Characterizing Co-located Insecure Coding Patterns in Infrastructure as Code Scripts

Farzana Ahamed Bhuiyan
Tennessee Technological University
Cookeville, Tennessee, USA
fbhuiyan42@tntech.edu

Akond Rahman
Tennessee Technological University
Cookeville, Tennessee, USA
arahman@tntech.edu

ABSTRACT

Context: Insecure coding patterns (ICPs), such as hard-coded passwords can be inadvertently introduced in infrastructure as code (IaC) scripts, providing malicious users the opportunity to attack provisioned computing infrastructure. As performing code reviews is resource-intensive, a characterization of co-located ICPs, i.e., ICPs that occur together in a script can help practitioners to prioritize their review efforts and mitigate ICPs in IaC scripts. **Objective:** *The goal of this paper is to help practitioners in prioritizing code review efforts for infrastructure as code (IaC) scripts by conducting an empirical study of co-located insecure coding patterns in IaC scripts.* **Methodology:** We conduct an empirical study with 1613, 2764 and 2845 Puppet scripts respectively collected from three organizations namely, Mozilla, Openstack, and Wikimedia. We apply association rule mining to identify co-located ICPs in IaC scripts. **Results:** We observe 17.9%, 32.9%, and 26.7% of the scripts to include co-located ICPs respectively, for Mozilla, Openstack, and Wikimedia. The most frequent co-located ICP category is hard-coded secret and suspicious comment. **Conclusion:** Practitioners can prioritize code review efforts for IaC scripts by reviewing scripts that include co-located ICPs.

CCS CONCEPTS

• **Security and privacy → Software security engineering**.

KEYWORDS

configuration script, co-location, devops, devsecops, empirical study, infrastructure as code, insecure coding pattern, puppet, security

ACM Reference Format:
Farzana Ahamed Bhuiyan and Akond Rahman. 2020. Characterizing Co-located Insecure Coding Patterns in Infrastructure as Code Scripts. In *35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3417113.3422154

1 INTRODUCTION

Infrastructure as code (IaC) is the practice of automatically managing configurations and provisioning computing environments using source code [8]. Information technology (IT) organizations, such as Ambit Energy [1] and KPN [2], use IaC scripts to automatically manage their software configurations and construct automated deployment pipelines [15, 21]. Practitioners use IaC tools, such as Puppet [3] and Terraform [4], that provide utilities to implement the practice of IaC [4]. Practitioners have reported the benefits of using IaC scripts. For example, Ambit Energy observed their deployment frequency to increase by a factor of 1,200 using Puppet scripts [15]. KPN, a Dutch telecommunications company, uses Puppet scripts to manage its 10,000 servers [16].

Despite reported benefits, IaC scripts are susceptible to including insecure coding patterns (ICPs) [19, 20]. ICPs are recurrent coding patterns that indicate security weaknesses in source code [19, 20]. The existence of ICPs in IaC scripts necessitates practitioners to perform rigorous code reviews.

However, performing code reviews is a human-centric activity that involves practitioners and requires resources [26]. Practitioners can benefit from a prioritization strategy that may help them to allocate their code review efforts efficiently. One prioritization strategy could be identifying IaC scripts with co-located ICPs. Co-located ICPs are ICPs that occur together in a source code file. Let us consider a hypothetical scenario in this regard, where Dolly, a practitioner is assigned to review 1,000 Puppet scripts so that ICPs are detected and mitigated. Dolly is aware that performing code reviews is time consuming. She wants to prioritize review efforts by inspecting scripts with co-located ICPs as she might be able to detect and repair more ICPs. She wonders if scientific evidence shows ICPs to co-locate in IaC scripts. Dolly can be informed by a research study that reports empirical evidence to exist for co-located ICPs. Furthermore, characterizing co-located ICPs can help researchers understand the nature of ICPs in IaC scripts.

We observe anecdotal evidence related to co-located ICPs in open source software (OSS) repositories. Let us consider Figure 1 in this regard. Figure 1 presents a Puppet code snippet downloaded from an OSS repository [28]. We observe two ICPs to co-locate: one instance of a hard-coded secret and one instance of invalid IP address binding. Hard-coded secret and invalid IP address binding respectively correspond to the ICPs of revealing sensitive information and using the IP address 0.0.0.0 for configuration [19]. We hypothesize that through systematic investigation we can characterize co-located ICPs similar to that of Figure 1.

