




Update dynamic reconfiguration for new gateway

[Browse files](#) master parsonsbots committed 27 minutes ago1 parent [5985d0a](#)commit [077c3023b12960e8c63c83cd7fcbb0c566d7628e](#) Showing 1 changed file with 21 additions and 82 deletions.

Unified

Split

▼ 103  dynamic_reconfiguration.php 

```
3      * Dynamic Proxy Node Reconfiguration
4      *
5      * Reconfigure and manage redundant proxy processes with
    no connection interruptions
6      - * in Squid (HTTP) and 3proxy (SOCKS 5) with an
    unlimited number of unique ACLs for
7      - * individual proxy nodes using multiple destination IPs
    on a single server.
8      *
9      * @author Will Parsons
10     - * @link https://parsonsbots.com
11
12     */
13     class DynamicProxyReconfiguration {
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36         $this->gatewaysData =
37         json_decode(file_get_contents('/scripts/cache/gatewaysData
38         '), true);
39
40         - if (empty($this->gatewaysData['data']
41         ['proxies'])) {
42
43             return false;
44         }
45
46         * @return boolean
47         */
48         protected function _applyReconfiguration() {
49         - // Create writable log and cache
50         directories
51
52             $this->_createDirectories();
53
54         - // Check for existing reconfiguration
55         process
56
57             if
58         (file_exists('/scripts/pid/reconfigure.pid')) {
59
60                 $lastRan =
61         file_get_contents('/scripts/pid/reconfigure.pid');
62
63         - // Start new reconfiguration
64         process if 15 minutes has passed
65
66             if ($lastRan < strtotime('-15
67         minutes')) {
68
69                 unlink('/scripts/pid/reconfigure.pid');
```

```
3      * Dynamic Proxy Node Reconfiguration
4      *
5      * Reconfigure and manage redundant proxy processes with
    no connection interruptions
6      + * with an unlimited number of unique ACLs for _
    individual proxy nodes using
7      + * multiple destination IPs on a single server.
8      *
9      * @author Will Parsons
10     + * @copyright 2019 Will Parsons
11     + * @license https://github.com/parsonsbots/dynamic-
    proxy-node-reconfiguration/blob/master/LICENSE MIT License
12     + * @link https://parsonsbots.com
13     + * @link https://eightomic.com
14     */
15     class DynamicProxyReconfiguration {
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39         $this->gatewaysData =
40         json_decode(file_get_contents('/scripts/cache/gatewaysData
41         '), true);
42
43         + if (empty($this->gatewaysData['data']
44         ['proxy_ips'])) {
45
46             return false;
47         }
48
49         * @return boolean
50         */
51         protected function _applyReconfiguration() {
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```

86         } else {
91             $gatewaysJsonData = shell_exec("curl " .
$this->api . " --connect-timeout 30");
92             $this->gatewaysData =
json_decode($gatewaysJsonData, true);
93
94             // Log API error timestamp
95             if (
96                 empty($this->gatewaysData['data'])
97             ||
98                 !is_dir('/etc/squid3')
99                 return false;
100             }
101
102             // Require Squid and sysctl configurations
103             from API
104             if (
105                 empty($this->gatewaysData['data']
106                 ['squid_conf']) ||
107                 empty($this->gatewaysData['data']
108                 ['sysctl_conf'])
109             ) {
110                 unlink('/scripts/pid/reconfigure.pid');
111                 return false;
112             }
113
114             // Cache new ACLs from API
115             file_put_contents('/scripts/cache/gatewaysData',
116             $gatewaysJsonData);
117
118             // Create new reconfiguration process ID
119             file_put_contents('/scripts/pid/reconfigure.pid', time());
120
121             // Save and apply sysctl settings
122             file_put_contents('/etc/sysctl.conf',
123             $this->gatewaysData['data']['sysctl_conf']);
124
125             shell_exec('sysctl -p');
126
127             // Don't run reconfiguration if there
128             aren't any ACLs to apply to proxy IPs
129             $proxies = $this->gatewaysData['data']
130             ['proxies'];
131
132             if (empty($proxies[0])) {
133                 unlink('/scripts/pid/reconfigure.pid');
134                 return false;
135             }
136
137             // Create new Squid user directories with
138             chunked sources and destinations
139             shell_exec('rm -rf /etc/squid3/users/');
140             shell_exec('mkdir -m 777
141             /etc/squid3/users/');
142
143             }
144
145             }

