Better WordPress Access & Security Controls with Access Policies

Build scalable WordPress solutions with better access management

Advanced Access Manager plugin

Is this book right for you? **2**

Why Access Policy? **3**

Story A. Administrator’s worth nightmare 4

Story B. $70,000 penalty for one wrong sentence 5

Story C. Troublesome page redirects 6

Story D. When a freelancer kills a project 6

What is Access Policy? **9**

Use-Case A. Admin access control based on geolocation 11

Use-Case B. Limit writer to a specific category(s) 12

Use-Case C. Make a private website 14

Use-Case D. Emulated default WordPress roles 16

Use-Case E. Post and page redirects 17

Use-Case F. Conditional user registration 18

How does Access Policy work? **20**

Multirole support 22

Multisite support 22

Summary **24**

Is this book right for you?

This material is primarily prepared for everybody who is interested in the WordPress website security and more precisely improving security pasture through better access management process.

From my observations supporting many businesses around the world, companies that are in the highly regulated areas like banking, insurance, healthcare as well as educational and government institutions implement access policies at the core of their WordPress access management procedures.

I believe that software engineers, chief technology officers, chief security officers, software project and product managers will benefit the most from reading this material.

Why Access Policy?

Managing access and security for the WordPress website sometimes is quite a challenging task. Even when everything is based on a simple concept of Roles & Capabilities, it is not always clear how to give the correct permissions to users so they can do only what is necessary.

Have you ever been in a situation where you had to recreate manually the same access and security settings over and over again on each WordPress website your company owns? Or, maybe, you are a software engineer who is tired of copy & pasting code that does one off tasks like capability removal, role creation, hiding metaboxes/widgets, etc.

You could also learn the hard way that giving too much access and permissions to a user may significantly increase the risk for a user error or for your website to be compromised.

> ***Note!*** *I have no intention to use any sort of scarcity tactics or create imaginary problems to justify our solutions. Below, I listed several real-life examples that were noted on multiple small and large WordPress projects. Most of these issues lead to substantial damages to business reputation, intellectual properties, and revenue.*

Access management is a very large topic and takes a big part of the security. That is why it is extremely important to pay extra attention to it and implement all the security measurements as early, in a project, as possible; and, honestly, that is much easier than you think.

From my observations, most of the WordPress projects do not have any access controls defined and those that do, 9 out of 10 times are very high level and clunky.

The weak and loosely defined access controls may stay unnoticed for quite a long period of time, however, when a project starts to scale, that becomes one of the biggest pain-points for organizations of all sizes. Eventually, it leads to hacked websites, data corruption, revenue losses and in some cases to business bankruptcy.

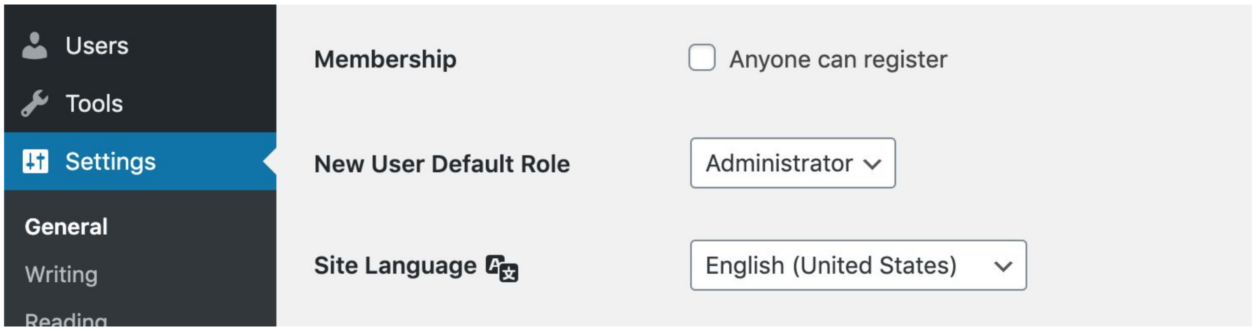
> *If you do not have solid access management in place, it is a matter of time until your website gets compromised.*

To show how easy things can get wrong, below I've listed just a few real-life stories of projects that were compromised because of weak access controls.

