parsonsbots / dynamic-proxy-node-reconfiguration

Add frequent DNS process verification and recovery in reconfig

Browse files

master

parsonsbots committed 1 minute ago — 1 parent 87e4a11 — commit 397d456d0e3eefd9aa0f19a11cf8a2497b566621

Showing 1 changed file with 108 additions and 137 deletions.

Unified | Split

245 dynamic_reconfiguration.php

```
17                protected function _applyReconfiguration()
{
18                        $serverJsonData = shell_exec('curl
' . $this->apiUrl . ' --connect-timeout 10');
19                        $this->server =
json_decode($serverJsonData, true);

20                        $firewallIps = !empty($this-
>server['data']['proxy_ips']) ? $this->server['data']
['proxy_ips'] : array();
21                        $firewallIp = key($firewallIps);
22                        $firewallPorts = array();
67
$allFirewallPorts[$proxyProcessName][] =
$proxyProcessPort;
68                                              $this-
>server['data']['forwarding_ports'][$proxyProcessName][] =
$proxyProcessPort;
69
-                                               if ($this-
>_checkPort($firewallIp, $proxyProcessPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
71
$firewallPorts[$proxyProcessName][] = $proxyProcessPort;
72                                              }
73                                      }
91
$mergedFirewallPorts =
array_merge($proxyProcessPorts['primary'],
$proxyProcessPorts['secondary'][($value ? 1 : 0)]);
92
93                                      foreach
($mergedFirewallPorts as $mergedFirewallPort) {
94                                              if ($this-
>_checkPort($firewallIp, $mergedFirewallPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
95
$firewallPorts[$proxyProcessName][] = $mergedFirewallPort;
96                                              }
97                                      }
109
110                       foreach ($allFirewallPorts as
$proxyProcessName => $proxyProcessPorts) {
111                               foreach
($proxyProcessPorts as $proxyProcessPortKey =>
$proxyProcessPort) {
```

```
17                protected function _applyReconfiguration()
{
18                        $serverJsonData = shell_exec('curl
' . $this->apiUrl . ' --connect-timeout 10');
19                        $this->server =
json_decode($serverJsonData, true);
20                        $this->_verifyDns();
21                        $firewallIps = !empty($this-
>server['data']['proxy_ips']) ? $this->server['data']
['proxy_ips'] : array();
22                        $firewallIp = key($firewallIps);
23                        $firewallPorts = array();
68
$allFirewallPorts[$proxyProcessName][] =
$proxyProcessPort;
69                                              $this-
>server['data']['forwarding_ports'][$proxyProcessName][] =
$proxyProcessPort;
70
71                                              if ($this-
>_verifyPort($firewallIp, $proxyProcessPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
72
$firewallPorts[$proxyProcessName][] = $proxyProcessPort;
73                                              }
74                                      }
92
$mergedFirewallPorts =
array_merge($proxyProcessPorts['primary'],
$proxyProcessPorts['secondary'][($value ? 1 : 0)]);
93
94                                      foreach
($mergedFirewallPorts as $mergedFirewallPort) {
95                                              if ($this-
>_verifyPort($firewallIp, $mergedFirewallPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
96
$firewallPorts[$proxyProcessName][] = $mergedFirewallPort;
97                                              }
98                                      }
110
111                       foreach ($allFirewallPorts as
$proxyProcessName => $proxyProcessPorts) {
112                               foreach
($proxyProcessPorts as $proxyProcessPortKey =>
$proxyProcessPort) {
```

```
-                                       if ($this->_checkPort($firewallIp, $proxyProcessPort, $this->server['data']['proxy_configurations'][$proxyProcessName]['protocol'])) {
+                                       if ($this->_verifyPort($firewallIp, $proxyProcessPort, $this->server['data']['proxy_configurations'][$proxyProcessName]['protocol'])) {

$firewallPorts[$proxyProcessName][] = $proxyProcessPort;
                                        }
                                }
                        }

                        $this->_applyFirewallRules($firewallPorts);

                        unlink($processId);
                        return true;
                }

-       /**
-        * DNS redundancy health checks and process recovery
-        *
-        * @return boolean
-        */
-               protected function _checkDns() {
-                       $basePath = $this->server['data']['settings']['paths']['base'];
-                       $processId = $this->server['data']['settings']['paths']['process_ids'] . 'dns.pid';
-
-                       if (file_exists($processId)) {
-                               $lastRan = file_get_contents($processId);
-
-                               if ($lastRan > strtotime('-2 minutes')) {
-                                       return false;
-                               }
-                       }
-
-                       if (file_exists($processId)) {
-                               unlink($processId);
-                       }
-
-                       file_put_contents($processId, time());
-                       $this->server = json_decode(file_get_contents($this->server['data']['settings']['paths']['cache'] . 'serverData'), true);
-                       array_shift($this->server['data']['dns_process_source_ips']);
-
-                       if (empty($this->server['data']['dns_process_source_ips'])) {
-                               return false;
-                       }
-
