503 lines (414 sloc) | 12 KB

```
1   package home
2
3   import (
4           "encoding/json"
5           "fmt"
6           "io"
7           "net"
8           "net/http"
9           "net/url"
10          "os"
11          "path/filepath"
12          "strings"
13          "time"
14
15          "github.com/AdguardTeam/AdGuardHome/internal/aghhttp"
16          "github.com/AdguardTeam/golibs/errors"
17          "github.com/AdguardTeam/golibs/log"
18          "github.com/miekg/dns"
19  )
20
21  // validateFilterURL validates the filter list URL or file name.
22  func validateFilterURL(urlStr string) (err error) {
23          if filepath.IsAbs(urlStr) {
24                  _, err = os.Stat(urlStr)
25                  if err != nil {
26                          return fmt.Errorf("checking filter file: %w", err)
27                  }
28
```

```
29                  return nil
30          }
31
32          url, err := url.ParseRequestURI(urlStr)
33          if err != nil {
34                  return fmt.Errorf("checking filter url: %w", err)
35          }
36
37          if s := url.Scheme; s != schemeHTTP && s != schemeHTTPS {
38                  return fmt.Errorf("checking filter url: invalid scheme %q", s)
39          }
40
41          return nil
42  }
43
44  type filterAddJSON struct {
45          Name      string `json:"name"`
46          URL       string `json:"url"`
47          Whitelist bool   `json:"whitelist"`
48  }
49
50  func (f *Filtering) handleFilteringAddURL(w http.ResponseWriter, r *http.Request) {
51          fj := filterAddJSON{}
52          err := json.NewDecoder(r.Body).Decode(&fj)
53          if err != nil {
54                  aghhttp.Error(r, w, http.StatusBadRequest, "Failed to parse request body json: %s"
55
56                  return
57          }
58
59          err = validateFilterURL(fj.URL)
60          if err != nil {
61                  err = fmt.Errorf("invalid url: %s", err)
62                  aghhttp.Error(r, w, http.StatusBadRequest, "%s", err)
63
64                  return
65          }
66
67          // Check for duplicates
68          if filterExists(fj.URL) {
69                  aghhttp.Error(r, w, http.StatusBadRequest, "Filter URL already added -- %s", fj.UR
70
71                  return
72          }
73
74          // Set necessary properties
75          filt := filter{
76                  Enabled: true,
77                  URL:     fj.URL,
```

```
78                  Name:    fj.Name,
79                  white:   fj.Whitelist,
80          }
81          filt.ID = assignUniqueFilterID()
82
83          // Download the filter contents
84          ok, err := f.update(&filt)
85          if err != nil {
86                  aghhttp.Error(
87                          r,
88                          w,
89                          http.StatusBadRequest,
90                          "Couldn't fetch filter from url %s: %s",
91                          filt.URL,
92                          err,
93                  )
94
95                  return
96          }
97
98          if !ok {
99                  aghhttp.Error(
100                         r,
101                         w,
102                         http.StatusBadRequest,
103                         "Filter at the url %s is invalid (maybe it points to blank page?)",
104                         filt.URL,
105                 )
106
107                 return
108         }
109
110         // URL is assumed valid so append it to filters, update config, write new
111         // file and reload it to engines.
112         if !filterAdd(filt) {
113                 aghhttp.Error(r, w, http.StatusBadRequest, "Filter URL already added -- %s", filt.
