intend to shift some employees to remote work permanently'. Gartner, 3 Apr 2020. Accessed Sep 2020. www.gartner.com/en/newsroom/press-releases/2020-04-03-gartner-cfo-surey-reveals-74-percent-of-organisations-to-shift-some-employees-to-remote-work-permanently2.
3. Bissette, Hayleigh. 'The threat of Covid-19 phishing attacks'. TheBusinessDesk, 7 Jul 2020. Accessed Sep 2020. www.thebusinessdesk.com/yorkshire/news/2058462-the-threat-of-covid-19-phishing-attacks.
4. 'Mitigate risks with PKI'. DigiCert. Accessed Sep 2020. www.digicert.com/pki/.
5. 'Behind the scenes of SSL cryptography'. DigiCert. Accessed Sep 2020. www.digicert.com/ssl-cryptography.htm.

Security controls in infrastructure as code

Sadiq Almuairfi and Mamdouh Alenezi, Prince Sultan University


Sadiq Almuairfi


Mamdouh Alenezi

The development, deployment and management of software applications have shifted dramatically in the past 10 years. This fundamental shift is what we now know as development operations (DevOps). Infrastructure as Code (IaC) is one of the main tenets of DevOps. Previously, manual configuration via cloud providers' UI consoles and physical hardware used to take place. But now, with the concept of IaC, the IT infrastructure can be automated by using blueprints that are easily readable by machines.

To accomplish IaC, various programming languages and tools must be used in order to define the infrastructure (network, servers, storage). The ability to use practices like code review, version control and unit testing impacts infrastructure automation positively. Two of the biggest benefits that IaC implementation provides are comparatively rapid iterations and increased speed of infrastructure deployment. The usage of IaC scripts is really helpful for practitioners in order to configure and provision their development environments. The scripts employed in IaC are also known as 'Configuration as code scripts' or 'configuration scripts.' How beneficial these IaC scripts are can be identified with the example of Fortune 500 company Intercontinental Exchange (ICE). ICE uses IaC scripts to maintain 75% of its 20,000 servers, which ultimately results in a time reduction of environment provisioning from one or two days to 21 minutes.[1]

Although IaC has gained popularity in recent years, when it comes to the quality of code, the research is very sparse. This limited research has introduced the term 'code smell'. Kent Beck and Martin Fowler introduced the concept of code smell and according to them, code smells are flaws in code that may cause problems.[2] There might be no run-time errors while code smell is present but the presence of code smell should be taken as an indication for improvement. Duplicate block (DB) smell is one such example of code smell that occurs whenever identical statements are repeated. The results of various researches prove that noting code smells is an appropriate method of assessing the quality of puppet code. Puppet is a popular provisioning tool that helps write puppet codes and uses IaC in order to specify the desired state of the environment.[3]

Since we know that IaC scripts are used by practitioners to develop environments and provision servers, there is a chance that they may introduce security smells inadvertently. Security smells are simply recurring coding patterns or, in simpler terms, are basically measures that indicate security weakness. There is also a possibility that the presence of security smells may lead to frequent security breaches. Thus, it becomes necessary for practitioners to study security smells in IaC scripts so that insecure coding practices can be avoided. If we study the responses of the Common Weakness Enumerator (CWE), it can be found that CWE considers hard-coded passwords as a security weakness. According to CWE: "If hard-coded passwords are used, it is almost certain that malicious users will gain access to the account in question." Thus, even if a security smell might not lead to a security breach at the time of its discovery, proper attention should be given to it and it should be inspected whenever needed.[4]

"Code smells are flaws in code that may cause problems. There might be no run-time errors while code smell is present but the presence of code smell should be taken as an indication for improvement"