```

```

86         } else {
91             $gatewaysJsonData = shell_exec("curl " .
$this->api . " --connect-timeout 30");
92             $this->gatewaysData =
json_decode($gatewaysJsonData, true);
93
94             if (
95                 empty($this->gatewaysData['data'])
96             ||
97                 !is_dir('/etc/squid3')
98                 return false;
99             }
100
101             // Cache new ACLs from API
102             file_put_contents('/scripts/cache/gatewaysData',
103             $gatewaysJsonData);
104
105             // Create new reconfiguration process ID
106             file_put_contents('/scripts/pid/reconfigure.pid', time());
107
108             // Save and apply sysctl settings
109             file_put_contents('/etc/sysctl.conf',
110             $this->gatewaysData['data']['sysctl_conf']);
111
112             shell_exec('sysctl -p');
113
114             // Don't run reconfiguration if there
115             aren't any ACLs to apply to proxy IPs
116             $proxies = $this->gatewaysData['data']
117             ['proxies'];
118
119             if (empty($proxies[0])) {
120                 unlink('/scripts/pid/reconfigure.pid');
121                 return false;
122             }
123
124             // Create new Squid user directories with
125             chunked sources and destinations
126             shell_exec('rm -rf /etc/squid3/users/');
127             shell_exec('mkdir -m 777
128             /etc/squid3/users/');
129
130             }
131
132             }

```

```

143         $firewallRules = $this->_configureFirewallRules(true);
144     -
145     -         // Save Squid ACLs from API to file
146         shell_exec('rm
/etc/squid3/proxy_ip_acl.conf');
147         shell_exec('touch
/etc/squid3/proxy_ip_acl.conf');
148     -
149     + file_put_contents('/etc/squid3/proxy_ip_acl.conf',
implode("\n", $this->gatewaysData['data']['acls']));
150     -
151     -         // Set proxy usernames and passwords using
htpasswd and basic_ncsa_auth for security
152         shell_exec('htpasswd -cb
/etc/squid3/passwords default default');
153         shell_exec('htpasswd -D
/etc/squid3/passwords default');
154
155     }
156
157     }
158
159
160     -         // Apply redundant firewall rules to begin
seamless reconfiguration
161     $this->_applyFirewallRules($firewallRules,
'redundant');
162
163     -         // Reconfigure existing SOCKS processes
first once redundant firewall is applied
164     if (
165         !empty($this->gatewaysData['data']
['socks']) &&
166         !empty($this->processes['socks']
[0])
167     ) {
168     -         // Reconfigure main SOCKS instance
169         $this->_reconfigure(
170             'socks',
171             'service 3proxy start',
172         );
173     }
174
175     // Reconfigure main HTTP instance
176     $this->_reconfigure(
177         'http',
178         'squid3 start',
179         0,
180         75,
181         '/etc/squid3/squid.conf',
182         str_replace(['pid'], 'pid_filename
/var/run/squid3.pid', str_replace(['ports'], 'http_port '
. implode("\n" . 'http_port ', $this->processes['http']
[0]), $this->gatewaysData['data']['http']))
183     );
184
185     -         $redundantProcesses = $this->
processes['http'];
186     unset($redundantProcesses[0]);
187     $redundantProcessChunks =
array_chunk($redundantProcesses,
round(count($redundantProcesses) / 2), true);
188
189     foreach ($redundantProcessChunks as

