Possum PIE[1]: An Integrated Approach to Cloud-Security Analysis

Motivation and Challenges

It is well known that protecting resources on the cloud is challenging. Various industry reports[2] [3] show that misconfiguration of security (e.g., access control) and unmonitored activity (e.g., who is accessing what) are the main causes of high-profile data breaches. This is because it is not easy to identify the set of security policies that are needed for an application to function correctly and securely. Permissions need to be specified at multiple levels including identity and access management at the resource level, network security groups and lists, creation of suitable subnets etc. The cumulative effect of all these policies is hard to analyse. This is because of various reasons, including fragmentation of cloud security which result in potentially conflicting policies, e.g., between application-level controls and deployment/infrastructure related controls. Because those in charge of cloud service deployments do not want to block legitimate behaviours, they tend to over-grant permissions leading to an increased attack surface. Such over-granting of permissions has led to leaking of sensitive data as well as take-over of the infrastructure.

The problem becomes more complicated when one considers the multi (or hybrid) cloud deployments[4]. One needs to be able to gather sufficient information from the different deployments before conducting the security analysis.

---

[1] From Possum PIE in keeping with naming our projects after exotic desserts. PIE stands for Policy Inference and Analysis Engine

[2] https://www.cybersecuritydive.com/news/cloud-attacks-weak-credentials/721573/

[3] https://snyk.io/learn/aws-security/aws-security-breaches/

[4] https://docs.oracle.com/en-us/iaas/Content/multicloud/Oraclemulticloud.htm

Suitable tooling that addresses all the following issues does not exist.

- What are the necessary permissions for a service principal to access a resource?
- Who has access to perform a sensitive operation (e.g., delete) on a resource?
- Given a particular security posture for an application, can the security posture be tightened?
- Can the necessary configurations, including permissions be automatically synthesised?

From a practical perspective, developers and cloud-deployment managers have a notion of intent. Because the intent is not captured formally, it is not possible to automate any analysis. For example, current technologies do not enable one to check if the permissions associated with a particular deployment satisfies the principle of least privilege. Ideally, generating the set of permissions that satisfy the principle of least privilege for a given application will provide a provable security guarantee. It is also necessary to have tooling to ensure that this guarantee continues to hold as the system evolves. Hence tool integration into the DevSecOps process is important. To simplify such an integration, a single tool that can automate all the required aspects of policy analysis is required.

Overview of Possum PIE

Our research project, called Possum PIE[5], is a cloud security policy inference and analysis engine. The aim is to support at least the following use-cases.

1. Analysis of given deployment: Check whether a given deployment satisfies the required security properties.
2. Least-privilege analysis: Derive the least set of permissions required based on source code and operation logs.
3. Self-provisioning: Infer and synthesise security-related properties that can be used as part of the deployment process.

To support the first two use-case, i.e., given a deployment of an application, Possum PIE extracts all the relevant permissions. This, typically, involves the following,

- Static analysis of available components such as source code, especially if it invokes standard APIs which have explicit permissions attached to them, infrastructure as code specifications such as Terraform, configuration information for deployment.

[5] https://www.allrecipes.com/recipe/218440/southern-possum-pie/

- Dynamic analysis of the execution, including gathering of information from event-logs that are captured (e.g., that support the CNCF standard called CloudEvents[6]).
- Automated test case generation to exercise various APIs of the application to improve coverage of required permissions. This could involve some form of introspection of actual deployment using relevant APIs.

The different analyses rely on a knowledge base that maps permissions to the behaviour exhibited by artifact under analysis. This knowledge base needs to be extracted from various sources. For example, the mapping of permissions to APIs is usually available either as comments in the source code or explicit documentation aimed at users of the services[7]. Once such a knowledge base is available, one can examine behavioural logs to determine which permissions are used by the application.

The high-level architecture is shown in Figure 1.

