

Refactor progress for IPBot proxy server reconfiguration API

master

parsonsbots committed 4 minutes ago

1 parent 4713d44

commit 59cc924570c914cf50b79255ec1f35db200f45c2

Showing 2 changed files with 197 additions and 345 deletions.

Unified

Split

31 README.md

25

26

27

28

29

30

31

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

- Without installing custom TCP congestion control algorithms

- Without opening port hijacking vulnerabilities with SO\_REUSEPORT, shared IPs, etc

- Without using excessive resources and large anycast server clusters

- Without integrating external proxy authentication methods (once API data is retrieved)

[Full explanation coming soon]

ini\_set('max\_execution\_time', 595); // Set time limit to 10 minutes for reconfiguration

require\_once('dynamic\_reconfiguration.php');

- \$api = 'https://example.com/gateways/api.json';

- \$processes = array(

'http' => array(

array('80', '8888', '55555'), //

Main process ports

- array('4444', '7777'),

- array('55540', '55541'),

- array('55546', '55547'),

- array('55520', '55521'),

- array('55524', '55525'),

- array('55526', '55527'),

- array('55528', '55529'),

- array('55530', '55531'),

- array('55532', '55533'),

- array('55534', '55535')

),

- 'socks' => array(

array('1085'), // Main process

ports

- array('1090')

)

);

- \$sshPorts = array('22');

- \$shell = '/bin/bash';

// Initiate dynamic proxy reconfiguration

- \$dynamicProxyReconfiguration = new DynamicProxyReconfiguration(\$api, \$processes, \$shell, \$sshPorts);

// Reconfigure all proxy processes

\$dynamicProxyReconfiguration->start('apply\_reconfiguration');

25

26

27

28

29

30

31

42

43

44

45

46

47

48

49

50

51

52

53

54

- Without installing custom TCP congestion control algorithms

- Without opening port hijacking vulnerabilities with SO\_REUSEPORT, shared IPs, etc

+ - Without using excessive resources or requiring large anycast server clusters

- Without integrating external proxy authentication methods (once API data is retrieved)

[Full explanation coming soon]

ini\_set('max\_execution\_time', 595); // Set time limit to 10 minutes for reconfiguration

require\_once('dynamic\_reconfiguration.php');

+ \$apiUrl = 'https://ipbot.com/api/servers';

+ \$sshPorts = array(

'22'

);

// Initiate dynamic proxy reconfiguration

+ \$dynamicProxyReconfiguration = new DynamicProxyReconfiguration(\$apiUrl, \$sshPorts);


// Reconfigure all proxy processes

\$dynamicProxyReconfiguration->start('apply\_reconfiguration');

https://github.com/parsonsbots/dynamic-proxy-node-reconfiguration/commit/59cc924570c914cf50b79255ec1f35db200f45c2

1/16

<pre> 73 74 - // Refresh firewall 75 - \$dynamicProxyReconfiguration-     &gt;start('apply_firewall'); 76 - 77 78 ## Coming Soon 79 * 1.0.1 </pre>	<pre> 55 56 57 ## Coming Soon 58 * 1.0.1 </pre>
--	---

511 ■■■■ dynamic\_reconfiguration.php 

<pre> ... @@ -1,281 +1,106 @@ 1  &lt;?php 2      class DynamicProxyReconfiguration { 3 4 -         public \$api; 5 -         public \$processes; 6 -         public \$shell; 7 -         public \$sshPorts; 8 9 -         public function __construct(\$api, 10 \$processes, \$shell, \$sshPorts) { 11 -             \$this-&gt;api = \$api; 12 -             \$this-&gt;processes = \$processes; 13 -             \$this-&gt;shell = \$shell; 14 -             \$this-&gt;sshPorts = \$sshPorts; 15         } 16 17 -         /** 18 -          * Apply firewall 19 -          * 20 -          * @return boolean 21 -          */ 22 -         protected function _applyFirewall() { 23 -             if 24 (file_exists('/scripts/pid/reconfigure.pid')) { 25 -                 return false; 26 -             } 27 28 -             \$this-&gt;gatewaysData = 29 json_decode(file_get_contents('/scripts/cache/gatewaysData 30 '), true); 31 32 -             if (empty(\$this- 33 &gt;gatewaysData['data']['proxy_ips'])) { 34 -                 return false; 35 -             } 36 37 -             \$overridePorts = array( 38 -                 'http' =&gt; \$this- 39 &gt;processes['http'] 40 -             ); 41 -             \$firewallRules = \$this- 42 &gt;_configureFirewallRules(false, \$overridePorts); 43 -             \$this- 44 &gt;_applyFirewallRules(\$firewallRules, 'elastic'); 45         } 46 47 -         /** 48 -          * Apply firewall rules 49 -          * 50 -          * @param array \$firewallRules Firewall rules 51 -          * @param string \$ruleSet Firewall rule set </pre>	<pre> 1  &lt;?php 2      class DynamicProxyReconfiguration { 3 4 +         public \$apiUrl; 5 6         public \$sshPorts; 7 8 +         public function __construct(\$apiUrl, 9 \$sshPorts) { 10 +             \$this-&gt;apiUrl = \$apiUrl; 11 12             \$this-&gt;sshPorts = \$sshPorts; 13         } </pre>
--	---

```

44 -      *
45 -      * @return
46 -      */
47 -      protected function
48 _applyFirewallRules($firewallRules, $ruleSet) {
49 -
50     unlink('/scripts/iptables/iptables-' . $ruleSet);
51     touch('/scripts/iptables/iptables-
52 ' . $ruleSet);
53 -
54     foreach ($firewallRules as
55 $ruleChunk) {
56 -
57         $saveRules = implode("\n",
58 $ruleChunk);
59 -
60         shell_exec('echo "' .
61 $saveRules . '" >> /scripts/iptables/iptables-' .
62 $ruleSet);
63 -
64     }
65 -
66     shell_exec('iptables-restore <
67 /scripts/iptables/iptables-' . $ruleSet);
68 -
69     return;
70 -
71 }
72 -
73 /**
74  * Apply seamless processes reconfiguration
75  *
76  * @return boolean
77  */
78 protected function _applyReconfiguration()
79 {
80 -
81     $this->_createDirectories();
82 -
83     if
84 (file_exists('/scripts/pid/reconfigure.pid')) {
85 -
86         $lastRan =
87 file_get_contents('/scripts/pid/reconfigure.pid');
88 -
89         if ($lastRan <
90 strtotime('-15 minutes')) {
91 -
92             unlink('/scripts/pid/reconfigure.pid');
93 -
94         } else {
95             return false;
96         }
97     }
98 -
99     $gatewaysJsonData =
100 shell_exec("curl " . $this->api . " --connect-timeout
101 30");
102 -
103     $this->gatewaysData =
104 json_decode($gatewaysJsonData, true);
105 -

