HTTP

> **CAUTION**
> HTTP flags are currently only available in certain Hosted CTFd tiers and CTFd Enterprise.

Overview

HTTP flags allow CTFd to delegate flag verification to an external server. Instead of storing the flag value directly in CTFd, the system sends the user's submission to a configured HTTP endpoint, which determines whether the flag is correct.

How It Works

HTTP flags use the same webhook system as Custom Webhooks. The endpoint must first be validated to prove ownership, and then CTFd will send flag verification requests to that endpoint.

Endpoint Validation

Before an HTTP flag endpoint can be used, it must be validated using the same process as webhook endpoints:

1. CTFd sends a `GET` request to your endpoint with a `token` query parameter
2. Your endpoint must respond with a JSON object containing the token hashed with your **Shared Secret** using **HMAC-SHA256**

For example, if CTFd sends:

```
GET http://example.com/?token=2b00042f6
```

Your application should respond with:

```
{ "response": "hmac_sha256(SHARED_SECRET, '2b00042f6')" }
```

> **INFO**
> The Shared Secret can be found in your CTFd instance's webhook settings. See the webhooks documentation for more details.

After the initial verification, CTFd will occasionally re-verify the endpoint to ensure it remains valid.

Flag Verification

Once the endpoint is validated, CTFd will send `POST` requests to verify flag submissions. The request payload is simple:

```
{
  "submission": "user_flag_submission"
}
```

Where `submission` contains the exact text that the user submitted as their flag.

Response Status Codes

Based on the status code returned by your endpoint the submission will be considered correct or incorrect:

- **202 Accepted** - The flag is **correct**
- **Any other status code** - The flag is **incorrect**

> **IMPORTANT**
> Note that a `200 OK` response is still considered **incorrect**. Only a `202` status code will mark the flag as correct. This is a deliberate design decision to mitigate accidental flag acceptance.

Example Implementation

Here's a simple example of an HTTP flag endpoint in Python using Flask:

```python
import hmac
import hashlib
from flask import Flask, request, jsonify

app = Flask(__name__)
SHARED_SECRET = "your_shared_secret_here"
CORRECT_FLAG = "flag{example_flag_value}"

@app.route("/submit", methods=["GET", "POST"])
def submit():
    if request.method == "GET":
        # Handle verification
        token = request.args.get("token")
        if not token:
            return jsonify({"error": "Missing token"}), 400

        # Create HMAC-SHA256 hash
        signature = hmac.new(
            SHARED_SECRET.encode(),
            token.encode(),
            hashlib.sha256
        ).hexdigest()

        return jsonify({"response": signature})

    elif request.method == "POST":
        # Handle flag verification
        data = request.get_json()
        submission = data.get("submission", "")

        # Check if the flag is correct
        if submission == CORRECT_FLAG:
            return "", 202  # Correct flag
        else:
            return "", 400  # Incorrect flag

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

Security Considerations

Since the endpoint receives user submissions, consider the following security hardening practices:

1. **Input validation** - Sanitize and validate all received data before processing. User submissions are untrusted input and should be handled carefully.
2. **Validate the webhook signature** - Verify that requests are actually coming from CTFd (and aren't forged) using the `CTFd-Webhook-Signature` header (see webhook security documentation)
3. **Use HTTPS** - Always use encrypted connections to protect flag submissions in transit
4. **Keep secrets secure** - Store your shared secret and flag values securely

Previous
« Programmable

Next
Custom Challenges »