Webhook Security Guidelines

Webhooks provide a mechanism where by a server-side application can notify a client-side application when a new event (that the client-side application might be interested in) has occurred on the server. Web hooks are incredibly useful and a resource-light way to implement event reactions. However, webhooks can also be abused and the following considerations must be taken into account to secure Webhooks. It should also be noted that webhook security considerations apply to both publisher/server and subscriber/client.

At the core of webhooks fundamental security best practices are available however are not enabled by default. This is a gap in the "default secure" principle however with focus and configuration this gap can be addressed.

One item to note is webhook urls by default carry a high level of confidentiality. This comes from the lack of default secure principle mentioned above. With this webhook urls should be kept secret and while the security community has generally accepted that obscurity is not security it is a best practice to follow. If your webhooks are sharing sensitive information the following considerations should be reviewed and implemented.

The high level, the following requirements outline the security mechanisms in various ways. These high level requirements allow engineers and developers to properly implement security controls.

- **Confidentiality: A passive attacker must not be able to read Webhook Notifications, as they may contain confidential data.**
- **Integrity: An active attacker must not be able to alter Webhook Notifications, as this may result in unexpected behavior affecting your infrastructure.**
- **Authenticity: An active attacker must not be able to forge Webhook Notifications as this may result in your receive code taking unwanted action.**

Control 1 - Securing Webhook URLs

- **Secure URLs** - Keep URLs out of source code, keep urls out of logs. Do not print Webhook URLs in logs, Do not store write URLs or hard code it in source. Webhook URLs should be securely stored in Vault.

  Abuse Case:

  - Find a Webhook URL that is in code and try to post a simple mesg to it.

Control 2 - Transport Security

- **Enable SSL/TLS** - The first step you should take to secure web application is to ensure that you are using HTTPS for web application's end point. Data transmitted over the Internet should always be encrypted. Your endpoint URL should support HTTPS, and you should add that secure URL to your webhook settings. Avoid connecting to an HTTPS URL with a self-signed certificate and weak Ciphers. Server must be correctly configured to support HTTPS with a valid server certificate.

  Abuse Case:

  - Use an existing webhook and try to make a request to it using http://.
  - Alternative abuse case is to negotiate weak SSL/TLS when making a webhook request.
  - Alternative abuse case is to look at the certificate and see if it is self-signed. If it does not follow a strong chain of trust it should not be trusted for sensitive data or production traffic unless other mechanisms are in place.

Control 3 - Basic Authentication

- **Basic Auth** - HTTP Basic Authentication will require Username and Password along with Webhook URL to Authenticate against WebServer. In an effort to enforce the high security standards, refrain from using Basic Auth and go for request signatures. If it becomes inevitable, make sure Intruder lockout is enabled and Strong password policy is enforced.

  Abuse Case:

- Bruteforce
- Disclosure from information in logs

Control 4 - TLS Authentication

- **Mutual TLS** - Mutual TLS to authenticate the client, it builds upon normal TLS by adding client Authentication in addition to Server Authentication to ensure traffic is both secure and trusted in both directions. Mutual TLS guarantees that both Client and Webhook Server present a certificate during TLS handshake which mutually proves identity.

  Abuse Case:

  - TLS hardening (do not use TLS 1.0 or 1.1)
  - Certificate theft from system

Control 5 - Oath Token

- **OAuth Authorization** - When you build your own endpoint for Authentication, make sure OAuth is set for Authorization OAuth 2.0 requires an Authentication token, which can be issued by Authorization server in order to connect to Webhook endpoint. OAuth token should be included in the Authorization header. Make sure that OAuth token has lifetime/Expiry set.

  Abuse Case:

  - Session hijacking
  - Other Oath attacks

Control 6 - Crude Authorized Access

- **IP Whitelisting** - When designing network architecture, you may wish to have one set of servers and a load balancer in a DMZ that receive webhook requests from Server, and then proxy those requests to your private network. Have proper whitelist configured to allow inbound traffic to be whitelisted in Firewall with host name which list the egress IPs under the host's A record. This can be technically complex to manage, especially when the IP addresses of the webhook provider change. If you're processing webhooks using an app developed by the webhook provider, or the open source community, and

you don't own the code, make sure to review their docs or ask them questions to confirm they abide by security best practices.

Abuse Case:

- Using a targeted webhook with IP whitelisting go to a system that would not be in the whitelist and try to use it. If you can GET or POST the control is not in place.

