Preventing Cross Site Request Forgery (CSRF) Attack using CSRF-Tokens on HP Printing Devices

Table of contents

Introduction 2

Detailed Description 2

CSRF Configuration on HP FutureSmart Printers 3

CSRFToken in HTTP POST Requests 5

Impact on Software Tools and Solutions 6

How to address it in Solutions 6

Using a Velocity Template for EWS Content 6

Programmatically Generating the CSRF Token 6

References 7

Introduction

Cross-Site Request Forgery (CSRF) is an exploit which hijacks the authenticated user session to send unauthorized requests to a server. For the server receiving the requests, it appears that the action is initiated by an authenticated user. The actions could weaken the security of the server which a hacker can exploit to take control over the server.

Browsers are inherently trusted and are designed to cache the authenticated session cookie (until it expires). When the user authenticates to a server, the server generates a session authentication cookie which gets cached by the web browser. This cookie will be automatically included by the browser for any subsequent request going out to the server.

When the user who has already logged into a target server clicks on a well formed link unknowingly, the link can execute a specific action on the target server using the cached session information in the browser. Attackers could formulate such links and hide them inside an email body or on a web portal that has been compromised.

There are methods by which CSRF attacks can be detected and prevented by the server. This technical whitepaper provides insight into techniques that will be used to prevent such attacks for the Embedded Web Server on HP FutureSmart devices.

Detailed Description

The Embedded Web Server (EWS) is one of the primary configuration and management interfaces on HP FutureSmart Devices. For security purposes, EWS requires a user to provide Admin credentials. EWS employs cookie based authentication with a default timeout value of 30 minutes. Authentication cookies are typically cached by browsers making the EWS susceptible for the CSRF attack.

The EWS server can employ security measures to prevent CSRF attacks. Standard industry practices include:

- Checking Origin Header
- Checking Referer header
- CSRFToken authentication

Origin Header:

The Origin header provides identity of the security contexts that caused the user agent to initiate an HTTP request. Web servers can prevent the CSRF attack by allowing the requests if the Origin header contains known or white-listed origins. This safeguards the server from unknown or cross domain attacks.

Referer Header:

The Referer header provides identity of the webpage or URI that referred the request being made. The web server can prevent the CSRF attack by examining the hostname in the Referer URI. The check would be similar to Origin header and thus it protects the device from unknown or cross domain attacks.

CSRFToken:

Origin and Referer headers provide safeguards against unknown domain attacks, however these are not mandatory headers. In the case both headers are not present in the incoming request, the web server would not know if this is an authentic request or an attack. In such cases, the CSRFToken provides a means to protect servers against the CSRF attacks.

