18

Denial of Service by requesting to reset a password

Share:

TIMELINE

makerlab submitted a report to Nextcloud. Mar 7th (3 ye

Description:

I believe that this is posible due to the brute force protection that makes all request last for 30 seconds which in this case is using all the PHP workers avalible in the pool, so the only way to defend yourself is setting up a limit or having a lot of resources.

How to reproduce:

- In the Nextcloud login screen click the "Forgot password?" button and then type something in the textbox (can be anything)
- Then open the developers tools and go to the network tab
- Hold the "enter" key after pressing the reset password button and in the network tab you will see a lot of request being made
- With just 1000 request I managed to make the demo server "https://demo2.nextcloud.com/" not respond for 1 hour

Impact

The attacker could make an entire nextcloud installation or even the entire server where it is hosted not respond for a very long time
Also, this attack can be made by almost anyone

2 attachments:
F740352: Report2.png
F740353: Report.png

OT: posted a comment. Mar 7th (3 ye

Thanks a lot for reporting this potential issue back to us!

Our security team will take a look at this issue as soon as possible. We will reply to your report within 72 hours, usually much faster. For obvious reasons we'd like t you to not disclose this issue to any other party.

nickvergessen Nextcloud staff changed the status to **Triaged**. Mar 9th (3 ye

Hmm, couple of reqeusts are killed with 412 fo me right away.
But yeah seems to be possible to make the server busy.

makerlab posted a comment. Mar 9th (3 ye

In my Nextcloud installation I get 429 and 504 at a very slow rate
The login page also seems to be vulnerable
A solution to this problem could be making the login system drop connections with a 429 and a message containing something like "Too many requests, try again i minutes" (like most web apps) instead of throttling the next login attempt to 30 seconds which easily fills the PHP pool

nickvergessen Nextcloud staff posted a comment. Mar 19th (3 ye

in the last 30 mins by an ip.
In case you want to test it, see the patch below.

