Technical Brief

Why Cloud Security Requires Strong Application Identity

Aporeto

Why Cloud Security Requires Strong Application Identity

Aporeto

IT evolves. The evolution wave is currently driven by a mass migration into the cloud infrastructure and the adoption of microservices architecture. Unless your company is a startup without any previous "IT debt," you are invariably integrating the cloud and microservice-based applications into your existing "brownfield" environment. The question should not be whether your security technologies and practices are evolving as fast because it should be clear that they are not.

Without the appropriate security measures, the very cloud and microservices innovations that you are adopting to propel your business forward create risks that could expose your most confidential data and bring the business down.

One may attempt to extend existing infrastructure security and segmentation techniques to the evolving cloud and microservices space, but this ernes attempt simply ignores the realities of the public cloud and the Zero Trust posture that would be wise to adopt. The ideal security posture is to protect workloads and their interactions with strong contextual identity that feeds an automatable, scalable policy engine.

Contextual Identity for Applications

Application and service deployment, whether microservices or older service-oriented architectures, are increasingly dependent on the concept of identity. Providing security for interconnected services requires a reliable and robust identity for applications to be able to authenticate and authorize themselves to each other. This interaction of identity and security can take several forms:

- **An application consuming a cloud service such as an S3 bucket or an API to scale up or down other applications.**
- **An application consuming an external business service (such as Salesforce).**
- **Different services of the same application interacting with each other, such as microservice interactions.**
- **Access to databases or other infrastructure components.**

The basis of all this interconnection model is a strong and secure identity. Nonetheless, there is a fundamental chicken-and-egg problem. Namely, applications need to retrieve dynamic trusted identities that are not checked into some static code repository for security reasons. Cloud service providers offer some mechanisms for this type of functionality by leveraging service account concepts. This approach, however, creates other system weakness when either external services or programmatic human interactions are required.

Strong Identity

In most cases, static API keys become the weakest security link in operations. Additionally, service account identity generally is useful within the same cloud environment. To put it simply, one cannot use an AWS service account to authenticate and authorize an access in a GCP tensorflow service without a significant configuration overhead.

To address this problem, we need a uniform and dynamic identity model that not only interfaces with existing identity components in the cloud, but also takes into account runtime information of services, and allows the translation of these identities to formats compatible with any third party service.

Two-factor authentication (2FA) is a well-known and recommended method for ensuring the right user identity when accessing a particular service. 2FA shrinks the threat surface because it minimizes vulnerabilities from stolen credentials. Newer systems, like iPhone X, use biometrics such as the user's voice, picture, or fingerprints to strengthen 2FA and strengthen the authentication process.

What is the analogous 2FA mechanism when machines interact with each other? Is there such a thing as an application's "fingerprint" or"biometric" information?

The Aporeto AppID model precisely answers this question because it creates a dynamic fingerprint that, in conjunction with static parameters like secrets, provides the equivalent of 2FA for applications. By understanding these dynamic fingerprints, not only do we get a stronger security model, but we also achieve a finer-granularity security model where application interactions can be managed based on environmental and real-time factors.

As an example, the Aporeto AppID model allows two applications to interact regardless of location: They can be in the same data center, or in separate clouds. Or, image vulnerabilities can be baked into the dynamic application identity to prevent specific actions. In essence, the Aporeto AppID model allows authorization based on run-time primitives. The corresponding effect in the human world is to grant or block user access based on his location or because of the profile of the device through which he is attempting access.

Creating Strong Identity

- **Creating dynamic identities that are trustworthy; and**
- **Using dynamic identity for direct authorization or exchanges this dynamic identity based on policy with a static identity for compatibility with other services.**

Metadata Extraction

The critical component for dynamic identities is a trusted local or remote component that can mine metadata for generating an application fingerprint. The Enforcer fills this role in the Aporeto system and runs on every workload server in the infrastructure. An Enforcer also needs its proper identity metadata because the Aporeto system must trust it (this topic is covered below).

The Enforcer uses a metadata extractor to collect an application's runtime information for generating its fingerprint. For example, in a Linux system, metadata will include user information, library dependencies of the application, network information, kernel version, checksum of the binary, vulnerabilities, and run-time or system call behavior. In a docker container, the metadata can include all docker manifest data, image information, vulnerabilities, runtime information from the docker daemon, and so on. These local data are discoverable by a user and become the foundation of identity.