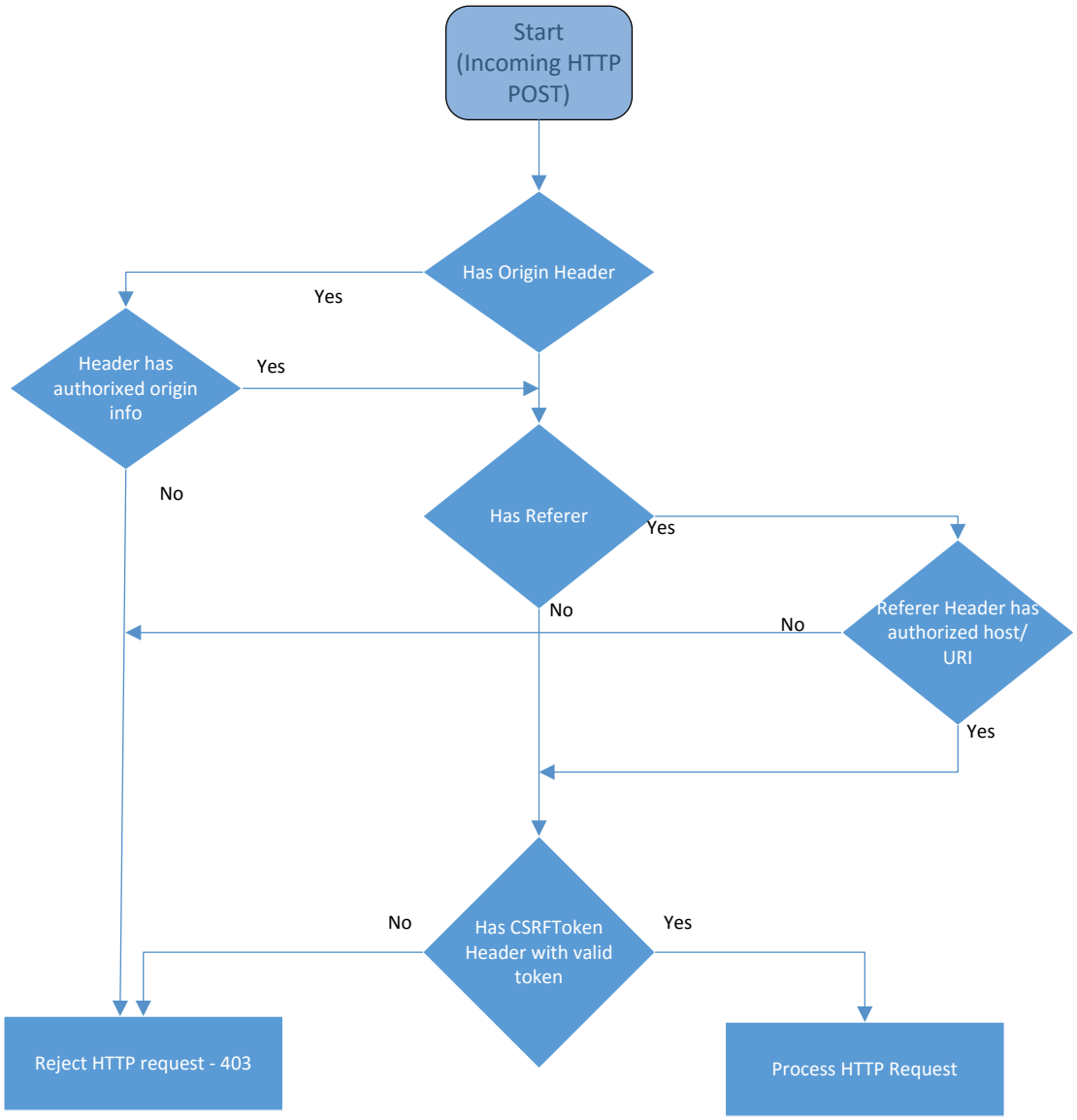
A CSRFToken is a cryptographic randomly generated value. These tokens can be generated on per session basis. Tokens are inserted within HTML forms as “CSRFToken” when the form is fetched from the web server. When the web client submits the HTTP request, the client application must include the CSRFToken “CSRFToken”.

The Server verifies if the token from the client matches the token provided earlier to the client for the current session. If the token is not-present or not-correct, the request would be rejected with “HTTP 403” error. If the “CSRFToken” is correct, the application request would be processed further.

