GoSecure: Securing Projects with Go

Maria Spichkova, Achal Vaish, David C. Highet, Isthi Irfan, Kendrick Kesley and Priyanga D. Kumar
School of Science, RMIT University, Melbourne, Victoria 3000, Australia

Keywords: Software Engineering, Cloud Security.

Abstract: This paper presents an automated solution for security vulnerability scanning of Google Cloud Platform (GCP) projects, to cover gaps in the capabilities of solutions to scan GCP projects for common security issues. The elaborated security inspection tool, GoSecure, can scan multiple GCP instances against industry recognised Center for Internet Security (CIS) benchmarks for GCP. GoSecure covers all categories listed under the CIS benchmarks for GCP, providing an overview of the existing security profile of all GCP projects, along with suggestions for improvement in configurations for the individual projects.

1 INTRODUCTION

Cloud Security is of increasing importance to many enterprises as companies migrate data and applications to cloud computing resources from in house data centres. Securing resources in the Cloud brings a different set of challenges to an enterprise.

Our research is focused on securing Google Cloud Platform (GCP) projects, and scaling the security management when dealing with multiple projects. GCP was introduced in 2008 with their first product, Google App Engine (Ciurana, 2009; Zahariev, 2009). In 2018, Google Cloud Platform has a worldwide market share of 9.5%, claiming the third position after Amazon Web Services and Microsoft Azure (Alto, 2019).

One of the most common and dangerous incorrectly configured resource are storage buckets such as those provided by Google Cloud (Cloud Storage) or Amazon S3 (Amazon's cloud storage solution) where the bucket has been incorrectly configured to provide public access to sensitive data. An example of this is a 2019 breach where Medico, Inc., a healthcare vendor left an S3 bucket exposed to public access. The bucket titled "medicoar" contained 1.7GB of data including insurance claims and personal medical information, see (Upguard Inc., 2019). Breaches such as these can be costly to companies in terms of reputation, fines for regulatory non-compliance and lost business. For individuals affected having personally identifiable information exposed (PII) can leave individuals susceptible to fraud.

The security requirements are initially anchored to CIS (Center for Internet Security) Benchmarks for Google Cloud Platform. These benchmarks focuses on 7 fundamental services which are Identity and Access Management, Logging and Monitoring, Networking, Virtual Machines, Storage, Cloud SQL Database Services, and Kubernetes Engine as they are seen as the foundations of most projects. Security Solutions based off similar benchmarks such as the Center for Internet Security (CIS) benchmarks can assist enterprises with ensuring that security vulnerability checks are provided consistently irrespective of which platform an application is deployed to.

Contributions. The purpose of our project is to provide solutions for gaps in the capabilities of existing software to scan GCP projects for common security issues. Our aim is to develop a security inspection tool in the form of a web application that scans multiple GCP instances against industry recognised CIS benchmarks for GCP. CIS provides a document listing benchmarks in 7 security categories.

To solve the problem, we developed a web-based tool, GoSecure, which could be deployed to multiple organizations with capabilities to manage users at the team, project or user level. The application uses a microservices architecture hosted in GCP which allows for automatic scaling of resources and high levels of availability. The tool covers all categories listed under the CIS benchmarks for GCP, providing an overview of the existing security profile of all GCP projects, along with suggestions for improvement in configurations for the individual projects.

Outline. The rest of the paper is organised as follows. Section 2 presents the background analysis and

Spichkova, M., Vaish, A., Highet, D., Irfan, I., Kesley, K. and Kumar, P.
GoSecure: Securing Projects with Go.
DOI: 10.5220/0009591605870594
In *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, pages 587-594
ISBN: 978-989-758-421-3

related work. The proposed solution is introduced in Section 3, which implementation is then discussed in Section 4. Finally, Section 5 summarises the paper.