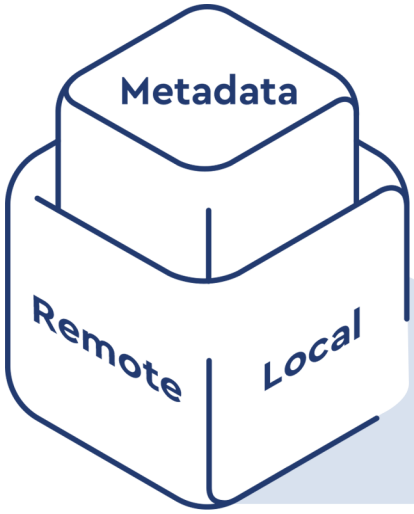
Other metadata sources also contribute to identity. For example, if a VM is running in AWS, we can extract relevant information, such as the VPC where the VM is running, the AMI, and tags through simple API calls. If a VM is running on ESX, we can extract vCenter information. If there is a code repository, we can correlate the running process with the corresponding code commits. In a CI/CD environment, we can associate the image checksum with the output of a third-party vulnerability scanner. The more metadata we can correlate, the more powerful the AppID system becomes, a viable goal in modern cloud environments because of the significant amount of data made available by the platforms.

Once metadata are extracted from a combination of static, user, and dynamic information sources, we can create a multi-attribute identity that is a cryptographically-signed JSON document. This dynamic identity can be now used to validate the application with other systems. It can be used directly for authorization by the Aporeto system or by transparently inserting authorization in the L3-L7 session handshakes. Moreover, this dynamic AppID can be used by the application for authorization with external services.

Metadata Extraction Implementation in the Aporeto System

- Local extraction is done by each Enforcer running on every server. Local extractors implement essential functions that are common to all users. An extensibility framework allows the creation of plugins to accumulate additional metadata through environment-specific functions. Metadata collection begins any time a process associated with an Aporeto-secured application starts and may continue over time to offer a more comprehensive profile.
- Remote and IT system metadata extraction is implemented through a hook function in the Aporeto policy system. After extracting local metadata, the Enforcer issues a request to the policy system. A plugin model in the policy system requests additional resources from any third-party component that is available in the infrastructure.
- Automated, periodic metadata extraction is enabled by a framework that allows users to write workflows in the policy system to decorate identity with additional metadata as new data becomes available. For example, because the environment in which an application is running can change, an automation workflow can continuously monitor an external database for threat vulnerabilities and update the application identity at runtime.

Bootstrapping Enforcers



In the above process, a key component is the bootstrapping of Enforcers so that they can be trusted by the Aporeto system and provide a credible communication path to the policy system. This process is done either through a one-time token or through method providing metadata and dynamic identity.

Bootstrapping with one-time-token: When the Enforcer is first installed on a server, the installation procedure can provide a token that can only be used once to register the server with the policy system and download a certificate. This certificate can be stored in memory or on disk and is periodically renewed. Obviously, in this case, the risk is that the certificate and key may be compromised.

Dynamic attestation: In several cloud environments, the Enforcer can automatically provide trusted metadata that can be used to authenticate the validity of the Enforcer. For example, in an AWS installation, every VM is provided with an identity document. This document is signed with the AWS private key, and it includes information such as the VPC and data center that the VM is running, the associated AWS account, and so on.


```
{
  "devpayProductCodes" : null,
  "marketplaceProductCodes" : [ "1abc2defghijklm3nopqrs4tu" ],
  "availabilityZone" : "us-west-2b",
  "privateIp" : "10.158.112.84",
  "version" : "2017-09-30",
  "instanceId" : "i-1234567890abcdef0",
  "billingProducts" : null,
  "instanceType" : "t2.micro",
  "accountId" : "123456789012",
  "imageId" : "ami-5fb8c835",
  "pendingTime" : "2016-11-19T16:32:11Z",
  "architecture" : "x86_64",
  "kernelId" : null,
  "ramdiskId" : null,
  "region" : "us-west-2"
}
```


The Enforcer retrieves this information and sends it to the Aporeto orchestrator. The Aporeto system validates this document by using the AWS public certificate and then correlates it with the AWS information. Azure and GCP have similar identity documents.

In more static environments like a VMware vSphere, the Aporeto system can be extended through plugins to validate new Enforcers. In these scenarios, the Enforcer collects information such as the VM MAC address, VM UUID from SMBIOS, processor information, and so on. The Enforcer then issues an authentication request with the orchestrator. At this point, the orchestrator correlates the data with the corresponding vCenter installation and uniquely identifies the VM in question.

Securing Your Workloads with Aporeto

Aporeto secures your applications whether they are built with microservices or have older service-oriented architectures. Aporeto's Zero Trust security for microservices, containers and the cloud work through the transparent generation of strong application identity that is used for authenticating and authorizing application component interaction with themselves, user interfaces, and external services. Aporeto's approach, analogous to two-factor authentication for user interfaces, allows enterprises to securely continue their journey to the cloud and adopt more agile microservices while leveraging their existing, brownfield infrastructure without compromising them.