SOCC ACM Symposium on Cloud Computing

Automatically Detecting Risky Scripts in Infrastructure Code

Ting Dai, Alexei Karve, Grzegorz Koper, Sai Zeng

IBM Research

IBM

Scripts in Infrastructure code

- Modern Infrastructure-as-Code (IaC) tools support **embedded scripting** languages such as **Shell** and **PowerShell** to manage infrastructure resources and interact with applications to execute automation procedures.

- Risky patterns in infrastructure scripts introduce bugs and expose vulnerabilities leading to widespread of
  - Business disruptions, e.g., Remove-Partition -DriveLetter 'C'
  - Application performance degradations,
  - Infrastructure with vulnerabilities.

Research Challenges

- Existing techniques and practices for checking risky patterns in IaC embedded scripts are rudimentary.

- Industry shifts to community-based approach
  - A team of contributors have mixed skills, experiences, and responsibilities.
  - Most contributors are system administrators (SAs) who lack the same level of understanding and debugging support vs software developers.
  - Nearly **75%** of system downtime is caused by human errors [2]. Many service outage incidents are caused by mistakes made by SAs [1].

Risky Infrastructure Script Incidents

Amazon service outage

Amazon S3 service became unavailable due to a **removal cmd** invoked by an SA who inadvertently removed a large set of servers. The service disruption lasted for **5 hours** with financial loss of **$150 million.**

Delete database accidently

A risky script accidently deleted the system directories. Mistakenly used **//** as the comment symbol (should be **#**).

```
#!/bin/sh
OUT_DIR=/data/backup/mongod/tmp    // bakup folder
rm -rf $OUT_DIR/*                  // delete tmp folder
```