HCSE&CS-2020, September, 2020,

ACM ISBN 978-1-4503-8128-4/20/09…$15.00
https://doi.org/10.1145/3417113.3422154

[1] https://www.ambitenergy.com/
[2] https://www.kpn.com/
[3] https://puppet.com/
[4] https://www.terraform.io/

```
1  http-socket => '0.0.0.0:5668', <-------- Invalid IP address binding
2  mount =>
       ↪ '/dynamicproxy-api=/usr/local/bin/invisible-unicorn.py',
3  callable => 'app',
4  },
5 cron { 'proxydb-bak':
6       ensure => present,
7       user => 'root', <---------- Hard-coded secret
8 }
```

Figure 1: Co-located ICPs in a Puppet code snippet downloaded from an OSS repository [28].

The goal of this paper is to help practitioners in prioritizing code review efforts for infrastructure as code (IaC) scripts by conducting an empirical study of co-located insecure coding patterns in IaC scripts.

We answer the following research questions:

- ***RQ1**: What categories of insecure coding patterns (ICPs) co-locate in infrastructure as code scripts? How frequently do ICPs co-locate?*
- ***RQ2**: What source code metrics characterize co-located insecure coding patterns in infrastructure as code scripts?*

We conduct an empirical study with 1613, 2764, and 2845 Puppet scripts respectively, collected from Mozilla [5], Openstack [6], and Wikimedia [7]. First, we identify what categories of ICPs are co-located using association rule mining [5]. Next, we identify which source code metrics characterize scripts that have co-located ICPs.

We make the following contributions:

- A list of ICPs that co-locate in IaC scripts;
- An evaluation of how frequently ICPs co-locate in IaC scripts; and
- A list of source code metrics that characterize co-located ICPs in IaC scripts.

2 BACKGROUND AND RELATED WORK

In this section, we provide the necessary background on infrastructure as code scripts and discuss relevant prior work.

2.1 Background

IaC is the process of automatically managing configurations and provisioning computing environments using source code [8]. A dedicated programming language allows specifying the necessary environment settings, such as required libraries or the amount of RAM for computing environments, such as virtual machine (VM). Performing these tasks is repetitive and needs a significant degree of expertise. IaC makes it possible to avoid the manual process of upgrading environment versions or setting up a new environment by executing a simpler script [9]. Automation and simplicity reduces the test and release cycle time to a great extent, lowers the chance of errors and increases deployment flexibility [8].

Puppet is considered as one of the most popular tools for infrastructure automation [9, 24]. Puppet provides support to define environment parameters, configure deployment, and customize Ruby plug-ins in the process of continuous integration. Puppet scripts are called manifests and are composed of resources, resource type declarations, and inter-resource dependencies [9]. These manifest files are written in domain-specific languages (DSL) [24].

2.2 Related Work

Our paper is closely related to prior research on IaC scripts. Sharma et al. [25] and Bent et al. [27], in separate studies investigated code maintainability aspects of Puppet scripts. Jiang and Adams [10] investigated the co-evolution of IaC scripts and other software artifacts, such as build files and source code. Rahman et al. [18] conducted a systematic mapping study with 32 IaC-related publications and observed a lack of security-related research in the domain of IaC. Hanappi et al. [6] investigated how the convergence of IaC scripts can be automatically tested, and proposed an automated model-based test framework. Rahman et al. [17] constructed a defect taxonomy for IaC scripts that included eight defect categories. In this paper, we build upon the research conducted by Rahman et al. [19], which identified seven categories of ICPs that are indicative of security weaknesses in IaC scripts. They identified 21,201 occurrences of ICPs but did not investigate co-located ICPs.

From the above-mentioned discussion, we observe a lack of research that investigates co-located ICPs in IaC scripts. We address this research gap in our paper.

3 METHODOLOGY

ICPs that occur together in one IaC script are co-located ICPs. ICPs are recurring coding patterns in the source code of a program that suggests a potential security weakness [19]. ICPs may not always cause a security breach but still needs attention and review. Rahman et al. [19] derived a list of seven ICPs in IaC scripts through an empirical study on 1,726 IaC scripts. The identified ICPs are: (i) Admin by default, (ii) Empty password, (iii) Hard-coded secret, (iv) Invalid IP address binding, (v) Suspicious comment, (vi) Use of HTTP without TLS/SSL and (vii) Use of weak cryptography algorithms.

3.1 Dataset

We collect our Mozilla, Openstack, and Wikimedia datasets from the prior work done by Rahman et al. [19]. To identify ICPs in the three datasets we apply the following steps: *first*, we apply SLIC [19] on every IaC script in the dataset. *Second*, we manually inspect each detected ICP occurrence as static analysis tools are susceptible to generating false positives [11]. The last author, with eight years of experience in software security manually perform the inspection by applying closed coding [3] to identify which scripts have what categories of ICPs. The last author maps each script to an ICP.