Writing the IaC code is quite a complex task as it is basically done by blending various infrastructure programming languages. Thus, some sort of approach is needed in order to create complex IaC designs that save time not only when being designed but also at the time of deployment and redeployment. Model-driven engineer-

ing (MDE) provides us with one such approach – data-intensive continuous engineering and rollout (Dicer) – that can be used to create language-agnostic models that can be transformed into IaC. Dicer is a model-driven approach and acts as a supporting tool for users who wish to develop IaC for big data frameworks. Although Dicer doesn't support all of the well-known big data frameworks, it provides the benefit of the addition of new frameworks.[5]

Similar to software source code, IaC scripts can face many issues, making these scripts susceptible to security defects. Defects in IaC scripts can have serious consequences, as these scripts are associated with setting up and managing cloud-system infrastructure and ensuring the availability of software devices. For example, in early 2017, execution of a defective IaC script removed the home directories of around 270 customers in cloud instances maintained by Wikimedia.[6] This evidence, plus research studies, motivated us to carefully study the security of IaC, which is the focus of this article.

Related work

Software teams are able to implement continuous deployment and make rapid changes by the use of configuration as code (CaC) tools.[7] The popularity of these tools is increasing day by day but investigating the challenges faced by the programmers while using these tools and finding solutions to those challenges can be really helpful in identifying potential technical challenges related to CaC.

We conducted research on what questions are asked by the programmers about CaC. In order to extract puppet-related questions asked by programmers on Stack Overflow from 2010 to 2016, qualitative analysis was applied. On the basis of responses extracted through this analysis, it was discovered that the three areas that disturb programmers the most about CaC are installation, security and data separation. While practitioners use IaC, certain recurring coding patterns may lead to weaknesses in security, indicating an increased chance of security breaches.

In order to make sure that practitioners make no mistakes when developing IaC scripts, researchers conducted a study of security smells in IaC scripts. They made use of 1,726 IaC scripts through the approach of qualitative analysis in order to find seven security smells. Among the 15,232 IaC scripts, the authors implemented the SLIC (security linter for infrastructure as code) tool to identify the occurrence of each smell. Through their research, Rahman et al in 'The Seven Sins' have tried to answer questions about the type of security smells, their frequency, lifetime and the opinion of practitioners about them. Apart from finding the answers to these questions, the authors made sure that the results of the SLIC tool were properly evaluated and created Oracle datasets in order to do that.

IaC uses blueprints that are easily readable by machines. Schwarz et al have discussed all the necessary elements and abstractions that are used in the writing and maintenance of the blueprint of IaC. The authors seem to outline the benefits of DevOps but at the same time keeping IaC as the accelerating tactic for DevOps. The standard that the authors have used for the expression is known as 'topology and orchestration specification for cloud applications' (Tosca). Tosca is defined as an industrial practice language that is used for automated industrial deployment and multi-cloud compliant applications. One other major benefit that Tosca offers is that it provides reusable nodes and edges. Also, the authors have discussed Dicer, which is a model-driven tool to quickly put together the infrastructure design for a big data cloud application as a part of continuous-data intensive architecting.

Increased complexity

IaC scripts are a blend of various infrastructural programming languages, and with the increased complexity of the infrastructure, the complexity of writing these IaC scripts is also increased. In order to exploit model-driven engineering (MDE), Artac et al attempted to create language-agnostic models that possess the capability to transform themselves into IaC scripts automatically. In addition, the authors have shown that a significant amount of time can be saved even while creating complex IaC by following the Dicer approach. Also, the authors described the Ops activities (server provisioning, monitoring, self-adaptation, etc) required to deploy and operate cloud applications continuously. Speaking of Dicer, the key feature it offers is that models that are written using the Dicer profile can be automatically translated into IaC scripts.