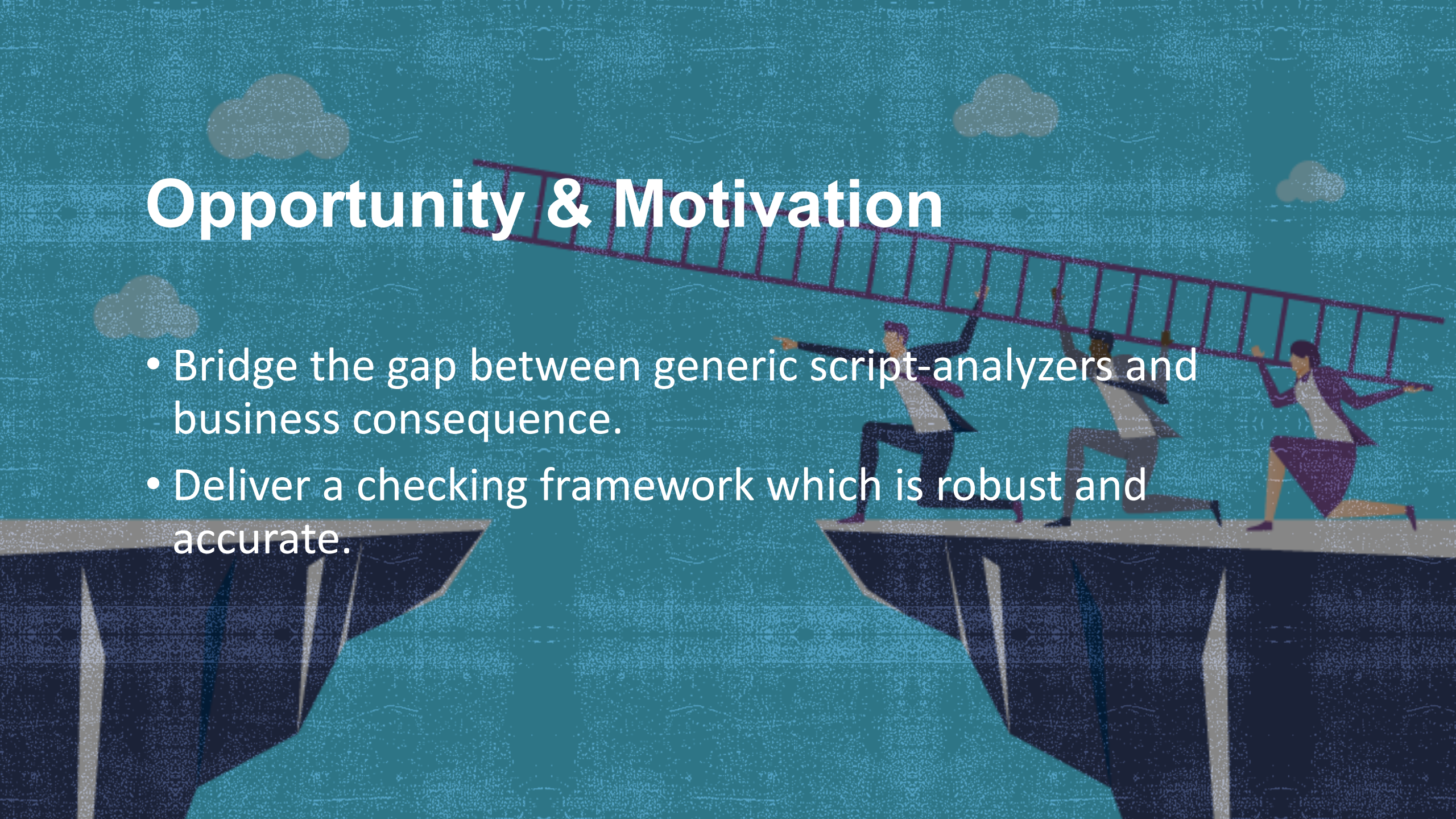
System directory

Qid: 55323391

State-of-the-Art

- Existing infrastructure code linters **lack** the capability of checking **IaC embedded scripts**, e.g., Shell, PowerShell in Ansible playbooks or Chef cookbooks
  - Such as Ansible-lint [13], Puppet-lint [15], SLIC [38], FSMoVe [41]

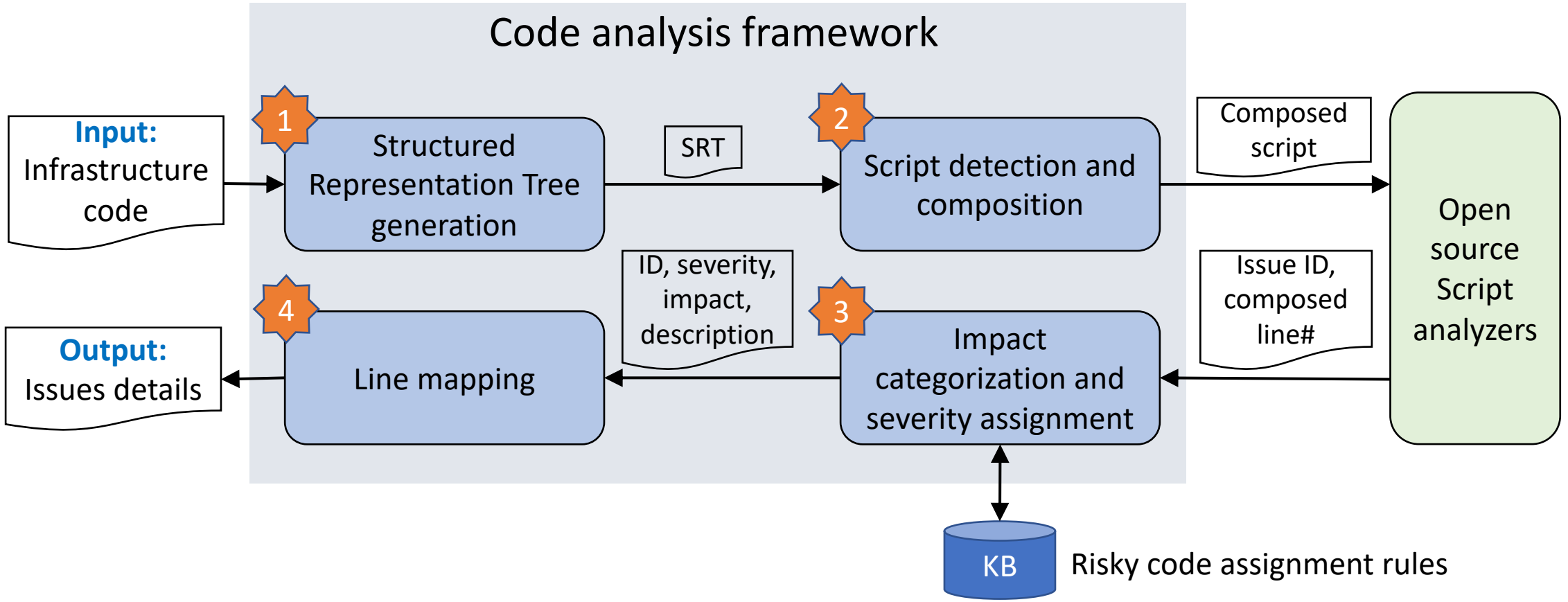
- Generic script-analyzers, e.g., Shellcheck [29], PSScriptAnalyzer [35], report issues in the scripts by checking their formats and syntaxes.
  - False positives and false negatives
  - Without correlating the identified issues with their risky behaviors
    - **How** the risks **manifest** in the production environment;
    - **What** the potential business **consequences** of the risks have;
    - **How severe** the negative consequences are.