```

```

12 /**
13  * Apply seamless processes reconfiguration
14  *
15  * @return boolean
16  */
17 protected function _applyReconfiguration()
18 {
19 +
20     $serverJsonData = shell_exec('curl
21 ' . $this->apiUrl . ' --connect-timeout 10');
22 +
23     $this->server =
24 json_decode($serverJsonData, true);
25 +
26     $firewallIps = $this-
27 >server['data']['proxy_ips'];
28 +
29     $firewallPorts = array();
30 +
31     $processId = $this->server['data']
32 ['settings']['paths']['process_ids'] . 'reconfigure.pid';
33 -
34     if (file_exists($processId)) {
35 +
36         $lastRan =
37 file_get_contents($processId);
38 -
39         if ($lastRan >
40 strtotime('-10 minutes')) {
41 -
42             return false;
43         }
44 +
45         unlink($processId);
46     }
47 -
48     $this->_createDirectories();
49 -
50     $this->_createFiles();
51 -

```

```

81         if (
82 -             empty($this->gatewaysData['data']) ||
83 -             !is_dir('/etc/squid3')
84         ) {
85 -             file_put_contents('/scripts/errors/api-error-' . time(),
86 -                             $this->gatewaysData);
87 -             unlink('/scripts/pid/reconfigure.pid');
88 -             return false;
89         }
90 -         file_put_contents('/scripts/cache/gatewaysData',
91 -                         $gatewaysJsonData);
92 -         file_put_contents('/scripts/pid/reconfigure.pid', time());
93 -         if (!empty($this->gatewaysData['data']['server'])) {
94 -             file_put_contents('/etc/sysctl.conf', implode("\n", $this->gatewaysData['data']['server']));
95 -             shell_exec('sysctl -p');
96         }
97 -
98 -         $proxyIps = $this->gatewaysData['data']['proxy_ips'];
99 -
100 -         if (empty($proxyIps)) {
101 -             unlink('/scripts/pid/reconfigure.pid');
102 -             return false;
103         }
104 -
105 -         shell_exec('rm -rf
106 - /etc/squid3/users/');
107 -         shell_exec('mkdir -m 777
108 - /etc/squid3/users/');
109 -
110 -         if (!empty($this->gatewaysData['data']['http']['files'])) {
111 -             foreach ($this->gatewaysData['data']['http']['files'] as $file) {
112 -                 shell_exec('mkdir
113 - m 777 ' . str_replace(array('s.txt', 'd.txt'), '',
114 - $file['path']));
115 -                 shell_exec('touch
116 - ' . $file['path']);

```

```

37         if (
38 +             empty($this->server['data']) ||
39 +             empty($firewallProxyIps)
40         ) {
41 +             unlink($processId);
42 -             return false;
43         }
44 -
45 +             file_put_contents($processId,
46 +                             time());
47 +             file_put_contents($this->server['data']['settings']['paths']['cache'] .
48 +                             'serverData', $serverJsonData);
49 +             if (
50 +                 !empty($this->server['data']['server_configuration']['kernel']) &&
51 +                 !empty($this->server['data']['server_configuration']['kernel']
52 +                 ['options']) &&
53 +                 !empty($this->server['data']['server_configuration']['kernel']['path'])
54 +                 &&
55 +                 !empty($this->server['data']['server_configuration']['kernel']['save'])
56 +                 &&
57 +             ) {
58 +                 file_put_contents($this->server['data']['server_configuration']['path'],
59 +                                 implode("\n", $this->server['data']
60 +                                 ['server_configuration']['options']));
61 +                 shell_exec($this->server['data']['server_configuration']['save']);
62         }
63 -
64 +             $this->server['data']
65 +             ['forwarding_ports'] = array();
66 -
67 +             foreach ($this->server['data']
68 +             ['proxy_process_ports'] as $proxyProcessName =>
69 +             $proxyProcessPorts) {
70 +                 foreach
71 +                 ($proxyProcessPorts['primary'] as $proxyProcessPortKey =>
72 +                 $proxyProcessPort) {
73 +                     $this->server['data']['forwarding_ports'][$proxyProcessName][] =
74 +                     $proxyProcessPort;