cheers nickvergessen

Code 10.98 KiB Wrap lines Copy Dow

```
diff --git a/core/templates/429.php b/core/templates/429.php
new file mode 100644
index 0000000000..caf8a3675e
--- /dev/null
+++ b/core/templates/429.php
@@ -0,0 +1,4 @@
+<div class="body-login-container update">
+    <h2><?php p($l->t('Too many requests')); ?></h2>
+    <p class="infogroup"><?php p($l->t('There were too many requests from your network. Retry later or contact your administrator if this is an error
+</div>
diff --git a/lib/private/AppFramework/Middleware/Security/BruteForceMiddleware.php b/lib/private/AppFramework/Middleware/Security/BruteForceMiddlewar
index 46c33083e4..6cb44da3ed 100644
--- a/lib/private/AppFramework/Middleware/Security/BruteForceMiddleware.php
+++ b/lib/private/AppFramework/Middleware/Security/BruteForceMiddleware.php
@@ -1,4 +1,5 @@
 <?php
+declare(strict_types=1);
 /**
  * @copyright Copyright (c) 2017 Lukas Reschke <lukas@statuscode.ch>
  *
@@ -25,9 +26,15 @@

 use OC\AppFramework\Utility\ControllerMethodReflector;
 use OC\Security\Bruteforce\Throttler;
+use OCP\AppFramework\Controller;
+use OCP\AppFramework\Http;
 use OCP\AppFramework\Http\Response;
+use OCP\AppFramework\Http\TooManyRequestsResponse;
 use OCP\AppFramework\Middleware;
+use OCP\AppFramework\OCS\OCSException;
+use OCP\AppFramework\OCSController;
 use OCP\IRequest;
+use OCP\Security\Bruteforce\MaxDelayReached;

 /**
  * Class BruteForceMiddleware performs the bruteforce protection for controllers
@@ -65,7 +72,7 @@ public function beforeController($controller, $methodName) {

        if($this->reflector->hasAnnotation('BruteForceProtection')) {
            $action = $this->reflector->getAnnotationParameter('BruteForceProtection', 'action');
-           $this->throttler->sleepDelay($this->request->getRemoteAddress(), $action);
+           $this->throttler->sleepDelayOrThrowOnMax($this->request->getRemoteAddress(), $action);
        }
    }

@@ -82,4 +89,23 @@ public function afterController($controller, $methodName, Response $response) {

        return parent::afterController($controller, $methodName, $response);
    }
+
+   /**
+    * @param Controller $controller
+    * @param string $methodName
+    * @param \Exception $exception
+    * @throws \Exception
+    * @return Response
+    */
+   public function afterException($controller, $methodName, \Exception $exception): Response {
+       if ($exception instanceof MaxDelayReached) {
+           if ($controller instanceof OCSController) {
+               throw new OCSException($exception->getMessage(), Http::STATUS_TOO_MANY_REQUESTS);
+           }
+
+           return new TooManyRequestsResponse();
+       }
+
+       throw $exception;
+   }
 }
diff --git a/lib/private/Security/Bruteforce/Throttler.php b/lib/private/Security/Bruteforce/Throttler.php
index b5a4dfbfaf..c11637692e 100644
--- a/lib/private/Security/Bruteforce/Throttler.php
+++ b/lib/private/Security/Bruteforce/Throttler.php
```

```
 /**
  * @copyright Copyright (c) 2016 Lukas Reschke <lukas@statuscode.ch>
  *
@@ -33,6 +34,7 @@
 use OCP\IConfig;
 use OCP\IDBConnection;
 use OCP\ILogger;
+use OCP\Security\Bruteforce\MaxDelayReached;

 /**
  * Class Throttler implements the bruteforce protection for security actions in
@@ -49,6 +51,8 @@
  */
 class Throttler {
    const LOGIN_ACTION = 'login';
+   const MAX_DELAY = 25;
+   const MAX_ATTEMPTS = 10;

    /** @var IDBConnection */
    private $db;
@@ -81,7 +85,7 @@ public function __construct(IDBConnection $db,
     * @param int $expire
     * @return \DateInterval
     */
-   private function getCutoff($expire) {
+   private function getCutoff(int $expire): \DateInterval {
       $d1 = new \DateTime();
       $d2 = clone $d1;
       $d2->sub(new \DateInterval('PT' . $expire . 'S'));
@@ -96,9 +100,9 @@ private function getCutoff($expire) {
     * @param array $metadata Optional metadata logged to the database
     * @suppress SqlInjectionChecker
     */
-   public function registerAttempt($action,
-                                   $ip,
-                                   array $metadata = []) {
+   public function registerAttempt(string $action,
+                                   string $ip,
+                                   array $metadata = []): void {
       // No need to log if the bruteforce protection is disabled
       if($this->config->getSystemValue('auth.bruteforce.protection.enabled', true) === false) {
           return;
@@ -138,7 +142,7 @@ public function registerAttempt($action,
     * @param string $ip
     * @return bool
     */
-   private function isIPWhitelisted($ip) {
+   private function isIPWhitelisted(string $ip): bool {
       if($this->config->getSystemValue('auth.bruteforce.protection.enabled', true) === false) {
           return true;
       }
@@ -196,7 +200,6 @@ private function isIPWhitelisted($ip) {
       }

       return false;
-
   }

   /**
@@ -204,20 +207,21 @@ private function isIPWhitelisted($ip) {
    *
    * @param string $ip
    * @param string $action optionally filter by action
+   * @param float $maxAgeHours
    * @return int
    */
-  public function getDelay($ip, $action = '') {
+  public function getAttempts(string $ip, string $action = '', float $maxAgeHours = 12): int {
      $ipAddress = new IpAddress($ip);
      if ($this->isIPWhitelisted((string)$ipAddress)) {
          return 0;
      }

      $cutoffTime = (new \DateTime())
-         ->sub($this->getCutoff(43200))
+         ->sub($this->getCutoff((int) ($maxAgeHours * 3600)))
          ->getTimestamp();