Control 7 - Message Protection and Secrecy

- **Get your Web Authentication Key setup** - *By default, a webhook URL is open and may receive a payload from anybody who knows the URL. The Key is the only thing that protects the webhook and should be handled with the highest secrecy. For security reasons, we recommend you to accept requests only from trusted sources.* This is accomplished by signing your requests with a Hash-based Message Authentication Code (or HMAC). Once you defined a secret HMAC, Server will only accept signed requests for that webhook. To sign a request you need to generate hash digests of your request body using the sha512 algorithm and send it in HTTP header. If it is tampered or left blank then Server would respond with HTTP/1.1 400 Bad Request header and message like "The request is expected to be signed".

  It's recommended to include a signature header as part of POST request, this allows additional layer of security to ensure requests are originating from legitimate source. You can have your own solution for this or leverage official libraries from third party integration.

  Abuse Case:

  - Using a targeted webhook capture a message and replay it with a removed or replaced HMAC. If the service accepts the message it is vulnerable to this attack as it is not validating the HMAC.

Control 8 - Cross Site Request Forgery

- **CSRF protection -** CSRF Protection for webhooks is an important Security feature to prevent Cross Site Request Forgery. If you're using Rails, Django, or another web framework, your site might automatically check that every POST request contains a *CSRF token*. However, this security measure might also prevent your site from processing legitimate requests. If so, you might need to exempt the webhooks route from CSRF protection and handle them.

  Abuse Case:

  - Using a targeted webhook try sending a webhook POST request with a known list of malformed Cross-Site Request Forgery payloads.

Control 9 - Rotation and Retry

- **Retry Logic** - Have provision to reset a webhook's authentication key. To ensure that you do not lose any webhook requests between the time you reset your key and when you update your application to start using that new key, your webhook processor should reject requests with failed signatures with custom Status code.

  Abuse Case:

  - Using a targeted webhook, capture HTTP Request with Auth key. Generate new Authkey for the Webhook URL, try POST ing with expired Auth key, Invalid Authkey, payload and validate.

Control 10 - Pass Secrets in Request Header

- **Protection from repudiation attacks** - Most webhook providers allow you to pass a secret or a "token" in every webhook request. Example Github. The secret is typically passed as an HTTP header, as a field in the JSON payload or appended to the request URL. When you confirm the token you received is the one you expected, it helps validate the request was sent by the webhook publisher.

  Abuse Case:

- Using a targeted webhook capture a known good message. Tamper or remove the Token from Request Header and POST the message to target URL. The webhook should respond that the message was invalid. If it does not it is vulnerable to repudiation attacks.

Control 11 - Fail Safe and Secure

- **Event Handling** - Handling webhook events correctly is crucial to making sure your integration's business logic works as expected. Webhook endpoints might occasionally receive the same event more than once. We advise you to guard against duplicated event receipts by making your event processing idempotent. One way of doing this is logging the events you've processed, and then not processing already-logged events.

  Abuse Case:

  - Using a targeted webhook try to send incorrect, corrupted or flawed messages to the webhook targeting processing flaws in the webhook service or any services processing webhook messages. If the service does not handle the corrupt/flawed message it is vulnerable to this attack.

Control 12 - Replay protections

- **Replay Attacks** - Anyone who is sniffing traffic can intercept the request and replay it. To protect against replay attacks, you can rely on adding timestamp for when the webhook event was generated. These should be added to the header. Timestamps are typically used as a seed for computing and verifying a signature. If you try to just change the timestamp, the signature verification will fail because it's not valid.

  Abuse Case:

  - Using a targeted webhook capture a webhook message and POST and replay the same message without modification. If you can do this, replay attacks are possible.

Control 13 - Payload best practices

- **Service Attack Vector** - In most cases webhooks are being used as input for a service account. In these scenarios you need to be mindful to validate the data coming using tight validations of the payload for injection, DDOS, XSS, and other attacks that can be found in the OWASP TOP 10.

  Abuse Case:

  - Using a targeted webhook use POST method to the webhook trying to take advantage of security weaknesses in the service or application that is receiving webhook messages. This can include simple attacks such as sending large amounts of data to complex attacks like injection.

Control 14 - HTTP Methods best practices

- **Supported HTTP methods**- While writing or exposing Webhook. Make sure only enable selected HTTP methods like GET and POST. Other methods like PUT, DELETE and even OPTIONS and HEAD should be disabled if not required.

  Abuse Case:

  - Using a targeted webhook URL, try sending payloads using different HTTP methods like DELETE and PUT and observe the HTTP response.

Reference Docs:

Python Webhook Tester: https://webhook.site/

https://www.fullstackpython.com/webhooks.html

https://www.nexmo.com/blog/2019/06/28/using-message-signatures-to-ensure-secure-incoming-webhooks-dr

https://community.spinnaker.io/t/how-to-secure-webhooks/1122