SLIC respectively, generated 1334, 5664, and 2987 ICP alerts of which 210, 1560, and 326 are false positives for Mozilla, Openstack, and Wikimedia. By filtering false positives we identify 1124, 4111, and 2661 ICPs respectively, in Mozilla, Openstack, and Wikimedia. All constructed datasets are available online [1].

3.2 Methodology to Answer RQ1

We provide the methodology to answer RQ1 in this section.

Co-location categories: To find what categories of ICPs co-locate in IaC scripts, we use association rule mining [5]. Association rule

[5] https://hg.mozilla.org/
[6] https://git.openstack.org/cgit
[7] https://gerrit.wikimedia.org/r/#/admin/projects/

mining is an unsupervised learning algorithm that identifies co-located patterns in datasets [5]. Using association rule mining we identify co-located ICP categories that are of length >= 2. As shown in Figure 1, one such example is *(Hard-coded secret, Invalid IP address binding)*, which is of length 2. We do not use any cutoffs for support and confidence as we want to identify all co-located ICP categories.

Frequency Metrics: We characterize how frequently ICPs co-locate using two metrics for each identified ICP co-location categories. The first metric quantifies how many of the ICPs are co-located. The second metric says how many scripts have co-located ICPs.

- **Proportion of Co-located ICPs (ICP Prop)**: This metric refers to the percentage of ICPs that belong to a certain co-location category.

$$\text{ICP Prop}\ (x) = \frac{\#\ \text{of instances with co-location category}\ x}{\#\ \text{of ICPs in the dataset}} * 100\% \quad (1)$$

- **Percentage of IaC Scripts (Script Prop)**: This metric refers to the percentage of scripts having co-location category, x.

$$\text{Script Prop}\ (x) = \frac{\#\ \text{of scripts with >= 1 co-location category}\ x}{\#\ \text{of scripts in the dataset}} * 100\% \quad (2)$$

3.3 Methodology to Answer RQ2

We answer RQ2 by computing statistical significance and correlation magnitude for each of the 10 metrics listed in Table 1. We select these code metrics as these code metrics characterize buggy IaC scripts according to Rahman and Williams [22]. Our hypothesis is that code metrics that characterize buggy IaC scripts will also correlate scripts with ICPs and co-located ICPs. We repeat the computation process of statistical significance and correlation magnitude for two cases: (i) neutral scripts and scripts with at least one ICP and (ii) neutral scripts, scripts with one ICP, and scripts with co-located ICPs.

Statistical Tests: We apply two statistical tests: *first*, we use the Mann-Whitney U test, a non-parametric test that compares two distributions [13]. While applying the Mann Whitney U test, we compare if each of the 10 code metrics is significantly different between scripts with no ICPs and scripts with at least one ICP. *Second*, we quantify the correlation between code metrics and scripts with co-located ICPs using the Kruskal Wallis H test, a non-parametric test that can compare more than two distributions [12]. For the Kruskal Wallis H test, we quantify if each of the 10 code metrics is significantly different among three groups: scripts with no ICPs, scripts with one ICP, and scripts with at least one co-located ICP.

Correlation Magnitude: We use two approaches to compute correlation magnitude: *first*, we use Cliff's Delta [2] to compare the correlation magnitude for a metric between insecure and neutral scripts. We interpret Cliff's Delta values according to Romano et al.[23]'s recommendations: 'negligible difference' when $\Delta < 0.147$, 'small difference' when $0.147 < \Delta < 0.33$, 'medium difference' when $0.33 < \Delta < 0.474$, and 'large difference' when $\Delta > 0.474$. *Second*, we use feature importance, similar to prior work [22] to compute the correlation magnitude between each metric and scripts with co-located ICPs. We use the Random Forest (RF) [7] algorithm using Scikit Learn API [14] to calculate the importance of each source code metrics with respect to the co-location of ICPs. The output value varies from zero to one, and a higher value for a source code metric indicates a higher correlation with co-located ICPs in IaC scripts. When constructing the RF model, the independent variables are each of the 10 metrics, whereas the dependent variable corresponds to three categories of scripts: neutral scripts, scripts with only one ICP, and scripts with co-located ICPs.

For both, Mann Whitney U and Kruskal Wallis H tests, we reject the null hypothesis is $p - value < 0.05$.