```

```
                ->from('bruteforce_attempts')
                ->where($qb->expr()->gt('occurred', $qb->createNamedParameter($cutoffTime)))
                ->andWhere($qb->expr()->eq('subnet', $qb->createNamedParameter($ipAddress->getSubnet())));
@@ -226,24 +230,37 @@ public function getDelay($ip, $action = '') {
                $qb->andWhere($qb->expr()->eq('action', $qb->createNamedParameter($action)));
            }

-           $attempts = count($qb->execute()->fetchAll());
+           $result = $qb->execute();
+           $row = $result->fetch();
+           $result->closeCursor();
+
+           return (int) $row['attempts'];
+       }

+       /**
+        * Get the throttling delay (in milliseconds)
+        *
+        * @param string $ip
+        * @param string $action optionally filter by action
+        * @return int
+        */
+       public function getDelay(string $ip, string $action = ''): int {
+           $attempts = $this->getAttempts($ip, $action);
            if ($attempts === 0) {
                return 0;
            }

-           $maxDelay = 25;
            $firstDelay = 0.1;
-           if ($attempts > (8 * PHP_INT_SIZE - 1))  {
+           if ($attempts > self::MAX_ATTEMPTS) {
                // Don't ever overflow. Just assume the maxDelay time:s
-               $firstDelay = $maxDelay;
-           } else {
-               $firstDelay *= pow(2, $attempts);
-               if ($firstDelay > $maxDelay) {
-                   $firstDelay = $maxDelay;
-               }
+               return self::MAX_DELAY;
            }
-           return (int) \ceil($firstDelay * 1000);
+
+           $delay = $firstDelay * 2**$attempts;
+           if ($delay > self::MAX_DELAY) {
+               return self::MAX_DELAY;
+           }
+           return (int) \ceil($delay * 1000);
        }

        /**
@@ -253,7 +270,7 @@ public function getDelay($ip, $action = '') {
         * @param string $action
         * @param string $metadata
         */
-       public function resetDelay($ip, $action, $metadata) {
+       public function resetDelay(string $ip, string $action, string $metadata): void {
            $ipAddress = new IpAddress($ip);
            if ($this->isIPWhitelisted((string)$ipAddress)) {
                return;
@@ -280,9 +297,28 @@ public function resetDelay($ip, $action, $metadata) {
         * @param string $action optionally filter by action
         * @return int the time spent sleeping
         */
-       public function sleepDelay($ip, $action = '') {
+       public function sleepDelay(string $ip, string $action = ''): int {
            $delay = $this->getDelay($ip, $action);
            usleep($delay * 1000);
            return $delay;
        }
+
+       /**
+        * Will sleep for the defined amount of time unless maximum was reached in the last 30 minutes
+        * In this case a "429 Too Many Request" exception is thrown
+        *
+        * @param string $ip
+        * @param string $action optionally filter by action
+        * @return int the time spent sleeping
+      $delay = $this->getDelay($ip, $action);
+      if (($delay === self::MAX_DELAY * 1000) && $this->getAttempts($ip, $action, 0.5) > self::MAX_ATTEMPTS) {
+          // If the ip made too many attempts within the last 30 mins we don't execute anymore
+          throw new MaxDelayReached('Reached maximum delay');
+      }
+      usleep($delay * 1000);
+      return $delay;
+  }
 }
diff --git a/lib/public/AppFramework/Http/TooManyRequestsResponse.php b/lib/public/AppFramework/Http/TooManyRequestsResponse.php
new file mode 100644
index 0000000000..160614ab33
--- /dev/null
+++ b/lib/public/AppFramework/Http/TooManyRequestsResponse.php
@@ -0,0 +1,51 @@
+<?php
+declare(strict_types=1);
+/**
+ * @copyright Copyright (c) 2020 Joas Schilling <coding@schilljs.com>
+ *
+ * @license GNU AGPL version 3 or any later version
+ *
+ * This program is free software: you can redistribute it and/or modify
+ * it under the terms of the GNU Affero General Public License as
+ * published by the Free Software Foundation, either version 3 of the
+ * License, or (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU Affero General Public License for more details.
+ *
+ * You should have received a copy of the GNU Affero General Public License
+ * along with this program.  If not, see <http://www.gnu.org/licenses/>.
+ *
+ */
+
+namespace OCP\AppFramework\Http;
+
+use OCP\Template;
+
+/**
+ * A generic 429 response showing an 404 error page as well to the end-user
+ * @since 19.0.0
+ */
+class TooManyRequestsResponse extends Response {
+
+  /**
+   * @since 19.0.0
+   */
+  public function __construct() {
+      parent::__construct();
+
+      $this->setContentSecurityPolicy(new ContentSecurityPolicy());
+      $this->setStatus(429);
+  }
+
+  /**
+   * @return string
+   * @since 19.0.0
+   */
+  public function render() {
+      $template = new Template('core', '429', 'blank');
+      return $template->fetchPage();
+  }
+}
diff --git a/lib/public/Security/Bruteforce/MaxDelayReached.php b/lib/public/Security/Bruteforce/MaxDelayReached.php
new file mode 100644
index 0000000000..817ef0e60c
--- /dev/null
+++ b/lib/public/Security/Bruteforce/MaxDelayReached.php
@@ -0,0 +1,30 @@
+<?php
+declare(strict_types=1);
+/**
+ * @copyright Copyright (c) 2020 Joas Schilling <coding@schilljs.com>
+ *
+ * @license GNU AGPL version 3 or any later version
+ * published by the Free Software Foundation, either version 3 of the
+ * License, or (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU Affero General Public License for more details.
+ *
+ * You should have received a copy of the GNU Affero General Public License
+ * along with this program.  If not, see <http://www.gnu.org/licenses/>.
+ *
+ */
+
+namespace OCP\Security\Bruteforce;
+
+/**
+ * Class MaxDelayReached
+ * @since 19.0
+ */
+class MaxDelayReached extends \RuntimeException {
+}
```

