use `automember-*` and `automember-default-*` commands to set the group membership automatically.

Once the individual systems are enumerated in the `allow_all_hosts` host group, new rules can be defined with possibly more targeted services to align some of those hosts to.

Now, for example, the organization runs the application `wikiapp`

and it wants to have the PAM service `wikiapp` with HBAC handled by the FreeIPA server for authentication and authorization. The undermentioned actions will achieve this.

Define the service:

```
ipa hbacsvc-add wikiapp
```

Then define the rule for this service:

```
ipa hbacrule-add allow_wikiapp
```

Add the service to the rule:

```
ipa hbacrule-add-service allow_wikiapp --hbacsvcs=wiki-
app
```

At any point the status of the rule can be checked:

`ipa hbacrule-find allow_wikiapp` Add the user `exusr` and the host `wikiapp.example.test` to the rule:

```
ipa hbacrule-add-user allow_wikiapp --user=exusr
```

```
ipa hbacrule-add-host allow_wikiapp --hosts=wikiapp.
example.test
```

The access to the service can be tested:

```
ipa hbactest --user=exusr --host=wikiapp.example.test
--service=wikiapp
```

If the rule `allow_wikiapp` matches, it is set up properly. However, `allow_all_users_services` matches as well, therefore the host should be removed from the host group. But beware – this might cut away access to the machine through ssh, if ssh is configured to use FreeIPA HBAC:

```
ipa hostgroup-remove-member allow_all_hosts --hosts=
wikiapp.example.test
```

All other users that should be allowed access to the host `wikiapp.example.test` by mod_authnz_pam must be also added to the `allow_wikiapp` HBAC rule.

2. On the server hosting the application, the `/etc/pam.d/wikiapp` file must be created with the configuration specifying SSSD as

the mechanism for authentication and authorization:

```
auth    required   pam_sss.so
account required   pam_sss.so
```

The `wikiapp` HBAC service name needs to match the PAM service name but it is just a string, it does not need to match the hostname. So, `wiki` or `test` could be used instead.
Furthermore, mod_authnz_pam must be installed:

```
sudo dnf install mod_authnz_pam -y
```

The following changes to the aforementioned mod_auth_gssapi configuration need to be done:

(a) addition of the directive specifying the loading of the mod_authnz_pam module in addition to mod_auth_gssapi:

    `LoadModule authnz_pam_module modules/mod_authnz_pam.so`,

(b) replacing the directive

    `require valid-user`

    with the directive specifying that for granting access, the user not only has to be successfully authenticated, but also it has to be allowed access to the `wikiapp` PAM service by HBAC rule(s):

    `require pam-account wikiapp`.

Finally, enable Apache to use the PAM stack and restart it:

```
sudo setsebool -P allow_httpd_mod_auth_pam 1
sudo systemctl restart httpd.service
```

The PAM authentication and authorization should now work as expected. If the user with a valid Kerberos ticket is also granted access by a HBAC rule, the overall authentication also succeeds, if there is no rule granting him access, the overall authentication fails.

HBAC in FreeIPA can control access of users to services running on various hosts. HBAC rules can use user groups in FreeIPA. Sometimes, the administrator might not want to manage access centrally

and might prefer to locally set the list of groups that can access the web application. Eventually, SSSD will provide the way to set up access control on per-PAM service basis. For now, with mod_authnz_pam, all the PAM modules are at disposal, including pam_access. For example, adding line

```
account required pam_access.so accessfile=/etc/http-access.
conf
```

to `/etc/pam.d/wikiapp` will enable the access control using file `/etc/http-access.conf`. If the content of that file is

```
+ : (wiki-group-test) : ALL
- : ALL : ALL
```

only the users in the `wiki-group-test` group will be granted access. Both the local groups from `/etc/group` and the FreeIPA-managed groups are considered for this access control check.

G.4 External identities for the login form – mod_intercept_form_submit

If connecting the web application's login form to the FreeIPA server is desired, it is done on the server hosting the web application as described in this section.

