Commit 406022e

thieung authored 2 weeks ago Verified

```
fix(security): close auth bypass + default-permit RBAC (issue #866) (#950)

Fixes a 3-vuln chain enabling unauthenticated/underprivileged access to
admin surfaces:

1. Auth bypass (router.go): invalid tokens silently fell through to
   anonymous role. Now reject on token presence with invalid/expired.
2. Default-permit RBAC (policy.go): unknown methods returned a writable
   role. Now fail-closed: unknown -> RoleNone.
3. Missing admin classifications: several sensitive methods weren't in
   any allowlist and slipped through default-permit. Now explicitly
   classified (admin/write/read).

Additional hardening:
- Remove isWriteMethod prefix fallbacks; use exact allowlist only to
  prevent silent scope creep when new methods are added.
- Classify Phase-3 methods (TTS, browser, zalo, whatsapp) into
  admin/write/read per sensitivity.
- Gate empty MethodRole in router dispatcher.

Tests:
- TestMethodRole_ApprovalsList_IsViewer: asserts exec.approval.list is
  viewer and approve/deny are operator.
- TestMethodRole_DriftCoverage_AllProtocolMethodsClassified: AST-parses
  pkg/protocol/methods.go at test time; asserts every Method* constant
  resolves to a non-None role. Permanent drift guard for future adds.
```

dev (#950) · v3.11.3 ••• lite-v3.9.0

1 parent a29938c commit 406022e

3 files changed +392 -36

Top

Filter files…

internal

- gateway
  - router.go
- permissions
  - policy.go
  - policy_test.go

Search within code

internal/gateway/router.go

```
@@ -67,12 +67,25 @@ func (r *MethodRouter) Handle(ctx context.Context,
client *Client, req *protocol
67  67      if req.Method != protocol.MethodConnect && req.Method !=
protocol.MethodHealth && req.Method != protocol.MethodBrowserPairingStatus {
68  68          if pe := r.server.policyEngine; pe != nil {
69  69              if !pe.CanAccess(client.role, req.Method) {
70      -               slog.Warn("permission denied", "method", req.Method, "role",
client.role, "client", client.id)
        70  +               required := permissions.MethodRole(req.Method)
        71  +               slog.Warn("security.permission_denied",
        72  +                   "method", req.Method,
        73  +                   "role", client.role,
        74  +                   "required", string(required),
        75  +                   "client", client.id,
        76  +               )
71  77              locale := i18n.Normalize(client.locale)
        78  +               var msg string
        79  +               if required == permissions.RoleNone {
        80  +                   // Unclassified method — fail-closed (issue #866 fix).
        81  +                   msg = i18n.T(locale, i18n.MsgPermissionDenied, req.Method+"
is not permitted for this session")
        82  +               } else {
        83  +                   msg = i18n.T(locale, i18n.MsgPermissionDenied, req.Method+"
requires "+string(required)+" role")
        84  +               }
72  85              client.SendResponse(protocol.NewErrorResponse(
73  86                  req.ID,
74  87                  protocol.ErrUnauthorized,
```

```
75        -                    i18n.T(locale, i18n.MsgPermissionDenied, req.Method+" requires "+string(permissions.MethodRole(req.Method))+" role"),
          88 +                 msg,
76        89                   ))
77        90                   return
78        91               }
          @@ -286,17 +299,22 @@ func (r *MethodRouter) handleConnect(ctx context.Context, client *Client, req *p
286       299          }
287       300      }
288       301
289       -      // Path 4: Fallback → viewer (wrong token or pairing not available)
290       -      client.role = permissions.RoleViewer
291       -      client.authenticated = true
292       -      client.userID = params.UserID
293       -      tid, errCode := r.resolveTenantHint(ctx, params.TenantHint, params.UserID)
294       -      if errCode != "" {
295       -          client.SendResponse(protocol.NewErrorResponse(req.ID, errCode, "tenant access revoked"))
296       -          return
297       -      }
298       -      client.tenantID = tid
299       -      r.sendConnectResponse(ctx, client, req.ID)
          302 +      // Path 4: Fail-closed — no valid token, no valid pairing → reject.
          303 +      // Previously fell back to viewer+authenticated=true, which allowed any
          304 +      // unauthenticated client to exercise the default-permit policy (CVE #866).
          305 +      slog.Warn("security.ws_connect_rejected",
          306 +          "reason", "no_valid_credentials",
          307 +          "client", client.id,
          308 +          "has_token", params.Token != "",
          309 +          "has_sender_id", params.SenderID != "",
          310 +      )
          311 +      client.authenticated = false
          312 +      locale := i18n.Normalize(client.locale)
          313 +      client.SendResponse(protocol.NewErrorResponse(
          314 +          req.ID,
          315 +          protocol.ErrUnauthorized,
          316 +          i18n.T(locale, i18n.MsgPermissionDenied, "valid token or active pairing required"),
          317 +      ))
```

```
300  318     }
301  319
302  320     func (r *MethodRouter) sendConnectResponse(ctx context.Context, client *Client,
                reqID string) {
```