114
115                 return
116         }
117
118         onConfigModified()
119         enableFilters(true)
120
121         _, err = fmt.Fprintf(w, "OK %d rules\n", filt.RulesCount)
122         if err != nil {
123                 aghhttp.Error(r, w, http.StatusInternalServerError, "Couldn't write body: %s", err
124         }
125 }
126
```

```
127 func (f *Filtering) handleFilteringRemoveURL(w http.ResponseWriter, r *http.Request) {
128         type request struct {
129                 URL       string `json:"url"`
130                 Whitelist bool   `json:"whitelist"`
131         }
132
133         req := request{}
134         err := json.NewDecoder(r.Body).Decode(&req)
135         if err != nil {
136                 aghhttp.Error(r, w, http.StatusBadRequest, "failed to parse request body json: %s"
137
138                 return
139         }
140
141         config.Lock()
142         filters := &config.Filters
143         if req.Whitelist {
144                 filters = &config.WhitelistFilters
145         }
146
147         var deleted filter
148         var newFilters []filter
149         for _, f := range *filters {
150                 if f.URL != req.URL {
151                         newFilters = append(newFilters, f)
152
153                         continue
154                 }
155
156                 deleted = f
157                 path := f.Path()
158                 err = os.Rename(path, path+".old")
159                 if err != nil {
160                         log.Error("deleting filter %q: %s", path, err)
161                 }
162         }
163
164         *filters = newFilters
165         config.Unlock()
166
167         onConfigModified()
168         enableFilters(true)
169
170         // NOTE: The old files "filter.txt.old" aren't deleted.  It's not really
171         // necessary, but will require the additional complicated code to run
172         // after enableFilters is done.
173         //
174         // TODO(a.garipov): Make sure the above comment is true.
175
```

```
176         _, err = fmt.Fprintf(w, "OK %d rules\n", deleted.RulesCount)
177         if err != nil {
178                 aghhttp.Error(r, w, http.StatusInternalServerError, "couldn't write body: %s", err
179         }
180 }
181
182 type filterURLReqData struct {
183         Name    string `json:"name"`
184         URL     string `json:"url"`
185         Enabled bool   `json:"enabled"`
186 }
187
188 type filterURLReq struct {
189         Data      *filterURLReqData `json:"data"`
190         URL       string            `json:"url"`
191         Whitelist bool              `json:"whitelist"`
192 }
193
194 func (f *Filtering) handleFilteringSetURL(w http.ResponseWriter, r *http.Request) {
195         fj := filterURLReq{}
196         err := json.NewDecoder(r.Body).Decode(&fj)
197         if err != nil {
198                 aghhttp.Error(r, w, http.StatusBadRequest, "json decode: %s", err)
199
200                 return
201         }
202
203         if fj.Data == nil {
204                 err = errors.Error("data cannot be null")
205                 aghhttp.Error(r, w, http.StatusBadRequest, "%s", err)
206
207                 return
208         }
209
210         err = validateFilterURL(fj.Data.URL)
211         if err != nil {
212                 err = fmt.Errorf("invalid url: %s", err)
213                 aghhttp.Error(r, w, http.StatusBadRequest, "%s", err)
214
215                 return
216         }
217
218         filt := filter{
219                 Enabled: fj.Data.Enabled,
220                 Name:    fj.Data.Name,
221                 URL:     fj.Data.URL,
222         }
223         status := f.filterSetProperties(fj.URL, filt, fj.Whitelist)
224         if (status & statusFound) == 0 {
```

```
225                 http.Error(w, "URL doesn't exist", http.StatusBadRequest)
226                 return
227         }
228         if (status & statusURLExists) != 0 {
229                 http.Error(w, "URL already exists", http.StatusBadRequest)
230                 return
231         }
232
233         onConfigModified()
234
235         restart := (status & statusEnabledChanged) != 0
236         if (status&statusUpdateRequired) != 0 && fj.Data.Enabled {
237                 // download new filter and apply its rules
238                 flags := filterRefreshBlocklists
239                 if fj.Whitelist {
240                         flags = filterRefreshAllowlists
241                 }
242                 nUpdated, _ := f.refreshFilters(flags, true)
243                 // if at least 1 filter has been updated, refreshFilters() restarts the filtering
244                 // if not - we restart the filtering ourselves
245                 restart = false
246                 if nUpdated == 0 {
247                         restart = true
248                 }
249         }
250
251         if restart {
252                 enableFilters(true)
253         }
254 }
255
256 func (f *Filtering) handleFilteringSetRules(w http.ResponseWriter, r *http.Request) {
257         // This use of ReadAll is safe, because request's body is now limited.