CSRF Configuration on HP FutureSmart Printers

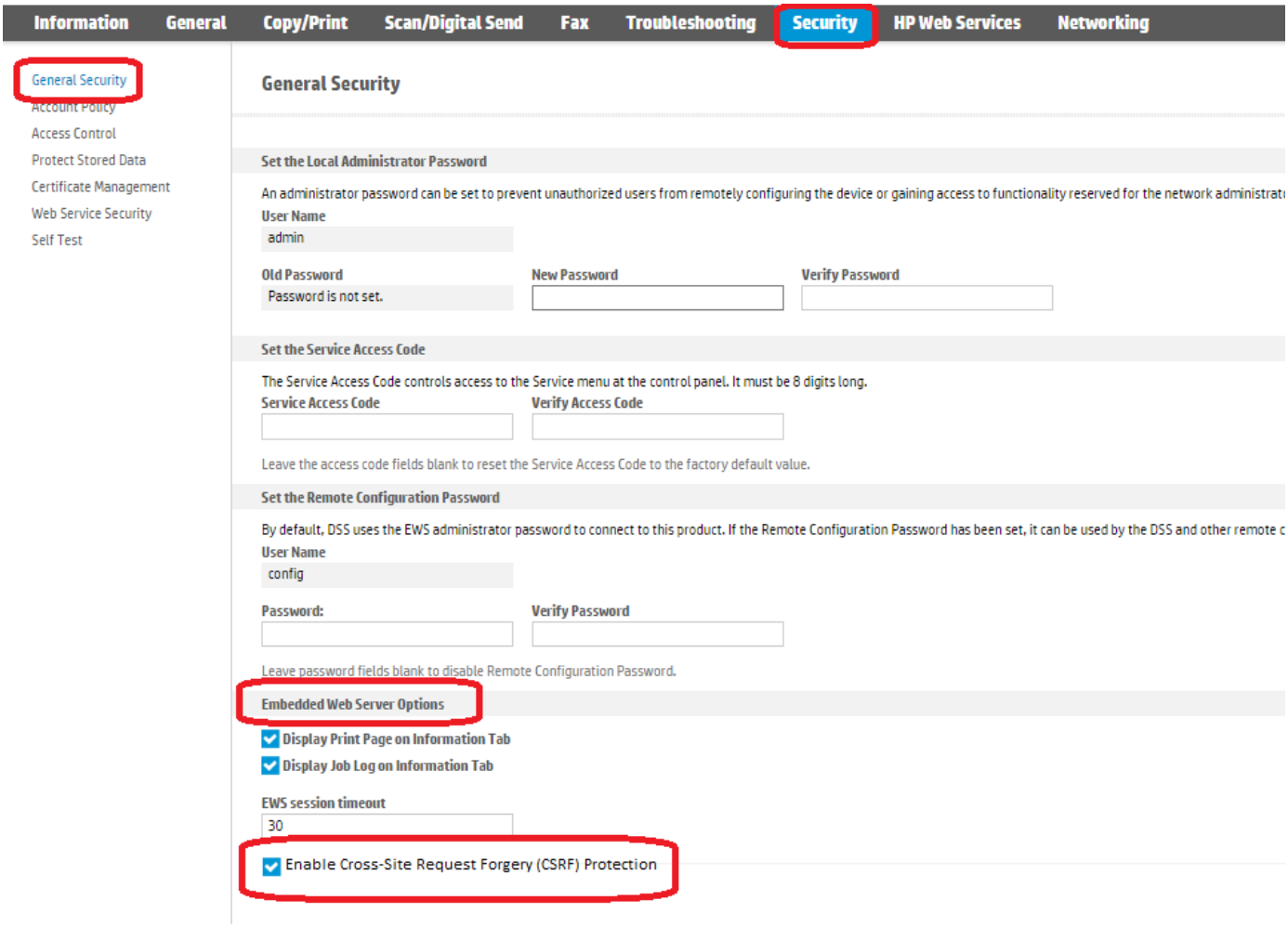
HP FutureSmart devices enforce CSRF protection by default. When this protection is enforced, all the incoming POST requests would be checked presence for of Origin header or Referer header. If they are present, they would be checked for correctness and allowed origin (that is the host-device).

Further, the HTML contents would be checked for CSRF token. If CSRF token verification is successful, the request would be processed further. If the verification fails, the request would be rejected by the server.



This security feature would have impact on any SW Tools or Solutions that uses HTML form data directly in HTTP POST; aka "**Screen Scraping**" to manage settings on the devices. Origin/Referer header or CSRFToken must be present without which request will not be serviced.

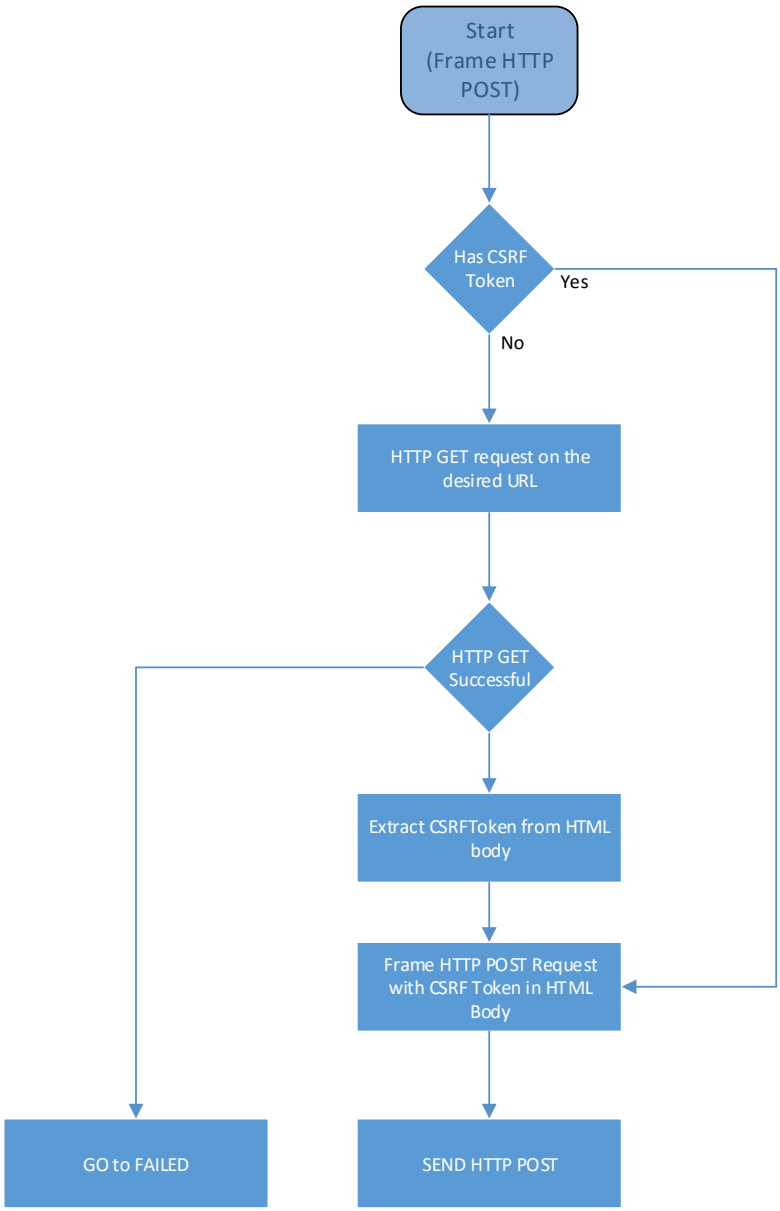
Note: CSRF Protection can be disabled from the EWS or using WS*. The EWS page configuration setting under the "Security" tab is shown below. The default setting is enabled.