internal/permissions/policy.go

```
                @@ -27,6 +27,11 @@ const (
27   27          RoleAdmin    Role = "admin"    // Full access to all methods
28   28          RoleOperator Role = "operator" // Read + write access (no admin operations)
29   29          RoleViewer   Role = "viewer"   // Read-only access
          30    +
          31    +     // RoleNone is a sentinel returned by MethodRole for methods that have no
          32    +     // explicit classification. The router treats it as deny-for-everyone so
          33    +     // newly-added RPCs are secure-by-default (fail-closed).
          34    +     RoleNone Role = ""
30   35     )
31   36
32   37     // Scope represents a specific permission scope.
                @@ -88,8 +93,14 @@ func (pe *PolicyEngine) IsOwner(senderID string) bool {
88   93     }
89   94
90   95     // CanAccess checks if a role has access to a gateway RPC method.
          96    + // Unclassified methods (MethodRole == RoleNone) are denied for every role
          97    + // including owner — callers must surface an explicit PERMISSION_DENIED/
          98    + // UNAUTHORIZED and never silently permit.
91   99     func (pe *PolicyEngine) CanAccess(role Role, method string) bool {
92   100        requiredRole := MethodRole(method)
          101   +     if requiredRole == RoleNone {
          102   +         return false
          103   +     }
93   104        return roleLevel(role) >= roleLevel(requiredRole)
94   105    }
95   106
                @@ -130,7 +141,18 @@ func RoleFromScopes(scopes []Scope) Role {
130  141    }
131  142
132  143    // MethodRole returns the minimum role required for a given RPC method.
```

```
144 + //
145 + // Policy is fail-closed (default-deny): methods absent from every allowlist
146 + // return RoleNone. The gateway dispatcher must reject RoleNone with an
147 + // UNAUTHORIZED / PERMISSION_DENIED error rather than granting viewer access.
148 + // This is the fix for issue #866 where default-permit let unauthenticated
149 + // clients invoke mutation/exfiltration RPCs (heartbeat.*, logs.tail, etc.).
133 150   func MethodRole(method string) Role {
151 +     // System methods that bypass auth entirely (pre-auth handshake only).
152 +     if isPublicMethod(method) {
153 +         return RoleViewer
154 +     }
155 +
134 156       // Admin-only methods
135 157       if isAdminMethod(method) {
136 158           return RoleAdmin
    @@ -141,8 +163,26 @@ func MethodRole(method string) Role {
141 163           return RoleOperator
142 164       }
143 165
144     -     // Everything else is read-only (viewer can access)
145     -     return RoleViewer
166 +     // Read-only methods (viewer and above)
167 +     if isReadMethod(method) {
168 +         return RoleViewer
169 +     }
170 +
171 +     // Fail-closed: unknown / unclassified method → deny.
172 +     return RoleNone
173 + }
174 +
175 + // isPublicMethod lists methods every authenticated (and some pre-auth) client
176 + // is allowed to call. Kept tiny on purpose — do not expand without review.
177 + func isPublicMethod(method string) bool {
178 +     switch method {
179 +     case protocol.MethodConnect,
180 +         protocol.MethodHealth,
181 +         protocol.MethodStatus,
182 +         protocol.MethodBrowserPairingStatus:
183 +         return true
184 +     }
```

```
        185 +     return false
146     186   }
147     187
148     188   // MethodScopes returns the scopes required for a method.
        ...   @@ -164,62 +204,218 @@ func MethodScopes(method string) []Scope {
164     204
165     205   func isAdminMethod(method string) bool {
166     206       adminMethods := []string{
        207 +         // Config — admin/owner only. Additional master-scope/owner guards live
        208 +         // in the handler middleware, but classify here for defense-in-depth.
        209 +         protocol.MethodConfigGet,
167     210           protocol.MethodConfigApply,
168     211           protocol.MethodConfigPatch,
        212 +         protocol.MethodConfigSchema,
        213 +         protocol.MethodConfigDefaults,
        214 +         protocol.MethodConfigPermissionsList,
        215 +         protocol.MethodConfigPermissionsGrant,
        216 +         protocol.MethodConfigPermissionsRevoke,
        217 +
        218 +         // Agents — create/update/delete and link mutations.
169     219           protocol.MethodAgentsCreate,
170     220           protocol.MethodAgentsUpdate,
171     221           protocol.MethodAgentsDelete,
172         -         protocol.MethodAgentsLinksList,
173     222           protocol.MethodAgentsLinksCreate,
