

projectdiscovery / nuclei Public[Code](#) [Issues](#) 102 [Pull requests](#) 31 [Discussions](#) [Actions](#) [Projects](#)

Commit 6c803c7



dwisiswant0 authored last week · 18 / 18 · Verified

fix(expressions): avoid helper eval in literal checks (#7321)

* fix(expressions): avoid helper eval in literal checks

``hasLiteralsOnly()`` currently evaluates helper expressions while deciding whether "`{{{...}}}`" contains unresolved variables, which makes validation paths run side-effectful helpers.

Just replace that runtime eval with a ``Vars()`` len check so unresolved-variable detection literally stays literal (am I writing it right?), and of course side-effect free.

Also make ``Evaluate()`` return template-authored expression compile/eval errors instead of logging and then keep continuing, so malformed helper calls still fail in the rendering path.

Fixes [#7320](#)

Signed-off-by: Dwi Siswanto <git@dw1.io>

* fix(javascript): avoid nil map panic on arg eval failure

``getArgsCopy()`` logs and counts argument evaluation failures, but continues to store Port in ``argsCopy`` even when ``evaluateArgs()`` returns. This causes malformed JavaScript args like "`{{{base64()}}}`" to panic instead of returning the original error.

Return the evaluation error immediately after reporting it to prevent the panic and make sure proper error propagation.

Signed-off-by: Dwi Siswanto <git@dw1.io>

Signed-off-by: Dwi Siswanto <git@dw1.io>

dev (#7321) · v3.8.0

1 parent 6f2ade6 commit 6c803c7

6 files changed +142 -17 lines changed

↑ Top ⚙️

Filter files...

- pkg/protocols
 - common/expressions
 - expressions.go
 - expressions_test.go
 - variables.go
 - variables_test.go
 - javascript
 - js.go
 - js_test.go

6 files changed +142 -17 lines changed

Search within code

...protocols/common/expressions/expressions.go

```

... @@ -1,10 +1,10 @@
1 1 package expressions
2 2
3 3 import (
4 + "fmt"
4 5 "strings"
5 6
6 7 "github.com/Knetic/govaluate"
7 - "github.com/projectdiscovery/gologger"
8 8
9 9 "github.com/projectdiscovery/nuclei/v3/pkg/operators/common/dsl"
10 10 "github.com/projectdiscovery/nuclei/v3/pkg/protocols/common/marker"
... @@ -53,28 +53,40 @@ func evaluate(data string, base map[string]interface{})
... (string, error) {
53 53 // - complex: containing helper functions [ + variables]

```

```

54 54 // literals like {{2+2}} are not considered expressions
55 55 for _, expression := range expressions {
56 +   originalExpression := expression
56 57 // replace variable placeholders with base values
57 58   expression = replacer.Replace(expression, base)
59 +
58 60 // turns expressions (either helper functions+base values or base
    values)
59 61   compiled, err :=
    govaluate.NewEvaluableExpressionWithFunctions(expression, dsl.HelperFunctions)
60 62   if err != nil {
61 -     gologger.Warning().Msgf("Failed to compile expression '%s': %v",
    expression, err)
62 -     continue
63 -   }
64 -   // propagate unresolved {...} markers from variable values so the
65 -   // downstream ContainsUnresolvedVariables check can detect them instead
66 -   // of having encoding functions (e.g. base64) hide them
67 -   if markers := unresolvedVarMarkers(compiled.Vars(), base); markers != ""
    {
68 -     data = replacer.ReplaceOne(data, expression, markers)
69 -     continue
63 +     return data, fmt.Errorf("failed to compile expression %q: %w",
    originalExpression, err)
70 64   }
65 +
71 66   result, err := compiled.Evaluate(base)
72 67   if err != nil {
73 -     gologger.Warning().Msgf("Failed to evaluate expression '%s': %v",
    expression, err)
74 -     continue
68 +     return data, fmt.Errorf("failed to evaluate expression %q: %w",
    originalExpression, err)
75 69   }
70 +
71 +   replacement := result
72 +   // Preserve unresolved markers only when a helper call would otherwise
73 +   // hide them from downstream validation. Plain expressions such as
74 +   // comparisons should evaluate normally.