Install the mod_intercept_form_submit Apache module:

```
sudo dnf install mod_intercept_form_submit -y
```

The module must be configured in the `/etc/httpd/conf.d/wikiapp_form_submit.conf` file as follows (*Listing G.3*):

```
LoadModule intercept_form_submit_module modules/
   mod_intercept_form_submit.so

<LocationMatch ^/drupal7/>
<If "%{QUERY_STRING} =~ /noext=1/">
  InterceptFormPAMService wikiapp
  InterceptFormLogin name
  InterceptFormPassword pass
</If>
</LocationMatch>
```

Listing G.3: mod_intercept_form_submit configuration for the whole Drupal site

Two important notes: firstly, mod_authnz_pam must also be loaded, the PAM service specified in the above configuration must be configured and SELinux boolean `allow_httpd_mod_auth_pam` must be enabled (as already described in the previous section) [62], because mod_intercept_form_submit internally calls mod_authnz_pam, as mentioned in the *Section 2.3*. Therefore it also observes the PAM access control. Secondly, on *Listing G.3*, mod_intercept_form_submit is configured to cover the whole Drupal site, because Drupal not only provides the login form at `user/login`, but it also displays the login form in the Login block on any page on the Drupal site, if it is accessed anonymously and does not require administrative or other special permissions (so, for example, accessing `admin/config` page redirects an anonymous user to the 'Access denied' page).

In general (not particularly in the case of Drupal), mod_intercept_form_submit must be configured either for the location of the fallback login form in the case of mod_auth_gssapi being used, or for the standard location of the login form if mod_auth_gssapi is not used. There may be – and when used with Drupal, there are – at least two more things to consider: how is optionality of GSSAPI authentication implemented, and whether mod_auth_gssapi is configured for the whole Drupal site, or only for a login page. As for the former matter, in regard to its implementation introduced by the third patch described in *Section 5.3*, when it is enabled, mod_intercept_form_submit must be configured for the standard location of the login form even if mod_auth_gssapi is used. And as for the latter matter, a situation, when mod_auth_gssapi is configured only for a login page, but mod_intercept_form_submit is configured for the whole Drupal site, must be avoided, as this causes mod_intercept_form_submit to interfere with all POST requests on the site – and this in turn blocks any changes to the site's configuration or content. Attempts to find a configuration of mod_intercept_form_submit that covers the whole Drupal site, but prevents these interferences, failed. Therefore in the case that mod_auth_gssapi is configured only for the GSSAPI login page in the test setup (`/drupal7/user/loginGSSAPI`), mod_intercept_form_submit is configured only for the standard login page (*Listing G.4*):

```
<LocationMatch ^/drupal7/user/login$>
  InterceptFormPAMService wikiapp
```

```
  InterceptFormLogin name
  InterceptFormPassword pass
</LocationMatch>
```

Listing G.4: mod_intercept_form_submit configuration only for the login page

This will prevent the aforementioned interferences, but will also limit the usage of mod_intercept_form_submit to `/drupal7/user/login` only.

Finally, Apache must be restarted:

```
sudo systemctl restart httpd.service
```

With `/etc/pam.d/wikiapp` in place and mod_intercept_form_submit, the `REMOTE_USER` will be set whenever the authentication through the PAM stack succeeds. If it fails, the web application will still have a chance to run local authentication.

G.5 Storing external users in internal databases – mod_lookup_identity

First of all, SSSD must be configured on the server hosting the web application.

```
sudo dnf install sssd-dbus -y
```

Enable and configure its ifp subsystem by the changes to the `/etc/sssd/sssd.conf` file as contained in this example patch (*Listing G.5*, auxiliary information omitted):

```
 ...
 chpass_provider = ipa
 ipa_server = _srv_, ipa.example.test
 dns_discovery_domain = example.test
+ldap_user_extra_attrs = mail, givenname, sn
+
 [sssd]
-services = nss, pam, ssh
+services = nss, pam, ssh, ifp
 config_file_version = 2

 domains = example.test
 ...
 [pac]
```

```
+[ifp]
+allowed_uids = apache, root
+user_attributes = +mail, +givenname, +sn
+
 ...
```

