Product Solutions Resources Open Source Enterprise Pricing Sign in Sign up

Cacti / cacti Public Sponsor Notifications Fork 405 Star 1.6k

Code Issues 188 Pull requests Discussions Actions Projects Wiki Security 42 Insights

Unauthenticated Command Injection

Critical **netniV** published **GHSA-6p93-p743-35gf** on Dec 5, 2022

Package	Affected versions	Patched versions
Cacti (-)	v1.2.22	1.2.23, 1.3.0

Description

Summary

A command injection vulnerability allows an unauthenticated user to execute arbitrary code on a server running Cacti, if a specific data source was selected for any monitored device.

Details

The vulnerability resides in the `remote_agent.php` file. This file can be accessed without authentication. In order to verify that the client is allowed the function `remote_client_authorized` is called:

```
if (!remote_client_authorized()) {
        print 'FATAL: You are not authorized to use this service';
        exit;
}
```

This function retrieves the IP address of the client via `get_client_addr` and resolves this IP address to the corresponding hostname via `gethostbyaddr`. After this, it is verified that an entry within the `poller` table exists, where the hostname corresponds to the resolved hostname. If such an entry was found, the function returns `true` and the client is authorized:

```
function remote_client_authorized() {
        // ...
        $client_addr = get_client_addr();
        // ...

        $client_name = gethostbyaddr($client_addr);
            // ...
        $pollers = db_fetch_assoc('SELECT * FROM poller', true, $poller_db_cnn_id);

        if (cacti_sizeof($pollers)) {
                foreach($pollers as $poller) {
                        if (remote_agent_strip_domain($poller['hostname']) == $client_name) {
                                return true;
                        } elseif ($poller['hostname'] == $client_addr) {
                                return true;
                        }
                }
        }

        cacti_log("Unauthorized remote agent access attempt from $client_name ($client_addr)");

        return false;
}
```

This authorization can be bypassed due to the implementation of the `get_client_addr` function. The function is defined in the file `lib/functions.php` and checks serval `$_SERVER` variables to determine the IP address of the client:

```
function get_client_addr($client_addr = false) {
        $http_addr_headers = array(
                'X-Forwarded-For',
                'X-Client-IP',
```

Severity

Critical 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2022-46169

Weaknesses

CWE-77

Credits

stefan-schiller-sonarsource Reporter

```
                'X-Real-IP',
                'X-ProxyUser-Ip',
                'CF-Connecting-IP',
                'True-Client-IP',
                'HTTP_X_FORWARDED',
                'HTTP_X_FORWARDED_FOR',
                'HTTP_X_CLUSTER_CLIENT_IP',
                'HTTP_FORWARDED_FOR',
                'HTTP_FORWARDED',
                'HTTP_CLIENT_IP',
                'REMOTE_ADDR',
        );

        $client_addr = false;
        foreach ($http_addr_headers as $header) {
                if (!empty($_SERVER[$header])) {
                        $header_ips = explode(',', $_SERVER[$header]);
                        foreach ($header_ips as $header_ip) {
                                if (!empty($header_ip)) {
                                        if (!filter_var($header_ip, FILTER_VALIDATE_IP)) {
                                                cacti_log('ERROR: Invalid remote client IP Addr
                                        } else {
                                                $client_addr = $header_ip;
                                                cacti_log('DEBUG: Using remote client IP Addres
                                                break 2;
                                        }
                                }
                        }
                }
        }

        return $client_addr;
}
```

The variables beginning with `HTTP_` can be arbitrarily set by an attacker. Since there is a default entry in the `poller` table with the hostname of the server running Cacti, an attacker can bypass the authentication e.g. by providing the header `Forwarded-For: <TARGETIP>`. This way the function `get_client_addr` returns the IP address of the server running Cacti. The following call to `gethostbyaddr` will resolve this IP address to the hostname of the server, which will pass the `poller` hostname check because of the default entry.

After the authorization of the `remote_agent.php` file is bypassed, an attacker can trigger different actions. One of these actions is called `polldata`:

```
switch (get_request_var('action')) {
        case 'polldata':
                // ...
                poll_for_data();
                // ...
                break;
```

The called function `poll_for_data` retrieves a few request parameters and loads the corresponding `poller_item` entries from the database. If the `action` of a `poller_item` equals `POLLER_ACTION_SCRIPT_PHP`, the function `proc_open` is used to execute a PHP script:

```
function poll_for_data() {
        global $config;

        $local_data_ids = get_nfilter_request_var('local_data_ids');
        $host_id        = get_filter_request_var('host_id');
        $poller_id      = get_nfilter_request_var('poller_id');
        $return         = array();

        $i = 0;

        if (cacti_sizeof($local_data_ids)) {
                foreach($local_data_ids as $local_data_id) {
                        input_validate_input_number($local_data_id, 'local_data_id');

                        $items = db_fetch_assoc_prepared('SELECT *
                                FROM poller_item
                                WHERE host_id = ?
                                AND local_data_id = ?',
                                array($host_id, $local_data_id));
                        // ...
                        if (cacti_sizeof($items)) {
                                foreach($items as $item) {
                                        switch ($item['action']) {
                                        // ...
                                        case POLLER_ACTION_SCRIPT_PHP: /* script (php script se
                                                // ...
                                                $cactiphp = proc_open(read_config_option('path_
                                                // ...
```

The attacker-controlled parameter `$poller_id` is retrieved via the function `get_nfilter_request_var`, which allows arbitrary strings. This variable is later inserted into the string passed to `proc_open`, which leads to a command injection vulnerability. By e.g. providing the `poller_id=;id` the `id` command is executed.

In order to reach the vulnerable call, the attacker must provide a `host_id` and `local_data_id`, where the `action` of the corresponding `poller_item` is set to `POLLER_ACTION_SCRIPT_PHP`. Both of these ids (`host_id` and `local_data_id`) can easily be bruteforced. The only requirement is that a `poller_item` with an `POLLER_ACTION_SCRIPT_PHP` action exists. This is very likely on a productive instance because this action is added by some predefined templates like `Device - Uptime` or `Device - Polling Time`.

Impact

This command injection vulnerability allows an unauthenticated user to execute arbitrary commands if a `poller_item` with the `action` type `POLLER_ACTION_SCRIPT_PHP` (`2`) is configured.

Remediation

The following suggestions should be applied to prevent the described vulnerability.

The authorization bypass should be prevented by not allowing an attacker to make `get_client_addr` (file `lib/functions.php`) return an arbitrary IP address. This could be done by not honoring the `HTTP_...` `$_SERVER` variables. If these should be kept for compatibility reasons it should at least be prevented to fake the IP address of the server running Cacti.

The command injection should be prevented by applying the following changes to the file `remote_agent.php`:

The variable `$poller_id` is supposed to be an integer and should thus be retrieved via the function `get_filter_request_var` instead of `get_nfilter_request_var`:

```
function poll_for_data() {
        // ...
           $poller_id      = get_filter_request_var('poller_id');
           // ...
```

For further hardening against command injections the `$poller_id` should be escaped with `escapeshellarg` before being passed to `proc_open`:

```
function poll_for_data() {
// ...
$cactiphp = proc_open(read_config_option('path_php_binary') . ' -q ' . $config['base_path'] . '
// ...
```

Patches

Version	Patches
1.2.x	`7f0e163`
1.3.x	`b43f13a`

For instances of 1.2.x running under PHP < 7.0, a further change `a8d59e8` is also required.