258         body, err := io.ReadAll(r.Body)
259         if err != nil {
260                 aghhttp.Error(r, w, http.StatusBadRequest, "Failed to read request body: %s", err)
261
262                 return
263         }
264
265         config.UserRules = strings.Split(string(body), "\n")
266         onConfigModified()
267         enableFilters(true)
268 }
269
270 func (f *Filtering) handleFilteringRefresh(w http.ResponseWriter, r *http.Request) {
271         type Req struct {
272                 White bool `json:"whitelist"`
273         }
```

```
274         type Resp struct {
275                 Updated int `json:"updated"`
276         }
277         resp := Resp{}
278         var err error
279 
280         req := Req{}
281         err = json.NewDecoder(r.Body).Decode(&req)
282         if err != nil {
283                 aghhttp.Error(r, w, http.StatusBadRequest, "json decode: %s", err)
284 
285                 return
286         }
287 
288         flags := filterRefreshBlocklists
289         if req.White {
290                 flags = filterRefreshAllowlists
291         }
292         func() {
293                 // Temporarily unlock the Context.controlLock because the
294                 // f.refreshFilters waits for it to be unlocked but it's
295                 // actually locked in ensure wrapper.
296                 //
297                 // TODO(e.burkov):  Reconsider this messy syncing process.
298                 Context.controlLock.Unlock()
299                 defer Context.controlLock.Lock()
300 
301                 resp.Updated, err = f.refreshFilters(flags|filterRefreshForce, false)
302         }()
303         if err != nil {
304                 aghhttp.Error(r, w, http.StatusInternalServerError, "%s", err)
305 
306                 return
307         }
308 
309         js, err := json.Marshal(resp)
310         if err != nil {
311                 aghhttp.Error(r, w, http.StatusInternalServerError, "json encode: %s", err)
312 
313                 return
314         }
315         w.Header().Set("Content-Type", "application/json")
316         _, _ = w.Write(js)
317 }
318 
319 type filterJSON struct {
320         URL         string `json:"url"`
321         Name        string `json:"name"`
322         LastUpdated string `json:"last_updated,omitempty"`
```

```
	ID          int64  `json:"id"`
	RulesCount  uint32 `json:"rules_count"`
	Enabled     bool   `json:"enabled"`
}

type filteringConfig struct {
	Filters          []filterJSON `json:"filters"`
	WhitelistFilters []filterJSON `json:"whitelist_filters"`
	UserRules        []string     `json:"user_rules"`
	Interval         uint32       `json:"interval"` // in hours
	Enabled          bool         `json:"enabled"`
}

func filterToJSON(f filter) filterJSON {
	fj := filterJSON{
		ID:          f.ID,
		Enabled:     f.Enabled,
		URL:         f.URL,
		Name:        f.Name,
		RulesCount: uint32(f.RulesCount),
	}

	if !f.LastUpdated.IsZero() {
		fj.LastUpdated = f.LastUpdated.Format(time.RFC3339)
	}

	return fj
}

// Get filtering configuration
func (f *Filtering) handleFilteringStatus(w http.ResponseWriter, r *http.Request) {
	resp := filteringConfig{}
	config.RLock()
	resp.Enabled = config.DNS.FilteringEnabled
	resp.Interval = config.DNS.FiltersUpdateIntervalHours
	for _, f := range config.Filters {
		fj := filterToJSON(f)
		resp.Filters = append(resp.Filters, fj)
	}
	for _, f := range config.WhitelistFilters {
		fj := filterToJSON(f)
		resp.WhitelistFilters = append(resp.WhitelistFilters, fj)
	}
	resp.UserRules = config.UserRules
	config.RUnlock()

	jsonVal, err := json.Marshal(resp)
	if err != nil {
		aghhttp.Error(r, w, http.StatusInternalServerError, "json encode: %s", err)
```

```
372
373                 return
374         }
375         w.Header().Set("Content-Type", "application/json")
376         _, err = w.Write(jsonVal)
377         if err != nil {
378                 aghhttp.Error(r, w, http.StatusInternalServerError, "http write: %s", err)
379         }
380 }
381
382 // Set filtering configuration
383 func (f *Filtering) handleFilteringConfig(w http.ResponseWriter, r *http.Request) {
384         req := filteringConfig{}
385         err := json.NewDecoder(r.Body).Decode(&req)
386         if err != nil {
387                 aghhttp.Error(r, w, http.StatusBadRequest, "json decode: %s", err)
388
389                 return
390         }
391
392         if !checkFiltersUpdateIntervalHours(req.Interval) {
393                 aghhttp.Error(r, w, http.StatusBadRequest, "Unsupported interval")
394
395                 return
396         }
397
398         func() {
399                 config.Lock()
400                 defer config.Unlock()
401
402                 config.DNS.FilteringEnabled = req.Enabled
403                 config.DNS.FiltersUpdateIntervalHours = req.Interval
404         }()
405
406         onConfigModified()
407         enableFilters(true)
408 }
409
410 type checkHostRespRule struct {
411         Text         string `json:"text"`
412         FilterListID int64  `json:"filter_list_id"`
413 }
414
415 type checkHostResp struct {
416         Reason string `json:"reason"`
417
418         // Rule is the text of the matched rule.
