

## 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     {
19         $serverJsonData = shell_exec('curl
20         ' . $this->apiUrl . ' --connect-timeout 10');
21         $this->server =
22         json_decode($serverJsonData, true);
23
24         $firewallIps = !empty($this-
25         >server['data']['proxy_ips']) ? $this->server['data']
26         ['proxy_ips'] : array();
27         $firewallIp = key($firewallIps);
28         $firewallPorts = array();
29
30         $allFirewallPorts[$proxyProcessName][] =
31         $proxyProcessPort;
32
33         $this-
34         >server['data']['forwarding_ports'][$proxyProcessName][] =
35         $proxyProcessPort;
36
37         - if ($this-
38         >_checkPort($firewallIp, $proxyProcessPort, $this-
39         >server['data']['proxy_configurations'][$proxyProcessName]
40         ['protocol'])) {
41
42         $firewallPorts[$proxyProcessName][] = $proxyProcessPort;
43         }
44     }
45
46     $mergedFirewallPorts =
47     array_merge($proxyProcessPorts['primary'],
48     $proxyProcessPorts['secondary'](($value ? 1 : 0)));
49
50     foreach
51     ($mergedFirewallPorts as $mergedFirewallPort) {
52         - if ($this-
53         >_checkPort($firewallIp, $mergedFirewallPort, $this-
54         >server['data']['proxy_configurations'][$proxyProcessName]
55         ['protocol'])) {
56
57         $firewallPorts[$proxyProcessName][] = $mergedFirewallPort;
58         }
59     }
60
61     foreach ($allFirewallPorts as
62     $proxyProcessName => $proxyProcessPorts) {
63         foreach
64         ($proxyProcessPorts as $proxyProcessPortKey =>
65         $proxyProcessPort) {
```

```
17         protected function _applyReconfiguration()
18     {
19         $serverJsonData = shell_exec('curl
20         ' . $this->apiUrl . ' --connect-timeout 10');
21         $this->server =
22         json_decode($serverJsonData, true);
23         +
24         $this->_verifyDns();
25         $firewallIps = !empty($this-
26         >server['data']['proxy_ips']) ? $this->server['data']
27         ['proxy_ips'] : array();
28         $firewallIp = key($firewallIps);
29         $firewallPorts = array();
30
31         $allFirewallPorts[$proxyProcessName][] =
32         $proxyProcessPort;
33
34         $this-
35         >server['data']['forwarding_ports'][$proxyProcessName][] =
36         $proxyProcessPort;
37         +
38         if ($this-
39         >_verifyPort($firewallIp, $proxyProcessPort, $this-
40         >server['data']['proxy_configurations'][$proxyProcessName]
41         ['protocol'])) {
42
43         $firewallPorts[$proxyProcessName][] = $proxyProcessPort;
44         }
45
46         $mergedFirewallPorts =
47         array_merge($proxyProcessPorts['primary'],
48         $proxyProcessPorts['secondary'](($value ? 1 : 0)));
49
50         foreach
51         ($mergedFirewallPorts as $mergedFirewallPort) {
52         +
53         if ($this-
54         >_verifyPort($firewallIp, $mergedFirewallPort, $this-
55         >server['data']['proxy_configurations'][$proxyProcessName]
56         ['protocol'])) {
57
58         $firewallPorts[$proxyProcessName][] = $mergedFirewallPort;
59         }
60
61         foreach ($allFirewallPorts as
62         $proxyProcessName => $proxyProcessPorts) {
63         foreach
64         ($proxyProcessPorts as $proxyProcessPortKey =>
65         $proxyProcessPort) {
```

```

112 -             if ($this-
>_checkPort($firewallIp, $proxyProcessPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
113
$firewallPorts[$proxyProcessName][] = $proxyProcessPort;
114             }
115         }
116     }
117
118     $this->
>_applyFirewallRules($firewallPorts);
119
        unlink($processId);
120         return true;
121     }
122
123 -     /**
124 -      * DNS redundancy health checks and process
recovery
125 -      *
126 -      * @return boolean
127 -      */
128     protected function _checkDns() {
129         $basePath = $this->server['data']
['settings']['paths']['base'];
130         $processId = $this->server['data']
['settings']['paths']['process_ids'] . 'dns.pid';
131
132         if (file_exists($processId)) {
133             $lastRan =
file_get_contents($processId);
134
135             if ($lastRan >
strtotime('-2 minutes')) {
136                 return false;
137             }
138         }
139
140         if (file_exists($processId)) {
141             unlink($processId);
142         }
143
144         file_put_contents($processId,
time());
145
        $this->server =
json_decode(file_get_contents($this->server['data']
['settings']['paths']['cache'] . 'serverData'), true);
146         array_shift($this->server['data']
['dns_process_source_ips']);
147
148         if (empty($this->server['data']
['dns_process_source_ips'])) {
149             return false;
150         }
151
152         $dnsIps = array_values($this->
server['data']['dns_process_source_ips']);
153
154         foreach ($dnsIps as $dnsIpKey =>
$dnsIp) {
155             $processName = $dnsIpKey
== 0 ? 'named' : 'named-redundant' . $dnsIpKey;

```