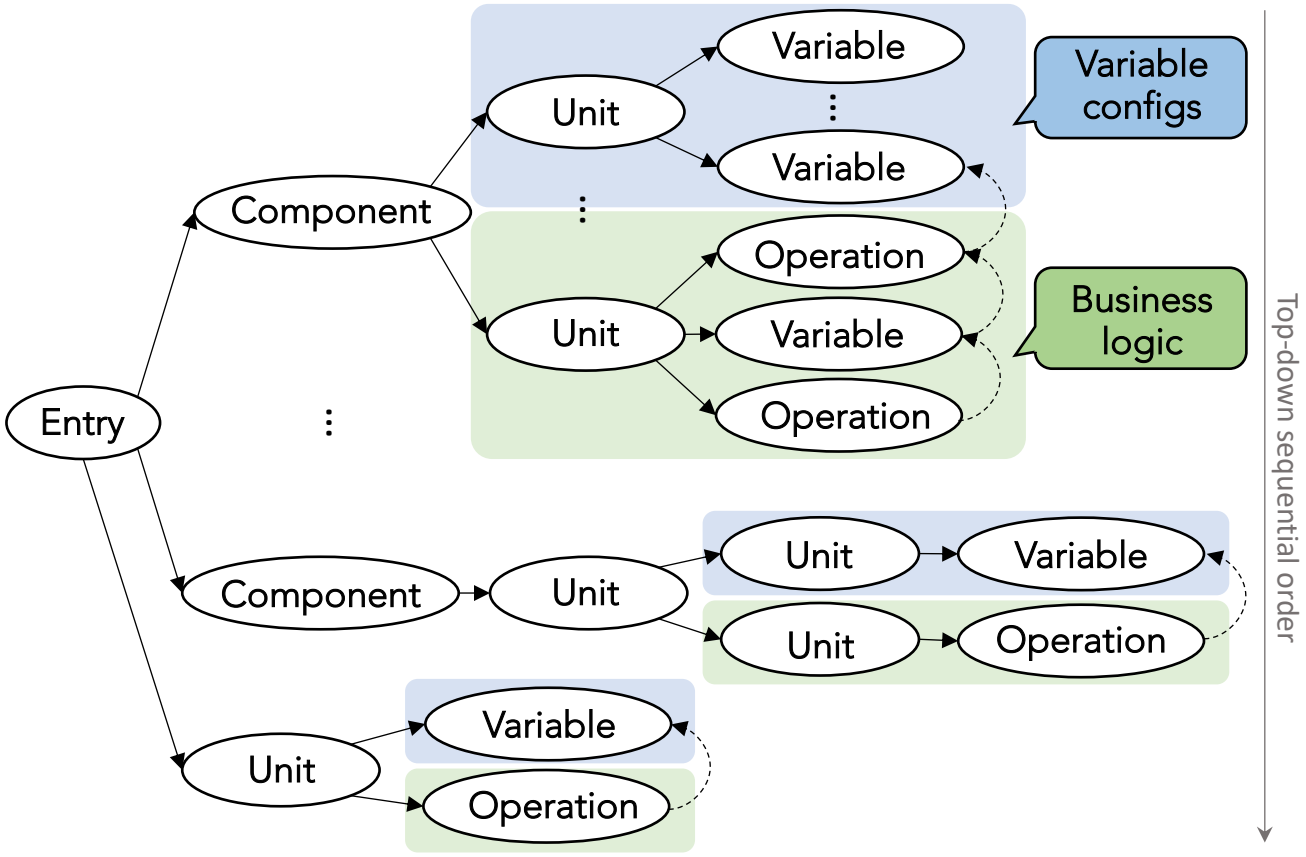
Opportunity & Motivation

- Bridge the gap between generic script-analyzers and business consequence.
- Deliver a checking framework which is robust and accurate.