```

<pre> 198 \$redundantProcessChunk) { 199 - // Define active redundant process 200 numbers 201 \$activeRedundantProcesses = 202 array_keys(\$redundantProcesses); 203 - 204 - // Define redundant process number 205 range for reconfiguration 206 reset(\$redundantProcessChunk); 207 \$redundantProcessStart = 208 key(\$redundantProcessChunk); 209 end(\$redundantProcessChunk); 210 unset(\$activeRedundantProcesses[\$redundantProcess - 1]); 211 } 212 - // Get list of active redundant 213 ports for firewall configuration 214 \$activeRedundantPorts = array(); 215 foreach (\$redundantProcesses as 216 \$key => \$redundantProcess) { 217 \$this-> 218 >_applyFirewallRules(\$firewallRules, 'elastic'); 219 } 220 foreach (\$redundantProcessRange as 221 \$redundantProcess) { 222 - // Reconfigure redundant 223 HTTP instances 224 \$this->_reconfigure(225 'http', 226 'squid3-redundant' 227 . \$redundantProcess . ' start -f /etc/squid3/squid- 228 redundant' . \$redundantProcess . '.conf', 229 0, 230 0, 231 '/etc/squid3/squid-redundant' . \$redundantProcess . 232 '.conf', 233 - 234 str_replace(['pid'], 'pid_filename /var/run/squid- 235 redundant' . \$redundantProcess . '.pid', 236 str_replace(['ports'], 'http_port ' . implode("\n" . 237 'http_port ', \$this->processes['http'] 238 [\$redundantProcess]), \$this->gatewaysData['data'] 239 ['http'])) 240); 241 } 242 } 243 - // Fixed delay necessary to 244 circumvent connection errors from varying downtime during 245 reconfiguration with bulk ACLs 246 sleep(75); 247 } 248 \$this->_applyFirewallRules(\$firewallRules, 249 'elastic'); 250 if (!empty(\$this->gatewaysData['data'] 251 ['socks-redundant'])) { 252 - // Reconfigure redundant SOCKS 253 instance 254 \$this->_reconfigure(</pre>	<pre> \$redundantProcessChunk) { 175 \$activeRedundantProcesses = 176 array_keys(\$redundantProcesses); 177 } 178 reset(\$redundantProcessChunk); 179 \$redundantProcessStart = 180 key(\$redundantProcessChunk); 181 end(\$redundantProcessChunk); 182 unset(\$activeRedundantProcesses[\$redundantProcess - 1]); 183 } 184 \$activeRedundantPorts = array(); 185 foreach (\$redundantProcesses as 186 \$key => \$redundantProcess) { 187 \$this-> 188 >_applyFirewallRules(\$firewallRules, 'elastic'); 189 } 190 foreach (\$redundantProcessRange as 191 \$redundantProcess) { 192 \$this->_reconfigure(193 'http', 194 'squid3-redundant' 195 . \$redundantProcess . ' start -f /etc/squid3/squid- 196 redundant' . \$redundantProcess . '.conf', 197 0, 198 0, 199 '/etc/squid3/squid-redundant' . \$redundantProcess . 200 '.conf', 201 + 202 str_replace(['pid'], 'pid_filename /var/run/squid- 203 redundant' . \$redundantProcess . '.pid', 204 str_replace(['ports'], 'http_port ' . implode("\n" . 205 'http_port ', \$this->processes['http'] 206 [\$redundantProcess]), \$this->gatewaysData['data'] 207 ['http'] 208 ['acls'])) 209); 210 } 211 } 212 sleep(75); 213 \$this->_applyFirewallRules(\$firewallRules, 214 'elastic'); 215 if (!empty(\$this->gatewaysData['data'] 216 ['socks-redundant'])) { 217 \$this->_reconfigure(</pre>
--	---

```

264         'socks',
265         'service 3proxy-redundant
start',
273     }
274
275     $this->_applyFirewallRules($firewallRules,
'elastic');
276 -
277 -         // Remove reconfiguration process ID
278         unlink('/scripts/pid/reconfigure.pid');
279
280         return true;
286     * @return boolean
287     */
288     protected function _checkDNS() {
289 -         // Check for existing DNS check process ID
290         if (file_exists('/scripts/pid/dns.pid')) {
291             $lastRan =
file_get_contents('/scripts/pid/dns.pid');
292
293 -         // Start new DNS check process if
2 minutes have passed
294             if ($lastRan < strtotime('-2
minutes')) {
295
296                 unlink('/scripts/pid/dns.pid');
297             } else {
298                 return false;
299             }
300
301 -         // Create new DNS check process ID
302         file_put_contents('/scripts/pid/dns.pid',
time());
303 -
304 -         // Get list of DNS IPs
305         $this->gatewaysData =
json_decode(file_get_contents('/scripts/cache/gatewaysData
'), true);
306         array_shift($this->gatewaysData['data']
['dns_ips']);
307
314         foreach ($dnsIps as $key => $dnsIp) {
315             $processName = $key == 0 ? 'named'
: 'named-redundant' . $key;
316             $dnsResponse = array();
317 -         exec('dig +time=2 +tries=1
ghostproxies @' . $dnsIp . ' 2>&1', $dnsResponse);
318
319             if (
320                 !empty($dnsResponse[3]) &&
321                 * @return array $rules Firewall rules
322             */
323             protected function
_configureFirewallRules($redundant = false, $overridePorts
= array()) {
324 -         // Begin input filter rules
325         $rules = array(
326             '*filter',
327             ':INPUT ACCEPT [0:0]',
328             ':FORWARD ACCEPT [0:0]',
329             ':OUTPUT ACCEPT [0:0]',
330             '-A INPUT -p icmp -m hashlimit --