Story A. Administrator's worth nightmare

The project had a very ambitious plan - to produce high-quality content that leads to 2,000 - 3,000 new user registrations per month. When it started to scale, the number of employees that got *Administrator* rights grew to 30+ and the website was getting around 1,500 new users per month. Things were going extremely well until one of the administrators made a fatal mistake.

When you go to the "Settings->General" page, there is one option called "New Users Default Role".



Membership: ☐ Anyone can register

New User Default Role: Administrator

Site Language: English (United States)

Either on purpose or by mistake, this option was changed to the "Administrator" and all newly registered users, got full admin access to the **entire website**. Because of insufficient monitoring, the mistake was identified only a few days later, however, that was enough to have about a hundred new admin users on the site.

The damage to the website's reputation and data was irreversible.

Story B. $70,000 penalty for one wrong sentence

Highly regulated industries like financial, insurance or healthcare do not forgive mistakes. Companies that produce content in these areas are paid extremely well, however, they are also under lots of pressure. Mistakes like missing disclaimers, invalid offering details, published content prior to established release-date are very expensive.

When you do not have a well-defined access management strategy, it is really easy to grant unwanted permissions to random users. This happened to one company. One of the content writers had the ability to edit published content. The capability "`edit_published_posts`" was assigned to the writer directly and she accidentally changed one of the disclaimers on the published page.

The company ended up paying close to $70,000 in penalty fees.

Story C. Troublesome page redirects

I call it the *microcode problem*. It is when one or multiple engineers while working on a project, inject a small snippet of code/configuration that does a one-off requirement. Typically it is a code that hides specific metaboxes, widgets, restricts access to certain pages, API endpoints or performs HTTP redirects.

These types of code snippets are injected all over the place. From plugins and themes to `wp-config.php`, `.htaccess` and application load balancers. Unfortunately, this is a very typical mistake that I see in many projects.

One of the recent examples is when the engineer implemented a few lines of code that were redirecting only visitors from one page to a different page. The content editor, while working on the first page, did not experience any issues, because he was an authenticated user and redirection did not occur for him.

The issue got attention only a few weeks later because analytics showed very poor page performance. The company lost tens of thousands of dollars in revenue and more importantly opportunities to acquire customers.

Story D. When a freelancer kills a project

This is probably one of the most dramatic issues between website owners and freelancers. It is so easy these days to find a freelancer that appears to be very seasoned, however, he may have very poor work ethics. While, in general, we have to trust people, it is much safer to "*trust but verify*" and the best way to apply this principle is by drawing boundaries on what freelancers can/cannot do.

It is easy to give the full administrator access to a website without knowing that your freelancer has very limited experience with WordPress and as soon as some fatal mistake is made, he may “ghost on you” for good.

You’ll be surprised how often this type of situation happens. I get at least two emails per month where a website owner claims that her freelancer disappeared leaving a completely broken website. For none tech-savvy business owners this is the worth nightmare, especially when business is fully online.

Above are just a few real-life examples that you either already experienced or may experience in the near future if you do not have a well-defined and structured access management strategy.

> *It is a huge mistake to think that by “stitching” together a few plugins that do authentication (login), role management, access controls, and user activity monitoring will solve all the potential problems. At first, it may appear so, however, at scale, you will quickly discover that it is near impossible to get the full picture of who can do what and when.*

The majority of the plugins offer fairly intuitive UI with dozens or even hundreds of options that privileged users can toggle. However, when the number of privileged users grows, the risk of misconfiguration can be waiting just around the corner.

For a website owner, it becomes crucial to have reliable, consistent and scalable control over the authorization process and this is what **Access Policy** is designed for.