2 BACKGROUND AND RELATED WORK

In this section, we would like to provide an introduction to the background of our work: Center for Information Security benchmarks, Google Cloud Platform (GCP) Services as well as existing security solutions for GCP. For a general analysis of the security issues as well as the corresponding comparison of cloud platforms we would like to refer to the following publications. A comprehensive analysis of existing challenges and issues involved in the cloud computing security problem, see (Almorsy et al., 2016). A survey on security and privacy in cloud computing was introduced in (Zhou et al., 2010). A more recent survey on security mechanisms of Amazon Web Services, Google Cloud Platform, Microsoft Azure, IBM SmartCloud, and Rackspace, was presented in (Panth et al., 2014).

2.1 CIS Benchmarks

Center for Internet Security (CIS)[1] is a nonprofit organisation, established in 2000, which provides a set of benchmarks and software intended to improve internet security. The CIS benchmarks are offered for all three major public Cloud providers (AWS, GCP and Azure). This is an advantage as tools can be built with consistent security standards across all three clouds using the benchmarks. As noted in the introduction the CIS benchmarks for Google Cloud Platform cover seven areas: Identity and Access Management, Logging and Monitoring, Networking, Virtual Machines, Storage, Cloud SQL Database Services, and Kubernetes Engine.

The CIS benchmarks have the concept of 2 levels. Level 1 is the minimum level of security that any project should have. Level 2 is a higher level aimed at projects which require a higher level of security. These are not recommended for every project as implementing “may negatively inhibit the utility or performance of the technology”, as per CIS definition. They are intended for defence in depth and where security is paramount. The benchmarks each have a note marking whether they are “scored” or “not scored”. These are used for the tools provided by CIS security and provide a compliance score. As the scoring method does not appear to be publicly available we have chosen not to implement a scoring method within the current project. As the solution we have created is intended to scan for all benchmarks included in the Google Cloud Platform Foundation Benchmarks a brief outline of each section follows to give some context to what the solution implements.

The core CIS benchmarks are as follows:

- *Identity and Access Management Benchmarks*: include service account, service and API key and credential standards. An example is a benchmark to ensure multifactor authentication is enabled for each GCP account.
- *Logging and Monitoring Benchmarks*: are concerned with ensuring that good logging and monitoring practices are followed so that security analysis, resource change tracking and compliance management can take place. Ensuring that log sinks are enabled for storage buckets is an example.
- *The Networking Benchmarks*: cover areas such as preventing SSH access, controlling VPC access and ensuring that potentially insecure default or legacy networks are not present.
- *Virtual Machines Benchmarks*: are rules for Google Compute VM instances. It includes rules around SSH keys, IP forwarding and oslogin issues.
- *Storage Benchmarks*: include IAM and logging rules for Storage buckets.
- *The Cloud SQL Benchmarks*: are largely concerned with ensuring that unauthorised or inappropriate users/systems can not access a Cloud SQL Database. This can include ensuring SSL is activated for all incoming connections or that users with administrative privileges connect with a password for example.
- *Kubernetes Engine Benchmarks*: are rules for Google Kubernetes Engine (GKE) deployments. It covers a wide range of recommendations including logging, authorisation and various other configuration recommendations.

2.2 Google Cloud Platform Services

Google Cloud Platform offers a very large range of services which are too numerous to mention here. This section provides an overview of some of the important services that the CIS benchmarks cover and some of the security considerations that need to be taken into account. Especially important from a security perspective are services which expose data to the open internet:

[1]https://www.cisecurity.org/

- *Compute Engine*: is an Infrastructure as a Service (IaaS) which enable creating and running Virtual Machines on Google Infrastructure.
- *Kubernetes Engine*: is a managed environment for deploying Kubernetes clusters. Kubernetes is system for automating deployment, scaling, and management of containerised applications.
- *App Engine*: is a file storage service for accessing data hosted on GCP infrastructure. It offer similar capabilities to the well known AWS S3 storage service.
- *CloudSQL*: is a managed service for administering and running Relational Databases hosted in Google Cloud. It works with either MySQL or PostgreSQL databases.