Listing G.5: SSSD and its ifp service configuration

Restart SSSD:

```
service sssd restart
```

Proper functionality can be tested by attempting to retrieve some information using dbus-send:

```
dbus-send -print-reply -system -dest=org.freedesktop.sssd.
infopipe /org/freedesktop/sssd/infopipe org.freedesktop.
sssd.infopipe.GetUserAttr string:exusr array:string:gecos,
mail
```

```
dbus-send -print-reply -system -dest=org.freedesktop.sssd.
infopipe /org/freedesktop/sssd/infopipe org.freedesktop.
sssd.infopipe.GetUserGroups string:exusr
```

Possibly, `setenforce 0` will need to set for the above part to work. For D-Bus calls from Apache, which will be done when testing and using the web application,

```
setenforce 1
sudo setsebool -P httpd_dbus_sssd on
```

should work provided there is recent enough selinux-policy package present. If retrieving information using D-Bus works, install mod_lookup_identity:

```
sudo dnf install mod_lookup_identity -y
```

Configure the module in the `/etc/httpd/conf.d/wikiapp_lookup.conf` file as follows (*Listing G.6*):

```
LoadModule lookup_identity_module modules/
   mod_lookup_identity.so

<LocationMatch ^/drupal7/>
  LookupUserAttr mail REMOTE_USER_EMAIL " "
  LookupUserAttr givenname REMOTE_USER_FIRSTNAME
```

```
  LookupUserAttr sn REMOTE_USER_LASTNAME
  LookupUserGroups REMOTE_USER_GROUPS ":"
  LookupUserGroupsIter REMOTE_USER_GROUP
</LocationMatch>
```

Listing G.6: mod_lookup_identity configuration

In general, mod_lookup_identity must be configured both for the location where GSSAPI authentication is configured to operate, and for the location of the login form (standard and/or fallback) so that mod_lookup_identity retrieves the additional information about the user regardless of the external authentication method used (not in the case of local authentication). However, when used with Drupal, the simplest way to achieve this is to configure it for the whole Drupal site, as is the case in *Listing G.6*. All attempts to configure it only for a limited subset of pages (mainly the login pages and the home page in various combinations) on the Drupal site caused it to cease operating properly. As mentioned in *Section G.2*, mod_lookup_identity also requires `REMOTE_USER` to be set at least on `/drupal7/`.

Finally, restart Apache:

```
sudo systemctl restart httpd.service
```

It does not matter if a user has authenticated using Kerberos (mod_auth_gssapi) or mod_intercept_form_submit – mod_ lookup_ identity uses `REMOTE_USER` to get additional information about the user and store it into `REMOTE_USER_*`. The web application will then be able to access and use the `REMOTE_USER_*` environment variables containing the information about the user.

In general, the group membership is retrieved into the `REMOTE_USER_GROUPS` environment variable and/or the `REMOTE_USER_GROUP_N`, `REMOTE_USER_GROUP_1`, `REMOTE_USER_GROUP_2`... environment variables (when configured as in *Listing G.6*, the group membership is stored to all of these environment variables). Then it can be expanded into the web application's roles. The administrator of the web application will need to set the initial mapping of groups to roles, but then, upon each login, roles can be updated from the central IdP. As a result, if an organization hires a new employee and his group membership is correctly set in the appropriate groups in the central FreeIPA server, system management tools across the organization can retrieve this group membership on the fly and not only make the new employee

able to log in to the system management tools, but assign him correct roles and permissions as well.