It is a security artifact that can be transferred between WordPress websites and applied to any role, user, visitors or to everybody at once. Because each policy is a separate custom post type record, all the changes to it are tracked and you can instantly know who, when and what changed. You can also quickly find out to whom any particular policy is attached.

What is Access Policy?

The *Access Policy* is a well-structured JSON document that contains one or more statements, params, and dependencies that collectively define access settings to a website's resources and actions that can be performed upon them.

> *You can find the detailed reference to all supported resources, actions and param on the Access Policy Reference page.*

In general, a policy may have 3 main sections. All are optional and used depending on the policy intent.

```
{
   "Dependency": {},
   "Statement": [],
   "Param": []
}
```

While a policy defines the access rules, there has to be a code that supports it. That is why the majority of policies come with the "`Dependency`" section that specifies a list of all dependencies that have to be installed. Typically you'll find dependencies like "WordPress" or "Advanced Access Manager" with the minimum required version.

```
{
   "Dependency": {
       "wordpress": ">=4.8",
       "advanced-access-manager": ">=6.4.0"
   }
}
```

The next important section is “`Statement`” that contains one or more policy statements with resources that are allowed or denied. Statements can be conditional and are applicable only if defined conditions are met.

```
{
   "Effect": "deny",
   "Resource": "PostType:post:posts",
   "Action": "Comment",
   "Condition": {
       "NotEquals": {
           "${IPSTACK.country_code}": "US"
       }
   }
}
```

The example policy above, is conditional and restricts the ability to write comments on all posts, if a user is not physically located in the US.

Last but not least, a policy may carry some parameters that support statements or can be used to override any WordPress core options. This section is mostly a wildcard where any custom parameter can be defined and either reused in statements or programmatically fetched from custom code.

```
{
   "Key": "option:users_can_register",
   "Value": 0,
   "Condition": {
       "In": {
           "${DATETIME.D}": [
               "Sat",
               "Sun"
           ]
       }
   }
}
```

The sample param above override WordPress core option that determines if new user registration is allowed and does not allow new registrations over the weekend.

Below there is the list of some of the most common use cases and how access policies allow fulfilling their requirements.

Use-Case A. Admin access control based on geolocation

Very powerful is managing access to a website or determining UI/UX based on the user's physical location. The example below demonstrates the policy that restricts access to the entire admin area of a WordPress website if a user is not coming from a very specific zip area. The policy uses external geolocation service ipstack.com to obtain geographical location based on a user’s IP address.

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=5.3.2",
       "advanced-access-manager": ">=6.3.3",
       "${CONST.AAM_IP_CHECK}": {
           "Name": "IP Check",
           "Version": ">=4.1.0",
           "URL": "https://aamplugin.com/pricing/ip-check"
       }
   },
   "Statement": [
       {
           "Effect": "deny",
           "Resource": "Capability:aam_access_dashboard",
           "Condition": {
               "NotEquals": {
                   "${IPSTACK.zip}": "28202"
               }
           }
       }
   ]
}
```

In this case, we deprive the user of the `aam_access_dashboard` capability that grants access to the admin area of a WordPress website if, based on the user's IP, she is not coming from an area that corresponds to `28202` zip code.

> ***Note!*** *The* `aam_access_dashboard` *capability is a custom capability that is introduced and supported only by the Advanced Access Manager plugin. For more details about this capability, refer to aam_access_dashboard documentation.*

Now, you can apply this policy, let's say to the *Administrator* role, knowing that all your admins are physically located in the city of Charlotte. Even if an admin account is compromised, a criminal has also to know which physical location he needs to tunnel through.

Use-Case B. Limit writer to a specific category(s)

When there is more than one content writer, it is typically a good practice to limit the number of directories the writer can produce content in. For example, one group of writers can post blogs only in the "Science" category, while others in the "Insurance". In most cases, it is critical to also limit what particular things the writer can do with content (e.g. publish, delete, edit or even see blog posts).

The policy below fulfils the following requirements:

- User can create and edit posts only in "Science" category;
- User is allowed only to save drafts or submit posts for review;
- All categories, except "Science", should be hidden;
- The "Science" is the default and preselected category for posts;
- User is not allowed to assign posts to any other category;