The quality of code is important in any software project. Schwarz et al have focused research on the quality of code while the concept of IaC is gaining popularity day by day, looking for flaws in the quality of code – that is, code smells. In their research, the authors tried to apply code smells to different technologies in order to investigate if similar results could be achieved. To proceed with their research, the authors formulated two questions and by the application of code smells, tried to find the answers to those questions. The questions were particularly concerned with the applicability of puppet smells to other configuration management tools and the existence of other programming smells relevant to IaC. Also, the authors have proposed three types of code smells in the paper: technology agnostic smells, technology-dependent smells and technology-specific smells. The results obtained through the research indicate the presence of IaC smells in other technologies and tools. In order to be sure of the results obtained, the authors further evaluated the results and conducted two more case studies that examined the properties of completeness and soundness.

System discrepancies and system outages are some of the outcomes that can be seen if the quality of IaC

scripts isn't up to the mark. In order to improve the quality of IaC scripts by practitioners, Rahman in 'Anti-Patterns in Infrastructure as Code' conducted research on the identification of anti-patterns in IaC scripts and then development of the IaC scripts. As per the author: "Through systematic investigation, we can identify anti-patterns in IaC that correlate with defects, and violate security and privacy objectives."

The methodology used by Rahman was based on characterising defective IaC scripts by extracting text features. The two text-mining techniques used were the 'bag-of-word' technique, and the 'term frequency-inverse document frequency' technique. In order to characterise the properties of defective IaC scripts, Rahman also applied Strauss-Corbin Grounded theory. As a result, the three properties that characterise defective IaC scripts turned out to be infrastructure provisioning, file system operations and management of user accounts. Rahman built prediction models using characteristics such as commits, number of multitasking practitioners, their age, etc.

Quality challenge

Even though IaC scripts are widely adopted, the development and maintenance of premium quality IaC scripts are challenging to the majority of the developers. Chen et al conducted research in order to identify error patterns for IaC.[8] The authors proposed an approach to handle frequently occurring IaC code errors. The approach works on the extraction of 'code changes' from historical commits and once they are extracted, an unsupervised machine learning algorithm is employed in order to cluster them in groups. Thus it can be said that the approach is formed by following three steps, which are: extraction of code changes; identification of error patterns; and suggesting constraint rules as outcome. The authors employed abstract syntax tree (AST) differencing in order to locate the code changes. Through their research, they were able to classify 41 cross-project error patterns such as variable related errors, file-related errors, OS-related errors, etc. The rationale that the authors propose as being behind their approach is: "Identical (or similar) code errors would be fixed with identical (or similar) code changes."

In order to keep up with the improvement in cloud system environments, Lavriv et al proposed the method of cloud system disaster recovery based on the concept of infrastructure as code.[9] In order to define a sample cloud infrastructure, the authors used a tool named Terraform by Harshicorp. This tool was developed using the Go programming language and is very promising. Its simple syntax is based on the key-value format, which makes it very popular in IT operations. Through their research, the authors have tried to show the benefits of using the IaC concept as compared to the manual approach in case of system failure. Also, the authors simulated the disaster of sudden manual destruction of all previously created infrastructure. The two types of recovery actions that were investigated were Terraform tool activation and manual infrastructure recreation. Through the results achieved, the authors were able to justify the difference between the recovery time manually and with a tool like Terraform. The difference becomes more visible with the increased complexity of the cloud system.

IaC scripts can be defective and may lead to large-scale service outages for end users.[10] These defects can be mitigated if somehow we are able to predict the defective IaC scripts and that was the goal of Rahman et al in conducting this research. The authors talk about proposing the matrices related to the defect prediction model in order to prioritise inspection efforts. Also, the authors employed constructivist ground theory (CGT) to identify the metrics that are suitable to IaC scripts. The methodology used by the authors consists of four steps: repository collection; commit message processing; determination of defect-related commits; and application of CGT theory respectively. As a result, the authors obtained 18 metrics indicating 18 different defect characteristics. Thus the authors, through their research, were able to prove that IaC scripts are susceptible to defects and these defect prediction models or metrics can help practitioners prioritise inspection efforts.