```

```

112 -
113     file_put_contents($file['path'], $file['contents']);
114     }
115 }
116
117     $configurations = array();
118     $firewallRules = $this-
119 >_configureFirewallRules(true);
120     $processes = $this-
121 >processes['http'];
122     $forwardingProcessChunks =
123 array_chunk($processes, round(count($processes) / 2),
124 false);
125
126     unset($forwardingProcessChunks[0]
127 [0]);
128
129     foreach ($forwardingProcessChunks
130 as $forwardingProcessChunkKey => $forwardingProcessChunk)
131 {
132     $forwardingAcls = $this-
133 >gatewaysData['data']['http']['acls'];
134     $this->forwardingPortIndex
135 = $this->staticPortIndex = 0;
136
137     $forwardingPorts =
138 array();
139
140     foreach
141 ($forwardingProcessChunk as $forwardingProcessPorts) {
142
143         foreach
144 ($forwardingProcessPorts as $forwardingProcessPort) {
145
146             $forwardingPorts[] = $forwardingProcessPort;
147         }
148     }
149
150     $this->forwardingPorts =
151 array(
152
153         $forwardingPorts
154
155     );
156
157     if
158 (strpos($forwardingAcls, '[forwarding_port]') !== false) {
159
160         $this-
161 >forwardingPorts = array_chunk($forwardingPorts,
162 count($forwardingPorts) / 2);
163     }
164
165     $forwardingAcls =
166 preg_replace_callback('/\[forwarding_port\]/', function ()
167 {
168     if (empty($this-
169 >forwardingPorts[1][$this->forwardingPortIndex])) {
170
171         $this-
172 >forwardingPortIndex = 0;
173     }
174
175     $forwardingPort =
176 $this->forwardingPorts[1][$this->forwardingPortIndex];

```

```

63     }
64
65     foreach
66 ($proxyProcessPorts['secondary'] as $proxyProcessPortKey
67 => $proxyProcessPorts) {
68
69         foreach
70 ($proxyProcessPorts as $proxyProcessPort) {
71
72             $this-
73 >server['data']['forwarding_ports'][$proxyProcessName][] =
74 $proxyProcessPort;
75
76             if ($this-
77 >_checkPort(key($firewallProxyIps), $proxyProcessPort,
78 $protocol)) {
79
80                 $firewallPorts[$proxyProcessName][] = $proxyProcessPort;
81             }
82         }
83     }
84
85     $this->server['data']
86 ['forwarding_ports'][$proxyProcessName] =
87 array_unique($this->server['data']['forwarding_ports']
88 [$proxyProcessName]);
89
90     shuffle($this-
91 >server['data']['forwarding_ports'][$proxyProcessName]);

```

```

147 -                                     $this-
>forwardingPortIndex++;
148 -                                     return
    $forwardingPort;
149 -                                     }, $forwardingAcls);
150 -                                     $forwardingAcls =
preg_replace_callback('/\[static_port\]/', function () {
151 -                                     if (empty($this-
>forwardingPorts[0][$this->staticPortIndex])) {
152 -                                     $this-
>staticPortIndex = 0;
153 -                                     }
154 -
155 -                                     $forwardingPort =
    $this->forwardingPorts[0][$this->staticPortIndex];
156 -                                     $this-
>staticPortIndex++;
157 -                                     return
    $forwardingPort;
158 -                                     }, $forwardingAcls);
159 -                                     $aclFilepath =
    '/etc/squid3/acls' . $forwardingProcessChunkKey . '.conf';
160 -                                     $configurations[] =
    str_replace('[acl_filepath]', $aclFilepath, $this-
>gatewaysData['data']['http']['configuration']);
161 -                                     shell_exec('rm ' .
    $aclFilepath);
162 -                                     shell_exec('touch ' .
    $aclFilepath);
163 -
    file_put_contents($aclFilepath, $forwardingAcls);
164 -                                     }
165 -
166 -                                     shell_exec('htpasswd -cb
    /etc/squid3/passwords default default');
167 -                                     shell_exec('htpasswd -D
    /etc/squid3/passwords default');
168 -
169 -                                     if (!empty($this-
>gatewaysData['data']['http']['users'])) {
170 -                                     foreach ($this-
>gatewaysData['data']['http']['users'] as $username =>
    $password) {
171 -
        shell_exec('htpasswd -b /etc/squid3/passwords ' .
    $username . ' ' . $password);
172 -                                     }
173 -
    }
174 -
175 -                                     $this-
>_applyFirewallRules($firewallRules, 'redundant');
176 -
177 -                                     if (
178 -                                     !empty($this-
>gatewaysData['data']['socks']) &&
179 -                                     !empty($this-
>processes['socks'][0])
180 -                                     ) {

```

```

77                                     }
78
79 +                                     $this-
>_applyFirewallRules($firewallPorts);
80 +                                     $firewallPorts = array();
81
82 +                                     foreach ($this->server['data']
    ['proxy_processes'] as $proxyProcessName =>
    $proxyProcesses) {
83 +                                     $this-
>_reconfigure($proxyProcesses[$proxyProcessName][0]);
84 +                                     unset($this-
>server['data']['proxy_processes'][$proxyProcessName][0]);
85 +                                     $this->server['data']
    ['proxy_processes'][$proxyProcessName] =