```
{
   "Dependency": {
       "wordpress": ">=5.3.2",
       "advanced-access-manager": ">=6.3.3",
       "${CONST.AAM_PLUS_PACKAGE}": {
           "Name": "Plus Package",
           "Version": ">=5.3.0",
           "URL": "https://aamplugin.com/pricing/plus-package"
       }
   },
   "Statement": [
       {
           "Effect": "deny",
           "Resource": "Taxonomy:category:terms",
           "Action": [
               "List",
               "Assign"
           ]
       },
       {
           "Effect": "allow",
           "Resource": "Term:category:science",
           "Action": [
               "List",
               "Assign"
           ]
       },
       {
           "Effect": "deny",
           "Resource": "PostType:post:posts",
           "Action": [
               "List",
               "Edit",
               "Delete",
               "Publish"
           ]
       },
       {
           "Effect": "allow",
           "Resource": "Term:category:science:posts",
           "Action": [
               "List",
               "Edit",
               "Delete"
```

```
            ]
        }
    ],
    "Param": [
        {
            "Key": "post:default:category",
            "Value": "science"
        }
    ]
}
```

In the policy above, we manage access to several resources. First, we hide and restrict the ability to assign content to all the terms that belong to the taxonomy “category” `Taxonomy:category:terms`. However, we override access only for one term - the “Science” category `Term:category:science`.

Similarly, we restrict the ability to see and manage all the posts `PostType:post:posts`. However, override this for all posts that belong to the “Science category” `Term:category:science:posts`.

Finally, the reserved parameter `post:default:category` defines the default category for all newly-created posts.

Use-Case C. Make a private website

Making a website private is one of the most popular policies in our official Access Policy Hub. It forces an unauthenticated user to login first before browsing any page on a website.

> ***FYI.*** *The policy also restricts physical access to media files if a website is configured to protect files. For more information about physical file protection, refer to the “How to manage access to the WordPress media library” article.*

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=5.3.2",
       "advanced-access-manager": ">=6.2.0",
       "${CONST.AAM_PLUS_PACKAGE}": {
           "Name": "Plus Package",
           "Version": ">=5.0.0",
           "URL": "https://aamplugin.com/pricing/plus-package"
       }
   },
   "Statement": [
       {
           "Effect": "deny",
           "Resource": "URI:*",
           "Metadata": {
               "Redirect": {
                   "Type": "login"
               }
           }
       },
       {
           "Effect": "allow",
           "Resource": "URI:/wp-login.php"
       }
   ]
}
```

The policy restricts access to all pages on the website `URI:*` and redirects a visitor to the standard WordPress login page. Upon successful authentication, the user is redirected back to the originally requested page.

> *If you are using a custom page for login that is not a default WordPress `wp-login.php`, then you can change `URI:/wp-login.php` to reflect your website setup. For example, if your custom login page is `/login`, then change resource to `URI:/login`*

Use-Case D. Emulated default WordPress roles

The default WordPress setup already comes with 5 predetermined roles: *Administrator*, *Editor*, *Author*, *Contributor,* and *Subscriber*. They are good enough for the majority of common use-cases. However, if you need to have better control over what capabilities are assigned to users, I recommend managing the entire website access with policies.

For example, the policy below emulates the default “Subscriber” role.