174     223           protocol.MethodAgentsLinksUpdate,
175     224           protocol.MethodAgentsLinksDelete,
        225 +
        226 +         // Channels.
176     227           protocol.MethodChannelsToggle,
        228 +         protocol.MethodChannelInstancesCreate,
        229 +         protocol.MethodChannelInstancesUpdate,
        230 +         protocol.MethodChannelInstancesDelete,
        231 +
        232 +         // Pairing management (approve/revoke/list/deny require admin).
177     233           protocol.MethodPairingApprove,
        234 +         protocol.MethodPairingDeny,
        235 +         protocol.MethodPairingList,
178     236           protocol.MethodPairingRevoke,
179         -         protocol.MethodTeamsList,
```

```
                237 +
                238 +         // Teams — create/delete/update/member management.
180             239           protocol.MethodTeamsCreate,
181             -             protocol.MethodTeamsGet,
182             240           protocol.MethodTeamsDelete,
183             -             protocol.MethodTeamsTaskList,
184             -             protocol.MethodTeamsTaskGet,
185             -             protocol.MethodTeamsTaskComments,
186             -             protocol.MethodTeamsTaskEvents,
                241 +         protocol.MethodTeamsUpdate,
                242 +         protocol.MethodTeamsMembersAdd,
                243 +         protocol.MethodTeamsMembersRemove,
                244 +         protocol.MethodTeamsTaskDelete,
                245 +         protocol.MethodTeamsTaskDeleteBulk,
                246 +
                247 +         // Tenants — write paths.
                248 +         "tenants.create",
                249 +         "tenants.update",
                250 +         "tenants.users.add",
                251 +         "tenants.users.remove",
                252 +
                253 +         // API keys expose secret material — gate list + mutations as admin.
187             254           protocol.MethodAPIKeysList,
188             255           protocol.MethodAPIKeysCreate,
189             256           protocol.MethodAPIKeysRevoke,
                257 +
                258 +         // Skills (can rewrite agent behavior).
190             259           protocol.MethodSkillsUpdate,
                260 +
                261 +         // Heartbeat — any write/test path (closes CVE #866 step 2 + step 4).
                262 +         protocol.MethodHeartbeatSet,
                263 +         protocol.MethodHeartbeatToggle,
                264 +         protocol.MethodHeartbeatTest,
                265 +         protocol.MethodHeartbeatChecklistSet,
                266 +
                267 +         // Live server logs — data exfiltration risk (closes CVE #866 step 3).
                268 +         protocol.MethodLogsTail,
                269 +
                270 +         // Hooks mutations (the handler middleware also enforces this).
                271 +         protocol.MethodHooksCreate,
```

```
                +         protocol.MethodHooksUpdate,
                +         protocol.MethodHooksDelete,
                +         protocol.MethodHooksToggle,
                +
                +         // Voice catalogue refresh touches provider credentials.
                +         protocol.MethodVoicesRefresh,
                +
                +         // TTS config mutations touch provider credentials / global state.
                +         protocol.MethodTTSEnable,
                +         protocol.MethodTTSDisable,
                +         protocol.MethodTTSSetProvider,
191  283        }
192  284        return slices.Contains(adminMethods, method)
193  285    }
194  286
195  287    func isWriteMethod(method string) bool {
196       -     writePrefixes := []string{
                +     writeExact := []string{
197  289            protocol.MethodChatSend,
198  290            protocol.MethodChatAbort,
                +         protocol.MethodChatInject,
199  292            protocol.MethodSessionsDelete,
200  293            protocol.MethodSessionsReset,
201  294            protocol.MethodSessionsPatch,
202  295            protocol.MethodCronCreate,
203  296            protocol.MethodCronUpdate,
204  297            protocol.MethodCronDelete,
205  298            protocol.MethodCronToggle,
206       -         "pairing.",
207       -         "device.pair.",
208       -         "approvals.",
209       -         "exec.approval.",
                +         protocol.MethodCronRun,
210  300            protocol.MethodSend,
                +         protocol.MethodAgentsFileSet,