Table 1: Metrics [22] used to answer RQ2

Metric	Description
Attribute	Count of attributes used to define and specify configuration values.
Command	Count of bash and batch commands that need to be executed.
Ensure	Count of checks used to ensure existence of a file.
File	Count of files, directories, and symbolic links using the 'file' syntax that need to be managed.
File mode	Count of file permissions i.e. file modes that are specified.
Hard-coded string	Count of hard-coded strings that are used to specify configuration values.
Include	Count of occurrences when external dependent modules are specified.
Lines of code (LOC)	Size of scripts as measured by the total number of lines of code.
Ordering	Count of functions that are used to specify the order of execution.
SSH_KEY	Count of SSH keys that are specified.

4 EMPIRICAL FINDINGS

In this section, we report our empirical findings.

4.1 Answer to RQ1

We answer **RQ1: What categories of insecure coding patterns (ICPs) co-locate in infrastructure as code scripts? How frequently do ICPs co-locate?** in this section. For Mozilla, Openstack, and Wikimedia we identify 290, 910, and 761 Puppet scripts with at least one occurrence of co-located ICPs. We present the co-located ICP categories along with their ICP proportion and script proportion values for Mozilla, Openstack, and Wikimedia respectively, in Tables 2, 3 and 4.

Table 2: Answer to RQ1: ICP prop. and script prop. values for Mozilla. The most frequent ICP co-location category is hard-coded secret and suspicious comment (S, C).

Co-located ICP Category[8]	ICP Prop	Script Prop	Count
(S, C)	2.79	11.03	1
(S, W)	1.39	5.52	1
(S, H)	0.96	3.79	1
(W, H)	0.70	2.76	1
(C, H), (W, C, H), (W, C)	0.61	2.41	3
(A, B), (W, B), (S, W, B), (S, W, H)	0.52	2.07	4
(S, C, H), (S, C, W), (S, C, H, W)	0.43	1.72	3
(A, S), (A, H), (A, C), (A, W), (E, S), (E, C), (A, S, H), (A, S, C), (A, S, W), (A, C, H), (A, W, H), (A, W, C), (A, S, C, H), (A, S, W, H), (A, S, C, W), (A, W, C, H), (A, S, C, W, H)	0.35	1.38	17
(E, H), (A, E), (E, W), (A, E, S), (A, E, H), (A, E, C), (A, E, W), (E, S, H), (E, S, C), (E, S, W), (E, C, H), (E, W, H), (E, W, C), (A, E, S, H), (A, E, S, C), (A, E, S, W), (A, E, C, H), (A, E, W, H), (A, E, W, C), (E, S, C, H), (E, S, W, H), (E, S, C, W), (E, W, C, H), (A, E, S, C, H), (A, E, S, W, H), (A, E, S, C, W), (A, E, C, W, H), (E, S, C, W, H), (A, E, S, C, W, H)	0.17	0.69	29
(C, B)	0.09	0.35	1

Table 3: Answer to RQ1: ICP prop. and script prop. values for Openstack. The most frequent ICP co-location category is hard-coded secret and HTTP without TLS/SSL (S, H).

Co-located ICP Category[8]	ICP Prop	Script Prop	Count
(S, H)	3.14	15.57	1
(C, S)	2.01	9.98	1
(S, B)	0.93	4.61	1
(C, B)	0.73	3.62	1
(S, A)	0.71	3.51	1
(C, H)	0.60	2.96	1
(C, S, H)	0.49	2.41	1
(H, A), (S, H, A)	0.40	1.97	2
(S, W), (C, S, B)	0.38	1.86	2
(E, S)	0.24	1.21	1
(H, B)	0.22	1.10	1
(S, H, B)	0.20	0.99	1
(C, W)	0.18	0.88	1
(E, C), (C, S, W)	0.13	0.66	2
(A, B), (C, A), (S, A, B), (C, S, A), (C, H, A), (C, S, H, A)	0.11	0.55	6
(E, C, S)	0.09	0.44	1
(S, H, W), (H, W), (C, H, B), (C, S, H, B)	0.07	0.33	4
(E, W), (E, H), (E, C, W), (E, S, W), (E, C, S, W)	0.04	0.22	5
(W, A), (E, B), (S, A, W), (H, A, W), (C, W, A), (E, S, B), (E, H, B), (E, S, H), (E, C, H), (C, H, W), (S, H, A, W), (C, S, A, W), (C, H, A, W), (E, S, H, B), (C, S, H, W), (A, C, H, S, W)	0.02	0.11	16

Table 4: Answer to RQ1: ICP prop. and script prop. values for Wikimedia. The most frequent ICP co-location category is hard-coded secret and suspicious comment (S, C).