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=4.0.0",
       "advanced-access-manager": ">=6.0.0"
   },
   "Statement": [
       {
           "Effect": "allow",
           "Resource": [
               "Capability:read",
               "Capability:level_0",
               "Capability:subscriber"
           ]
       }
   ]
}
```

It grants exactly the same list of capabilities as if a user had the “Subscriber” role assigned to him on the edit profile page.

From the Access Policy Hub, you can install ready-to-use policies that emulate all the default WordPress roles:

- Administrator role policy
- Editor role policy
- Author role policy

- Contributor role policy
- Subscriber role policy

Use-Case E. Post and page redirects

It is common for evolving websites to move content around. This comes with the need to define redirects because content URLs typically change.

This policy may solve the problem of *microcodes* where all redirects are defined in one place.

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=5.3.2",
       "advanced-access-manager": ">=6.4.0"
   },
   "Statement": {
       "Effect": "deny",
       "Resource": "Post:page:sample-page",
       "Action": "Read",
       "Metadata": {
           "Redirect": {
               "Type": "url",
               "URL": "/new-sample-page",
               "Code": 410
           }
       }
   }
}
```

The policy above, redirects users from `sample-page` to a different URL `/new-sample-page` and notifies browser that the original page is gone with HTTP 410 code.

Use-Case F. Conditional user registration

Sometimes it is very useful to control the user registration process. Let's say you want to allow registration only for users that come from a certain geographical area or maybe narrow it to a specific time window.

In the sample policy below, we allow user registration only between January 1st and February 28th.

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=5.2.4",
       "advanced-access-manager": ">=6.3.3"
   },
   "Param": [
       {
           "Key": "option:users_can_register",
           "Value": 1,
           "Condition": {
               "Between": {
                   "${DATETIME.z}": [
                       0,
                       57
                   ]
               }
           }
       }
   ]
}
```

The "`DATETIME.z`" marker returns the current day of the year where January 1st is `0` and February 28th is `57`. Any day in between allows user registration.

For this policy to work properly, you need to disable the "Membership (anyone can register)" option on the "Settings->General" page.

Another popular way to do similar is based on a user's geolocation. For example, the "Allow registration only for European users" access policy will not allow new user registration if, based on a visitor's IP address, he/she is not located in a European country.