```

```

234         'socks',
235         'service 3proxy-redundant
start',
243     }
244
245     $this->_applyFirewallRules($firewallRules,
'elastic');
246
247         unlink('/scripts/pid/reconfigure.pid');
248
249         return true;
254     * @return boolean
255     */
256     protected function _checkDNS() {
257         if (file_exists('/scripts/pid/dns.pid')) {
258             $lastRan =
file_get_contents('/scripts/pid/dns.pid');
259
260             if ($lastRan < strtotime('-2
minutes')) {
261
262                 unlink('/scripts/pid/dns.pid');
263             } else {
264                 return false;
265             }
266
267         file_put_contents('/scripts/pid/dns.pid',
time());
268
269         $this->gatewaysData =
json_decode(file_get_contents('/scripts/cache/gatewaysData
'), true);
270         array_shift($this->gatewaysData['data']
['dns_ips']);
271
272         foreach ($dnsIps as $key => $dnsIp) {
273             $processName = $key == 0 ? 'named'
: 'named-redundant' . $key;
274             $dnsResponse = array();
275 +         exec('dig +time=2 +tries=1 proxies
@' . $dnsIp . ' 2>&1', $dnsResponse);
276
277             if (
278                 !empty($dnsResponse[3]) &&
279                 * @return array $rules Firewall rules
280             */
281             protected function
_configureFirewallRules($redundant = false, $overridePorts
= array()) {
282 -         // Begin input filter rules
283         $rules = array(
284             '*filter',
285             ':INPUT ACCEPT [0:0]',
286             ':FORWARD ACCEPT [0:0]',
287             ':OUTPUT ACCEPT [0:0]',
288             '-A INPUT -p icmp -m hashlimit --

```

```

hashlimit-name icmp --hashlimit-mode srcip --hashlimit
1/second --hashlimit-burst 2 -j ACCEPT', // Rate limit
ICMP to prevent PoD attacks
412     );
413
414 -           // Rate limit SSH ports to protect from
brute-forcing
415         if (
416             !empty($this->sshPorts) &&
417             is_array($this->sshPorts)
418         ) {
419             foreach ($this->sshPorts as
$sshPort) {
420                 if (is_numeric($sshPort))
421                 {
422                     $rules[] = '-A
INPUT -p tcp -m tcp --dport ' . $sshPort . ' -m connlimit
--connlimit-above 4 --connlimit-mask 32 --connlimit-saddr
-j REJECT --reject-with tcp-reset';
423                 }
424             }
425         }
426
427 -           // Apply custom firewall rules from API
(e.g. disabling ports for specific destinations)
428         if (
429             !empty($this->gatewaysData['data']
['firewall_filter']) &&
430             is_array($this-
>gatewaysData['data']['firewall_filter'])
434         ) {
435             }
436
437 -           // Allow specific DNS IPs for internal
round-robin DNS
438         if (
439             !empty($this->gatewaysData['data']
['dns_ips']) &&
440             is_array($this-
>gatewaysData['data']['dns_ips'])
441         ) {
442             $rules[] = '-A OUTPUT -d ' .
implode(',', $this->gatewaysData['data']['dns_ips']) . ' -
p udp -m udp -j ACCEPT';
443         }
444
445 -           // Prevent spoof attacks when using DNS on
localhost
446         $rules[] = '-A OUTPUT -s 127.0.0.0/24 -p
udp -j DROP';
447 -
448 -           // End input filter rules
449         $rules[] = 'COMMIT';
450 -
451 -           // Prepare default rules for NAT
452         $rules[] = '*nat';
453         $rules[] = ':PREROUTING ACCEPT [0:0]';
454         $rules[] = ':INPUT ACCEPT [0:0]';

```

```

hashlimit-name icmp --hashlimit-mode srcip --hashlimit
1/second --hashlimit-burst 2 -j ACCEPT'
374     );
375
376         if (
377             !empty($this->sshPorts) &&
378             is_array($this->sshPorts)
379         ) {
380             foreach ($this->sshPorts as
$sshPort) {
381                 if (is_numeric($sshPort))
382                 {
383                     $rules[] = '-A
+ INPUT -p tcp -m tcp --dport ' . $sshPort . ' -m connlimit
--connlimit-above 4 --connlimit-mask 32 --connlimit-saddr
-j DROP';
384                 }
385             }
386         }
387
388         if (
389             !empty($this->gatewaysData['data']
['firewall_filter']) &&
390             is_array($this-
>gatewaysData['data']['firewall_filter'])
394         ) {
395             }
396
397         if (
398             !empty($this->gatewaysData['data']
['dns_ips']) &&
399             is_array($this-
>gatewaysData['data']['dns_ips'])
400         ) {
401             $rules[] = '-A OUTPUT -d ' .
implode(',', $this->gatewaysData['data']['dns_ips']) . ' -
p udp -m udp -j ACCEPT';
402         }
403
404         $rules[] = '-A OUTPUT -s 127.0.0.0/24 -p
udp -j DROP';
405
406         $rules[] = '*nat';
407         $rules[] = ':PREROUTING ACCEPT [0:0]';
408         $rules[] = ':INPUT ACCEPT [0:0]';