Co-located ICP Category [8]	ICP Prop	Script Prop	Count
(S, C)	3.26	9.97	1
(S, H)	1.59	4.86	1
(C, H)	0.77	2.36	1
(S, B), (S, C, H)	0.43	1.31	2
(S, W)	0.34	1.05	1
(S, E)	0.30	0.92	1
(S, A)	0.26	0.79	1
(C, B)	0.21	0.66	1
(H, B), (W, C)	0.13	0.39	2
(B, A), (H, A), (S, B, A), (H, S, B), (S, C, B), (S, E, C), (H, C, B), (E, H), (S, H, A), (E, C), (S, W, C), (H, S, C, B)	0.09	0.26	12
(E, A), (W, E), (W, H), (S, E, A), (S, E, H, A), (W, E, H), (S, E, H), (E, H, A)	0.04	0.13	8

We present the co-located ICP categories along with their ICP proportion and script proportion values for Mozilla, Openstack, and Wikimedia respectively, in Tables 2, 3 and 4. In all three tables 'S', 'C', 'H', 'E', 'W', 'B' and 'A' respectively denote hard-coded secret, suspicious comments, HTTP without TLS/SSL, empty password, weak cryptography algorithms, invalid IP address binding, and admin by default.

In Tables 2, 3 and 4 'ICP Prop' and 'Script Prop' respectively, presents the percentage of ICPs and percentage of scripts for which we observe ICP co-location. 'Count' represents the number of unique co-located ICP categories. Tables 2, 3 and 4 are sorted based on 'ICP Prop.' values. For example, from Table 2 we observe hard-coded secret and suspicious comment to co-locate for 11.03% of the scripts in the Mozilla dataset. The co-location category is ('S' , 'C') indicating hard-coded secrets and comments that have an ICP Prop. value of 2.79 and Script Prop. value of 11.03.

One possible explanation for ICP co-location can be attributed to the operations that are implemented in the script. For example, to setup a virtual network a practitioner may use '0.0.0.0' as IP address and 'http' for data transmission protocol. '0.0.0.0' and 'http' are respectively instances of invalid IP address binding and HTTP without TLS.

[8]'S', 'C', 'H', 'E', 'W', 'B' and 'A' respectively denotes hard-coded secret, suspicious comments, HTTP without TLS/SSL, empty password, weak cryptography algorithms, invalid IP address binding, and admin by default.

4.2 Answer to RQ2

In this section, we answer **RQ2: What source code metrics characterize co-located insecure coding patterns in infrastructure as code scripts?**

We report the median, maximum and minimum values of each metric for insecure scripts and neutral scripts in Table 5. For example, in the case of Mozilla, the median value for the metric 'Attribute' is respectively 12 and 2 for insecure and neutral scripts. We also report the p-value and Cliff's Delta value respectively in the 'p-value' and 'Δ' columns. The metrics 'Attribute' and 'Hard-coded string' have a large' Cliff's Delta value for all three datasets. The metric 'Lines of code' has a 'large' Cliff's Delta value for two datasets. We observe 8 of the 10 metrics to show correlation with IaC scripts that have co-located ICPs.

In Table 6, we report the distribution of each metric for the scripts with co-located ICPs, scripts with only one ICP and neutral scripts with no ICP. We provide the median and maximum, minimum values of each metric. For the Mozilla dataset, the median value of 'Attribute' is respectively 14, 8, and 2. We also report the p-value in the 'p-value' columns.

In Table 7, we report a ranked order of each identified source code metrics that characterize the co-location of ICPs in IaC scripts. For Mozilla and Wikimedia dataset, we observe LOC to show the strongest correlation with the co-location of ICPs in IaC. For Openstack, we observe the strongest correlation to be 'Hard-coded string'. This ranking is in accordance with our earlier findings presented in Table 5 as we see the metrics 'Lines of code', 'Hard-coded string' and 'Attribute' have a stronger correlation for all three datasets.

5 DISCUSSION AND CONCLUSION

We discuss the implications of our findings in this section.

Implications for practitioners: Our findings have implications for practitioners on how they can prioritize their review efforts for IaC scripts. IT organizations can maintain a large number of Puppet scripts, e.g. as many as 3,143 Puppet scripts [17]. Security reviews for a large number of scripts can be time-consuming and can benefit from code review prioritization strategies. One strategy can be prioritizing Puppet scripts that have co-located ICPs. For example, for Mozilla, Openstack, and Wikimedia respectively, security reviews can be prioritized for the 11.0%, 15.5%, and 9.9% scripts as these scripts contain the most frequently occurring co-located ICP categories.

