



2nd Annual
MidPoint Community Meetup

Secure by Design: MidPoint Expression Language

Agenda

- Customization problem (Open-closed principle)
- Danger of Groovy/Python/JavaScript
- Security by design
- MidPoint expression language
- Plan



Customization Problem

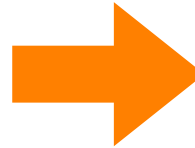
- We want midPoint to be **flexible** and **customizable**
i.e. we want midPoint to be *open*
- We do **not want to change** midPoint source code
i.e. we want midPoint to be *closed*

Open-closed principle (Bertrand Meyer, 1988)

Customization Problem

- We want midPoint to be **flexible** and **customizable**
i.e. we want midPoint to be *open*
- We do **not want to change** midPoint source code
i.e. we want midPoint to be *closed*

Open-closed principle (Bertrand Meyer, 1988)

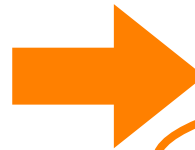


- **Configuration properties**
handle *predictable* customization
- **Expressions**
handle *unpredictable* customization

Customization Problem

- We want midPoint to be **flexible** and **customizable**
i.e. we want midPoint to be *open*
- We do **not want to change** midPoint source code
i.e. we want midPoint to be *closed*

Open-closed principle (Bertrand Meyer, 1988)



- **Configuration properties**
handle *predictable* customization
- **Expressions**
handle *unpredictable* customization

Expressions

- Magic dust that provides ultimate flexibility
- Scripting expressions:
Heroes that saved a lot of projects
- Groovy as primary scripting language
since midPoint 2.0 (2012)

```
givenName + ' ' + familyName

jobCode == 'X007'

'X' + input

basic.composeDnWithSuffix('uid', ...)

for ( a in focus.assignment ) {
    if (a.targetRef.type == ... ) {
        ...
    }
}
```

Dangers of Groovy/Python/JavaScript

- This magic dust provides a bit too much flexibility
- Groovy: built-in remote code execution (RCE) vulnerability
- “Industry standard” approach: ignore the problem
- Sandboxing does not really work
 - *default permit vs default deny* problem
 - upgrade problem
- Scripting languages are **not** designed for security

```
System.exit(0)
```

```
"${System.exit(0)}"
```

```
new File('/etc/passwd').text
```

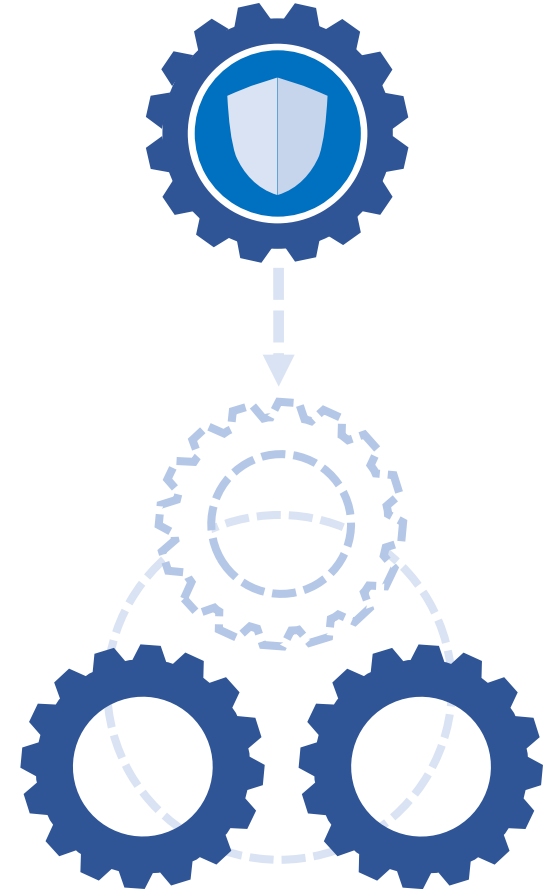
```
'rm -rf /'.execute()
```

```
['rm', '-rf', '/'].execute()
```

... and access to everything in Java runtime, all libraries, reflection ...
everything

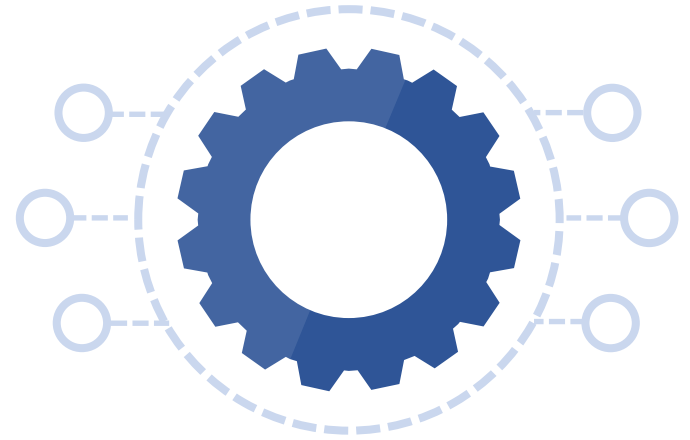
Security by Design

- Security cannot be an afterthought, it has to be integral part of the design
- Security is much more than “no vulnerabilities”
- We need language which is **designed to be secure**
- We were not overwhelmed by our choices
 - Rich languages, not secure (Groovy, Python, etc.)
 - Safe languages, not powerful (Liquid, Jakarta EL, etc.)
- Winner: Common expression language (CEL) by Google et al.