Infrastructure Scripts Analysis Framework



Structure Representation Tree



Infrastructure code example

- **An Ansible playbook directory**

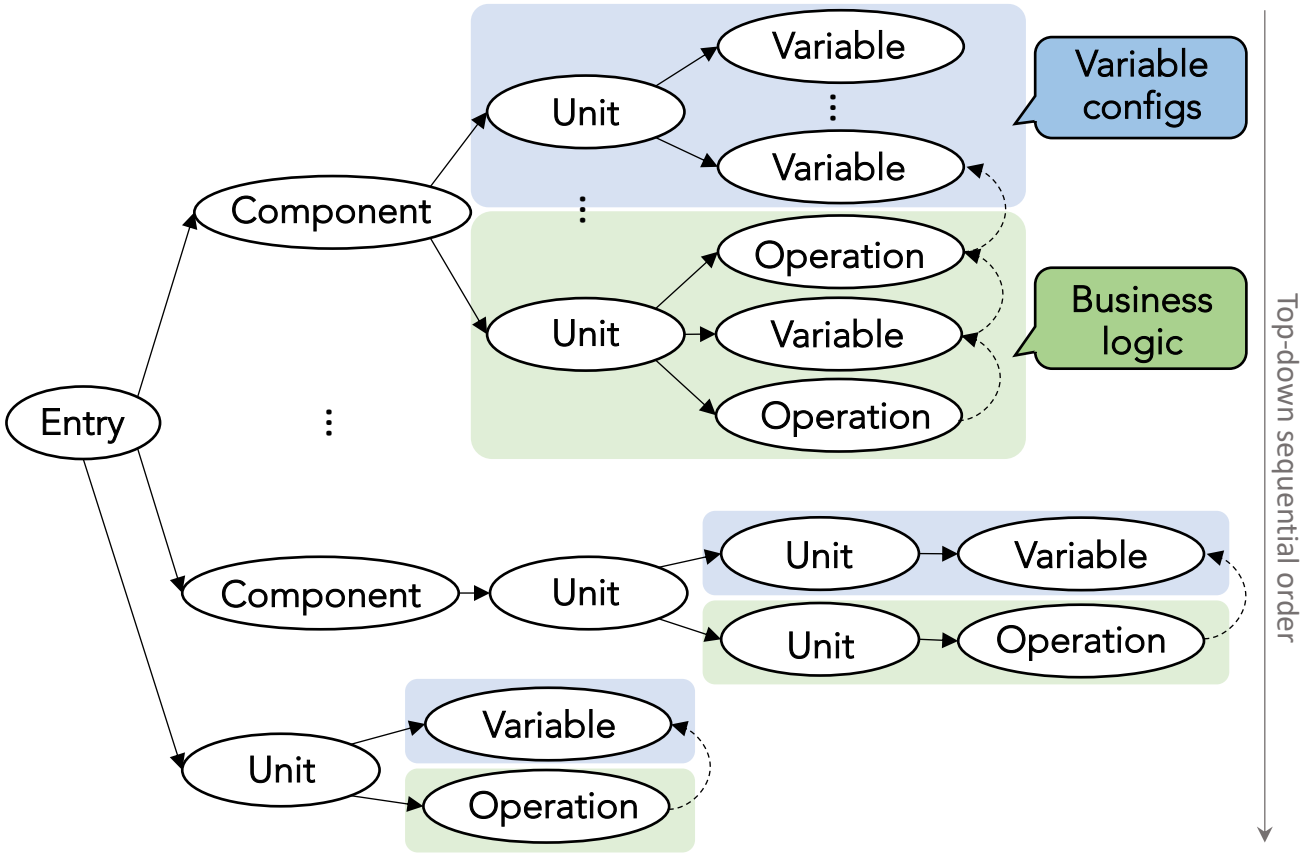
```
.
├── README.md
├── main.yml
└── roles
    └── backup_missed_unix
        ├── tasks
        │   ├── agent_dsmc.yml
        │   ├── check_agent.yml
        │   ├── delete_rst_sched.yml
        │   ├── kill_agent.yml
        │   ├── main.yml
        │   ├── rerun_bkp.yml
        │   ├── set_pass.yml
        │   └── unlock_node.yml
        └── vars
            └── main.yml
```

Entry · Component · Business logic units · Variable config unit

- **An operation**

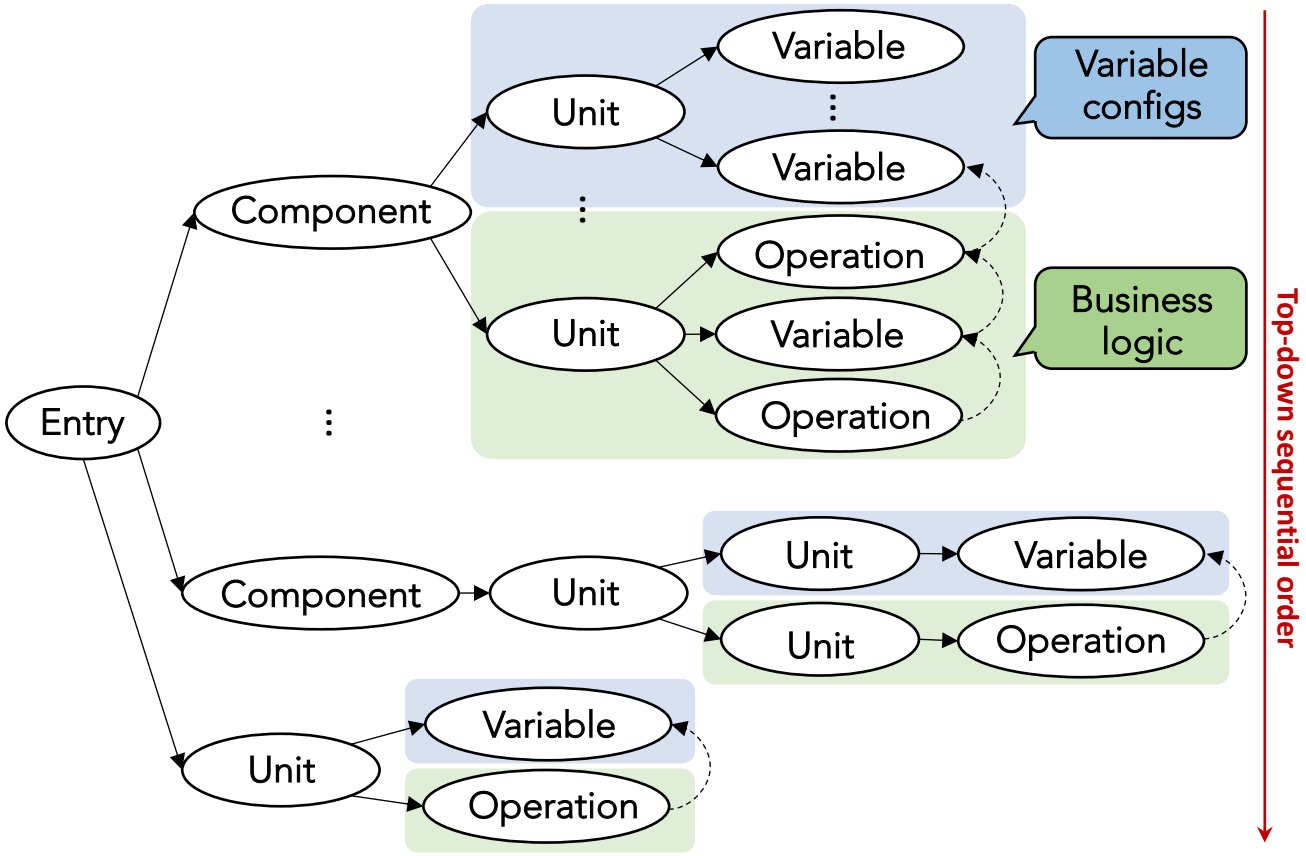
```
23   - name: Search Backups Running
24     shell: "{{ tsm_command }} \"select count(*)
         from sessions where client_name='{{
         ansible_node }}' having count(*)>1 \""
```