2.3 Existing Security Solutions

The existing solutions to Cloud Security on the GCP platform and beyond can be divided in two groups: enterprise solutions (such as GCP Audit, Forseti Security, Cloud Security Scanner, and AWS Trusted Advisor) and non-enterprise solutions (auch as Hayat). In what follows we present them in more details.

GCP Audit[2]: was initially introduced by Spotify as an internal tool, which was used to scan their 800+ GCP projects for security vulnerabilities. This tool was based on an internal rules repository which had rulesets defined that was customised to their needs. The overall goal of Spotify's GCP Audit tool was to highlight common security issues such as permissions, misconfigured SQL and exposed instances to the public internet. This tool was later open sourced and discontinued whilst Spotify moved on to contribute to the Forsetti Security tool.

Forseti Security[3]: is a set of open source security tools for GCP. It is is built as a traditional server application running in dedicated Virtual Machines. Forseti Security uses a system of modules which includes Inventory for information about resources, Real-Time Enforcer for remediation and Scanner for scanning projects. The Scanning functionality looks for misconfiguration and security bugs.

Cloud Security Scanner[4]: is a web security scanner provided by Google Cloud to scan App Engine, Compute Engine, and Google Kubernetes Engine apps for common vulnerabilities. Scans can be run on-demand or at scheduled times. It is aimed at common web vulnerabilities and can scan for XSS, Flash injection, Mixed-content, Clear text passwords and usage of insecure JavaScript libraries. Cloud Security Scanner could be considered complementary to the solution we have developed as it has a limited amount of vulnerabilities but these are not covered under the CIS benchmarks.

AWS Trusted Advisor[5]: is a well known SaaS offered by Amazon for their public cloud that implements seven basic checks. More checks are available on higher tier support plans. It covers not just Security, but also performance, Fault Tolerance and Service Limit checks. It is comparable to our solution in that it does not need to be self-hosted as it is offered as a SaaS.

Hayat[6]: is a GCP Auditing and hardening script developed by an independent programmer. It's available as an open source Github project and employs the bash scripting structure with a combination of Unix and gcloud commands. The domains covered by this project are based on CIS benchmarks and is closely comparable to our solution. However, the script-based tool is considered more as a non-enterprise solution since it is suited for an ad-hoc scanning of a GCP project but cannot be used for a multi-project instance.

3 PROPOSED SOLUTION

In this section we introduce the proposed solution: its architecture as well as the way the project can be managed within the system. The back-end is written in the Go language(Donovan and Kernighan, 2015), which also has given the name of the proposed system: *GoSecure*

3.1 System Architecture

Figure 1 presents the architecture of our proposed solution. The front-end of the application is a static website stored in a Cloud Storage bucket. All subsystems will be deployed into a Google Cloud Platform project. We propose to use *CloudBuild* to build and deploy those subsystems to production environment.

Google Cloud Functions are a Function as a Service (FaaS) similar in operation to Amazon Lambda. They are Google Cloud's solution for providing event-triggered stateless compute containers and we use them to perform all the business logic in our application. A function or a group of functions is executed in a dedicated Cloud Function. As shown in

