# Security Header Configurations

First, install the `flask-talisman` library using pip:

```
pip install flask-talisman
```

Now, let's create a Python file for our plugin, which we'll call `security_headers.py`:

```
from flask import Flask
from flask_talisman import Talisman

app = Flask(__name__)

# Define the security policy
csp = {
    'default-src': "'self'",
    'script-src': ["'self'", 'cdnjs.cloudflare.com'],
    'style-src': ["'self'", 'maxcdn.bootstrapcdn.com'],
    'img-src': ['data:', 'cdn.example.com'],
    'object-src': "'none'",
}

# Initialize the Talisman extension with the security policy
talisman = Talisman(
    app,
    content_security_policy=csp,
    content_security_policy_nonce_in=['script-src'],
    feature_policy={
        'geolocation': "'none'",
        'midi': "'none'",
        'sync-xhr': "'none'",
        'microphone': "'none'",
        'camera': "'none'",
        'usb': "'none'",
        'magnetometer': "'none'",
    },
    strict_transport_security=True,
    force_https_permanent=True,
)

# Define your routes and other application logic below
```

```
if __name__ == '__main__':
    app.run()
```

In this code:

1. We import the necessary modules and create a Flask app.
2. We define a Content Security Policy (CSP) that specifies which sources are allowed for different types of content, such as scripts, styles, and images.
3. We initialize the `flask-talisman` extension with our security policy and additional security configurations like Strict Transport Security (HSTS) and enforcing HTTPS.
4. You can customize the CSP and feature policy rules according to your application's requirements.

Now, you need to integrate this middleware into your Flask application. Define your routes and other application logic below the `app.run()` line.

The `flask-talisman` library will automatically add security headers to your responses based on the defined policy.