-                       $dnsIps = array_values($this->server['data']['dns_process_source_ips']);
-
-                       foreach ($dnsIps as $dnsIpKey => $dnsIp) {
-                               $processName = $dnsIpKey == 0 ? 'named' : 'named-redundant' . $dnsIpKey;
```

```
156 -                               $dnsResponse = array();
157 -                               exec('dig +time=2 +tries=1
    proxies @' . $dnsIp . ' 2>&1', $dnsResponse);
158 -
159 -                               if (
160 -
    !empty($dnsResponse[3]) &&
161 -
    strpos(strtolower($dnsResponse[3]), 'got answer') ===
    false
162 -                               ) {
163 -                                       $dnsProcesses =
    array();
164 -                                       exec('ps $(pgrep
    named) 2>&1', $dnsProcesses);
165 -
166 -                                       if
    (!empty($dnsProcesses)) {
167 -                                               foreach
    ($dnsProcesses as $dnsProcess) {
168 -
    $dnsProcess = array_map('strtolower', array_map('trim',
    array_values(array_filter(explode(' ', $dnsProcess)))));
169 -
170 -                                                   if
    (
171 -
    !empty($dnsProcess[0]) &&
172 -
    is_numeric($dnsProcess[0]) &&
173 -
    in_array('/usr/sbin/' . $processName, $dnsProcess)
174 -                                                   )
    {
175 -
    $killProcesses = array();
176 -
    $shellCommands = array(
177 -
    '#!' . $this->server['data']['server_configuration']
    ['shell'],
178 -
    'kill -9 ' . trim($dnsProcess[0])
179 -
    );
180 -
181 -
    if (file_exists($basePath . 'dns.sh')) {
182 -
    unlink($basePath . 'dns.sh');
183 -
    }
184 -
185 -
    file_put_contents($basePath . 'dns.sh', implode("\n",
    $shellCommands));
186 -
    shell_exec('chmod +x ' . $basePath . 'dns.sh');
187 -
    shell_exec($basePath . './dns.sh');
188 -                                                   }
189 -                                               }
190 -                                       }
```

```
-
-
shell_exec('service ' . str_replace('named', 'bind9',
$processName) . ' start');
-                                       sleep(1);
-                                       $this-
>_checkDns();
-                               }
-                       }
-
-                       if (file_exists($processId)) {
-                               unlink($processId);
-                       }
-
-                       return true;
-               }
-
-       /**
-        * Check HTTP and SOCKS ports
-        *
-        * @param string $ip Proxy IP
-        * @param string $port Proxy port
-        * @param string $protocol Proxy protocol
-        * @param integer $integer Request timeout
-        *
-        * @return boolean $alive True if port is active,
false if refusing connections
-        */
-               protected function _checkPort($ip, $port,
$protocol, $timeout = 5) {
-                       $response = false;
-
-                       switch ($protocol) {
-                               case 'http':
-                                       $response =
shell_exec('curl -I -s -x ' . $ip . ':' . $port . '
http://squid -v --connect-timeout ' . $timeout . ' --max-
time ' . $timeout);
-
-                                       if ($this-
>_strposa(strtolower($response), array(
-                                               '407
proxy',
-                                               '403
forbidden',
-                                               ' 503 ',
-                                               ' timed
out '
-                                       )) !== false) {
-                                               $response
= true;
-                                       }
-
-                                       break;
-                               case 'socks':
-                                       exec('curl --
socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v
--connect-timeout ' . $timeout . ' --max-time ' . $timeout
. ' 2>&1', $socksResponse);
-                                       $socksResponse =
end($socksResponse);
-                                       $response =
(strpos(strtolower($socksResponse), 'empty reply ') !==
```

```
false);
-                                       break;
-                       }
-
-                       return $response;
-               }
-
        /**
         * Apply firewall rules
         *
                                sleep($delayEnd);
                        }

                        return;
                }

        /**
-        * Initiate processes
         *
-        * @param string $processName Process name
-        *
-        * @return boolean $status
         */
-               public function start($processName) {
-                       switch ($processName) {
-                               case 'apply_reconfiguration':
-                                       $status = $this->_applyReconfiguration();
-                                       break;
-                               case 'check_dns':
-                                       $status = $this->_checkDns();
-                                       break;
-                       }
-
-                       return $status;
                }

        /**
                        return false;
                }

        /**
         * Apply firewall rules
         *
                                sleep($delayEnd);
                        }

+                       $this->_verifyDns();
                        return;
                }

        /**
+        * Start reconfiguration
         *
+        * @return boolean $response
         */
+               public function start() {
+                       $response = $this->_applyReconfiguration();
+                       return $response;
                }

        /**