Even though build automation tools provide benefits such as reduction in errors and reduction in rapid release of software changes, these are still considered complex among software practitioners. Build automation is considered to be a technology that "automatically compiles and tests software changes, packages the software changes into a binary, and prepares the created binary for deployment".[11] The authors in 'Which Factors Influence Practitioners' Usage of Build Automation Tools?' focus their research on the identification of adoption factors that may influence the usage of these build automation tools among software practitioners. In addition, the authors conducted a survey to identify these adoption factors, and responses from 268 software professionals were recorded. The responses revealed that complexity wasn't the main factor hindering the use of these tools by practitioners – instead, it was compatibility. On the basis of responses received, the authors suggest that the use of build automation tools can be increased if, "build automation tools fit well with practitioners' existing workflow and tool usage, and usage of build automation tools are made more visible among practitioners' peers."

Security (code) smells

The concept of code smells describes flaws in code that may lead to a problem. Rahman et al in 'The Seven Sins' identified seven security smells in IaC by performing an experiment on open source repositories. They concluded that security smells can have a long lifespan – for example, a hard-coded secret can remain as long as 98 months, with a median lifetime of 20 months. On the other hand, Schwarz et al presented a catalogue of 17 code smells in IaC which

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="pathKey1" value="C:\realPath1"/>
    <add key="pathKey2" value="C:\realPath2"/>
    <add key="pathKey3" value="C:\realPath3"/>
  </appSettings>
</configuration>
```

Figure 1: An example of a config file.

were applied and focused on the Chef configuration management tool. This indicates that a further investigation of code smells in the area of IaC is needed. Therefore, we will list some code smells (process metrics) for any deployment tools that are not covered by the above-mentioned studies as follows:

Permissions: The source (deployment) server must have write-only permission to destination servers and client machines during the deployment process. This action will avoid reading the source server information from destination servers and clients. In fact, read permission does not always lead to a security breach but makes it easier to gather information on the deployment server. Thus, we consider read permission is a security smell in IaC.

Path configuration: In this smell, it's recommended to save all paths in the config file, which will save time and effort in the case of any path issues brought up. The configuration file is not compiled during the deployment process, so if a case of a path error comes along on a production server, practitioners can change the path from the config file easier and faster. Figure 1 shows an example of a config file protected with a key.

Threat model

Threat modelling is the process of identifying and investigating potential threats in order to find any architectural bugs before they breach security. Although threat modelling can be done at any stage, doing it in the early stages provides the additional benefit of early determination of threats. Figure 2 shows the model that represents the security assurance workflow as proposed by Jim Bird.[12]

Pre-commit stage