```

```

181 -             $this->_reconfigure(
182 -                 'socks',
183 -                 'service 3proxy
start',
184 -                 '3proxy.cfg',
185 -
186 -                 0,
187 -                 20,
188 -
189 -                 '/usr/local/etc/3proxy/3proxy.pid',
190 -                 $this->gatewaysData['data']['socks']
191 -             );
192 -         }
193 -
194 -         $this->_reconfigure(
195 -             'http',
196 -
197 -             'squid3 start',
198 -
199 -             'squid3',
200 -             '/var/run/squid3.pid',
201 -             0,
202 -             75,
203 -             '/etc/squid3/squid.conf',
204 -             str_replace('[pid]',
205 -             'pid_filename /var/run/squid3.pid', str_replace('[ports]',
206 -             'http_port ' . implode("\n" . 'http_port ', $this->processes['http'][0]), $configurations[0]))
207 -         );
208 -         unset($processes[0]);
209 -         $processChunks =
array_chunk($processes, round(count($processes) / 2),
true);
210 -
211 -         foreach ($processChunks as
$processChunkKey => $processChunk) {
212 -             $activeProcesses =
array_keys($processes);
213 -             reset($processChunk);
214 -             $processStart =
key($processChunk);
215 -             end($processChunk);
216 -             $processEnd =
key($processChunk);
217 -             $processRange =
range($processStart, $processEnd);
218 -
219 -             foreach ($processRange as
$process) {
220 -                 unset($activeProcesses[$process - 1]);
221 -             }
222 -
223 -             $activePorts = array();
224 -
225 -             foreach ($processes as
$processKey => $process) {

```

```

array_chunk(proxyProcesses, round(count($proxyProcesses) /
2), true);

```

```

86 -         }
87 -
88 +         foreach (array(0, 1) as $value) {
89 +             foreach ($this->server['data']['proxy_process_ports'] as
$proxyProcessName => $proxyProcessPorts) {
90 +                 $firewallPorts[$proxyProcessName] =
$proxyProcessPorts['primary'];
91 -
92 +                 foreach
($proxyProcessPorts['secondary'][$value ? 1 : 0]) as

```

```

221 -                                     if
(in_array($processKey, $activeProcesses)) {
222 -
$activePorts = array_merge($activePorts,
$processes[$processKey]);
223 -                                     }
224 -                                     }
225 -
226 -                                     $overridePorts = array(
227 -                                     'http' =>
array(array_merge(array(
228 -                                     '80',
229 -                                     '8888',
230 -                                     '55555'
231 -                                     )), $activePorts),
232 -                                     'socks' => array(
233 -                                     array(
234 -                                     '1090'
235 -                                     )
236 -                                     )
237 -                                     );
238 -
239 -                                     $firewallRules = $this->
_configureFirewallRules(false, $overridePorts);
240 -                                     $this->
_applyFirewallRules($firewallRules, 'elastic');
241 -
242 -                                     foreach ($processRange as
$process) {
243 -                                     $this->
_reconfigure(
244 -                                     'http',
245 -                                     'squid3-
redundant' . $process . ' start -f /etc/squid3/squid-
redundant' . $process . '.conf',
246 -                                     'squid3-
redundant' . $process,
247 -                                     '/var/run/squid-redundant' . $process . '.pid',
248 -                                     0,
249 -                                     0,
250 -                                     '/etc/squid3/squid-redundant' . $process . '.conf',
251 -                                     str_replace('[pid]', 'pid_filename /var/run/squid-
redundant' . $process . '.pid', str_replace('[ports]',
'http_port ' . implode("\n" . 'http_port ', $this->
processes['http'][$process])),
$configurations[$processChunkKey]))
252 -                                     );
253 -                                     }
254 -
255 -                                     sleep(75);
256 -                                     }
257 -
258 -                                     $overridePorts = array(
259 -                                     'http' => $this->

```

```

93 + $proxyProcessPort) {
+ $firewallPorts[$proxyProcessName][] = $proxyProcessPort;
94 +                                     }
95 +                                     }
96 +
97 +                                     foreach ($this->
server['data']['proxy_processes'] as $proxyProcessName =>
$splitProxyProcesses) {
98 +                                     foreach
($splitProxyProcesses[$$proxyProcessName][($value ? 0 :
1)] as $proxyProcess) {
99 +                                     $this->
_reconfigure($proxyProcess);
100 +                                     }
101 +                                     }
102 +                                     }
103 +

```



```

>processes['http']
260 -         );
261 -         $firewallRules = $this-
>_configureFirewallRules(false, $overridePorts);
262 -         $this-
>_applyFirewallRules($firewallRules, 'elastic');
263 -
264 -         if (!empty($this-
>gatewaysData['data']['socks-redundant'])) {
265 -             $this->_reconfigure(
266 -                 'socks',
267 -                 'service 3proxy-
redundant start',
268 -                 '3proxy-
redundant.cfg',
269 -                 '/usr/local/etc/3proxy/3proxy-redundant.pid',
270 -                 0,
271 -                 40,
272 -                 '/usr/local/etc/3proxy/3proxy-redundant.cfg',
273 -                 $this-
>gatewaysData['data']['socks-redundant']
274 -             );
275 -         }
276 -
277 -         $this-
>_applyFirewallRules($firewallRules, 'elastic');
278 -
unlink('/scripts/pid/reconfigure.pid');
279 -         return true;
280 -     }
281
282     *
283     * @return boolean
284     */
285
286     protected function _checkDNS() {
287 -         if
288         (file_exists('/scripts/pid/dns.pid')) {
289 -             $lastRan =
290             file_get_contents('/scripts/pid/dns.pid');
291 -             if ($lastRan <
292             strtotime('-2 minutes')) {
293 -                 unlink('/scripts/pid/dns.pid');
294 -             } else {
295 -                 return false;
296 -             }
297 -         }
298 -
299 -         file_put_contents('/scripts/pid/dns.pid', time());
300 -         $this->gatewaysData =
301         json_decode(file_get_contents('/scripts/cache/gatewaysData
'), true);
302 -         array_shift($this-
>gatewaysData['data']['dns_ips']);