[2]https://github.com/spotify/gcp-audit

[3]https://forsetisecurity.org

[4]https://cloud.google.com/security-scanner

[5]https://aws.amazon.com/premiumsupport/echnology/trusted-advisor/

[6]https://github.com/DenizParlak/hayat

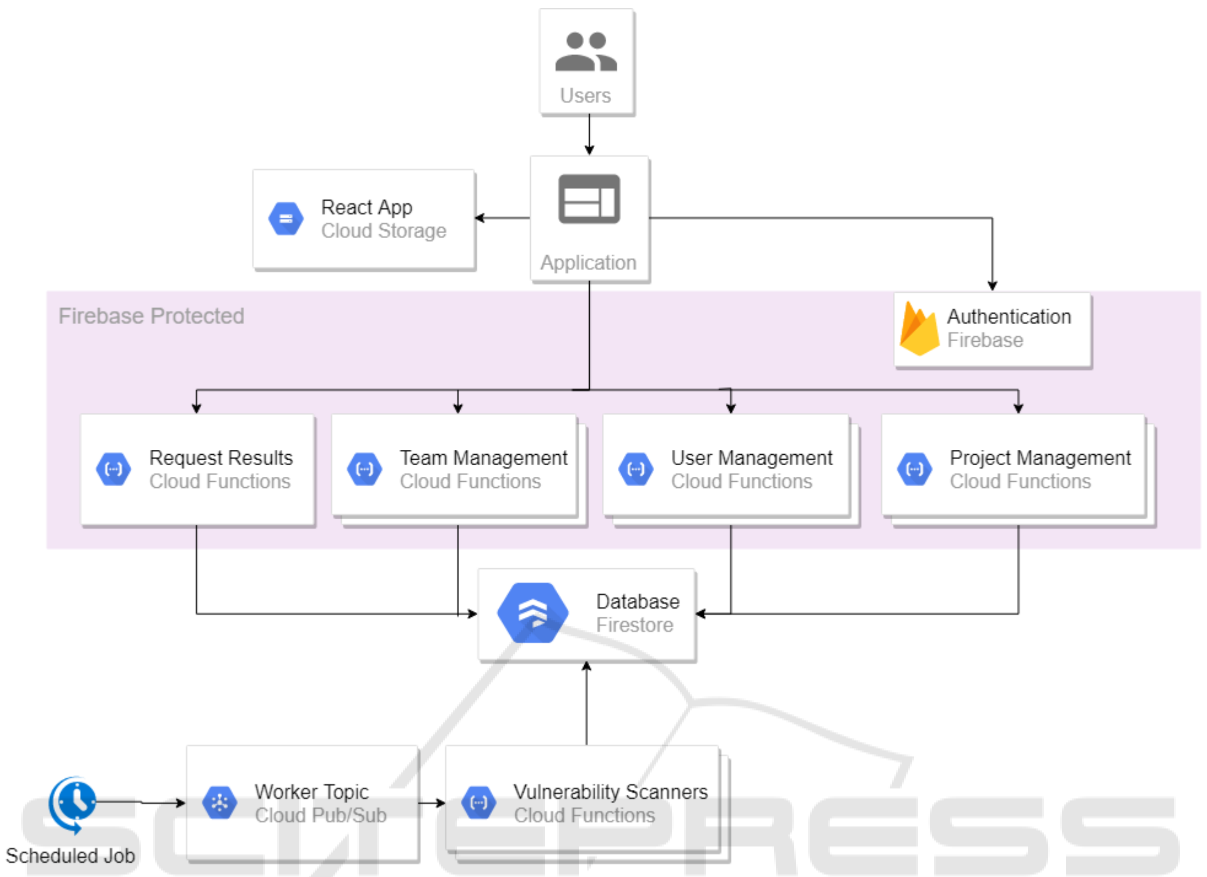

Figure 1: Architecture of the proposed solution.

Figure 1 processing of scanning results, management of team, users and projects and the vulnerability scanners themselves are all handled by Cloud Functions which communicate with the database and the front end to process and serve results and managerial functions.

The solution applies serverless architecture to define the scanners individually, specifically using Google Cloud Function. Each function will be responsible for a single service, and will hold multiple tests to be checked against the projects. The scanners are using Google Cloud Platform API for Go to check services configurations. A serverless architecture provides benefits in terms of scalability, reliability and ease of development. For example, Google Cloud will automatically create new instances of Cloud Functions where the number of incoming requests exceed the number of instances available. Each instance of a Cloud Function can only handle one concurrent request at a time so new instances must be created for each request. Moreover, compared to hosting the application in a traditional instance, serverless architecture provides a more cost-efficient solution as it only charges while the function is running. However, this is only true until a certain point. When the requests are very large, hosting an instance is cheaper as the serverless will count every request as an invocation.