At this stage the continuous integration server executes automatic tests on all the proposed changes and team members are engaged in reviewing code. Changes to software and configuration are checked into a source code repo. Security checks and controls at this stage are as follows: lightweight iterative threat modelling and risk assessment; static analysis checking in the IDE of the engineer; and peer code reviews.

Continuous integration stage

This stage is triggered automatically by a check-in. Build and basic automated testing of the system is performed. Fast feedback to developers is returned verifying whether the change breaks the build. The stage is usually completed in a few minutes. The security checks at this stage are as follows:

- Compile and build checks that ensure that the steps are clean with no errors and warnings.
- Software component analysis in the build with identification of risk in third-party components.
- Incremental static analysis scanning for bugs and security vulnerabilities.
- Generating alerts on high-risk code changes with the help of static analysis checks and tests.
- Automatic unit testing of security functions with the help of code coverage analysis.
- Signing binary artifacts digitally and storing them in repositories.

Acceptance stage

The continuous integration server executes a set of automatic acceptance tests. This stage is triggered by a successful commit. The latest good commit build is selected and dispatched to an acceptance test environment. Automated acceptance tests – which involve functional, integration, performance and security tests – are executed. In order to minimise the time required to perform the test, in most cases the tests are fanned out to heterogeneous test servers and executed in parallel. By following the 'fail fast' approach, the most time-consuming and expensive tests are left until as late as possible in the test cycle and are only executed if other tests have already passed.

Security control and tests at this stage are as follows:

- Provisioning of runtime environment and secure automated configuration management.
- Deployment of the latest good build from the binary artifact repository automatically.
- Smoke tests, which are designed to catch mistakes in configuration or deployment.
- Dynamic application security testing.
- Automatic functional and integration testing of security features.
- Automated security attacks with the help of Gauntlt or other security tools.
- Deep static analysis scanning.
- Fuzzing (APIs, files).
- Manual penetration testing.

Production deployment and post-deployment

Upon passing the above-mentioned tests, the change is ready to be deployed to production. Security checks and controls are needed at this stage, including:

- Provisioning of a run-time environment and secure automated configuration management.

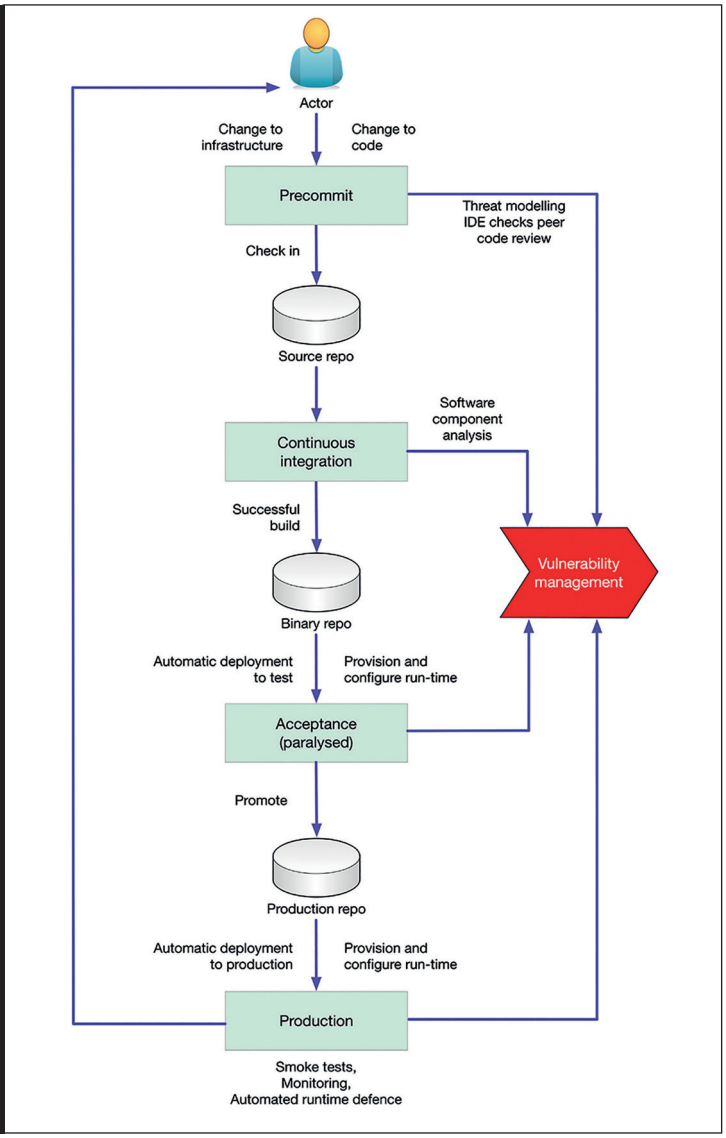

Figure 2: Security assurance workflow for IaC.

- Automated deployment and release orchestration.
- Post-deployment smoke tests.
- Automated run-time assets and compliance checks.
- Production monitoring/feedback.
- Runtime defence.
- Red teaming.
- Bug bounties.
- Blameless postmortems.

All of the above-mentioned practices are implemented according to the risk profile of the organisation.

Vulnerability management module

In order to track vulnerabilities, assess risk and understand trends, this module helps us to view the status of the pipeline, systems and portfolios. There is a need for metrics for compliance and risk-management purposes. These metrics will help us understand how to prioritise testing and training efforts, which helps in assessing the application security program.

By collecting data on vulnerabilities, important information can be gained, such as: the number of vulnerabilities discovered, how the vulnerabilities are discovered and what tools are giving the best returns, what are serious vulnerabilities, how long it's taking to get the vulnerabilities fixed, etc. All this information can be obtained by providing security testing results from continuous delivery pipelines into a vulnerability manager.