Coming back to the hypothetical example mentioned in Section 1 Dolly now has evidence that shows scripts with co-located ICPs can vary between 17.9%∼32.9%.

Table 5: Answer to RQ2: Comparing source code metrics between two groups: scripts with no ICPs and at least one ICP. Metrics with $p - value < 0.05$ for all datasets are indicated by ✪. Correlation magnitude using Cliff's Delta is reported in the Δ column.

Source Code Metric	Type	Mozilla				Openstack				Wikimedia			
		Median	(Max,Min)	p-value	Δ	Median	(Max,Min)	p-value	Δ	Median	(Max,Min)	p-value	Δ
Attribute ✪	Insecure	12	(168,0)	1.14e-44	0.52	14	(463,0)	2.93e-97	0.49	15	(267,0)	2.18e-100	0.52
	Neutral	2	(173,0)			5	(189,0)			5	(193,0)		
Command ✪	Insecure	0	(5,0)	1.84e-19	0.22	0	(10,0)	4.59e-26	0.14	0	(23,0)	2.95e-60	0.29
	Neutral	0	(6,0)			0	(15,0)			0	(15,0)		
Ensure ✪	Insecure	0.5	(29,0)	1.23e-04	0.13	0	(31,0)	4.20e-05	0.08	1	(28,0)	4.48e-38	0.29
	Neutral	0	(22,0)			0	(57,0)			0	(27,0)		
File ✪	Insecure	0	(6,0)	3.41e-08	0.17	0	(15,0)	1.05e-11	0.10	1	(24,0)	1.93e-40	0.28
	Neutral	0	(10,0)			0	(15,0)			0	(26,0)		
File mode ✪	Insecure	0	(12,0)	7.07e-11	0.18	0	(32,0)	5.47e-08	0.07	1	(19,0)	9.55e-41	0.27
	Neutral	0	(8,0)			0	(6,0)			0	(17,0)		
Hard-coded string ✪	Insecure	7	(341,0)	6.43e-65	0.63	10	(381,0)	1.95e-153	0.61	11	(258,0)	1.42e-106	0.53
	Neutral	1	(129,0)			3	(94,0)			3	(235,0)		
Include	Insecure	1	(180,0)	0.06	0.06	1	(37,0)	8.38e-06	0.09	0	(41,0)	1.79e-01	0.02
	Neutral	1	(48,0)			0.5	(81,0)			0	(110,0)		
Lines of code ✪	Insecure	60	(1178,8)	1.85e-55	0.59	77.5	(1623,7)	1.28e-84	0.46	58	(2247,6)	1.93e-111	0.55
	Neutral	22	(407,0)			40	(612,0)			23	(518,0)		
Ordering ✪	Insecure	0	(11,0)	3.33e-11	0.19	0	(33,0)	1.92e-31	0.18	0 0	(20,0)	1.53e-34	0.24
	Neutral	0	(11,0)			0	(12,0)			0	(18,0)		
SSH	Insecure	0	(0,0)	1.0	0.00	0	(1,0)	2.1e-03	0.004	0	(1,0)	4.91e-02	0.001
	Neutral	0	(0,0)			0	(0,0)			0	(0,0)		

Table 6: Answer to RQ2: Comparing source code metrics among three groups: scripts with no ICPs, one ICP, and co-located ICPs. Source code metrics with $p - value < 0.05$ for all datasets are indicated by ✪.