The API requires a form of authentication. Our proposed solution uses a custom service account for each project which will be uploaded by the users. Scanners which does not have sufficient permissions will give an empty result. However, the system as a whole allows partial permissions (e.g., the service account has *IAM* permissions but not Storage).

Vulnerability Scanning is triggered by a Cron job. Thus, each scanner is be triggered repeatedly using a *CRON* job which is managed by a service called Cloud Scheduler. Each project will be assigned to a single *CRON* job. Therefore, users can customise the interval for each of their projects. At the end of every scanning job, the function will save the test result. This result are saved independently in which each result would have a *uuid* of their own. The result structure is presented in Figure 2, where *group* denotes the service for individual service (in this case it's referring to *IAM* service), and *project* denotes the project which the result is for. *issues* are an array of combined tests for the service. Each test is identified by *id* field which will have a *resources* key to hold the issues, if exist, for each resource.

```
{
   "group": "giam",
   "previous_uuid": "xxxxxxxx-xxxx-xxxx-xxxx-
       xxxxxxxxxxxx",
   "project": "gosecure",
   "status": 200,
   "timestamp": "last scanning time",
   "uuid": "yyyyyyyy-yyyy-yyyy-yyyy-
       yyyyyyyyyyyy",
   "issues" [
       {
           "id": "1.1",
           "resources": [
               {
                   "name": "gosecure",
                   "resource_issues": [
                       {
                           "issue_time": "recorded
                               time for this issue
                               ",
                           "text": "issue
                               description"
                       }
                   ]
               }
           ]
       }
   ]
}
```

Figure 2: Result structure of the vulnerability scanning.

The front-end of the application is hosted by the Google Cloud storage which serves the front end part of the system. One significant reason for using the cloud storage is to increase the availability of the designed website as cloud storage has redundant servers.The application interacts with the cloud function APIs to display the security vulnerabilities category wise along with their recommendations to resolve each issue. A centralized state management is achieved by the system with the use of React-Redux technology. Redux aids in the consistency of the application regardless of the environment it is hosted. Another API is also proposed to get any result based on the *uuid* of the result. This is used to compare the latest result with the previous one to generate *diffs* from both results. *Diffs* can be used to quickly identify new reported vulnerabilities and user/organisation can act accordingly.

The core aim of the proposed *GoSecure* application is to identify the issues. An important extension is also how users may correct the issues that are discovered. The CIS benchmarks provide remediation instructions for each issue. The application shows this remediation verbatim to the user. No automatic remediation is provided as the application does not have write access to projects.

3.2 Project Management

A Google Cloud Platform's customer can have multiple projects in their organisation, and each of their project might have multiple instances with varying configurations. Our proposed solution has the ability to manage projects which include creating, updating, and removing projects from our scanning schedules. To be able to add a project, users need to upload a service account key which give our proposed solution access to read their deployment configurations.

For every project, an encryption key is created using Google *KMS* (Key Management Service), and stores the project's encrypted service account key into a *Firestore* collection. However, Google Cloud Platform does not allow key deletion. Therefore, when the project is deleted from our system, the key won't be deleted as well. Each project in *Firestore* will also hold a key to their latest results for all services. We propose to only use the result's *uuid* in here to reduce duplication. Figure 3 illustrates this idea on a sample project. *schedule* will hold the interval between scanning. It is broken down into the *interval* and *time_unit* so we can have flexible timings.

```
{
   "id": "gosecure",
   "key": "encrypted service account key",
   "schedule": {
      "interval": 3,
      "time_unit": "hour"
   },
   "latest_result": {
      "giam": { "uuid": "aaaaaaaa-aaaa-aaaa" }
         ,
      "gkubernetes": { "uuid": "bbbbbbbb-bbbb"
         },
      "glogging": { "uuid": "cccccccc-cccc" },
      "gnetwork": { "uuid": "dddddddd-dddd" },
      "gsql": { "uuid": "eeeeeeee-eeee-eeee" }
         ,
      "gstorage": { "uuid": "ffffffff-ffff" },
      "gvm": { "uuid": "gggggggg-gggg-gggg-
         gggg" }
   }
}
```

Figure 3: Sample project.