The above-mentioned security assurance workflow can be made more secure with the best practices discussed in the following section.

Best practices to secure IaC

To achieve consistency, speed of deployment, simplicity and security in IaC, best practice should be followed. Related efforts in this direction can be found in the references.[13-19] We have worked on consolidating these studies and present the best of the best practices as follows:

Manual security assessment: This step involves manually inspecting the live infrastructure after deployments and reviewing the architecture/templates before they are deployed to a live environment.

Codify everything: All infrastructure specifications should be explicitly coded in configuration files, such as AWS CloudFormation templates, Chef recipes, Ansible playbooks, or any other IaC tool. These configuration files represent the single source of truth of your infrastructure specifications and describe exactly what cloud components you'll use, how they relate to one another and how the entire environment is configured. Infrastructure can then be deployed quickly and seamlessly, and – ideally – no one should log into a server to manually make changes.

IaC documentation: This approach advises that writing documentation should be avoided because the code itself will register the machine status automatically. That means the infrastructure documentation is always up to date. Additional documentation, such as diagrams and other setup instructions, may be necessary to educate employees who are less familiar with the infrastructure deployment process. But most of the deployment steps will be performed by the configuration code, so this documentation should ideally be kept to a minimum.

Version everything: The configuration files should be managed in a version-controlled way. Because all configuration data are written in code, any modifications to the codebase can be controlled, tracked and reconciled. The version control system (VCS) is a core part of managing IaC. The VCS is the source of truth for the overall status of the infrastructure. Any changes in infrastructure will be performed by

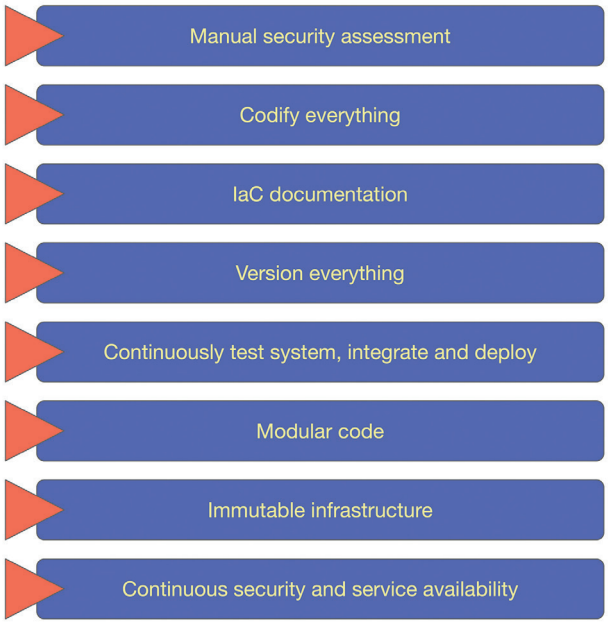

Figure 3: Best practice summary.

changes committed to the VCS. In addition, VCS is important for IaC because it provides the following functions:

- Traceability: Record all changes that have been made.
- Rollback: Restore things back in case of any failures.
- Correlation: Useful for tracing and fixing complex problems when these occur.
- Visibility: All team members can see when changes are committed to VCS.
- Actionability: VCS can trigger actions automatically when a change is committed.

Continuously test system, integrate and deploy: The repeatability of IaC is an enabler of this approach, where the changes first deploy into a test environment. Automated tests verify that there were no security and compliance regressions before deploying the changes to a production server. All of this is managed by an automated CI/CD (continuous integration/continuous deployment) pipeline. Unit tests for configuration code should include security checks such as the following:

- Disable unnecessary services.
- Close unused ports.
- Look for hardcoded credentials and secrets.
- Check and review permissions on files and directories.
- Ensure that development tools are not installed in production servers.
- Check auditing and logging policies and configurations.