Source Code Metric	Type	MOZI			OSTK			WIKI		
		Median	(Max,Min)	p-value	Median	(Max,Min)	p-value	Median	(Max,Min)	p-value
Attribute ✪	COLOCATE	14	(168,0)		14	(463,0)		18	(267,0)	
	ONLY_ONE	8	(107,0)	2.55e-44	12	(187,0)	5.13e-96	10	(102,0)	4.70e-107
	Neutral	2	(173,0)		5	(189,0)		5	(193,0)	
Command ✪	COLOCATE	0	(5,0)		0	(6,0)		0	(23,0)	
	ONLY_ONE	0	(3,0)	2.88e-23	0	(10,0)	3.87e-25	0	(6,0)	9.99e-70
	Neutral	0	(6,0)		0	(15,0)		0	(15,0)	
Ensure ✪	COLOCATE	0	(14,0)		0	(31,0)		2	(28,0)	
	ONLY_ONE	1	(29,0)	8.22e-06	1	(28,0)	8.76e-06	1	(27,0)	4.48e-38
	Neutral	0	(22,0)		0	(57,0)		0	(27,0)	
File ✪	COLOCATE	0	(6,0)		0	(14,0)		1	(24,0)	
	ONLY_ONE	1	(5,0)	1.96e-07	0	(15,0)	4.85e-11	0	(16,0)	3.80e-42
	Neutral	0	(10,0)		0	(15,0)		0	(26,0)	
File mode ✪	COLOCATE	0	(12,0)		0	(32,0)		1	(19,0)	
	ONLY_ONE	0	(5,0)	4.50e-10	0	(5,0)	7.45e-07	0	(16,0)	8.35e-46
	Neutral	0	(8,0)		0	(6,0)		0	(17,0)	
Hard-coded string ✪	COLOCATE	9	(341,0)		10	(381,0)		13	(258,0)	
	ONLY_ONE	4	(46,0)	3.21e-65	9	(123,0)	1.95e-153	7	(132,0)	7.07e-114
	Neutral	1	(129,0)		3	(94,0)		3	(235,0)	
Include	COLOCATE	1	(180,0)		1	(37,0)		0	(41,0)	
	ONLY_ONE	1	(34,0)	0.31	0	(8,0)	1.33e-05	0	(15,0)	9.31e-03
	Neutral	1	(48,0)		0.5	(81,0)		0	(110,0)	
Lines of code ✪	COLOCATE	67	(1178,8)		79	(1623,8)		64	(2247,6)	
	ONLY_ONE	51	(211,15)	4.89e-54	64	(478,7)	1.69e-84	46	(325,8)	3.20e-113
	Neutral	22	(407,0)		40	(612,0)		23	(518,0)	
Ordering ✪	COLOCATE	0	(7,0)		0	(33,0)		1	(20,0)	
	ONLY_ONE	0	(11,0)	5.57e-10	0	(10,0)	5.59e-30	0	(10,0)	9.44e-42
	Neutral	0	(11,0)		0	(12,0)		0	(18,0)	
SSH	COLOCATE	0	(0,0)		0	(1,0)		0	(1,0)	
	ONLY_ONE	0	(0,0)	1.0	0	(0,0)	6.2e-03	0	(0,0)	1.22e-01
	Neutral	0	(0,0)		0	(0,0)		0	(0,0)	

Table 7: Correlation magnitude of the source code metrics based on feature importance analysis.

Rank	Mozilla	Openstack	Wikimedia
1	Lines of code (0.28)	Hard-coded string (0.28)	Lines of code (0.27)
2	Hard-coded string (0.19)	Lines of code (0.26)	Hard-coded string (0.19)
3	Attribute (0.18)	Attribute (0.20)	Attribute (0.18)
4	Include (0.09)	Ensure (0.07)	Ensure (0.07)
5	Ensure (0.07)	Include (0.06)	Include (0.07)
6	Ordering (0.05)	Ordering (0.05)	Command (0.06)
7	File (0.05)	Command (0.04)	Ordering (0.06)
8	Command (0.05)	File (0.03)	File (0.05)
9	File mode (0.03)	File mode (0.02)	File mode (0.04)
10	SSH_KEY (0.00)	SSH_KEY (0.01)	SSH_KEY (0.01)

Implications for researchers: We have provided one possible explanation related to why co-located ICPs occur in Section 4.1. We advocate for future research that will apply mixed methods analysis to identify the root causes for co-location occurrence. As static analysis is language-specific researchers can leverage our findings in Section 4.2 to construct models that will predict scripts with co-located ICPs.

Limitations: Our empirical study is limited to the datasets that we analyzed. We use Puppet to construct our datasets, which is a declarative language. We acknowledge that there may exist other ICPs that we have not considered.

Conclusion: ICPs in IaC scripts can provide malicious users the opportunity to perform large-scale data breaches. Prioritization of security code reviews can help practitioners detect and mitigate ICPs in IaC scripts. We conduct an empirical study with 7,222 Puppet scripts to characterize co-located ICPs in IaC scripts. First, we identify the categories of ICPs that co-locate. Next, we quantify the correlation between source code metrics and IaC scripts with co-located ICPs.

We observe ICPs in IaC scripts to co-locate. Of the 7,222 Puppet scripts, 21.06% have at least one instance of co-located ICPs. The most frequently occurring co-located ICPs are (i) hard-coded secret and suspicious comment, and (ii) hard-coded secret and HTTP without TLS/SSL. We observe 8 of the 10 source code metrics to characterize ICPs in IaC scripts.