419         //
420         // Deprecated: Use Rules[*].Text.
```

```
421         Rule string `json:"rule"`
422
423         Rules []*checkHostRespRule `json:"rules"`
424
425         // for FilteredBlockedService:
426         SvcName string `json:"service_name"`
427
428         // for Rewrite:
429         CanonName string   `json:"cname"`    // CNAME value
430         IPList    []net.IP `json:"ip_addrs"` // list of IP addresses
431
432         // FilterID is the ID of the rule's filter list.
433         //
434         // Deprecated: Use Rules[*].FilterListID.
435         FilterID int64 `json:"filter_id"`
436 }
437
438 func (f *Filtering) handleCheckHost(w http.ResponseWriter, r *http.Request) {
439         q := r.URL.Query()
440         host := q.Get("name")
441
442         setts := Context.dnsFilter.GetConfig()
443         setts.FilteringEnabled = true
444         setts.ProtectionEnabled = true
445         Context.dnsFilter.ApplyBlockedServices(&setts, nil, true)
446         result, err := Context.dnsFilter.CheckHost(host, dns.TypeA, &setts)
447         if err != nil {
448                 aghhttp.Error(
449                         r,
450                         w,
451                         http.StatusInternalServerError,
452                         "couldn't apply filtering: %s: %s",
453                         host,
454                         err,
455                 )
456
457                 return
458         }
459
460         resp := checkHostResp{}
461         resp.Reason = result.Reason.String()
462         resp.SvcName = result.ServiceName
463         resp.CanonName = result.CanonName
464         resp.IPList = result.IPList
465
466         if len(result.Rules) > 0 {
467                 resp.FilterID = result.Rules[0].FilterListID
468                 resp.Rule = result.Rules[0].Text
469         }
```

```
470
471         resp.Rules = make([]*checkHostRespRule, len(result.Rules))
472         for i, r := range result.Rules {
473                 resp.Rules[i] = &checkHostRespRule{
474                         FilterListID: r.FilterListID,
475                         Text:         r.Text,
476                 }
477         }
478
479         js, err := json.Marshal(resp)
480         if err != nil {
481                 aghhttp.Error(r, w, http.StatusInternalServerError, "json encode: %s", err)
482
483                 return
484         }
485         w.Header().Set("Content-Type", "application/json")
486         _, _ = w.Write(js)
487 }
488
489 // RegisterFilteringHandlers - register handlers
490 func (f *Filtering) RegisterFilteringHandlers() {
491         httpRegister(http.MethodGet, "/control/filtering/status", f.handleFilteringStatus)
492         httpRegister(http.MethodPost, "/control/filtering/config", f.handleFilteringConfig)
493         httpRegister(http.MethodPost, "/control/filtering/add_url", f.handleFilteringAddURL)
494         httpRegister(http.MethodPost, "/control/filtering/remove_url", f.handleFilteringRemoveURL)
495         httpRegister(http.MethodPost, "/control/filtering/set_url", f.handleFilteringSetURL)
496         httpRegister(http.MethodPost, "/control/filtering/refresh", f.handleFilteringRefresh)
497         httpRegister(http.MethodPost, "/control/filtering/set_rules", f.handleFilteringSetRules)
498         httpRegister(http.MethodGet, "/control/filtering/check_host", f.handleCheckHost)
499 }
500
501 func checkFiltersUpdateIntervalHours(i uint32) bool {
502         return i == 0 || i == 1 || i == 12 || i == 1*24 || i == 3*24 || i == 7*24
503 }
```