CSRFToken in HTTP POST Requests

CSRFTokens are cryptographically random values generated by the printer's web server. Further, these tokens are unique to every EWS session.

The client must first fetch the token from the printer using an HTTP GET request. During the GET request the device generates a SessionId (if not already included in GET request) and a mapping CSRFToken. The CSRFToken is included as a hidden form value with tag CSRFToken. The client app must make use of the SessionId and the extracted CSRFToken from the GET request in its HTTP POST request.



Impact on Software Tools and Solutions

Any web client, whether it is an embedded solution hosted on the device or an external software, will be impacted by the CSRF security measures when enforced by the device. All HTTP POST requests will be scrutinized by the Embedded Web Server for CSRF vulnerability (when the CSRF Protection is enforced). If the POST request fails the vulnerability check, the requests will be rejected with HTTP error code 403. This will break the solutions installed on the device. If such a request is originating from an external software client, the software client's functionality will be broken. The web clients MUST take measures so that they continue to work when CSRF protection if enforced on the device.

Here is what is expected from the solutions and software tools

I. Client MUST have a valid CSRFToken in the HTTP form data.
II. CSRFTokens are unique per HTTP session. SessionId and Token are a pair. Token needs to be fetched by the client using GET before the POST.
III. Origin and Referer are not mandatory headers. If the web client is including these headers, it MUST have trusted host-FQDN.

How to address it in Solutions

When the CSRF Prevention feature is enabled, the EWS within the device will require a CSRFToken parameter to be present in any HTTP POST request. This CSRFToken must match the value originally provided to the client via a previous HTTP GET request. There are two ways in which a Level-1 EWS controller may generate a valid CSRFToken parameter.

Using a Velocity Template for EWS Content

If the Level-1 solution is using a Velocity template to render the HTML content for a client's HTTP GET request, then the template may simply be modified to reference a new FutureSmart Velocity directive in order to have the CSRFToken generated. This directive is:

```
$Security.GetCSRFToken()
```

The form input is expected to have the name "CSRFToken", so in order to generate the complete input tag simply include the following content within the HTML form definition:

```
<input type="hidden" id="CSRFToken" name="CSRFToken" value="$Security.GetCSRFToken()" />
```

Programmatically Generating the CSRF Token

If the Level-1 solution is NOT using Velocity templates to generate content, the CSRF Token may be pulled out of the IExtHttpSessionState instance of the ExtContext. Specifically, when executing on a FW version that supports the CSRF Prevention feature, the CSRF token will present in the session data. The following code demonstrates one mechanism in which it can be pulled out of the header for subsequent use in generating HTML content to return to the client:

```
public class Level1Controller : ExtWebContentControllerBase
        {
          /// <summary>
          /// web request entry point
          /// </summary>
          public override void Load()
          {
            string                    token                    =                    string.Empty;

            if (ExtContext.SessionState.GetData.ContainsKey("CSRFToken") &&
              ExtContext.SessionState.GetData["CSRFToken"] != null)
            {
              token = SessionState.GetData["CSRFToken"].ToString();
            }
           ...
          }
        }
```

In the above example, the non-empty “token” variable will be a valid CSRFToken that can be passed back to a client, and which can then be passed back into the EWS for a subsequent HTTP POST request.

References

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

Doc ID: c05428973
Created November 2016
Version 2.2 April 2017
Public