```

113 +             if ($this-
>_verifyPort($firewallIp, $proxyProcessPort, $this-
>server['data']['proxy_configurations'][$proxyProcessName]
['protocol'])) {
114
$firewallPorts[$proxyProcessName][] = $proxyProcessPort;
115             }
116         }
117     }
118
119     $this->
>_applyFirewallRules($firewallPorts);
120 +
        $this->_verifyDns();
121         unlink($processId);
122         return true;
123     }
124

```

```

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 -
}

```

```

191 -
192 -
193 -                                     sleep(1);
194 -                                     $this-
>_checkDns();
195 -                                     }
196 -                                     }
197 -
198 -                                     if (file_exists($processId)) {
199 -                                         unlink($processId);
200 -                                     }
201 -
202 -                                     return true;
203 -                                     }
204 -
205 - /**
206 -  * Check HTTP and SOCKS ports
207 -  *
208 -  * @param string $ip Proxy IP
209 -  * @param string $port Proxy port
210 -  * @param string $protocol Proxy protocol
211 -  * @param integer $integer Request timeout
212 -  *
213 -  * @return boolean $alive True if port is active,
false if refusing connections
214 -  */
215 -     protected function _checkPort($ip, $port,
$protocol, $timeout = 5) {
216 -         $response = false;
217 -
218 -         switch ($protocol) {
219 -             case 'http':
220 -                 $response =
shell_exec('curl -I -s -x ' . $ip . ':' . $port . '
http://squid -v --connect-timeout ' . $timeout . ' --max-
time ' . $timeout);
221 -
222 -                 if ($this-
>_strpos(strtolower($response), array(
223 -                     '407
proxy',
224 -                     '403
forbidden',
225 -                     ' 503 ',
226 -                     ' timed
out '
227 -                 )) !== false) {
228 -                     $response
= true;
229 -                 }
230 -
231 -                 break;
232 -             case 'socks':
233 -                 exec('curl --
socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v
--connect-timeout ' . $timeout . ' --max-time ' . $timeout
. ' 2>&1', $socksResponse);
234 -                 $socksResponse =
end($socksResponse);
235 -                 $response =
(strpos(strtolower($socksResponse), 'empty reply ') !==

```

```

false);
236 -                                     break;
237 -                                     }
238 -
239 -                                     return $response;
240 -                                     }
241 -
242 - /**
243 -  * Apply firewall rules
244 -  *
497 -                                     sleep($delayEnd);
498 -                                     }
499 -
500 -                                     return;
501 -                                     }
502 -
503 - /**
504 -  * Initiate processes
505 -  *
506 -  * @param string $processName Process name
507 -  *
508 -  * @return boolean $status
509 -  */
510 -     public function start($processName) {
511 -         switch ($processName) {
512 -
513 -             case
514 -             'apply_reconfiguration':
515 -                                     $status = $this->_applyReconfiguration();
516 -                                     break;
517 -             case 'check_dns':
518 -                                     $status = $this->_checkDns();
519 -                                     break;
520 -
521 -             }
522 -
523 -             return $status;
524 -
525 -         }
526 -
527 -         /**
528 -         return false;
529 -         }

```

```

125 - /**
126 -  * Apply firewall rules
127 -  *
380 -                                     sleep($delayEnd);
381 -                                     }
382 -
383 +                                     $this->_verifyDns();
384 +                                     return;
385 +                                     }
386 -
387 - /**
388 +  * Start reconfiguration
389 +  *
390 +  * @return boolean $response
391 -  */
392 +     public function start() {
393 +         $response = $this->_applyReconfiguration();
394 +         return $response;
395 -     }
396 -
397 - /**
398 -         return false;
399 -     }
400 -
401 + /**
402 +  * DNS redundancy health checks and process
403 +  * recovery
404 +  *
405 +  * @return boolean
406 +  */
407 +     protected function _verifyDns() {
408 +         if (empty($this->server['data']
409 +         ['dns_process_source_ips'])) {
410 +             return false;
411 +         }
412 +
413 +         $dnsIps = array_values($this->server['data']['dns_process_source_ips']);
414 +         $basePath = $this->server['data']
415 +         ['settings']['paths']['base'];
416 +
417 +         foreach ($dnsIps as $dnsIpKey =>

```