```

```

455     $rules[] = ':OUTPUT ACCEPT [0:0]';
456     $rules[] = ':POSTROUTING ACCEPT [0:0]';
457
458 -         // Remove 127.0.0.1 from array of DNS IPs
and sort keys for load balancing
459     $dnsIps = array_values($this->gatewaysData['data']['dns_ips']);
460     unset($dnsIps[0]);
461     krsort($dnsIps);
462
463 -         // Load balance DNS
464     foreach ($dnsIps as $key => $dnsIp) {
465         $loadBalancer = '';
466
467         $rules[] = '-A OUTPUT -d
127.0.0.1/32 -p udp -m udp --dport 53 ' . $loadBalancer .
'-j DNAT --to-destination ' . $dnsIp;
472     }
473
474 -         // Begin DNAT load balancing for each
process
475     if (
476         !empty($this->processes) &&
477         is_array($this->processes)
478     ) {
479         $ports =
480         array_unique($basePorts);
481
482 -         // Remove ports from
routing list that are refusing connections
483     foreach ($ports as $key =>
$port) {
484 -         if ($this->_checkPort($this->gatewaysData['data']['proxies'][0],
$port, $protocol) === false) {
485         unset($ports[$key]);
486     }
487     }
488
489 -         /*
Prevent individual
490 -         listening ports / sockets from overloading using this
simple, persistent internal load balancing
491 -         method for DNAT
ports only (without changing the destination IP). This
also prevents malicious processes from hijacking
492 -         shared ports (when
the combination of SO_REUSEPORT and SO_REUSEADDR are used)
while still allowing a large number of sockets
493 -         */
494     shuffle($ports);
495     $ports =
496     array_values($ports);
497     krsort($ports);
498     }
499 }
500
501 -         // End DNAT load balancing for each
process
502     $rules[] = 'COMMIT';
503
504 -         // Chunk firewall rules to write to file

```

```

409     $rules[] = ':OUTPUT ACCEPT [0:0]';
410     $rules[] = ':POSTROUTING ACCEPT [0:0]';
411
412     $dnsIps = array_values($this->gatewaysData['data']['dns_ips']);
413     unset($dnsIps[0]);
414     krsort($dnsIps);
415
416     foreach ($dnsIps as $key => $dnsIp) {
417         $loadBalancer = '';
418
419         $rules[] = '-A OUTPUT -d
127.0.0.1/32 -p udp -m udp --dport 53 ' . $loadBalancer .
'-j DNAT --to-destination ' . $dnsIp;
424     }
425
426     if (
427         !empty($this->processes) &&
428         is_array($this->processes)
429     ) {
430         $ports =
431         array_unique($basePorts);
432
433     foreach ($ports as $key =>
$port) {
434 +         if ($this->_checkPort($this->gatewaysData['data']['proxy_ips'][0],
$port, $protocol) === false) {
435         unset($ports[$key]);
436     }
437     }
438
439 -         shuffle($ports);
440     $ports =
441     array_values($ports);
442     krsort($ports);
443     }
444 }
445
446     $rules[] = 'COMMIT';

```

<pre> 541 \$rules = array_chunk(\$rules, 100); 542 - 543 return \$rules; 544 } 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 \$shellCommands[] = 'kill -9 ' . trim(\$killProcess); 633 } 634 635 - 636 // Run shell commands through bash script 637 if (count(\$shellCommands > 1)) { 638 unlink('/scripts/' . \$protocol . '.sh'); 639 file_put_contents('/scripts/' . \$protocol . '.sh', implode("\n", \$shellCommands)); 640 shell_exec('chmod +x /scripts/' . \$protocol . '.sh'); 641 shell_exec('/scripts/./' . \$protocol . '.sh'); 642 } 643 - 644 // Update configuration file for process 645 if (646 !empty(\$configurationFile) && 647 !empty(\$configurationData) 648) { 649 file_put_contents(\$configurationFile, \$configurationData); 650 } 651 - 652 // Process recovery 653 unlink(\$processId); 654 sleep(2); 655 shell_exec(\$startCommand); 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 } 708 ?> </pre>	<pre> 483 \$rules = array_chunk(\$rules, 100); 484 485 } 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 </pre>
---	---

0 comments on commit 077c302