Structure Representation Tree



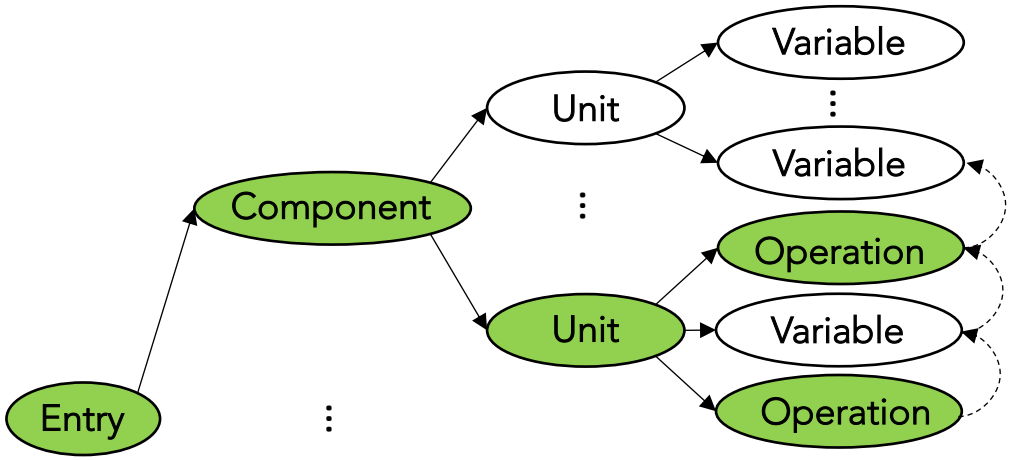
- A tree data-structure
- The syntax abstraction of the infrastructure code
- Extract and organize the key information
- Exclude details
- Ease the process of accurate script extraction

Structure Representation Tree



- Top-down sequential order
  - Reference-after-define policy between variables and operations
  - Happens-before relationship among operations

Script Detection




```
23   - name: Search Backups Running
24     shell: "{{ tsm_command }} \"select count(*)
         from sessions where client_name='{{
         ansible_node }}' having count(*)>1 \""
```

- Traverse the SRTs, and check whether the operation leaf nodes use the **script-related IaC libraries**.