makerlab posted a comment. Mar 19th (3 ye

Thanks for the response and the patch!

I will test it with the patch added and I'll see how it does with variants of this attack

By the way I love this aproach, combining both, the original idea and the new 429 response with too many requests

makerlab posted a comment. Mar 19th (3 ye

It works but in my nextcloud installation (which is still in 18.0.2) I get the following error: `Sabre\DAV\Exception\ServiceUnavailable: TypeError: Argument 3 passed t` `OC\Security\Bruteforce\Throttler::resetDelay() must be of the type string, array given, called in /var/www/html/cloud/lib/base.php on line 816`

Which as a workaround I deleted the "string" in the " `public function resetDelay(string $ip, string $action, this->string $metadata): void {` " as for some rea was getting an array instead

The patch needs some polishing but I think it fixed the vulnerability so for me it is ok

nickvergessen Nextcloud staff posted a comment. Mar 20th (3 ye

you are right:

Code 716 Bytes Wrap lines Copy Dow

```
diff --git a/lib/private/Security/Bruteforce/Throttler.php b/lib/private/Security/Bruteforce/Throttler.php
index c11637692e..563b7099c9 100644
--- a/lib/private/Security/Bruteforce/Throttler.php
+++ b/lib/private/Security/Bruteforce/Throttler.php
@@ -268,9 +268,9 @@ public function getDelay(string $ip, string $action = ''): int {
      *
      * @param string $ip
      * @param string $action
-     * @param string $metadata
+     * @param array $metadata
      */
-    public function resetDelay(string $ip, string $action, string $metadata): void {
+    public function resetDelay(string $ip, string $action, array $metadata): void {
         $ipAddress = new IpAddress($ip);
         if ($this->isIPWhitelisted((string)$ipAddress)) {
             return;

```

makerlab posted a comment. Jul 17th (2 ye

Any update on this?

Nextcloud 19 is still vulnerable to this attack

nickvergessen Nextcloud staff posted a comment. Jul 29th (2 ye

Yeah we are still discussing the PR, but it is planned to finish until the Nextcloud 20 release.

We might still backport it then to older versions.

nickvergessen Nextcloud staff closed the report and changed the status to ○ **Resolved**. Jan 15th (2 ye

Sorry this issue somehow stayed open although we fixed it already in august.

Thanks a lot for your report again. This has been resolved in our latest maintenance releases and we're working on the advisories at the moment.

Please let us know how you'd like to be credited in our official advisory. We require the following information:

- Name / Pseudonym
- Email address (optional)
- Website (optional)
- Company (optional)