Modular code (small changes rather than big batches): Modular infrastructure limits the number of changes that can be made to the configuration. Smaller changes make errors easier to detect and allow the team to fix them. There are many reasons to prefer small changes over big batches, including the following:

- Easier and less effort in case of testing the changes and evaluation.
- Faster to find the cause of the bugs and errors, then easy to fix them.

Immutable infrastructure: This approach takes IaC to the next level. The idea behind immutable infrastructure is that IT infrastructure elements are replaced for each deployment, instead of making changes on current components. This process provides consistency, avoids configuration drift and restricts the impact of undocumented changes. Also, it improves security and makes fixes easier due to the lack of configuration edits.

Continuous security and service availability: Complexity is the enemy of security, so it's important to secure not only the application and its runtime environment but also the continuous delivery toolchain and test environment. Also, it is important to protect the pipeline from insider attacks by ensuring that all changes are fully transparent and traceable from end to end. That will provide secure service availability and ensure the continuous flow of the delivery pipeline.

Figure 3 shows the summary of the best practice approach to secure IaC.

Conclusion

The aim of this article was not only to introduce the reason behind the increasing popularity of 'infrastructure as code' but also to dig deep into an area that is not much researched. Our research in this paper led us to the identification of code smells and their classification in categories such as technology-agnostic and technology-dependent smells. Thus, it can be concluded that these smells provide us with adequate means to test the quality/security of IaC scripts.

The research leads us to information on how IaC scripts help companies in the current IT industry to automatically configure their production environment and how security smells lead to an increment in system weakness and what practices can be used by software practitioners in order to formulate a quality code or IaC script.

Through our research on security in IaC, one thing that became clear is that IaC might be one of the fundamental pillars to DevOps but it is susceptible to defects. In order to eliminate the occurrence of these defects, we have to learn about defect prediction models. Lastly, we built a threat model in order to secure the controls of an IaC. The employability of threat models and their importance in the development stage is what makes them the preferred tool of developers. This practice of making threat models should also be promoted as it can protect end users from a lot of disturbance.

About the authors

Dr Sadiq Almuairfi is currently an e-service director and researcher at the Security Engineering Lab at Prince Sultan University, Riyadh, Saudi Arabia. He received a bachelor's degree in computer engineering from KFUPM, Dhahran, Saudi Arabia in 2001, a master's degree in information manage-

ment from King Abdulaziz University, Jeddah, Saudi Arabia, in 2005 and a PhD in cyber security from La Trobe University, Melbourne, Australia in 2014. His research interests include cyber security, network security and e-commerce security.

Dr Mamdouh Alenezi is currently the dean of educational services at Prince Sultan University. He received his MS and PhD degrees from DePaul University and North Dakota State University in 2011 and 2014 respectively. He has extensive experience in data mining and machine learning, where he applied several data mining techniques to solve software engineering problems. He has worked in several research areas and developed predictive models using machine learning to predict fault-prone classes, comprehend source code and predict the appropriate developer to be assigned to a new bug.