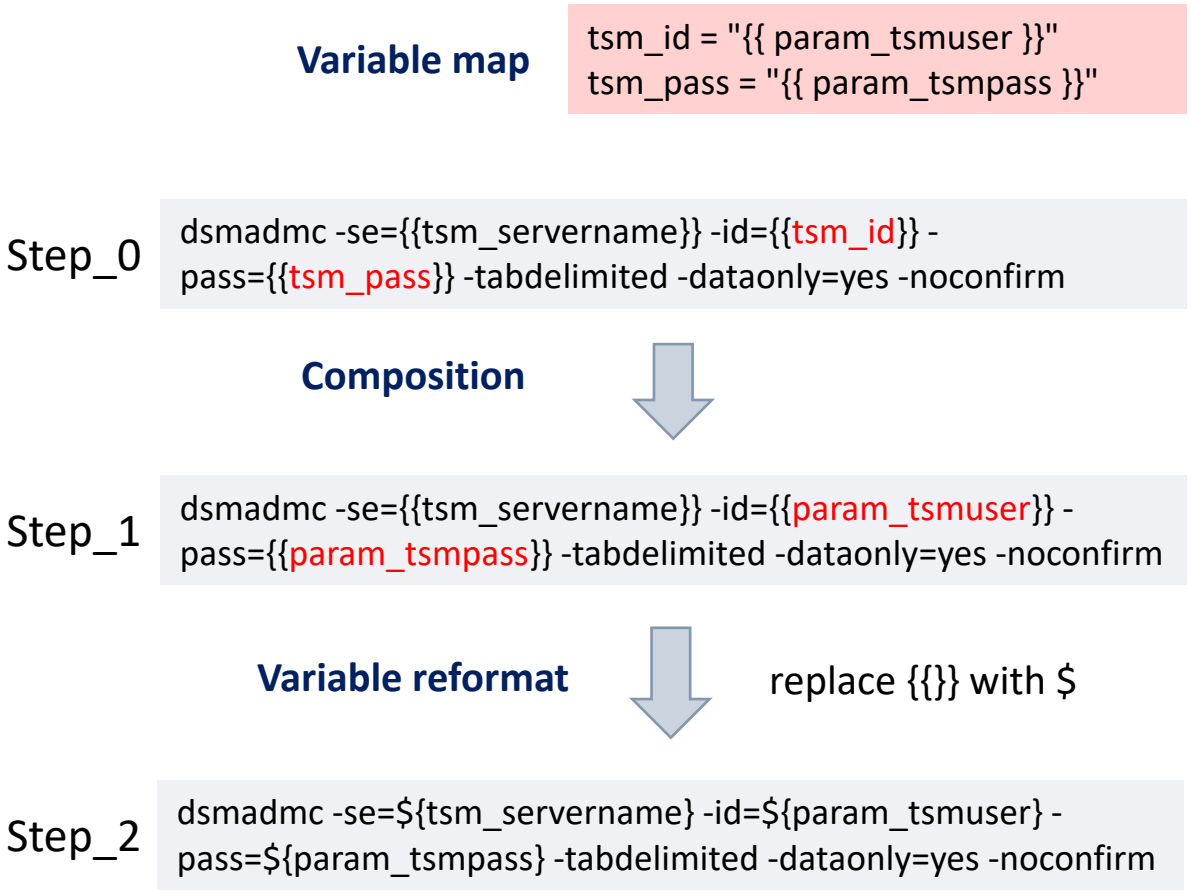
Ansible modules:
command, shell, win_shell, etc.

Chef resources:
execute, script, bash, etc.

Puppet resource:
exec

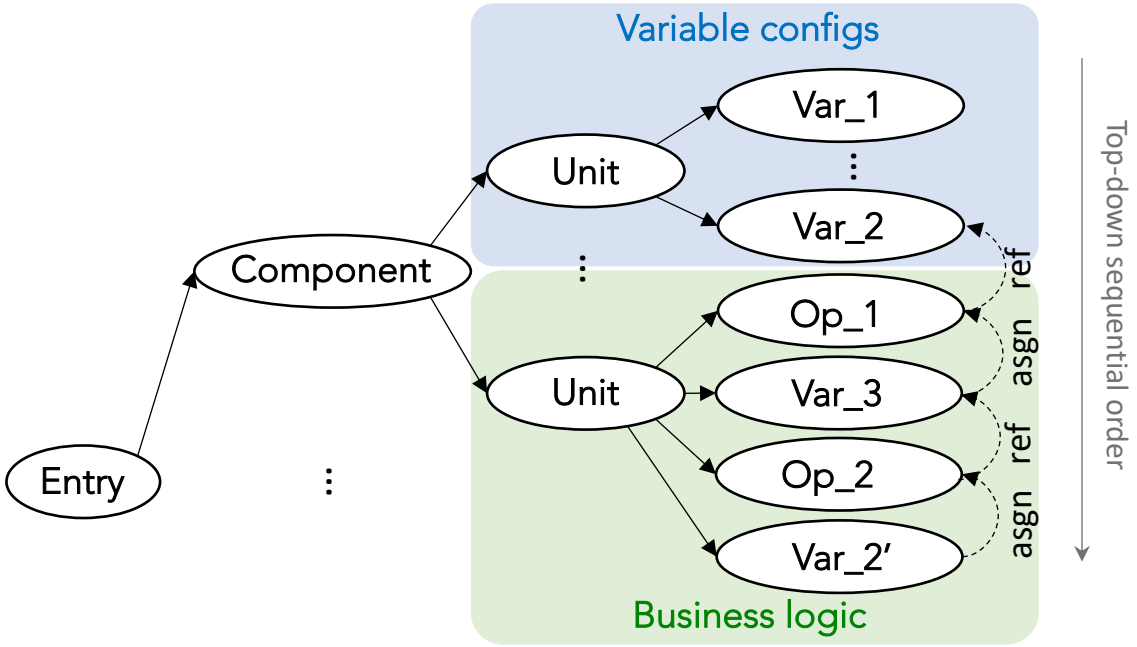
Terraform provisioners:
local-exec, remote-exec

Script Composition & Templated Variable Reformat



- Recursively replace a templated variable in raw scripts with its value extracted from the **variable map**.
- Remove the templating language from the composed scripts while still preserving the variable reference.