```
{
   "Version": "1.0.0",
   "Dependency": {
       "wordpress": ">=5.2.4",
       "advanced-access-manager": ">=6.3.3",
       "${CONST.AAM_IP_CHECK}": {
           "Name": "IP Check",
           "Version": ">=4.1.0",
           "URL": "https://aamplugin.com/pricing/ip-check"
       }
   },
   "Param": [
       {
           "Key": "option:users_can_register",
           "Value": 0,
           "Condition": {
               "NotEquals": {
                   "${IPSTACK.continent_name}": "Europe"
               }
           }
       }
   ]
}
```

How does Access Policy work?

In the previous chapter you've learned that access policies declare statements, params and code dependencies that collectively define access controls for a WordPress website.

Each policy is a separate custom post type `aam_policy` that is stored in the database table `wp_posts`. Technically speaking, there is not much difference between a normal WordPress post or page and a policy. However, while regular posts can be managed by users that have at least `edit_posts` capability, the access policies are allowed to be managed only by administrators that have access to the Advanced Access Manager plugin.

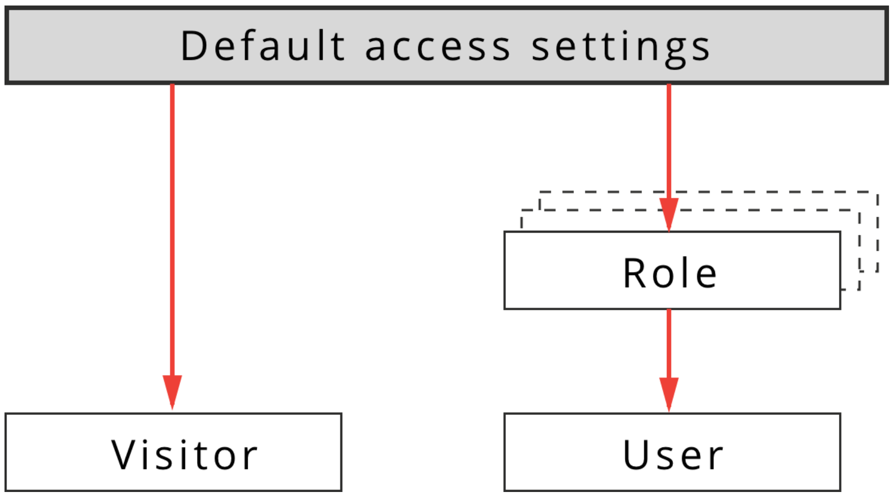
WordPress core has a very solid authorization layer that controls what users can manage with capabilities (e.g. `edit_posts`, `edit_pages`, `edit_order`) and AAM plugin leverages this to prevent unauthorized users from accessing or altering policies.

A completed policy, in order to take effect, has to be attached to some role, user, visitors or to the default access settings. Collectively we are going to call them *assignees*.

Any policy can be attached to assignee(s) in four different ways:

- Manually with AAM UI;
- Installed from the public or private Access Policy Hub;
- Through WP CLI with `wp aam policy-install` command;
- Programmatically with AAM API;

AAM comes with sophisticated access settings inheritance mechanism, where access settings are propagated down the hierarchical tree to the current user or visitor. For example, attached policy to the *Editor* role, is automatically propagated to all users that belong to this role. Visually this is shown on the diagram below.



FYI! To learn more about AAM inheritance mechanism, refer to the "AAM Access Settings Inheritance Mechanism" article.

All the attached and published policies are loaded as soon as WordPress core determines the current user. AAM plugin listens to the `set_current_user` core action and when it is triggered, it fetches attached policies and parses all the declared statements and params.

AAM also takes into consideration and handles more complicated WordPress websites with multirole or multisite setups.

Multirole support

WordPress core allows users to have two or more roles assigned. AAM, combines the list of all attached access policies from all roles. However, you might face a classic duality problem when a user has “Role A” with attached “Policy A” and “Role B” does not. In this case, should the user obey rules defined in “Policy A” or not?

AAM solves this problem by allowing a website administrator to explicitly define how access settings will be merged with ConfigPress (AAM internal configuration engine). By default, AAM honors attached policy, so the user will obey rules defined in “Policy A”, however, with following ConfigPress setup, the behavior can be reversed.

```
[aam]
core.settings.policy.merge.preference = "allow"
```

This configuration tells AAM core to enforce access policy only if all roles have it attached.

FYI! To learn more about multirole support, refer to the “WordPress access controls for users with multiple roles” article.

Multisite support

WordPress multisite setup can be challenging when it comes to synchronising access settings across the entire network of blogs.

To preserve integrity of access settings, all policies are stored in the main blog’s `wp_posts` database table.

This way, disregarding which blog/site user browses, all the attached access policies are loaded from the main site.

FYI! To learn more about multisite support, refer to the "AAM and WordPress Multisite Support" article.

Summary

Access management is a substantial part of any size project and organization. The majority of WordPress projects are built with little to no access controls in place. However, this is one of the most common reasons why websites get compromised.

Project owners and software engineers put so much emphasis on the "happy path" and what type of features need to be implemented that they leave no time to work on better security and access controls.

Access Policy is a unique solution that allows you to manage access to WordPress websites and keep track of all the access changes that happened. It is a well-structured and documented security artifact that can be distributed to a WordPress website and attached to any user, role, visitors or everybody at once.

It already supports a wide range of resources and actions as well as the ability to make access and security rules conditional. It offers developer API that allows to extend and add support for custom resources and conditions which turns it into a powerful framework. This way you can keep all the access and security configurations in one place and avoid the troublesome issue of *microcodes*.

> ***Access Policy** is a cohesive and centralized place where all access and security settings can be defined for everything that users can see or interact with.*