211  302            protocol.MethodTeamsTaskApprove,
212  303            protocol.MethodTeamsTaskReject,
213  304            protocol.MethodTeamsTaskComment,
214  305            protocol.MethodTeamsTaskCreate,
215  306            protocol.MethodTeamsTaskAssign,
```

```
307 +         protocol.MethodTeamsWorkspaceDelete,
308 +         protocol.MethodHooksTest,
309 +         protocol.MethodPairingRequest,
310 +         protocol.MethodApprovalsApprove,
311 +         protocol.MethodApprovalsDeny,
312 +
313 +         // TTS synthesis — invokes provider API (quota/credentials).
314 +         protocol.MethodTTSConvert,
315 +
316 +         // Browser automation — performs side-effecting actions.
317 +         protocol.MethodBrowserAct,
318 +
319 +         // Channel pairing starts (QR scan flows).
320 +         protocol.MethodZaloPersonalQRStart,
321 +         protocol.MethodWhatsAppQRStart,
216 322       }
217     -     for _, prefix := range writePrefixes {
218     -         if strings.HasPrefix(method, prefix) {
219     -             return true
220     -         }
323 +     return slices.Contains(writeExact, method)
324 + }
325 +
326 + // isReadMethod is the explicit viewer-allowlist for read-only RPCs. Keeping
327 + // this as a closed list (rather than "everything else") is what turns the
328 + // policy into fail-closed. New read RPCs must be added here explicitly.
329 + func isReadMethod(method string) bool {
330 +     readMethods := []string{
331 +         // Agent identity / wait
332 +         protocol.MethodAgent,
333 +         protocol.MethodAgentWait,
334 +         protocol.MethodAgentIdentityGet,
335 +
336 +         // Chat read
337 +         protocol.MethodChatHistory,
338 +         protocol.MethodChatSessionStatus,
339 +
340 +         // Agents read
341 +         protocol.MethodAgentsList,
342 +         protocol.MethodAgentsFileList,
```

```
343 +         protocol.MethodAgentsFileGet,
344 +         protocol.MethodAgentsLinksList,
345 +
346 +         // Sessions read
347 +         protocol.MethodSessionsList,
348 +         protocol.MethodSessionsPreview,
349 +
350 +         // Skills read
351 +         protocol.MethodSkillsList,
352 +         protocol.MethodSkillsGet,
353 +
354 +         // Cron read
355 +         protocol.MethodCronList,
356 +         protocol.MethodCronStatus,
357 +         protocol.MethodCronRuns,
358 +
359 +         // Channels read
360 +         protocol.MethodChannelsList,
361 +         protocol.MethodChannelsStatus,
362 +         protocol.MethodChannelInstancesList,
363 +         protocol.MethodChannelInstancesGet,
364 +
365 +         // Usage / quota
366 +         protocol.MethodUsageGet,
367 +         protocol.MethodUsageSummary,
368 +         protocol.MethodQuotaUsage,
369 +
370 +         // Heartbeat read
371 +         protocol.MethodHeartbeatGet,
372 +         protocol.MethodHeartbeatLogs,
373 +         protocol.MethodHeartbeatChecklistGet,
374 +         protocol.MethodHeartbeatTargets,
375 +
376 +         // Voices
377 +         protocol.MethodVoicesList,
378 +
379 +         // Tenants read
380 +         "tenants.list",
381 +         "tenants.get",
382 +         "tenants.users.list",
```

```
383 +         "tenants.mine",
384 +
385 +         // Teams read
386 +         protocol.MethodTeamsList,
387 +         protocol.MethodTeamsGet,
388 +         protocol.MethodTeamsTaskList,
389 +         protocol.MethodTeamsTaskGet,
390 +         protocol.MethodTeamsTaskGetLight,
391 +         protocol.MethodTeamsTaskComments,
392 +         protocol.MethodTeamsTaskEvents,
393 +         protocol.MethodTeamsTaskActiveBySession,
394 +         protocol.MethodTeamsWorkspaceList,
395 +         protocol.MethodTeamsWorkspaceRead,
396 +         protocol.MethodTeamsEventsList,
397 +         protocol.MethodTeamsKnownUsers,
398 +         protocol.MethodTeamsScopes,
399 +
400 +         // Hooks read
401 +         protocol.MethodHooksList,
402 +         protocol.MethodHooksHistory,
403 +
404 +         // Approvals read-only listing
405 +         protocol.MethodApprovalsList,
406 +
407 +         // TTS read-only (status/providers listing)
408 +         protocol.MethodTTSStatus,
409 +         protocol.MethodTTSProviders,
410 +
411 +         // Browser observation (no side effects)
412 +         protocol.MethodBrowserSnapshot,
413 +         protocol.MethodBrowserScreenshot,
414 +
415 +         // Zalo personal contacts listing
416 +         protocol.MethodZaloPersonalContacts,
221 417       }
222     -     return false
    418 +     return slices.Contains(readMethods, method)
223 419   }
224 420
225 421   // HasMinRole checks if the given role meets the minimum required level.
```