Variable Map



t_0

Var_1
Var_2

t_1

Var_1
Var_2
Var_3

t_2

Var_1
Var_2'
Var_3

- Lexically scope variables

  Var_2' overrides Var_2

- Any op **cannot** access Var_3 **before** t_1

- Initialization: contains all statically configured vars.
- In-time update

- Guarantee the **define-reference** order:
  - Statically configured vars can be accessed by all the ops.
  - Dynamically defined vars can only be accessed by the later ops.

Impact Categorization & Severity Assignment

- Empirical study all **345** rules from ShellCheck [29] and **64** rules from PSScriptAnalyzer [35].

Non-risk

- E.g., echo $(cat foo.txt)

Security

- E.g., find . -name '*.txt' -exec sh -c 'echo "{}"' \;

Availability

- E.g., rm -rf /

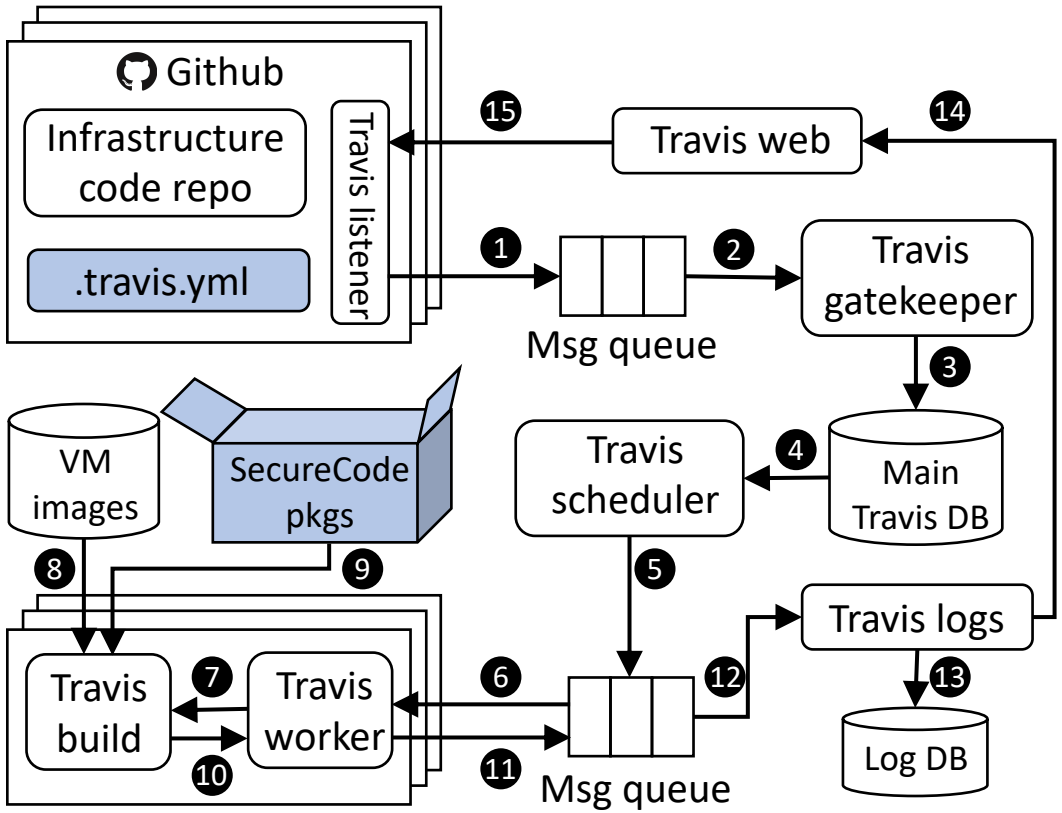
Performance

- E.g., if ["$(find . | grep 'IMG[0-9]')"]