Our proposed solution for identity and access management includes a fine-grained permission control for projects. Permissions are attached to a concept called *pod*. Each *pod* can have multiple users. Every user who belongs to a *pod* will derive the permissions of that *pod*. Users may belong to multiple *pods*. In that case, we aggregate the permissions of a user's pods, and use the aggregated permissions to interact with the system. Figure 4 presents a sample of our *pod* structure. *roles* will determine which actions and resources are allowed for a pod. The *key* denotes allowed actions, while the *value* denotes allowed resources for that specific action. We currently have 8 action types, which can be seen below.

- *project/read*: read projects which are registered through the system and their configurations.
- *project/write*: add new projects, update existing project configurations, and delete projects from the system.
- *project/result*: view the vulnerability reports for allowed projects.
- *pod/read*: read pods which are registered through the system.
- *pod/write*: add new pods, update existing pods, and delete pods from the system.
- *pod/member*: manage members in allowed pods. Users with this permission can add or remove members from the pods.
- *user/read*: read users who are registered through the system.
- *user/write*: add new users, update existing users, and delete users from the system.

```
{
   "uuid": "aaaaaaaa-aaaa-aaaa-aaaa-
        aaaaaaaaaaaa",
   "name": "Security Reviewer",
   "roles": {
      "project/result": ["project-a", "project
          -b", "project-c"]
   },
   "members": ["user-a", "user-b"]
}
```

Figure 4: Sample of the *pod* structure.

Our solution uses Google *Firebase Authentication* to authenticate users. Google Firebase is a mobile and web application development platform which offers a drop in authentication solution. However, we won't have any correlation between users and pods in *Firebase*. Therefore, we're not relying only in *Firebase*, but also in our customised function to authorise users. We also need to replicate certain data from *Firebase* to our *Firestore* collection as there are some performance benefit when listing users in pods. Figure 5 presents a sample of user model in our *Firestore*.

```
{
   "uid": "user-a",
   "email": "john.doe@example.com",
   "display_name": "John Doe",
   "pods": ["aaaaaaaa-aaaa-aaaa-aaaa-
        aaaaaaaaaaaa"]
}
```

Figure 5: Sample of user model.

The proposed solution will replicate *uid*, *email*, and *display_name* from *Firebase*. Every time users change their *email* or *display_name* in our system, we reflect the updates to *Firebase* as well.

We also propose to store users' membership into both *pods* and *users* collections. This is so that we can list all users when reading pods efficiently and vice versa. Without this duplication, if we store the membership in *pods* collection, we would have to search for members in *pods* collection if we want to display the pods that a user belongs to. On the other hand, if we store the membership in *users* collection, we would have to search for pods in *users* collection if we want to display all members of a pod.

4 DISCUSSION

The proposed GoSecure system is highly relevant to many segments of end users and meet their expected needs in scanning applications for security vulnerabilities. Firstly, a major advantage of the design would be that, its loosely coupled serverless functions which means additional features can be added and irrelevant features can be removed too.

Similarly, the security standards that define rules are based off CIS Benchmarks which is an industry accepted standard for application security benchmarking that our application follows which makes the application maintainable as standards continuously reviewed, making it efficient in updating rules in GoSecure.

From a client perspective no deployment is necessary as our solution operates as 'Software as a Service'. To set up a user must be authorised for an account and can then set up users, pods and add projects by providing a key for that project. This is different to solutions that require deployment on the users own infrastructure. As the end user does not need to deploy the software this saves time and does not require knowledge of how to deploy the GoSecure software.