internal/permissions/policy_test.go

```
@@ -1,6 +1,12 @@
package permissions

import (
    "go/ast"
    "go/parser"
    "go/token"
    "path/filepath"
    "runtime"
    "strings"
    "testing"

    "github.com/nextlevelbuilder/goclaw/pkg/protocol"
@@ -147,11 +153,71 @@ func TestCanAccess_ReadMethods_AnyRole(t *testing.T) {
    }
}

-func TestCanAccess_UnknownMethod_DefaultsToViewer(t *testing.T) {
+// TestCanAccess_UnknownMethod_DeniedForAll locks in the fail-closed behavior
+// introduced by issue #866: any method not present in the public / admin /
+// write / read allowlists MUST be denied for every role including owner.
+// This is the inverse of the previous default-permit behavior.
+func TestCanAccess_UnknownMethod_DeniedForAll(t *testing.T) {
    pe := NewPolicyEngine(nil)
-    // Unknown method defaults to viewer role requirement
-    if !pe.CanAccess(RoleViewer, "totally.unknown.method") {
-        t.Fatal("viewer should access unknown method (defaults to viewer)")
+    unknown := "totally.unknown.method"
+    for _, role := range []Role{RoleViewer, RoleOperator, RoleAdmin, RoleOwner} {
+        if pe.CanAccess(role, unknown) {
+            t.Fatalf("%s must NOT access unclassified method %q (fail-closed)", role, unknown)
+        }
+    }
+    if MethodRole(unknown) != RoleNone {
```

```
169 +         t.Fatalf("MethodRole(%q) = %q, want RoleNone", unknown,
    MethodRole(unknown))
170 +     }
171 + }
172 +
173 + // TestCanAccess_CVE866_HeartbeatAndLogs asserts that the three RPCs exploited
174 + // in the issue-#866 chain now require admin. Viewer/operator must be denied.
175 + func TestCanAccess_CVE866_HeartbeatAndLogs(t *testing.T) {
176 +     pe := NewPolicyEngine(nil)
177 +     adminOnly := []string{
178 +         protocol.MethodHeartbeatSet,
179 +         protocol.MethodHeartbeatChecklistSet,
180 +         protocol.MethodLogsTail,
181 +         protocol.MethodHeartbeatToggle,
182 +         protocol.MethodHeartbeatTest,
183 +     }
184 +     for _, method := range adminOnly {
185 +         t.Run(method, func(t *testing.T) {
186 +             if pe.CanAccess(RoleViewer, method) {
187 +                 t.Fatalf("viewer must NOT access %s (CVE #866)", method)
188 +             }
189 +             if pe.CanAccess(RoleOperator, method) {
190 +                 t.Fatalf("operator must NOT access %s (CVE #866)", method)
191 +             }
192 +             if !pe.CanAccess(RoleAdmin, method) {
193 +                 t.Fatalf("admin should access %s", method)
194 +             }
195 +             if !pe.CanAccess(RoleOwner, method) {
196 +                 t.Fatalf("owner should access %s", method)
197 +             }
198 +         })
199 +     }
200 + }
201 +
202 + // TestCanAccess_PublicMethods ensures pre-auth RPCs remain reachable by every
203 + // role (including the zero-value RoleNone placeholder that pre-connect clients
204 + // hold) — otherwise legitimate clients cannot even complete the handshake.
205 + func TestCanAccess_PublicMethods(t *testing.T) {
206 +     pe := NewPolicyEngine(nil)
207 +     public := []string{
```

```
208 +         protocol.MethodConnect,
209 +         protocol.MethodHealth,
210 +         protocol.MethodStatus,
211 +         protocol.MethodBrowserPairingStatus,
212 +     }
213 +     for _, method := range public {
214 +         t.Run(method, func(t *testing.T) {
215 +             for _, role := range []Role{RoleViewer, RoleOperator, RoleAdmin,
      RoleOwner} {
216 +                 if !pe.CanAccess(role, method) {
217 +                     t.Fatalf("%s must access public method %s", role, method)
218 +                 }
219 +             }
220 +         })
155 221       }
156 222   }
157 223
        @@ -241,6 +307,82 @@ func TestValidScope(t *testing.T) {