```

$dnsIp) {
434 +                                     $processName = $dnsIpKey
== 0 ? 'named' : 'named-redundant' . $dnsIpKey;
435 +                                     $dnsResponse = array();
436 +                                     exec('dig +time=2 +tries=1
proxies @' . $dnsIp . ' 2>&1', $dnsResponse);
437 +
438 +                                     if (
439 +
!empty($dnsResponse[3]) &&
440 +
strpos(strtolower($dnsResponse[3]), 'got answer') ==
false
441 +                                     ) {
442 +                                     $dnsProcesses =
array();
443 +                                     exec('ps $(pgrep
named) 2>&1', $dnsProcesses);
444 +
445 +                                     if
(!empty($dnsProcesses)) {
446 +                                     foreach
($dnsProcesses as $dnsProcess) {
447 +
$dnsProcess = array_map('strtolower', array_map('trim',
array_values(array_filter(explode(' ', $dnsProcess))));
448 +
449 +                                     if
(
450 +
!empty($dnsProcess[0]) &&
451 +
is_numeric($dnsProcess[0]) &&
452 +
in_array('/usr/sbin/' . $processName, $dnsProcess)
453 +                                     )
{
454 +
$killProcesses = array();
455 +
$shellCommands = array(
456 +
'#' . $this->server['data']['server_configuration']
['shell'],
457 +
'kill -9 ' . trim($dnsProcess[0])
458 +
);
459 +
460 +
if (file_exists($basePath . 'dns.sh')) {
461 +
unlink($basePath . 'dns.sh');
462 +
}
463 +
464 +
file_put_contents($basePath . 'dns.sh', implode("\n",
$shellCommands));
465 +
shell_exec('chmod +x ' . $basePath . 'dns.sh');
466 +
shell_exec($basePath . './dns.sh');

```

```

467 +
468 +
469 +
470 +
471 +
    shell_exec('service ' . str_replace('named', 'bind9',
    $processName) . ' start');
472 +
    sleep(1);
473 +
    $this->_verifyDns();
474 +
    }
475 +
    }
476 +
477 +
    return true;
478 +
    }
479 +
480 +
    /**
481 +
    * Check HTTP and SOCKS ports
482 +
    *
483 +
    * @param string $ip Proxy IP
484 +
    * @param string $port Proxy port
485 +
    * @param string $protocol Proxy protocol
486 +
    * @param integer $integer Request timeout
487 +
    *
488 +
    * @return boolean $alive True if port is active,
    false if refusing connections
489 +
    */
490 +
    protected function _verifyPort($ip, $port,
    $protocol, $timeout = 5) {
491 +
    $response = false;
492 +
493 +
    switch ($protocol) {
494 +
    case 'http':
495 +
    $response =
    shell_exec('curl -I -s -x ' . $ip . ':' . $port . '
    http://squid -v --connect-timeout ' . $timeout . ' --max-
    time ' . $timeout);
496 +
497 +
    if ($this->_strpos(strtolower($response), array(
498 +
    '407
    proxy',
499 +
    '403
    forbidden',
500 +
    ' 503 ',
501 +
    ' timed
    out '
502 +
    )) !== false) {
503 +
    $response
    = true;
504 +
    }
505 +
506 +
    break;
507 +
    case 'socks':
508 +
    exec('curl --
    socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v
    --connect-timeout ' . $timeout . ' --max-time ' . $timeout
    . ' 2>&1', $socksResponse);
509 +
    $socksResponse =
    end($socksResponse);
510 +
    $response =
    (strpos(strtolower($socksResponse), 'empty reply ') !==
    false);

```

2/20/2020

Add frequent DNS process verification and recovery in reconfig · parsonsbots/dynamic-proxy-node-reconfiguration@397d456

		511	+		break;
		512	+		}
		513	+		
		514	+		return \$response;
		515	+		}
		516	+		
546		517		}	
547	?>	518	?>		

0 comments on commit 397d456