Deploying an alternative solution to a Virtual Machine (VM) can entail significantly more time and knowledge to deploy. For example, Forseti Security requires creating a new GCP project, deploying a server VM, a Forseti security service account, creating a database and installing the client to use as an end user. An alternative is to deploy to Kubernetes but this requires an IT administrator with Kubernetes knowledge.

Forseti Security uses an IAM plugin called "IAM Explain" to facilitate Identity and Access Management for the solution. There are additional IAM roles needed for the forseti-gcp-reader service account over what is required for normal Forseti Security operation. None of the additional IAM roles are documented on the Forseti Security site (Lesperance, 2018). The additional roles needed for the forseti-gcp-reader service account to allow Forseti Security IAM Explainer to function:

`Organization Level Roles`

- `Container Analysis Notes Attacher`
- `DLP Jobs Reader`
- `Organization Creator`
- `StackDriver Maintenance Window Editor`
- `StackDriver Maintenance Window Viewer`

In contract to this solution, GoSecure manages roles and permission using the concept of pods and users to ensure security. Pods are essentially groups of users with the same roles and permissions. Each pod has certain permissions and resources associated with it:

```
Permissions
  a Admin:  Members can read/write to the
    resource and add other non-admin members to
    the pod.
  b Read:  Members can view or read from the
    resource
  c Write:  Members can make changes to the
    resource
  d Member Manager (only for pods):  A
    non-admin member
Resources
  a Project:  Permissions apply to project-wide
    scope
  b User:  Permissions apply to user data
    management
  c Pod:  Permissions apply to pod data
    management
```

For instance, the project admins for the various projects under a GCP instance are assigned to the same pod. In this scenario, admin is defined as the type of role and project as type of the resource, which is beneficial for the project management.

5 CONCLUSIONS

This paper presents a web-based solution for security vulnerability scanning. Our tool, GoSecure, provides an effective mean for managing multiple Google Cloud Projects across an organisation with fine-grained permissions. This covers a large gaps in the capabilities of solutions to scan GCP projects for common security issues.

Implementing the CIS benchmarks with regular scheduled scans helps an organization to ensure that security standards are maintained consistently across the organization's GCP projects. GoSecure not only covers all categories listed under the CIS benchmarks for GCP, but also provides an overview of the existing security profile of all GCP projects, along with suggestions for improvement in configurations for the individual projects.

This research project was conducted under the initiative *Research embedded in teaching*, see (Spichkova, 2019; Simic et al., 2016; Spichkova and Simic, 2017). This initiative was proposed at the RMIT University (Melbourne, Australia) within the Software Engineering projects in collaboration with industrial partners. The aim of this initiative is to encourage students' curiosity for Software Engineering and Computer Science research. To reach this aim we include research components as bonus tasks in the final year projects (on both undergraduate and postgraduate levels), which typically focus on software and system development. Few weeks long research projects have been sponsored by industrial partners, who collaborated with the students and academic advisers through the final year projects. Respectively, the topics of these short research projects focus align the topics final year projects. The successful results of this initiative are presented in (Christianto et al., 2018; Clunne-Kiely et al., 2017; Spichkova, 2018; Spichkova et al., 2018; Spichkova et al., 2019b; Sun et al., 2018; Chugh et al., 2019; Gaikwad et al., 2019; Spichkova et al., 2019a).

ACKNOWLEDGEMENTS

We would like to thank Shine Solutions Group Pty Ltd for sponsoring this project under the research grant RE-04210. We also would like to thank the experts from the Shine Solutions Group, especially Aaron Brown, for numerous discussions as well as their valuable advice and feedback.

REFERENCES

Almorsy, M., Grundy, J., and Müller, I. (2016). An analysis of the cloud computing security problem. *arXiv*.