G.6 SAML – mod_auth_mellon

Provided that the version 0.11.0 or newer is used, as mentioned in *Section 2.5*, mod_auth_mellon can be configured as follows (*Listing G.7*) [8]:

```
MellonSetEnvNoPrefix "REMOTE_USER_GROUP" "groups"
MellonEnvVarsIndexStart 1
MellonEnvVarsSetCount On
MellonSetEnvNoPrefix "REMOTE_USER_LASTNAME" "surname"
MellonSetEnvNoPrefix "REMOTE_USER_FIRSTNAME" "givenname"
MellonSetEnvNoPrefix "REMOTE_USER_EMAIL" "email"
```

Listing G.7: mod_auth_mellon configuration

Note that with this configuration, group membership is stored only into the `REMOTE_USER_GROUP_*` environment variables, but exactly these are expected by the code introduced by the fourth patch described in *Section 5.4*.

mod_auth_mellon authenticates users against SAML IdP. For example, Ipsilon server and clients can be installed besides an existing FreeIPA infrastructure, with the Ipsilon server being connected to the FreeIPA server, retrieving users and all data about them from the FreeIPA server.

H Functionality of the extended Webserver authentication module in pictures

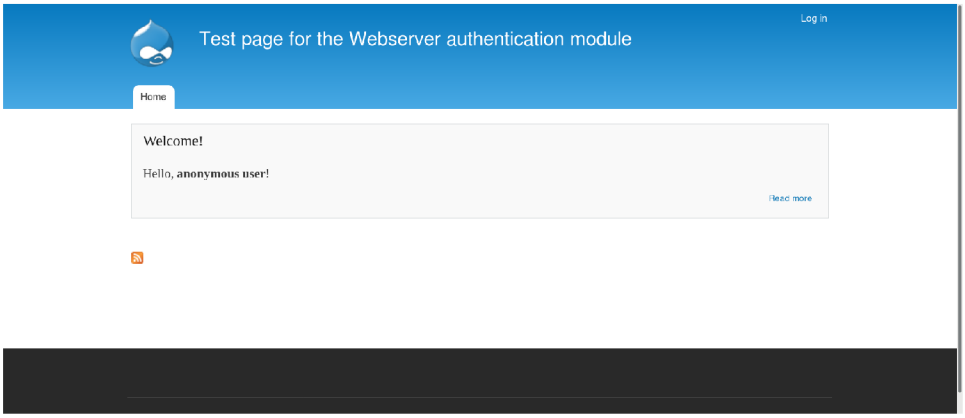

Figure H.1: The test page viewed by an anonymous user after enabling the module

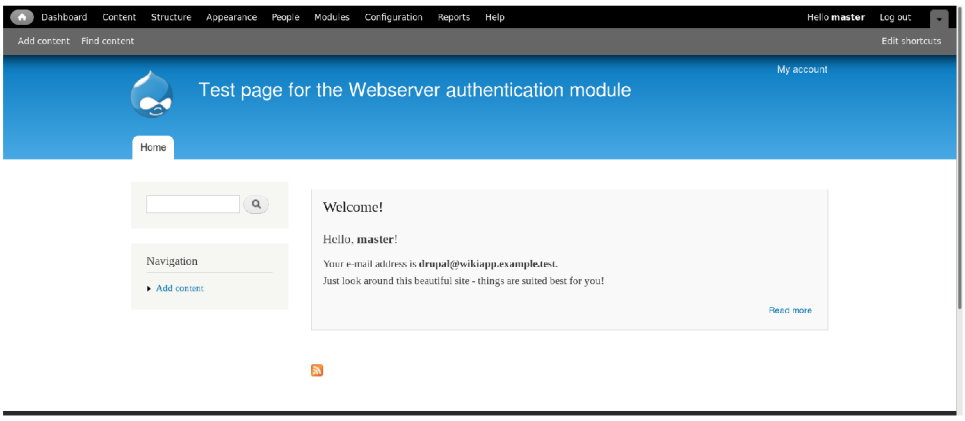

Figure H.2: The test page viewed by `master` after enabling the module with mod_auth_gssapi configured only for the GSSAPI login page

