Argon

The Plug ‘n Play Backend

Hacking Guide

Introduction

This guide explains how **argon.js** and **argon.php** - the client-side and server-side scripts respectively - function, and how you may contribute. Before you begin modifying any of the aforementioned files, it is helpful to understand the inner workings of Argon.

This guide assumes that you have fully read the Argon documentation.

The Goal of Argon

Argon is meant to be **flexible, easy to understand, nimble, and reliable**. Make your contributions match those goals and we should be good to go.

Do's and Don'ts

So that we avoid any confusion down the road, here are some basic guidelines that you should keep in mind when modifying the source of Argon. Some of these may seem like common sense, but it's better safe than sorry.

• Spelling out variables is always preferable in the place of abbreviations (refrain from variables like 'ai', 'np', 'dr', etc.).
• Comments should be used when something might not be clear. There's no need for a detailed explanation of every part of your code, however, comments may be necessary for some sections.
• Do not minify or compress your contributions.
• Your code should be concise, while still maintaining a clear purpose.
• Put your code where it makes sense. If you see something that could be improved, don't be afraid to fix it!
• Test, test, and retest.

Ideas

Here's a list of our most pressing feature requests and/or bugs: http://www.link.com.

Argon.js

Below you'll find an explanation of every method, function, and significant variable of **argon.js** in order of definition.

Function/Method/Var	Explanation	Used By

Mg.validate.username()	Validates a username (as a string).	1. Argon.ajax() 2. Argon.get() 3. Argon.update() 4. Argon.set() 5. Argon.user.signin() 6. Argon.user.create()
Mg.validate.password()	Validates a password (as a string).	1. Argon.ajax() 2. Argon.user.password() 3. Argon.user.signin() 4. Argon.user.create()
Mg.isJSON()	Determines if a string is valid JSON.	1. Mg.convertData()
Mg.countObj()	Gets the length of an object.	1. Mg.requests.sender()
Mg.requests.removeRequest()	Removes a request to the server from **Mg.requests.queue[]**.	1. Mg.requests.send()
Mg.requests.add()	Adds a request to the server to **Mg.requests.queue[]**	1. Argon.ajax()
Mg.requests.send()	Sends a request to the server.	1. Mg.requests.sender()
Mg.requests.sender()	An interval that sends requests to the server in bulk to **Mg.requests.send()** every 300 milliseconds.	Initiated independently.
Argon.user.clientID	Used to identify the current instance of Argon.	
Argon.groupObject.group ””	The Group (subdirectory) Argon should operate under, instead of the default group.	
Argon.url	Url to argon.php.	

Argon.user.scanForToken()	Looks for a token on the current computer.	Initiated independently.
Argon.ajax()	Used to validate a request, before sending to requests queue.	1. Argon.get() 2. Argon.update() 3. Argon.set() 4. Argon.user.remove() 5. Argon.user.permissions() 6. Argon.user.password() 7. Argon.user.signin() 8. Argon.user.signout() 9. Argon.user.create()
Argon.get()	Used to download data under the current user from server.	Initiated foreignly.
Argon.update()	Used to push new, or update existing data under the current user on the server.	Initiated foreignly.
Argon.set()	Used to replace existing data under the current user on the server with new data.	Initiated foreignly.
Argon.user.remove()	Removes the current user from the server.	Initiated foreignly.
Argon.user.permissions()	Updates permissions of current user to permissions defined in an object assigned to the first parameter.	Initiated foreignly.
Argon.user.remember()	Saves the current user's authentication information to localStorage.	1. Argon.user.signin()
Argon.user.forget()	Removes saved authentication information from localStorage.	1. Argon.user.remove() 2. Argon.user.signout()

Argon.user.password()	Changes the password of the current user.	Initiated foreignly.
Argon.user.signin()	Signs into a user	Initiated foreignly.
Argon.user.signout()	Signs out out of the current user, and into the default user.	Initiated foreignly.
Argon.user.create()	Creates a new user on the server.	Initiated foreignly.
Argon.user.info()	Passes information on current user to the callback function.	Initiated foreignly.

Argon.php

Below you'll find an explanation of every method, function, and significant variable of **argon.php** in order of definition.

Function/Method/Var	**Explanation**
openFile()	Opens a file and returns its contents.
actionIsAllowed()	Checks if an action is allowed by looking at config.json.
genToken()	Generates a token used in place of a user's password when sending and receiving requests.
splitString()	Splits a string.
saveFile()	Saves data to a file.

Requests

Client Side

In **argon.js**, when a method is called that requires the server to complete a command, the method calls the method **.ajax()** which then validates the request.

Once validated, **.ajax()** sends the request to **Mg.requests.add()**. Multiple requests are combined and sent in bulk in a single request every 300 milliseconds to avoid flooding the server with individual requests.

Here is the basic format of a request:

```
{
    “data”: {},
    “action”: “”
}
```

As you can see, a request is simply a json object with two properties, ***“data”*** and ***“action”***.
In this example, the property ***“data”*** is json object, though it could also be a string, or another data type. The property ***“action”*** is a string that tells the server exactly what to do with the request. A request can also contain other properties, for instance, a .update() request for another user’s data would also contain the property ***“otherUser”***.

This single request would be combined with any other requests made in the last 300 milliseconds, and sent to the server in the following format:

```
[
    {
        "data": {
            "a8": true,
        },
        "action": "update"
    },
    {
        "data": "all",
        "action": "get"
    }
]
```

The above request contains multiple requests in an array. The first request in the array tells the server to update the property *“a8”* in the current user. The second request in the array tells the server to get *“all”* the properties of the current user.

Server Side

When a bulk request like the previous example is sent to **argon.php**, the request is authenticated, and then fed into a for loop that cycles through the requests within.

Each request's action is then sent through a series of if statements until the appropriate set of instructions is found. For instance, if a request had the following syntax:

```
{
    "data": "all",
    "action": "get"
}
```

Argon.php would go through a list of if statements until the following was found:

```
... else if ($action === "get" &&
actionIsAllowed($action,$groupName,$userName) === true) {
```

The instructions within the correct statement would be run, and in this case, the data contained within the current user would be returned.

Once a request has been completed, its results are pushed to the array **$main_output** and echoed back to **argon.js**. This array has much the same format as a request sent to the server.

What is a Token?

A token is used by Argon to authenticate each request to the server. A token can be thought of as a strong temporary password. Instead of sending the user's password - which will remain relatively static - with each request, a token is sent that is then **renewed with each signin/signout**. This keeps a potential attacker who is able to capture a user's token from being able to maintain access to the user.

If one or more of your questions was not answered in this document, don’t hesitate to email us!
contact@loadfive.com.