```

```

75 +         if markers := unresolvedVarMarkers(compiled.Vars(), base); markers != ""
      {
76 +             usesFunctions := false
77 +             for _, token := range compiled.Tokens() {
78 +                 if token.Kind == govaluate.FUNCTION {
79 +                     usesFunctions = true
80 +                     break
81 +                 }
82 +             }
83 +             if usesFunctions && ContainsUnresolvedVariables(fmt.Sprint(result))
      == nil {
84 +                 replacement = markers
85 +             }
86 +         }
87 +
76 88         // replace incrementally
77 -         data = replacer.ReplaceOne(data, expression, result)
89 +         data = replacer.ReplaceOne(data, expression, replacement)
78 90     }
79 91     return data, nil
80 92 }

```



...cols/common/expressions/expressions_test.go



```

@@ -105,3 +105,50 @@ func TestEvaluateDoesNotReinterpretResolvedValues(t
 *testing.T) {

```

```

105 105     })
106 106     }
107 107 }
108 +
109 + func TestEvaluateDoesNotExecuteHelpersFromResolvedValues(t *testing.T) {
110 +     var calls int
111 +
112 +     withTestHelperFunction(t, "test_side_effect", func(args ...interface{})
      (interface{}, error) {
113 +         calls++
114 +         return "ok", nil
115 +     })
116 +
117 +     value, err := Evaluate("#{body}", map[string]interface{}{

```

```
118 +     "body": "{{test_side_effect(1)}}",
119 + })
120 + require.NoError(t, err)
121 + require.Equal(t, "{{test_side_effect(1)}}", value)
122 + require.Zero(t, calls)
123 + }
124 +
125 + func TestEvaluateReturnsErrorForInvalidTemplateExpression(t *testing.T) {
126 +     _, err := Evaluate("{{base64()}}", map[string]interface{}{})
127 +     require.Error(t, err)
128 +     require.ErrorContains(t, err, `failed to evaluate expression "base64()"`)
129 + }
130 +
131 + func TestEvaluateErrorDoesNotLeakResolvedValues(t *testing.T) {
132 +     _, err := Evaluate("{{base64('{{secret_token}}', 'extra')}}",
133 +         map[string]interface{}{
134 +             "secret_token": "top-secret-cia-mi6-kgb-mossad-classified",
135 +         })
136 +     require.Error(t, err)
137 +     require.ErrorContains(t, err, `failed to evaluate expression
138 + "base64('{{secret_token}}', 'extra')"`)
139 +     require.NotContains(t, err.Error(), "top-secret-cia-mi6-kgb-mossad-
140 + classified")
141 + }
142 +
143 + func TestEvaluatePlainExpressionsWithMarkerLikeValues(t *testing.T) {
144 +     value, err := Evaluate("{{body != ''}}", map[string]interface{}{
145 +         "body": "{{contact_id}}",
146 +     })
147 +     require.NoError(t, err)
148 +     require.Equal(t, "true", value)
149 + }
150 +
151 + func TestEvaluatePreservesVisibleMarkersFromHelperResults(t *testing.T) {
152 +     value, err := Evaluate("{{concat(body, '-x')}}", map[string]interface{}{
153 +         "body": "{{contact_id}}",
154 +     })
155 +     require.NoError(t, err)
156 +     require.Equal(t, "{{contact_id}}-x", value)
157 + }
```

```

  pkg/protocols/common/expressions/variables.go
  @@ -120,9 +120,10 @@ func hasLiteralsOnly(data string) bool {
120 120     if err != nil {
121 121         return false
122 122     }
123 123     - if expr != nil {
124 124     -     _, err = expr.Evaluate(nil)
125 125     -     return err == nil
123 123     + if expr == nil {
124 124     +     return true
125 125     +
126 126     }
127 127     - return true
127 127     +
128 128     + return len(expr.Vars()) == 0
128 129     }

```

```

  ...tocols/common/expressions/variables_test.go
  @@ -4,9 +4,26 @@ import (
4 4     "errors"
5 5     "testing"
6 6
7 7     + "github.com/Knetic/govaluate"
8 8     + "github.com/projectdiscovery/nuclei/v3/pkg/operators/common/dsl"
7 9     "github.com/stretchr/testify/require"
8 10    )
9 11
12 12    + func withTestHelperFunction(t *testing.T, name string, fn
    govaluate.ExpressionFunction) {
13 13        + t.Helper()
14 14        +
15 15        + originalFn, hadFn := dsl.HelperFunctions[name]
16 16        + dsl.HelperFunctions[name] = fn
17 17        +
18 18        + t.Cleanup(func() {
19 19            + if hadFn {
20 20                + dsl.HelperFunctions[name] = originalFn
21 21            + return