MidPoint Expression Language (MEL)

- Heavily based on Common expression language (CEL)
- *Expression language: functional* rather than imperative
- Designed for security/safety
- CEL is more what you'd call "framework" than actual language
- MEL = CEL engine + midPoint extensions



```
focus.givenName + ' ' + focus.familyName  
  
focus.activation.effectiveStatus == 'enabled'  
  
matches(focus.telephoneNumber, '^[\d-9]+$')
```

MEL Expressions in MidPoint

```
<expression>
  <script>
    <language>http://midpoint.evolveum.com/.../language#mel</language>
    <code>
      focus.givenName + ' ' + focus.familyName
    </code>
  </script>
</expression>
```

MEL Basics

Structured data

```
focus.givenName  
focus.activation.effectiveStatus  
focus.extension.jobCode
```

Operators

```
+ - * / % ! == != < <= > >= && || () ?: in . .? [] [?]
```

Functions (selection)

```
size(x)                s.contains(substring)  
matches(s, regexp)    s.trim()  
isEmpty(s)            list.join(separator)
```

MEL Examples

Username generator

```
focus.givenName.norm.substring(0,1) + focus.familyName.norm
```

Role autoassignment expression

```
jobCode == 'X007'
```

Name formatting (e.g. custom column)

```
'%s, %s'.format([focus.familyName, focus.givenName])
```

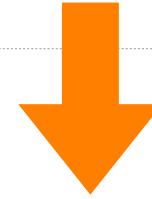
Telephone number validator

```
matches(focus.telephoneNumber, '^[\d-9]+$')
```

Expression Language

- MEL is an *expression* language
- Goal: return a value
- MEL is not a sequence of commands (not *imperative*)
- Evaluating values and functions (*functional* language)
- Functions are free of side effects (no “writes”)
- MEL has limitations, but it is still powerful

```
command;  
command;  
command;  
command;
```



```
func(func(func(func())))
```

Conditional

```
if (input == null) {  
  return 'foobar'  
} else {  
  return input  
}
```



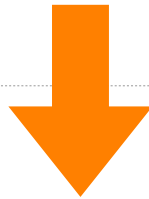
```
isNull(input) ? 'foobar' : input
```



```
default(input, 'foobar')
```

No Loops

```
for ( a in focus.assignment ) {  
    if (a.targetRef.type == PolicyType.COMPLEX_TYPE) {  
        return a.targetRef.oid  
    }  
}
```



```
focus.assignment  
    .filter(a, a.targetRef.type == 'PolicyType' )  
    .map(a, a.targetRef.oid)
```

MEL Extensions

Format

```
format.concatName([focus.givenName, focus.familyName])  
format.strftime(timestamp, '%d/%m/%Y %H:%M:%S')
```

Log

```
log.debug('User {} was inactivated due to {}'.format(focus.name, reason))
```

MidPoint objects

```
projection.primaryIdentifiers()
```

MidPoint functions

```
midpoint.getObject('UserType', oid)
```

MEL Security

- Does **not** have access to any Java runtime functionality, except those explicitly exposed through extensions.
- No access to files, network, processes, OS, midPoint internals, ...
- No direct way to cripple (DoS) the system, e.g. no infinite loops
- Secure enough for delegation, e.g. role auto-assignment, role engineering, reports, GUI customization
- Expression profiles are properly applied to allow/deny MEL extensions, e.g. `midpoint.*` and `secret.*` extensions denied by default
- MEL is secure in itself, yet expression result may still do damage (!!!) e.g. autoassign privileged role to all users



Remarks and Drawbacks

- Polystrings work as strings
- QNames are usually not needed
- Nulls and “optionals” are quite painful in CEL
- **focus.?activation.?activationStatus**
- **isNull(x)** rather than **x == null**
- **stringify()** and **default()** to the rescue



MEL Reality Check

Username generator

```
stringify(focus.?givenName.?norm).replace(' ', '').substring(0,1) +  
stringify(focus.?familyName.?norm).replace(' ', '').substring(0,7) +  
iterationToken
```

