Nimbusec Webhooks

Overview

Webhooks allow you to build or set up applications which subscribe to certain alerts in your Nimbusec account. When one of those alerts is triggered, we'll send a HTTP POST payload to the webhook's configured URL. Webhooks can be used to update an external issue tracker, feed into a SIEM, or even deploy a backup to your production server. You're only limited by your imagination.

Webhooks can be installed and configured exactly like your email and text notifiations. Once installed, the webhook will be triggered each time one or more subscribed alerts occur.

Setting up a webhook

Creating a webhook is a two-step process. You'll first need to set up how you want your webhook to behave through Nimbusec--what alerts it should listen to. After that, you'll set up your server to receive and manage the payload.

Setting up a webhook is similiar to setting up email or text notifications. Go to Settings > Notifications and click on "Add Webhook".

`Settings > Notifications > Add Webhook`

Webhooks require a few configuration options before you can make use of them. We'll go through each of these settings below.

Add Webhook Dialog

Webhook URL: This is the server endpoint that will receive the webhook payload. This URL must be accessible by the Nimbusec services, so make sure it is not a private URL like in the example image, but publicly accessible.

We recommend that you protect the URL via HTTPS as well.

Domains: Your webhook will only receive alerts for domains that you select. Select the domains you want to receive alert for in "Available Domains" and you the arrow buttons to move then to "Domains with notifications".

Notification levels: The webhook receives alerts for each new issue Nimbusec detecs (just like email and text notifications). Nimbusec issues are either rated as "*medium risk*" or "*severe risk*". You can select with the dropdowns in which alerts you are interested.

Payload

The second step to process webhooks is the server to handle the request. Nimbusec will send a `POST` request to the specified webhook URL for each new alert it detects.

The body of the request has always the content type `application/json` and the following format:

```
{
    "domain": {
        "id": "string",
        "name": "string",
        "url": "string",
        "responseIP": "string",
    },
    "issues": [
        {
            "id": "string",
            "event": "string",
            "category": "string",
            "severity": "int",
                        "regions": ["string"],
                        "viewports": ["string"],
            "details": { /* ... */ }
        },
        // ...
    ]
}
```

A sample application to receive webhooks can be found in our GitHub repository [nimbusec-oss/nimbusec-gelf](). This small example implementation receives any Nimbusec alerts and converts them to a GELF message that can be pushed to any GELF processing server like Graylog. It is used internally at nimbusec to feed our own alerts into our SIEM.

Securing your webhooks

Once your server is configured to receive payloads, it'll listen for any payload sent to the endpoint you configured. For security reasons, you probably want to limit requests to those coming from Nimbusec.

Nimbusec uses a hash signature to sign each webhook payload. This hash signature is passed along with each request in the headers as `X-Nimbusec-Signature`. Nimbusec uses PKCS 1 using RSAwithSHA512 to calculate the signature, which is sent encoded as BASE64 in the headers value.

You can use the following example codes to validate the Nimbusec signature:

Using Java:

```
public boolean verifySignature(byte[] signature, byte[] data) {
    try {
        PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(Base64.decodeBase64(PRIVATE_KEY));
        PublicKey key = KeyFactory.getInstance("RSA").generatePublic(spec);

        Signature sig = Signature.getInstance("SHA512WithRSA");
        sig.initVerify(key);
        sig.update(data);

        return sig.verify(signature);
    } catch(InvalidKeySpecException | SignatureException | NoSuchAlgorithmException |
InvalidKeyException ex) {
        return false;
    }
}
```

Using Go:

```
package example

import (
    "crypto"
    "crypto/rsa"
    "crypto/sha512"
)

func VerifySignature(signature []byte, data []byte) bool {
    // Example code to get the Nimbusec signature
    // header := request.Header.Get("X-Nimbusec-Signature")
    // signature, _ := base64.StdEncoding.DecodeString(header)
```

```
    // Example code to get the response body data
    // data, _ := ioutil.ReadAll(r.Body)

    hashed := sha512.Sum512(data)
    err := rsa.VerifyPKCS1v15(PublicKey, crypto.SHA512, hashed[:], signature)
    return err == nil
}
```

The repository [nimbusec-oss/nimbusec-gelf] showcases how to verify Nimbusec webhook signatures as well.

The public key used to verify all Nimbusec webhook signatures is:

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAlgbfxMniLiDMRYhRYY0f
fOLEACXSCWmX0/rSL+qib/3cbZAMOEknXUudich4ZuCHulZ9ApaPx/7u+x5jQSj4
aiZXJE+S+LecUqwbq1CSfByLPViyYu2xt2I0tqYdsQK6KmQs2Gl00UP/yxrHtcEz
NaZO8Z7bdL1AY3eW6oPjWeORK91FAEONbnCvXmPoGa/4+AUWr6FmMrjFiG8yM72K
eUvfzyWtZYNeFxJ+2UmqTco1oEdGmwJJYKgPAg4mRXOPBs1Il6W9+bwomUed/Rxd
GHuNPy4b9BOgSyFFoEHQJ2eL+W9IMpWegwV7VxXc37WlHQxoZ1886gO+u3hxvo++
+v0ami3JT1BZriTYdjSydktyUARQQzDaxAsYwUMTs/G++yiF3jt+J43pKvZ+ZSTP
+vXAKd+acbsUmH6WIxsu915BVPcnMgyeUWOK6NojiW4Z4BEuCWVKfqMKRU+LypFN
Hqpd3wxT26jnykJOm0a2xloXlmjS9x/LcHd6onN6I6wdPz8zSAU6lr0T2kWgPY+l
u0Ral9lpafe/Rq6GjPIvrlWNy2hjJhJ1FtzMCgySCs+XEqjFbM2GEOSK4M/NGY9+
zzkNgL4B0HpMHgRNeRfx0q+LuZtuHvNEDxmp/OvvfRqQGo5qqDhojm3rRi5qbsLa
k3siF46a7ml6ONtAD/Eib1kCAwEAAQ==
-----END PUBLIC KEY-----
```

Revision #1
Created 7 June 2023 09:44:09 by Lena
Updated 7 June 2023 09:44:32 by Lena