Reliability

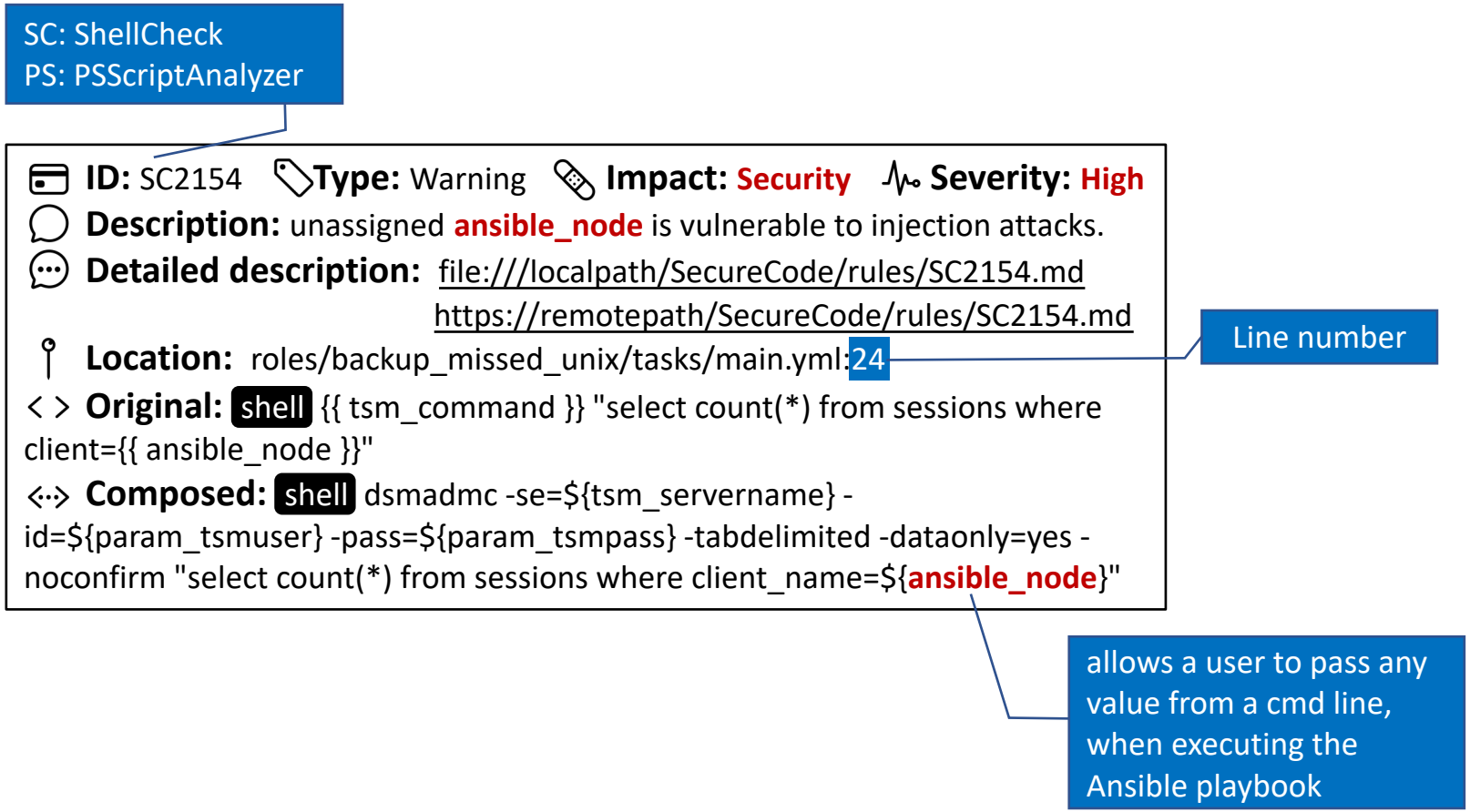
- Context-specific based on the infrastructure operation criticality
  - Critical: reboot, restart, etc
  - Moderate: file r/w, etc
  - Trivial: print on terminal, etc

Implementation & Set Up



- Implement SecureCode
  - Check Ansible playbooks
  - Propose new template parsers
  - Reuse parsing functions in the Ansible-lint [13]
- Integrate with IBM DevOps CI/CD pipeline.
- Test 45 IBM Services community github repos.

Output Format



Detection Accuracy & Statistics

- SecureCode identifies **3535** issues in total from the 45 repositories which contain 1492 automation files.
- **116** out of 3535 issues are false positives.

	High	Medium	Low	**Total**
Non-risk	0	0	862	862
Availability	2	0	0	2
Performance	0	51	0	51
Security	1204	0	0	1204
Reliability	485	247	568	1300
Total	1691	298	1430	3419

User Experience

- We evaluate code quality using
  - The number of detected issues, **range [0, 1414]**
  - The number of detected issues per LOC (i.e., ipl ratio), **range [0, 0.45]**

- We access user experience by comparing SecureCode with others
  - **Throughput Improvement**: LOCs reviewed per person per day
    - **5x** vs manual, **2-5x** vs ShellCheck, **2-3x** vs PSScriptAnalyzer
  - **Efficiency Gain**: the number of issues to be identified
    - **5x** vs manual, **2-3x** vs ShellCheck, **2-3x** vs PSScriptAnalyzer

Conclusion & Future work

- IaC embeded script analysis framework: a first step towards the direction by identifying risky scripts in infrastructure code.
- To bridge the gap between generic state-of-the-practice script-analyzers and business consequences.
- SecureCode is accurate.
- Next Steps
  - supporting more scripting languages,
  - supporting checking infrastructure code in other IaC tools