Value presence check

```
isPresent(focus.?activation.?administrativeStatus)
```

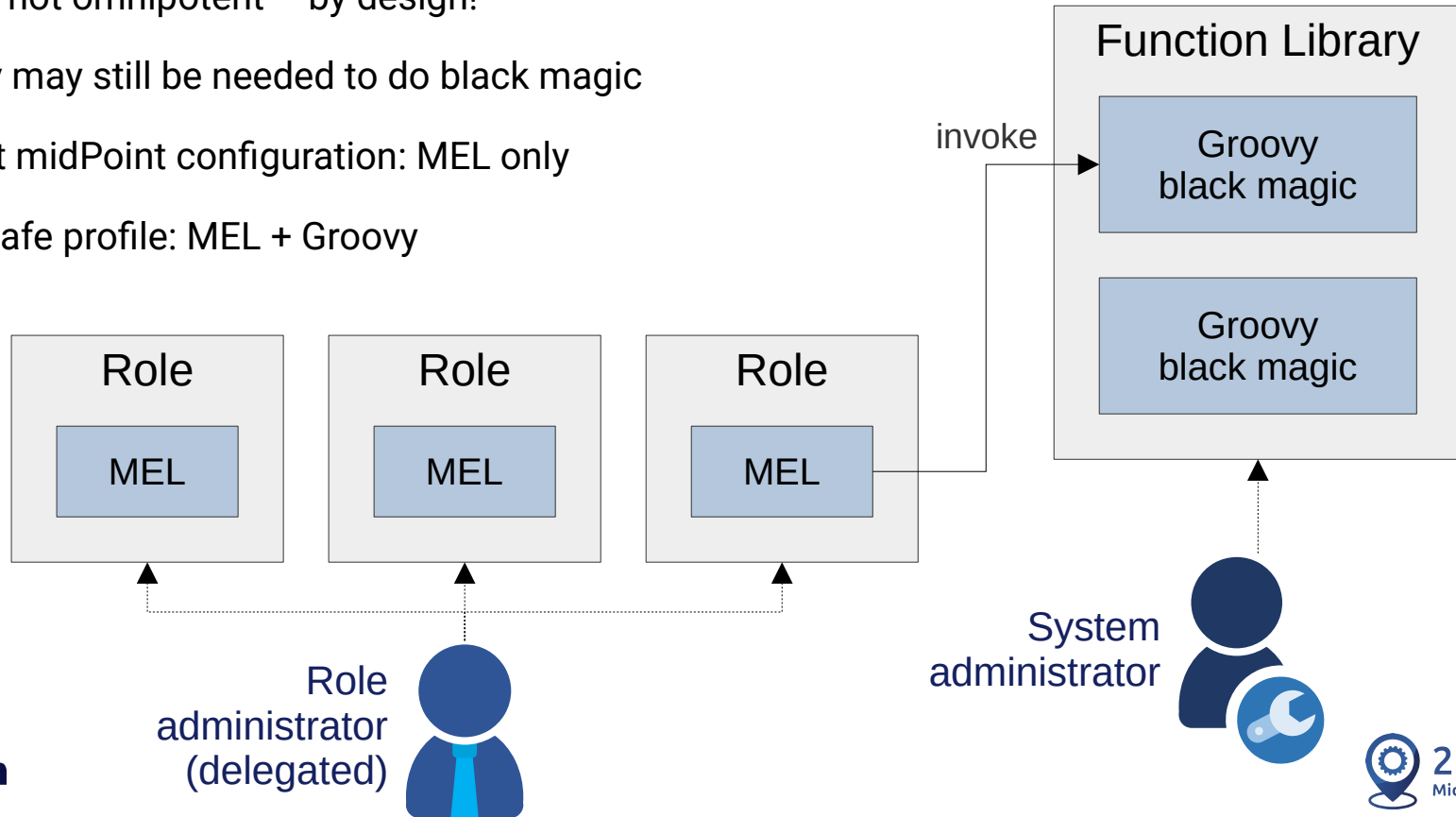
Null-safe string formatting

```
'resource: %s'.format(resource.?name)
```

```
'resource: ' + default(resource.?name, 'null')
```

MEL and Groovy

- MEL is not omnipotent – by design!
- Groovy may still be needed to do black magic
- Default midPoint configuration: MEL only
- Semi-safe profile: MEL + Groovy



Documentation, etc.

- MEL Documentation:

docs.evolveum.com → midPoint reference documentation → development branch

<https://docs.evolveum.com/midpoint/reference/master/expressions/expressions/script/mel/>

- AI Chatbots

They know CEL, they work fine

Explicitly include documentation link in the prompt to let them know about MEL extensions
(at least for now)

Plan

- MEL will be available in midPoint 4.11
- Baseline MEL implementation & documentation is done
- Some parts still missing (e.g. audit records, deltas, GUI)
- Integration with midPilot
- Testing and gathering feedback
- Groovy may still remain a default scripting language in midPoint 4.11
- This plan is subject to change



Conclusion

- Cybersecurity: more important than ever
- MidPoint: secure by design
- Our commitment to advance cybersecurity
- MidPoint expression language (MEL)
- Designed to be secure
- Availability: midPoint 4.11



Cybersecurity made in Europe

Evolveum

Thank you for your attention

Feel free to ask your questions now!



2nd Annual MidPoint Community Meetup