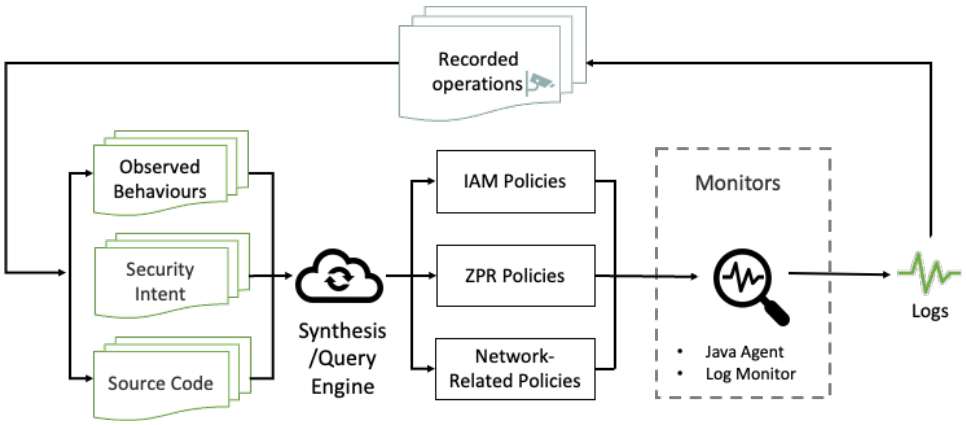

Figure 1: Architecture of Possum PIE

Once the relevant permissions are extracted, Possum PIE translates them into a suitable intermediate representation. This translation enables Possum PIE to analyse the interactions between the different policies. For example, an IAM policy may allow a service principal to access a resource. However, this may not be feasible because of the network topology and associated network security group policies. In this case, flagging a system error is incorrect, although one can report that the IAM and network policies are not consistent. Ultimately, Possum PIE needs to support all the different ways security

[6] https://cloudevents.io/

[7] https://docs.public.oneportal.content.oci.oraclecloud.com/en-us/iaas/Content/Identity/policyreference/corepolicyreference_topic-Permissions_Required_for_Each_API_Operation.htm

restrictions can be placed on accessing resources be they via IAM, network topology or other information flow restrictions.

The third use-case of self-provisioning extends the technologies developed to support the first two use-cases. This is part of our shift-left strategy where the information from the developer(s) is analysed, and the required permissions are inferred. This process can involve static analysis as well as dynamic analysis as part of the staging environment. The inferred permissions are then used as part of an automated deployment process which ensures that the desired security properties are met. Hence, no further introspection of a running system is required.

Our prototype implementation supports the analysis of a snapshot. That is, the analysis considers only a specific state of the system. All the policies are expressed in a suitable logic which enables the use of an SMT solver like Z3 to check the security requirements. In future, we will expand Possum PIE to perform temporal analysis, where the system configuration changes, for instance, when resources are moved from one compartment to another.

Possum PIE currently permits the specification of security requirements, e.g., on reachability, information flow involving declassification and endorsement, and referring to standards such as the CIS Benchmarks.

Simple Example

In this document we present the high-level details of the analyses of a public example called the Supremo System[8] which has two types of applications (one is restricted while the other is open to the public), two databases (one which is restricted and linked to the restricted application while the second one is linked to the public application), network configurations including Zero Trust Packing Routing (ZPR) and a declassifier (which is usually part of the application that is aware of sensitive data) that permits certain flows from the secure database to the public database. For the sake of readability, we do not describe all the technical details here and only present the security analysis that we can conduct at a high-level. The summary of the analysis that Possum PIE performs is given below.

Type of Property	**Informal Description**	**Results of Analyses**
Networking	• Detect misconfigurations in network security groups (NSG), Security	• Flag security violations in the configurations and show that an

[8] https://blogs.oracle.com/cloud-infrastructure/post/first-principles-zero-trust-packet-routing

	Lists (SL) and ZPR enforcement.	attacker (a general public user) can write to the secure database • Make recommendations that show this can be fixed at various levels including changing NSG, SL or adding suitable ZPR tags.
Database Security	• Model IAM-based authentication in the DB	• Show that attacker cannot connect to secure DB.
IAM	• Each service has its own IAM policy.	• Report violations of the least privilege principle. • Check that various confidentiality requirements are satisfied by the declassifier.
Security Zones (SZ)	• These are policies that deny access for certain operations which are not supported in all versions of the IAM.	• Show how with suitable SZ policies, actions such as the creation of public buckets can be prevented, even if the IAM policy is overly permissive.
Information Flow	• Apart from IAM related declassification, allow analyst to specify tag-based information flow requirements.	• Show that IAM, SZ policies are not always adequate. One needs the declassifier to have tags (based on the intent of the user) to prevent information flow leakage.

Conclusion

To summarise, Possum PIE allows the security analyst to verify the full security posture of a particular deployment. The tool also allows the analyst to explore “what-if” analysis. This can be used to demonstrate the robustness of the system by showing the number of security-features need to be misconfigured for a breach to occur. For example, even if IAM is overly permissive, correct SZ, ZRP and information flow configurations can prevent attacks. We are actively working on the synthesis of correct policies so that one can claim that the system is correct-by-construction.