References

1. Artac, M; Borovssak, T; Di Nitto, E; Guerriero, M; Tamburri, D. 'DevOps: Introducing Infrastructure-As-Code'. 2017 IEEE/ACM 39Th International Conference On Software Engineering Companion (ICSE-C).
2. Fowler, M; Beck, K; Brant, J; Opdyke, W; Roberts, D; 'Roberts, Refactoring: Improving the Design of Existing Code'. Addison-Wesley Longman Publishing, 2018.
3. Schwarz, J; Steffens, A; Lichter, H. 'Code Smells in Infrastructure as Code'. 11th International Conference On the Quality of Information and Communications Technology (QUATIC), 2018.
4. Rahman, A; Parnin, C; Williams, L. 'The Seven Sins: Security Smells in Infrastructure as Code Scripts'. IEEE/ACM 41st International Conference On Software Engineering (ICSE), 2019.
5. Artac, M; Borovsak, T; Di Nitto, E; Guerriero, M; Perez-Palacin, D; Tamburri, D. 'Infrastructure-As-Code for Data-Intensive Architectures: A Model-Driven Development Approach'. IEEE International Conference On Software Architecture (ICSA), 2018, pp.156-165.
6. Rahman, A. 'Anti-Patterns in Infrastructure as Code'. IEEE 11th International Conference On Software Testing, Verification and Validation (ICST), 2018.
7. Rahman, A; Partho, A; Morrison, P; & Williams, L, 'What Questions Do Programmers Ask about Configuration as Code?', 2018 ACM/IEEE 4Th International Workshop On Rapid Continuous Software Engineering
8. Chen, W; Wu, G; Wei, J. 'An Approach to Identifying Error Patterns for Infrastructure as Code'. 2018 IEEE International Symposium On Software Reliability Engineering Workshops (ISSREW).
9. Lavriv, O; Klymash, M; Grynkevych, G; Tkachenko, O; Vasylenko, V. 'Method of cloud system disaster recovery based on "Infrastructure as a code" concept'. 14th International Conference On Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), 2018.
10. Rahman, A; Stallings, J; Williams, L. 'Defect Prediction Metrics for Infrastructure as Code Scripts in DevOps'. 2018 ACM/IEEE 40th International Conference On Software Engineering: Companion Proceedings.
11. Rahman, A; Partho, A; Meder, D; Williams, L. 'Which Factors Influence Practitioners' Usage of Build Automation Tools?'. IEEE/ACM 3rd International Workshop On Rapid Continuous Software Engineering (Rcose), 2017.
12. Bird, Jim. 'Security as Code: Security Tools and Practices in Continuous Delivery'. O'Reilly Media, 2016. Accessed Sep 2020. www.oreilly.com/library/view/devopssec/9781491971413/ch04.html.
13. Feintuch, Roy. 'New Security Challenges with Infrastructure-as-Code and Immutable Infrastructure'. The New Stack, 4 Apr 2018. Accessed Jan 2020. https://the-newstack.io/new-security-challenges-with-infrastructure-as-code-and-immutable-infrastructure.
14. Chan, Mike. 'Infrastructure as Code: 6 best practices to get the most out of IaC'. Thorn Technologies, 27 Feb 2018. Accessed Jan 2020. www.thorntech.com/2018/02/infrastructure-as-code-best-practices/.
15. 'What Is Infrastructure as Code? How It Works, Best Practices, Tutorials'. Stackify, 5 Sep 2019. Accessed Jan 2020. https://stackify.com/what-is-infrastructure-as-code-how-it-works-best-practices-tutorials/.
16. Null, Christopher. 'Infrastructure as code: The engine at the heart of DevOps'. TechBeacon. Accessed Jan 2020, https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops.
17. Hornbeek, Mike. '9 Pillars of Continuous Security Best Practices'. DevOps.com, 8 Jan 2019. Accessed Jan 2020, https://devops.com/9-pillars-of-continuous-security-best-practices/.
18. Morris, Kief. 'Challenges and Principles'. In: 'Infrastructure as Code', O'Reilly Media, 2015.
19. 'Secure development and deployment guidance'. National Cyber Security Centre (NCSC). Accessed Jan 2020, www.ncsc.gov.uk/collection/developers-collection.