241 307       }
242 308   }
243 309
310 + // --- MethodApprovalsList: locks in fix for writePrefixes shadowing bug.
311 + // Before the fix, the "exec.approval." prefix in isWriteMethod's writePrefixes
312 + // short-circuited the public→admin→write→read ordering in MethodRole,
313 + // wrongly classifying exec.approval.list as RoleOperator. exec.approval.list
314 + // is an explicit entry in isReadMethod and must resolve to RoleViewer.
315 +
316 + func TestMethodRole_ApprovalsList_IsViewer(t *testing.T) {
317 +     if got := MethodRole(protocol.MethodApprovalsList); got != RoleViewer {
318 +         t.Fatalf("exec.approval.list must be RoleViewer (listed in
      isReadMethod); got %q", got)
319 +     }
320 +     if got := MethodRole(protocol.MethodApprovalsApprove); got != RoleOperator
      {
321 +         t.Fatalf("exec.approval.approve must be RoleOperator; got %q", got)
322 +     }
323 +     if got := MethodRole(protocol.MethodApprovalsDeny); got != RoleOperator {
324 +         t.Fatalf("exec.approval.deny must be RoleOperator; got %q", got)
325 +     }
```

```
326 + }
327 +
328 + // --- Drift coverage: parses pkg/protocol/methods.go at test time, enumerates
329 + // every const Method* = "...", and asserts none resolve to RoleNone. New RPCs
330 + // added without a matching allowlist entry will be caught here before shipping
331 + // as fail-closed rejections in production.
332 +
333 + func TestMethodRole_DriftCoverage_AllProtocolMethodsClassified(t *testing.T) {
334 +     _, thisFile, _, _ := runtime.Caller(0)
335 +     methodsGoPath := filepath.Join(filepath.Dir(thisFile), "..", "..", "pkg",
      "protocol", "methods.go")
336 +
337 +     fset := token.NewFileSet()
338 +     f, err := parser.ParseFile(fset, methodsGoPath, nil, 0)
339 +     if err != nil {
340 +         t.Fatalf("parse %s: %v", methodsGoPath, err)
341 +     }
342 +
343 +     var methods []string
344 +     for _, decl := range f.Decls {
345 +         gd, ok := decl.(*ast.GenDecl)
346 +         if !ok || gd.Tok != token.CONST {
347 +             continue
348 +         }
349 +         for _, spec := range gd.Specs {
350 +             vs, ok := spec.(*ast.ValueSpec)
351 +             if !ok {
352 +                 continue
353 +             }
354 +             for i, name := range vs.Names {
355 +                 if !strings.HasPrefix(name.Name, "Method") {
356 +                     continue
357 +                 }
358 +                 if i >= len(vs.Values) {
359 +                     continue
360 +                 }
361 +                 lit, ok := vs.Values[i].(*ast.BasicLit)
362 +                 if !ok || lit.Kind != token.STRING {
363 +                     continue
364 +                 }
```

```
365 +                 methods = append(methods, strings.Trim(lit.Value, `"`))
366 +             }
367 +         }
368 +     }
369 +
370 +     if len(methods) < 100 {
371 +         t.Fatalf("expected 100+ Method* constants, parser collected %d — methods.go layout may have changed", len(methods))
372 +     }
373 +
374 +     var unclassified []string
375 +     for _, m := range methods {
376 +         if MethodRole(m) == RoleNone {
377 +             unclassified = append(unclassified, m)
378 +         }
379 +     }
380 +     if len(unclassified) > 0 {
381 +         t.Fatalf("RBAC drift — %d protocol method(s) resolve to RoleNone:\n%s\n\nAdd each to isPublicMethod, isAdminMethod, isWriteMethod, or isReadMethod in policy.go.",
382 +             len(unclassified), strings.Join(unclassified, "\n  "))
383 +     }
384 + }
385 +
244 386   // --- MethodScopes: verify pairing/approvals special routes ---
245 387
246 388   func TestMethodScopes_PairingMethod(t *testing.T) {
```

Comments 0

Please sign in to comment.