```

```

22 +     }
23 +     delete(dsl.HelperFunctions, name)
24 + })
25 + }
26 +
10 27     func TestUnresolvedVariablesCheck(t *testing.T) {
11 28         tests := []struct {
12 29             data string
@@ -26,3 +43,15 @@ func TestUnresolvedVariablesCheck(t *testing.T) {
26 43             require.Equal(t, test.err, err, "could not get unresolved variables")
27 44         }
28 45     }
46 +
47 + func TestUnresolvedVariablesCheckDoesNotExecuteHelpers(t *testing.T) {
48 +     var calls int
49 +     withTestHelperFunction(t, "test_side_effect", func(args ...interface{})
(interface{}, error) {
50 +         calls++
51 +         return "ok", nil
52 +     })
53 +
54 +     err := ContainsUnresolvedVariables("{{test_side_effect(1)}}")
55 +     require.NoError(t, err)
56 +     require.Zero(t, calls)
57 + }

```

▼ pkg/protocols/javascript/js.go

...

```

@@ -701,6 +701,7 @@ func (request *Request) getArgsCopy(input
*contextargs.Context, payloadValues ma
701 701     if err != nil {
702 702         requestOptions.Output.Request(requestOptions.TemplateID,
input.MetaInput.Input, request.Type().String(), err)
703 703         requestOptions.Progress.IncrementFailedRequestsBy(1)
704 +     return nil, err
704 705     }
705 706     // "Port" is a special variable that is considered as network port
706 707     // and is conditional based on input port and default port specified in
input

```

```

  pkg/protocols/javascript/js_test.go
  @@ -9,8 +9,11 @@ import (
    9     9     "github.com/projectdiscovery/nuclei/v3/pkg/catalog/config"
    10    10     "github.com/projectdiscovery/nuclei/v3/pkg/catalog/disk"
    11    11     "github.com/projectdiscovery/nuclei/v3/pkg/loader/workflow"
    12    +     "github.com/projectdiscovery/nuclei/v3/pkg/output"
    12    13     "github.com/projectdiscovery/nuclei/v3/pkg/progress"
    13    14     "github.com/projectdiscovery/nuclei/v3/pkg/protocols"
    15    +     "github.com/projectdiscovery/nuclei/v3/pkg/protocols/common/contextargs"
    16    +     javascript "github.com/projectdiscovery/nuclei/v3/pkg/protocols/javascript"
    14    17     "github.com/projectdiscovery/nuclei/v3/pkg/templates"
    15    18     "github.com/projectdiscovery/nuclei/v3/pkg/testutils"
    16    19     "github.com/projectdiscovery/ratelimit"
  @@ -73,3 +76,35 @@ func TestCompile(t *testing.T) {
    73    76     }
    74    77     }
    75    78     }
    79    +
    80    + func TestExecuteWithResultsReturnsArgEvaluationErrorWithoutPanic(t *testing.T)
    81    + {
    82    +     options := testutils.DefaultOptions
    83    +     tmplInfo := &testutils.TemplateInfo{ID: "execute-with-results-arg-
    84    +         evaluation-error"}
    85    +
    86    +     testutils.Init(options)
    87    +
    88    +     t.Cleanup(func() {
    89    +         testutils.Cleanup(options)
    90    +     })
    91    +
    92    +     executorOptions := testutils.NewMockExecuterOptions(options, tmplInfo)
    93    +     executorOptions.JsCompiler = templates.GetJsCompiler()
    94    +
    95    +     request := &javascript.Request{
    96    +         Args: map[string]interface{}{
    97    +             "token": "{{base64()}}",
    98    +         },
    99    +         Code: `module.exports = { success: true, response: "ok" }`,
  
```

```
98 +   }
99 +   require.NoError(t, request.Compile(executorOptions))
100 +
101 +   target := contextargs.NewWithInput(context.Background(),
102 +     "https://example.com:443")
103 +
104 +   var err error
105 +   require.NotPanics(t, func() {
106 +     err = request.ExecuteWithResults(target, nil, nil,
107 +     func(*output.InternalWrappedEvent) {
108 +       t.Fatal("unexpected callback on argument evaluation failure")
109 +     })
110 +   })
111 +   require.ErrorContains(t, err, `failed to evaluate expression "base64()`)
112 + }
```

Comments 0



Please [sign in](#) to comment.