Alto, P. (2019). Cloud infrastructure spend grows 46% in q4 2018 to exceed us$80 billion for full year. Canalys Press Release.

Christianto, A., Chen, P., Walawedura, O., Vuong, A., Feng, J., Wang, D., Spichkova, M., and Simic, M. (2018). Enhancing the user experience with vertical transportation solutions. *Procedia computer science*, 126:2075–2084.

Chugh, R., Chawla, N., Gracias, R. M., Padda, J. S., Li, S., Nguyen, M. T., Spichkova, M., and Mantri, N. (2019). Automated gathering and analysis of cannabinoids treatment data. *Procedia Computer Science*, 159:2570–2579.

Ciurana, E. (2009). *Developing with Google App engine*. Apress.

Clunne-Kiely, L., Idicula, B., Payne, L., Ronggowarsito, E., Spichkova, M., Simic, M., and Schmidt, H. (2017). Modelling and implementation of humanoid robot behaviour. In *21st International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, pages 2249–2258. Elsevier Science Publishers BV.

Donovan, A. A. and Kernighan, B. W. (2015). *The Go programming language*. Addison-Wesley Professional.

Gaikwad, P. K., Jayakumar, C. T., Tilve, E., Bohra, N., Yu, W., and Spichkova, M. (2019). Voice-activated solutions for agile retrospective sessions. *Procedia Computer Science*, 159:2414–2423.

Lesperance, J. P. (2018). A review of forseti security for gcp. https://www.jplesperance.me/2018/02/a-review-of-forseti-security-for-gcp.

Panth, D., Mehta, D., and Shelgaonkar, R. (2014). A survey on security mechanisms of leading cloud service providers. *International Journal of Computer Applications*, 98(1):34–37.

Simic, M., Spichkova, M., Schmidt, H., and Peake, I. (2016). Enhancing learning experience by collaborative industrial projects. In *ICEER 2016*, pages 1–8. Western Sydney University.

Spichkova, M. (2018). Automated analysis of the impact of weather conditions on medicine consumption. In *2018 25th Australasian Software Engineering Conference (ASWEC)*, pages 166–170. IEEE.

Spichkova, M. (2019). Industry-oriented project-based learning of software engineering. In *ICECCS*, pages 51–61. IEEE.

Spichkova, M., Bartlett, J., Howard, R., Seddon, A., Zhao, X., and Jiang, Y. (2018). Smi: Stack management interface. In *23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 156–159.

Spichkova, M. and Simic, M. (2017). Autonomous systems research embedded in teaching. In *Intelligent Interactive Multimedia Systems and Services*, pages 268–277. Springer.

Spichkova, M., van Zyl, J., Sachdev, S., Bhardwaj, A., and Desai, N. (2019a). Comparison of computer vision approaches in application to the electricity and gas meter reading. In *ENASE*, pages 303–318. Springer.

Spichkova, M., van Zyl, J., Sachdev, S., Bhardwaj, A., and Desai, N. (2019b). Easy mobile meter reading for non-smart meters. In *14th International Conference on Evaluation of Novel Approaches to Software Engineering*. IEEE.

Sun, C., Zhang, J., Liu, C., King, B. C. B., Zhang, Y., Galle, M., Spichkova, M., and Simic, M. (2018). Software development for autonomous and social robotics systems. In *International Conference on Intelligent Interactive Multimedia Systems and Services*, pages 151–160. Springer.

Upguard Inc. (2019). Medical procedure: How a misconfigured storage bucket exposed medical data. https://www.upguard.com/breaches/data-leak-hipaa-medico-s3.

Zahariev, A. (2009). Google app engine. *Helsinki University of Technology*, pages 1–5.

Zhou, M., Zhang, R., Xie, W., Qian, W., and Zhou, A. (2010). Security and privacy in cloud computing: A survey. In *2010 Sixth International Conference on Semantics, Knowledge and Grids*, pages 105–112. IEEE.