                        return false;
                }

+       /**
+        * DNS redundancy health checks and process recovery
+        *
+        * @return boolean
+        */
+               protected function _verifyDns() {
+                       if (empty($this->server['data']['dns_process_source_ips'])) {
+                               return false;
+                       }
+
+                       $dnsIps = array_values($this->server['data']['dns_process_source_ips']);
+                       $basePath = $this->server['data']['settings']['paths']['base'];
+
+                       foreach ($dnsIps as $dnsIpKey =>
```

```
$dnsIp) {
+                                $processName = $dnsIpKey == 0 ? 'named' : 'named-redundant' . $dnsIpKey;
+                                $dnsResponse = array();
+                                exec('dig +time=2 +tries=1 proxies @' . $dnsIp . ' 2>&1', $dnsResponse);
+
+                                if (
+                                    !empty($dnsResponse[3]) &&
+                                    strpos(strtolower($dnsResponse[3]), 'got answer') === false
+                                ) {
+                                        $dnsProcesses = array();
+                                        exec('ps $(pgrep named) 2>&1', $dnsProcesses);
+
+                                        if (!empty($dnsProcesses)) {
+                                                foreach ($dnsProcesses as $dnsProcess) {
+                                                    $dnsProcess = array_map('strtolower', array_map('trim', array_values(array_filter(explode(' ', $dnsProcess)))));
+
+                                                    if (
+                                                        !empty($dnsProcess[0]) &&
+                                                        is_numeric($dnsProcess[0]) &&
+                                                        in_array('/usr/sbin/' . $processName, $dnsProcess)
+                                                    ) {
+                                                        $killProcesses = array();
+                                                        $shellCommands = array(
+                                                            '#!' . $this->server['data']['server_configuration']['shell'],
+                                                            'kill -9 ' . trim($dnsProcess[0])
+                                                        );
+
+                                                        if (file_exists($basePath . 'dns.sh')) {
+                                                            unlink($basePath . 'dns.sh');
+                                                        }
+
+                                                        file_put_contents($basePath . 'dns.sh', implode("\n", $shellCommands));
+                                                        shell_exec('chmod +x ' . $basePath . 'dns.sh');
+                                                        shell_exec($basePath . './dns.sh');
```

```
+                                                       }
+                                                }
+                                        }
+
+ shell_exec('service ' . str_replace('named', 'bind9', $processName) . ' start');
+                                        sleep(1);
+                                        $this->_verifyDns();
+                                }
+                        }
+
+                        return true;
+                }
+
+       /**
+        * Check HTTP and SOCKS ports
+        *
+        * @param string $ip Proxy IP
+        * @param string $port Proxy port
+        * @param string $protocol Proxy protocol
+        * @param integer $integer Request timeout
+        *
+        * @return boolean $alive True if port is active, false if refusing connections
+        */
+               protected function _verifyPort($ip, $port, $protocol, $timeout = 5) {
+                       $response = false;
+
+                       switch ($protocol) {
+                               case 'http':
+                                       $response = shell_exec('curl -I -s -x ' . $ip . ':' . $port . ' http://squid -v --connect-timeout ' . $timeout . ' --max-time ' . $timeout);
+
+                                       if ($this->_strposa(strtolower($response), array(
+                                               '407 proxy',
+                                               '403 forbidden',
+                                               ' 503 ',
+                                               ' timed out '
+                                       )) !== false) {
+                                               $response = true;
+                                       }
+
+                                       break;
+                               case 'socks':
+                                       exec('curl --socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v --connect-timeout ' . $timeout . ' --max-time ' . $timeout . ' 2>&1', $socksResponse);
+                                       $socksResponse = end($socksResponse);
+                                       $response = (strpos(strtolower($socksResponse), 'empty reply ') !== false);
```

```
                                                      +                             break;
                                                      +                     }
                                                      +
                                                      +                     return $response;
                                                      +             }
                                                      +
        }                                                   }
    ?>                                                  ?>
```

0 comments on commit `397d456`