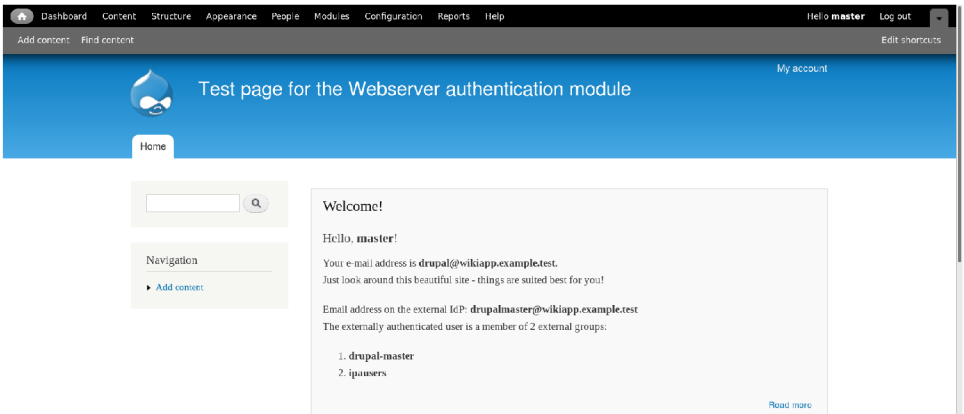

Figure H.3: The test page viewed by `master` *after enabling the module with mod_auth_gssapi configured for the whole Drupal site*

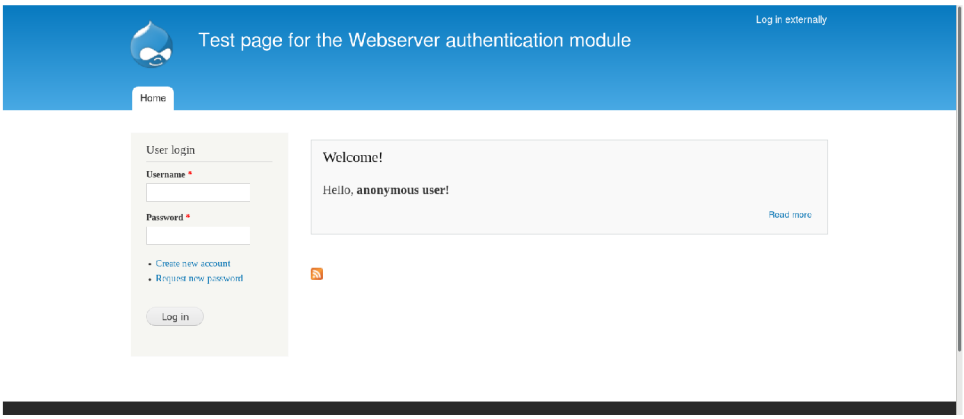

Figure H.4: The test page viewed by an anonymous user after enabling the optionality of GSSAPI authentication

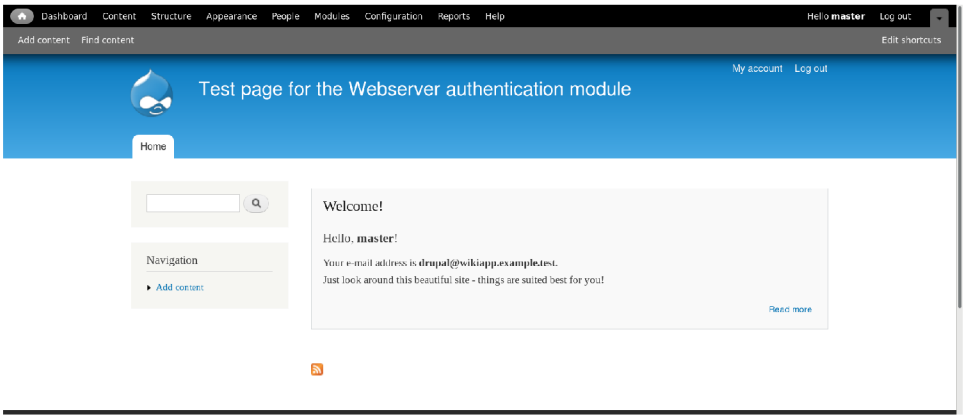

Figure H.5: The test page viewed by `master` *after enabling the optionality of GSS-API authentication with mod_auth_gssapi configured only for the GSSAPI login page*