```

```

104 -         return true;
105 -     }
106
107     *
108     * @return boolean
109     */
110
111     + protected function _checkDns() {
112     +         $basePath = $this->server['data']
113     +         ['settings']['paths']['base'];
114     +         $processId = $this->server['data']
115     +         ['settings']['paths']['process_ids'] . 'dns.pid';
116     +         if (file_exists($processId)) {
117     +             $lastRan =
118     +             file_get_contents($processId);
119     +             if ($lastRan >
120     +             strtotime('-2 minutes')) {
121     +                 return false;
122     +             }
123     +         }
124     +         unlink($processId);
125     +         file_put_contents($processId,
126     +         time());
127     +         $this->server =
128     +         json_decode(file_get_contents($this->server['data']
129     +         ['settings']['paths']['cache'] . 'serverData'), true);
130     +         array_shift($this->server['data']
131     +         ['dns_process_source_ips']);

```

```

301
302 -             if (empty($this->gatewaysData['data']['dns_ips'])) {
303                 return false;
304             }
305
306 -             $dnsIps = array_values($this->gatewaysData['data']['dns_ips']);
307
308             foreach ($dnsIps as $dnsIpKey => $dnsIp) {
309                 $processName = $dnsIpKey
310 == 0 ? 'named' : 'named-redundant' . $dnsIpKey;
311
312                 {
313
314                 $killProcesses = array();
315
316                 $shellCommands = array(
317 -                     '#!' . $this->shell,
318
319                 'kill -9 ' . trim($dnsProcess[0])
320
321                 );
322 -                 unlink('/scripts/dns.sh');
323 -                 file_put_contents('/scripts/dns.sh', implode("\n", $shellCommands));
324 -                 shell_exec('chmod +x /scripts/dns.sh');
325 -                 shell_exec('/scripts/.dns.sh');
326
327                 }
328
329                 }
330
331                 }
332
333                 shell_exec('service ' . str_replace('named', 'bind9', $processName) . ' start');
334
335                 sleep(1);
336 -                 unlink('/scripts/pid/dns.pid');
337 -                 $this->_checkDNS();
338
339                 }
340
341                 }
342
343                 unlink('/scripts/pid/dns.pid');
344                 return true;
345             }
346
347
348
349 -             *
350             * @param string $ip Proxy IP
351             * @param string $port Proxy port
352             * @param string $port Proxy protocol (http or socks)
353             * @param integer $integer Request timeout
354             *
355             * @return boolean $alive True if port is active, false if refusing connections

```

```

128
129 +             if (empty($this->server['data']['dns_process_source_ips'])) {
130                 return false;
131             }
132
133 +             $dnsIps = array_values($this->server['data']['dns_process_source_ips']);
134
135             foreach ($dnsIps as $dnsIpKey => $dnsIp) {
136                 $processName = $dnsIpKey
137 == 0 ? 'named' : 'named-redundant' . $dnsIpKey;
138
139                 {
140
141                 $killProcesses = array();
142
143                 $shellCommands = array(
144 +                     '#!' . $this->server['data']['server_configuration']['shell'],
145
146                 'kill -9 ' . trim($dnsProcess[0])
147
148                 );
149 -                 unlink($basePath . 'dns.sh');
150 +                 file_put_contents($basePath . 'dns.sh', implode("\n", $shellCommands));
151 +                 shell_exec('chmod +x ' . $basePath . 'dns.sh');
152 +                 shell_exec($basePath . './dns.sh');
153
154                 }
155
156                 }
157
158                 }
159
160                 shell_exec('service ' . str_replace('named', 'bind9', $processName) . ' start');
161
162                 sleep(1);
163 +                 $this->_checkDns();
164
165                 }
166
167                 }
168
169                 }
170
171                 unlink($processId);
172                 return true;
173             }
174
175
176
177
178
179
180
181             *
182             * @param string $ip Proxy IP
183             * @param string $port Proxy port
184 +             * @param string $protocol Proxy protocol
185             * @param integer $integer Request timeout
186             *
187             * @return boolean $alive True if port is active, false if refusing connections

```

```

362      */
363      protected function _checkPort($ip, $port,
$protocol, $timeout = 5) {
364      -          $alive = false;
365
366          switch ($protocol) {
367              case 'http':
373                  ' 503 ',
374                  ' timed
out '
375                  )) != false) {
376      -          $alive =
true;
377                  }
378
379                  break;
380              case 'socks':
381                  exec('curl --
socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v
--connect-timeout ' . $timeout . ' --max-time ' . $timeout
. ' 2>&1', $socksResponse);
382                  $socksResponse =
end($socksResponse);
383      -          $alive =
(strpos(strtolower($socksResponse), 'empty reply ') !=
false);
384                  break;
385              }
386      -          return $alive;
387      }
388
389      /**
390      -      * Configure firewall rules
391      *
392      -      * @param boolean $redundant Route to redundant
process ports only if true, all ports if false
393      -      * @param array $overridePorts Override default
base ports
394      *
395      -      * @return array $rules Firewall rules
396      */
397      protected function
_configureFirewallRules($redundant = false, $overridePorts
= array()) {
398      -          $rules = array(
399              '*filter',
400              ':INPUT ACCEPT [0:0]',
401              ':FORWARD ACCEPT [0:0]',
402              ':OUTPUT ACCEPT [0:0]',
403              '-A INPUT -p icmp -m
hashlimit --hashlimit-name icmp --hashlimit-mode srcip --
hashlimit 1/second --hashlimit-burst 2 -j ACCEPT'
404              );
405
406      -          if (
407              !empty($this->sshPorts) &&
is_array($this->sshPorts)
408              ) {
409                  foreach ($this->sshPorts
as $sshPort) {
410                      if
411                      (is_numeric($sshPort)) {