ACKNOWLEDGMENTS

We thank the PASER group at Tennessee Tech. University for their valuable feedback. The research was partially funded by the National Science Foundation (NSF) grant#2026869.

REFERENCES

[1] Farzana Ahamed Bhuiyan and Akond Rahman. 2020. Verifiability Package for Paper. https://figshare.com/s/f64f56ec9eaf2eab1af4. [Online; accessed 28-Jun-2020].

[2] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin* 114, 3 (1993), 494.

[3] Benjamin F Crabtree and William L Miller. 1999. *Doing qualitative research*. sage publications.

[4] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. 2019. Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 580–589.

[5] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques*. Elsevier.

[6] Oliver Hanappi, Waldemar Hummer, and Schahram Dustdar. 2016. Asserting Reliable Convergence for Configuration Management Scripts. *SIGPLAN Not.* 51, 10 (Oct. 2016), 328–343. https://doi.org/10.1145/3022671.2984000

[7] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[8] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation* (1st ed.). Addison-Wesley Professional.

[9] Yujuan Jiang and Bram Adams. 2015. Co-evolution of infrastructure and source code-an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 45–55.

[10] Yujuan Jiang and Bram Adams. 2015. Co-evolution of Infrastructure and Source Code: An Empirical Study. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (Florence, Italy) *(MSR '15)*. IEEE Press, Piscataway, NJ, USA, 45–55. http://dl.acm.org/citation.cfm?id=2820518.2820527

[11] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why Don't Software Developers Use Static Analysis Tools to Find Bugs?. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 672–681.

[12] William H Kruskal and W Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association* 47, 260 (1952), 583–621.

[13] Henry B Mann and Donald R Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[14] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2825–2830. http://dl.acm.org/citation.cfm?id=1953048.2078195

[15] Puppet. 2018. *Ambit Energy's Competitive Advantage? It's Really a DevOps Software Company*. Technical Report. Puppet. 3 pages. https://puppet.com/resources/case-study/ambit-energy

[16] Puppet. 2020. About KPN. https://puppet.com/resources/customer-story/kpn-0. [Online; accessed 19-June-2020].

[17] Akond Rahman, Effat Farhana, Chris Parnin, and Laurie Williams. 2020. Gang of Eight: A Defect Taxonomy for Infrastructure As Code Scripts. In *Proceedings of the 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. to appear, pre-print: https://akondrahman.github.io/papers/icse20_acid.pdf.

[18] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2018. A systematic mapping study of infrastructure as code research. *Information and Software Technology* (2018). https://doi.org/10.1016/j.infsof.2018.12.004

[19] Akond Rahman, Chris Parnin, and Laurie Williams. 2019. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 164–175.

[20] Akond Rahman, Md. Rayhanur Rahman, Chris Parnin, and Laurie Williams. 2020. Security Smells in Ansible and Chef Scripts: A Replication Study. *ACM Trans. Softw. Eng. Methodol.* (2020), 31. To appear. pre-print: https://arxiv.org/pdf/1907.07159.pdf.

[21] A. Rahman and L. Williams. 2018. Characterizing Defective Configuration Scripts Used for Continuous Deployment. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 34–45. https://doi.org/10.1109/ICST.2018.00014

[22] Akond Rahman and Laurie Williams. 2019. Source Code Properties of Defective Infrastructure as Code Scripts. *Information and Software Technology* (2019). https://doi.org/10.1016/j.infsof.2019.04.013

[23] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, and Jeff Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys. In *annual meeting of the Florida Association of Institutional Research*. 1–33.

[24] Rian Shambaugh, Aaron Weiss, and Arjun Guha. 2016. Rehearsal: A configuration verification tool for puppet. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 416–430.

[25] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. ACM, New York, NY, USA, 189–200. https://doi.org/10.1145/2901739.2901761

[26] Christopher Theisen, Kim Herzig, Patrick Morrison, Brendan Murphy, and Laurie Williams. 2015. Approximating Attack Surfaces with Stack Traces. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2* (Florence, Italy) *(ICSE '15)*. IEEE Press, 199–208.

[27] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 2018. How good is your puppet? An empirically defined and validated quality model for puppet. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 164–174. https://doi.org/10.1109/SANER.2018.8330206

[28] Wikimedia. 2020. operations/puppet. https://gerrit.wikimedia.org/r/#/admin/projects/operations/puppet. [Online; accessed 09-June-2020].