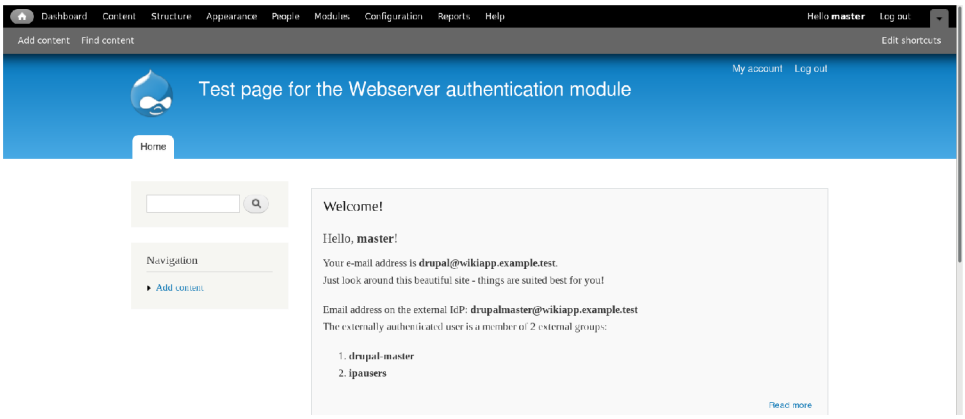

Figure H.6: The test page viewed by `master` *after enabling the optionality of GSS-API authentication with mod_auth_gssapi configured for the whole Drupal site*