```

```

188      */
189      protected function _checkPort($ip, $port,
$protocol, $timeout = 5) {
190      +          $response = false;
191
192          switch ($protocol) {
193              case 'http':
199                  ' 503 ',
200                  ' timed
out '
201                  )) != false) {
202      +          $response =
true;
203                  }
204
205                  break;
206              case 'socks':
207                  exec('curl --
socks5-hostname ' . $ip . ':' . $port . ' http://socks/ -v
--connect-timeout ' . $timeout . ' --max-time ' . $timeout
. ' 2>&1', $socksResponse);
208                  $socksResponse =
end($socksResponse);
209      +          $response =
(strpos(strtolower($socksResponse), 'empty reply ') !=
false);
210                  break;
211              }
212      +          return $response;
213      }
214
215      /**
216      +      * Apply firewall rules
217      *
218      +      * @param array $firewallPorts Firewall ports
219      *
220      *
221      +      * @return array $firewallRules Firewall rules
222      */
223      protected function
_applyFirewallRules($firewallPorts) {
224      +          $firewallRules = array(
225              '*filter',
226              ':INPUT ACCEPT [0:0]',
227              ':FORWARD ACCEPT [0:0]',
228              ':OUTPUT ACCEPT [0:0]',
229              '-A INPUT -p icmp -m
hashlimit --hashlimit-name icmp --hashlimit-mode srcip --
hashlimit 1/second --hashlimit-burst 2 -j ACCEPT'
230              );
231
232      +          /*if (
233              !empty($this->sshPorts) &&
is_array($this->sshPorts)
234              ) {
235                  foreach ($this->sshPorts
as $sshPort) {
236                      if
237                      (is_numeric($sshPort)) {

```

```

413 -                                     $rules[] =
'-A INPUT -p tcp -m tcp --dport ' . $sshPort . ' -m
connlimit --connlimit-above 4 --connlimit-mask 32 --
connlimit-saddr -j DROP';
414 -                                     $rules[] =
'-A INPUT -p tcp -m tcp --dport ' . $sshPort . ' -m
hashlimit --hashlimit-upto 15/hour --hashlimit-burst 3 --
hashlimit-mode srcip --hashlimit-name ssh --hashlimit-
htable-expire 500000 -j ACCEPT';
415                                     }
416                                     }
417                                     }
418
419                                     if (
420 -                                     !empty($this-
>gatewaysData['data']['firewall_filter']) &&
421 -                                     is_array($this-
>gatewaysData['data']['firewall_filter'])
422                                     ) {
423 -                                     foreach ($this-
>gatewaysData['data']['firewall_filter'] as $rule) {
424 -                                     $rules[] = $rule;
425                                     }
426                                     }
427
428 -                                     array_unshift($this-
>gatewaysData['data']['dns_ips'], '127.0.0.1');
429 -                                     $dnsIps = array_unique($this-
>gatewaysData['data']['dns_ips']);
430 -                                     $rules[] = '-A OUTPUT -d ' .
implode(',', $dnsIps) . ' -p udp -m udp -j ACCEPT';
431 -                                     $rules[] = '-A OUTPUT -s
127.0.0.0/24 -p udp -j DROP';
432 -                                     $rules[] = 'COMMIT';
433 -                                     $rules[] = '*nat';
434 -                                     $rules[] = ':PREROUTING ACCEPT
[0:0]';
435 -                                     $rules[] = ':INPUT ACCEPT [0:0]';
436 -                                     $rules[] = ':OUTPUT ACCEPT [0:0]';
437 -                                     $rules[] = ':POSTROUTING ACCEPT
[0:0]';
438                                     unset($dnsIps[0]);
439                                     krsort($dnsIps);
440
441                                     $loadBalancer = '-
m statistic --mode nth --every ' . $dnsIpKey . ' --packet
0 ' ;
442                                     }
443
444 -                                     $rules[] = '-A OUTPUT -d
127.0.0.1/32 -p udp -m udp --dport 53 ' . $loadBalancer .
'-j DNAT --to-destination ' . $dnsIp;
445                                     }
446
447                                     if (
448                                     $ports =
array_unique($basePorts);
449
450                                     foreach ($ports as
238 +
$firewallRules[] = '-A INPUT -p tcp -m tcp --dport ' .
$sshPort . ' -m connlimit --connlimit-above 4 --connlimit-
mask 32 --connlimit-saddr -j DROP';
239 +
$firewallRules[] = '-A INPUT -p tcp -m tcp --dport ' .
$sshPort . ' -m hashlimit --hashlimit-upto 15/hour --
hashlimit-burst 3 --hashlimit-mode srcip --hashlimit-name
ssh --hashlimit-htable-expire 500000 -j ACCEPT';
240                                     }
241                                     }
242                                     }
243
244                                     if (
245 +                                     !empty($this-
>server['data']['firewall_filter']) &&
246 +                                     is_array($this-
>server['data']['firewall_filter'])
247                                     ) {
248 +                                     foreach ($this-
>server['data']['firewall_filter'] as $rule) {
249 +                                     $firewallRules[] =
$rule;
250                                     }
251                                     }
252
253 +                                     array_unshift($this-
>server['data']['dns_ips'], '127.0.0.1');
254 +                                     $dnsIps = array_unique($this-
>server['data']['dns_ips']);
255 +                                     $firewallRules[] = '-A OUTPUT -d
127.0.0.1 ' . implode(',', $dnsIps) . ' -p udp -m udp -j
ACCEPT';
256 +                                     $firewallRules[] = '-A OUTPUT -s
127.0.0.0/24 -p udp -j DROP';
257 +                                     $firewallRules[] = 'COMMIT';
258 +                                     $firewallRules[] = '*nat';
259 +                                     $firewallRules[] = ':PREROUTING
ACCEPT [0:0]';
260 +                                     $firewallRules[] = ':INPUT ACCEPT
[0:0]';
261 +                                     $firewallRules[] = ':OUTPUT ACCEPT
[0:0]';
262 +                                     $firewallRules[] = ':POSTROUTING
ACCEPT [0:0]';
263                                     unset($dnsIps[0]);
264                                     krsort($dnsIps);
265
266                                     $loadBalancer = '-
m statistic --mode nth --every ' . $dnsIpKey . ' --packet
0 ' ;
267                                     }
268
269 +                                     $firewallRules[] = '-A
OUTPUT -d 127.0.0.1/32 -p udp -m udp --dport 53 ' .
$loadBalancer . '-j DNAT --to-destination ' . $dnsIp;
270                                     }
271
272                                     if (
273                                     $ports =
array_unique($basePorts);
274
275                                     foreach ($ports as
300                                     $ports =
array_unique($basePorts);
301
302                                     foreach ($ports as

