Deng Chen, Yan-duo Zhang, Wei Wei, Shi-xun Wang, Ru-bing Huang, Xiao-lin Li, Bin-bin Qu, Sheng Jiang, 2017. Efficient vulnerability detection based on an optimized rule-checking static analysis technique. *Frontiers of Information Technology & Electronic Engineering*,**18**(3):332-345. http://dx.doi.org/10.1631/FITEE.1500379

Efficient vulnerability detection based on an optimized rule-checking static analysis technique

Key words: Rule-based static analysis; Software quality; Software validation; Performance improvement

Corresponding author: Deng Chen
E-mail: chendeng8899@hust.edu.cn
ORCID: http://orcid.org/ 0000-0001-6359-801X

Motivation

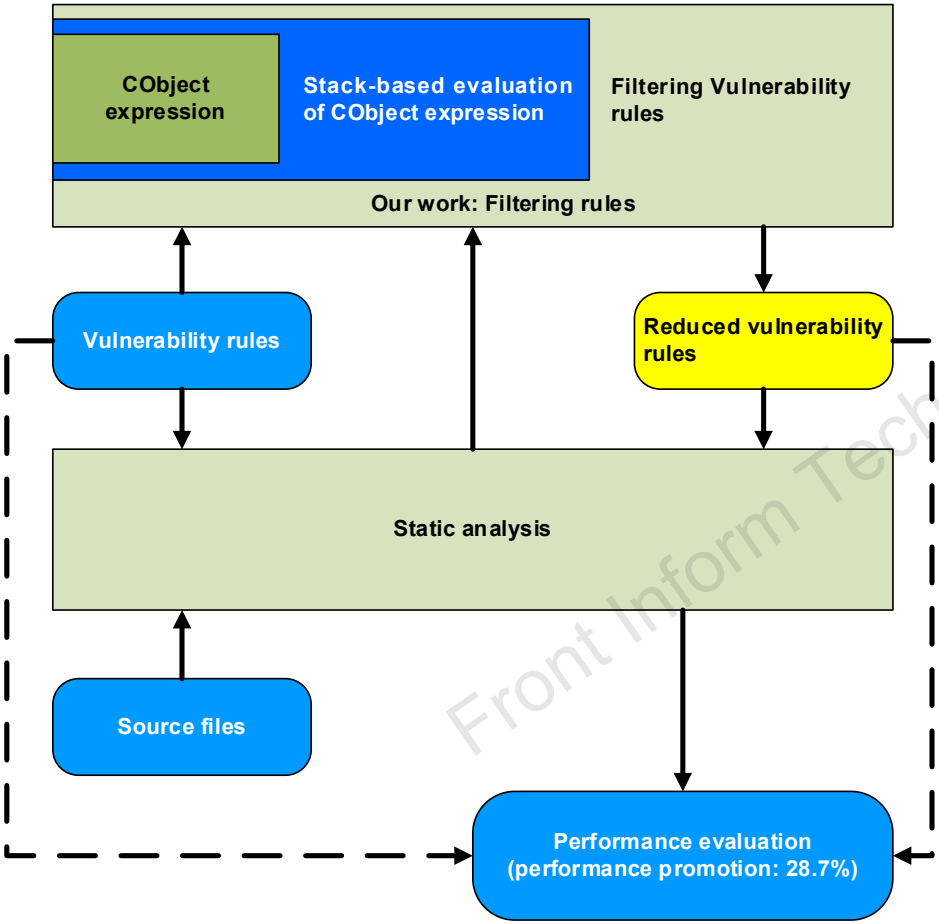
- Interactive usage of program static analysis has a high performance requirement.
- Rule-checking algorithm is crucial for the performance of rule-based static analysis tools, which always considers many redundant vulnerability rules.

Main idea

Filtering Vulnerability Rules

- CObject expression
- Stack-based evaluation of CObject expression
- Filtering vulnerability rules

Filtering vulnerability rules



1. Composing CObject expression

```
<![CDATA[
//TypeDeclaration/ClassOrInterfaceDeclaration
[
   (
      (./ImplementsList/ClassOrInterfaceType
      [ends-with(@Image,'SessionBean')]) or
      (./ImplementsList/ClassOrInterfaceType
      [ends-with(@Image,'MessageDrivenBean')])
   )
   and not (ends-with(@Image,'Bean'))
]]>
```

$or(exist(SessionBean), exist(MessageDrivenBean))$

2. Evaluation of CObject expression

$evaluate(CObj(r), f)evaluate(CObj(r)(f))$ → True

$evaluate(exist(\lambda, f))evaluate(exist(\lambda, P_f))$ → FALSE

3. Filtering vulnerability rules

Experimental results

We implemented our technique in an open source static analyzer PMD and used it to conduct comparison test. Experimental results show that our approach can improve the static analysis perform by 28.7%.

Table 3 Results of our comparison test with EPMD being run five times for each group of subjects and rule sets

Subject program	Rule set	Execution time (s)					AVG (s)	RDT (s)	RDT ratio
		1	2	3	4	5			
Eclipse	RS	469.8	437.4	444.0	438.9	467.1	451.4	70.4	15.6%
	RS*	380.3	380.2	380.3	382.9	381.6	381.1		
ElasticSearch	RS	40.7	36.9	37.1	36.7	39.7	38.2	6.6	17.3%
	RS*	31.5	31.7	31.7	31.6	31.5	31.6		
PMD	RS	9.5	6.6	6.7	6.6	8.6	7.6	3.4	45.3%
	RS*	4.1	4.2	4.1	4.2	4.2	4.2		
Tomcat	RS	15.5	14.8	14.6	14.6	15.2	14.9	5.4	36.4%
	RS*	9.6	9.5	9.3	9.6	9.5	9.5		
SQuirreL SQL Client	RS	45.2	31.2	31.4	48.1	31.1	37.4	11.1	29.6%
	RS*	26.4	26.5	26.2	25.8	26.8	26.3		
Jboss	RS	57.8	49.0	49.2	66.3	49.6	54.4	15.4	28.2%
	RS*	38.8	39.2	39.0	39.0	39.1	39.0		

AVG: average execution time of EMPD; RDT: average reduction of execution time

Conclusions

We proposed an optimized rule-checking algorithm that can improve the performance of static analysis tools without side effects on their detection capability and precision. Our rule-checking algorithm filters unnecessary vulnerability rules automatically by evaluating their CObject expressions. our method is a general approach that is applicable for most of mainstream programming languages and other types of vulnerability rules. We performed a comparison test and found that our technique can achieve an average performance promotion by 28.7%.