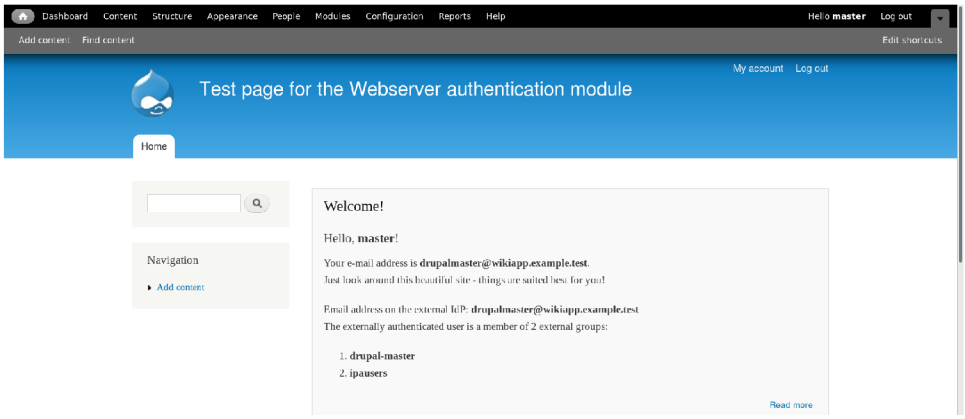

Figure H.7: The test page viewed by `master` *with an updated email*

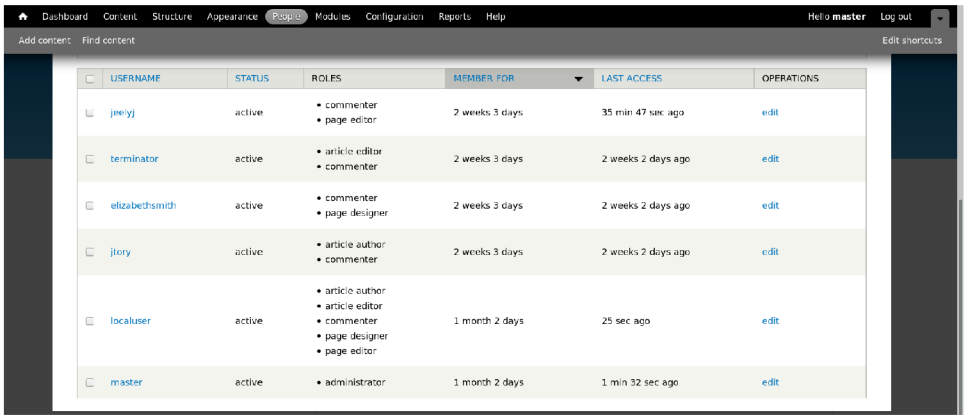

Figure H.8: Correctly mapped roles for all users

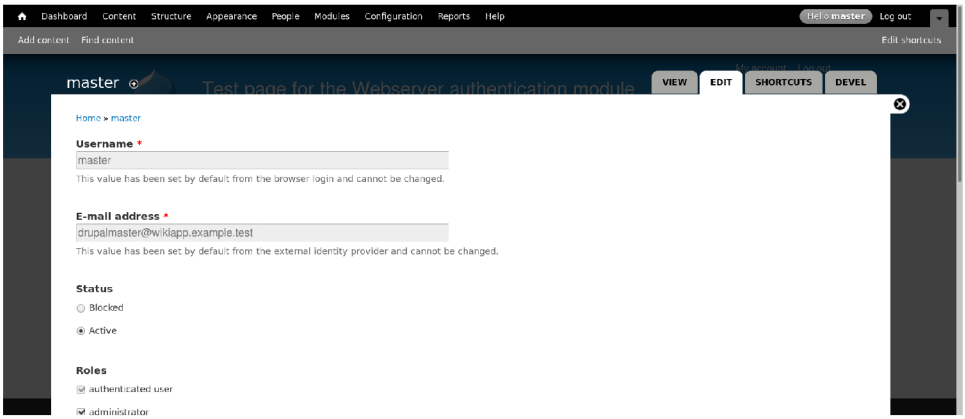

Figure H.9: The email field of the account modification form is disabled due to retrieval of the email from an external IdP

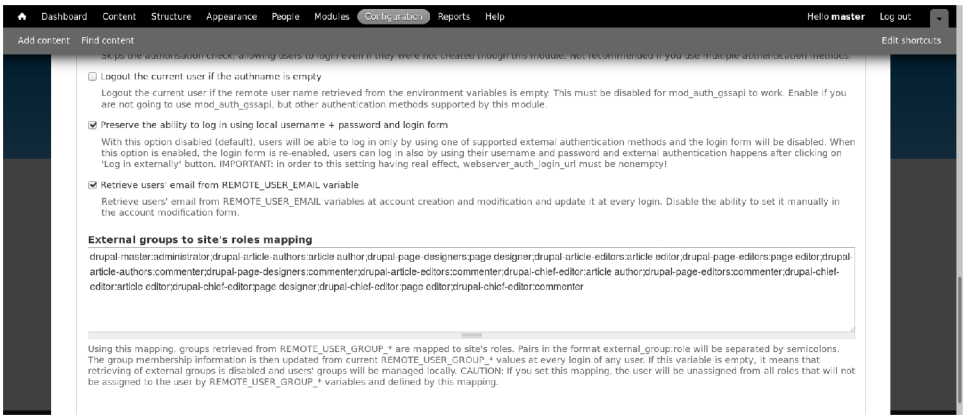

Figure H.10: The new settings on the module's configuration page

Electronic appendices

1. `webserver_auth.zip` – the original version of the module, for details about the files and the structure of the module see *Appendix D*
   (a) `LICENSE.txt`
   (b) `README.txt`
   (c) `webserver_auth.info`
   (d) `webserver_auth.install`
   (e) `webserver_auth.module`
2. `webserver_auth_extended.zip` – the extended version of the module
   (a) `LICENSE.txt` – unchanged, as the license remains the same
   (b) `README.txt` – unchanged yet, as the extension will be reviewed and potentially modified
   (c) `webserver_auth.info` – unchanged yet, as the extension is not yet available as a new release
   (d) `webserver_auth.install` – extended
   (e) `webserver_auth.module` – extended
3. `patches.zip` – the patches making up the extension that I published on *drupal.org*
   (a) `patch1.patch` – the first patch, for details see *Sections 5.1 and F.1*
   (b) `patch2.patch` – the second patch, for details see *Sections 5.2 and F.2*
   (c) `patch3.patch` – the third patch, for details see *Sections 5.3 and F.3*
   (d) `patch4.patch` – the fourth patch, for details see *Sections 5.4 and F.4*