```

```

$portKey => $port) {
478 -                                     if ($this->
>_checkPort(key($this->gatewaysData['data']['proxy_ips']),
$port, $protocol) === false) {
479
unset($ports[$portKey]);
480                                     }
481                                     }
497
$loadBalancer = '-m statistic --mode nth --every ' .
($portKey + 1) . ' --packet 0 ';
498
}
499
500 -
$rules[] = '-A PREROUTING -p tcp -m multiport --dports ' .
$dports . ' ' . $loadBalancer . '-j DNAT --to-destination
:' . $port . ' --persistent';
501                                     }
502                                     }
503                                     }
504                                     }

505
506
507 - $rules[] = 'COMMIT';
508
509 - $rules = array_chunk($rules, 100);
509 - return $rules;
510
}
511
/**
512 - * Create writable log and cache directories
513 - *
514 - * @return
515 - */
516
protected function _createDirectories() {
517 - $directories = array(
518 -
519 -         '/scripts',
520 -
521 -         '/scripts/cache',
521 -         '/scripts/errors',
522 -
522 -         '/scripts/iptables',

```

```

$portKey => $port) {
303 +                                     if ($this->
>_checkPort(key($this->server['data']['proxy_ips']),
$port, $protocol) === false) {
304
unset($ports[$portKey]);
305                                     }
306                                     }
322
$loadBalancer = '-m statistic --mode nth --every ' .
($portKey + 1) . ' --packet 0 ';
323
}
324
325 +
$firewallRules[] = '-A PREROUTING -p tcp -m multiport --
dports ' . $dports . ' ' . $loadBalancer . '-j DNAT --to-
destination :' . $port . ' --persistent';
326                                     }
327                                     }
328                                     }
329                                     }
330 +                                     }*/
331 +
332 + $firewallRules[] = 'COMMIT';
333 + $firewallRuleChunks =
array_chunk($firewallRules, 100);
334 + $firewallRulePath = $this->
>server['data']['settings']['paths']['firewall_rules'] .
'rules';
335 + unlink($firewallRulePath);
336 + touch($firewallRulePath);
337 +
338 + foreach ($firewallRuleChunks as
$firewallRuleChunk) {
339 +                                     $saveRules = implode("\n",
$firewallRuleChunk);
340 +                                     shell_exec('echo "' .
$saveRules . '" >> ' . $firewallRulePath);
341 +                                     }
342
343 + shell_exec('iptables-restore < ' .
$firewallRulePath);
344 + return $firewallRules;
345
}
346
/**
347 + * Create directories
348 + *
349 + * @return
350 + */
351
protected function _createDirectories() {
352 + $paths = $this->server['data']
['settings']['paths'];
353 +
354 + shell_exec('rm -rf ' .
$paths['configurations'] . ' ' . $paths['rules'] . ' ' .
$paths['users']);
355 +
356 + foreach ($this->server['data']
['proxy_configurations'] as $proxyConfigurationType =>
$proxyConfiguration) {
357 +                                     $paths =

```

```

523 -             '/scripts/pid'
524 -         );

525
526 -         foreach ($directories as
$directory) {
527             if (!is_dir($directory)) {
528 -                 shell_exec('mkdir
-m 777 ' . $directory);

```

```

529             }
530         }
531
532     /**

```

```

array_merge(array_values($proxyConfiguration['paths']));
+         }
359 +
360 +         foreach ($paths as $path) {
361 +             $directory = substr($path,
0, strpos($path, '/'));
362
363             if (!is_dir($directory)) {
364 +                 shell_exec('mkdir
-m 777 -p ' . $directory);
365 +             }
366 +         }
367 +
368 +         return;
369 +     }
370 +
371 +     /**
372 +      * Create files
373 +      *
374 +      * @return
375 +      */
376 +     protected function _createFiles() {
377 +         $passwordsPath = $this-
>server['data']['settings']['paths']['passwords'];
378 +
379 +         if (!empty($this->server['data']
['files'])) {
380 +             foreach ($this-
>server['data']['files'] as $file) {
381 +                 $directory =
substr($file['path'], 0, strpos($file['path'], '/'));
382 +
383 +                 if
(!is_dir($directory)) {
384 +                     shell_exec('mkdir -m 777 -p ' . $directory);
385 +                 }
386 +
387 +                 shell_exec('touch
' . $file['path']);
388 +
389 +                 file_put_contents($file['path'], $file['contents']);
390 +             }
391 +
392 +             shell_exec('htpasswd -cb ' .
$passwordsPath . ' default default');
393 +             shell_exec('htpasswd -D ' .
$passwordsPath . ' default');
394 +
395 +             if (!empty($this->server['data']
['users'])) {
396 +                 foreach ($this-
>server['data']['users'] as $username => $password) {
397 +                     shell_exec('htpasswd -b ' . $passwordsPath . ' ' .
$username . ' ' . $password);
398 +                 }
399 +             }
400
401     /**

```

<pre> 573      * Reconfigure specific process 574      * 575      -      * @param string \$protocol Proxy protocol (http or socks) 576      -      * @param string \$startCommand Command to start process 577      -      * @param string \$processName Process name 578      -      * @param string \$processId Full path to process ID 579      -      * @param integer \$delayStart Delay at beginning of reconfiguration 580      -      * @param integer \$delayEnd Delay at end of reconfiguration 581      -      * @param string \$configurationFile Full path to process configuration file 582      -      * @param string \$configurationData Process configuration data 583      * 584      * @return 585      */ 586      -      protected function _reconfigure(\$protocol, \$startCommand, \$processName, \$processId, \$delayStart = 0, \$delayEnd = 0, \$configurationFile, \$configurationData) { 587      -          if (\$delayStart) {  sleep(\$delayStart); }  \$killProcesses = array(); \$shellCommands = array(     '#!' . \$this-&gt;shell  ); -          \$killProcesses = \$this- -&gt;_getProcessIds(\$processName, \$configurationFile);  foreach (\$killProcesses as \$killProcess) {     \$shellCommands[] = 'kill -9 ' . trim(\$killProcess); }  if (count(\$shellCommands &gt; 1)) {     unlink('/scripts/' . \$protocol . '.sh'); - file_put_contents('/scripts/' . \$protocol . '.sh', implode("\n", \$shellCommands)); 604      -          shell_exec('chmod +x </pre>	<pre> 442      * Reconfigure specific process 443      * 444      +      * @param array \$proxyProcess Proxy process data  * * @return */ 448      +      protected function _reconfigure(\$proxyProcess) { 449      +          \$basePath = \$this-&gt;server['data'] ['settings']['paths']['base']; 450      +          \$configurationPath = \$proxyProcess['paths']['configuration']; 451      +          \$delayEnd = \$proxyProcess['delays']['end']; 452      +          \$delayStart = \$proxyProcess['delays']['start']; 453      +          \$scriptFile = \$proxyProcess['protocol'] . '.sh'; 454      + 455      +          if ( 456      +              !empty(\$delayStart) &amp;&amp; 457      +              is_numeric(\$delayStart) 458      +          ) { 459      +              sleep(\$delayStart); 460      +          } 461 462      +          \$killProcesses = array(); 463      +          \$shellCommands = array( 464      +              '#!' . \$this- &gt;server['data']['server_configuration']['shell'] 465      +          ); 466      +          \$killProcesses = \$this- &gt;_getProcessIds(\$proxyProcess['name'], \$configurationPath);  foreach (\$killProcesses as \$killProcess) {     \$shellCommands[] = 'kill -9 ' . trim(\$killProcess); }  if (count(\$shellCommands &gt; 1)) {     unlink(\$basePath . \$scriptFile); + file_put_contents(\$basePath . \$scriptFile, implode("\n", \$shellCommands)); 475      +          shell_exec('chmod +x ' . </pre>
--	--

<pre> 605 /scripts/' . \$protocol . '.sh'); -                                     shell_exec('/scripts/./' . \$protocol . '.sh'); 606                                     } 607 608                                     if ( 609 -                                     !empty(\$configurationFile) &amp;&amp; 610 -                                     !empty(\$configurationData) 611                                     ) { 612 -                                     file_put_contents(\$configurationFile, \$configurationData); 613                                     } 614 615 -                                     unlink(\$processId); 616 617                                     sleep(2); 617 -                                     shell_exec(\$startCommand); 618 619 -                                     if (\$delayEnd) { 620 621                                     sleep(\$delayEnd); 622                                     } 623 624                                     /** 630                                     */ 631                                     public function start(\$processName) { 632                                     switch (\$processName) { 633 -                                     case 'apply_firewall': 634 -                                     \$status = \$this-&gt;_applyFirewall(); 635 -                                     break; 636                                     case 'apply_reconfiguration': 637                                     \$status = \$this-&gt;_applyReconfiguration(); 638                                     break; 639                                     case 'check_dns': 640 -                                     \$status = \$this-&gt;_checkDNS(); 641                                     break; 642                                     } 643 </pre>	<pre> 476 \$basePath . \$scriptFile); +                                     shell_exec(\$basePath . './' . \$scriptFile); 477                                     } 478 479                                     if ( 480 +                                     !empty(\$configurationPath) &amp;&amp; 481 +                                     !empty(\$proxyProcess['parameters']) 482                                     ) { 483 +                                     file_put_contents(\$configurationPath, \$proxyProcess['parameters']); 484                                     } 485 486 +                                     unlink(\$proxyProcess['paths'] ['process_id']); 487                                     sleep(2); 488 +                                     shell_exec(\$proxyProcess['start_command']); 489 490 +                                     if ( 491 +                                     \$delayEnd &amp;&amp; 492 +                                     is_numeric(\$delayEnd) 493 +                                     ) { 494 +                                     sleep(\$delayEnd); 495 +                                     } 496 + 497 +                                     return; 498                                     } 499 500                                     /** 506                                     */ 507                                     public function start(\$processName) { 508                                     switch (\$processName) { 509                                     case 'apply_reconfiguration': 510                                     \$status = \$this-&gt;_applyReconfiguration(); 511                                     break; 512                                     case 'check_dns': 513 +                                     \$status = \$this-&gt;_checkDns(); 514                                     break; 515                                     } 516 </pre>
--	--

0 comments on commit 59cc924