CSAP

COMMON SECURITY ARCHITECTURE
for PRODUCTION

VERSION 1.3

PART 5B:
IMPLEMENTATION CONSIDERATIONS – CSAP CORE

Motion Picture Laboratories, Inc. August 1, 2023

Contents

1 Introduction

CSAP v1.3 is presented in six parts:

Part 1: Architecture Description the main architecture document.

Part 2: Interfaces describes the possible interfaces between the modules in a canonical form.

Part 3: Security Levels presents a metric-based approach to scaling security.

Part 4: Securing Software-Defined Workflow discusses how the security architecture can be applied to software-defined workflows that are managed using a service bus.

Part 5: Implementation Considerations is broken into sub-documents (5A, 5B, and 5C), which cover different aspects of CSAP implementation.

Part 6: Policy Description describes how policies and rules are defined. This part has not been published as of July 2023.

It is assumed that the reader is familiar with the previously published parts of CSAP, 1 to 5, and we do not reiterate the concepts described in those parts.

Changes from CSAP Part 1 v1.1

- The name of the *authorization policies* has been changed to *authorization rules*.
- The functions of the policy manager moved into the authorization service, the policy service in v1.0 is now the Authorization Rules Distribution Service (ARDS). In v1.0 this was called the policy engine. This does not change the functions necessary to create an authorization rule (formerly authorization policy), but consolidation simplifies this part of the architecture.
- Security initialization has been added.

Changes from CSAP Part 1 v1.2

- The functions of the Asset Protection Service have been merged into the authorization service. There is no change in function.
- The diagrammatic representation is now three services (authorization, authentication and the ARDS) as the part of the CSAP infrastructure.
- The CSAP supporting security functions Trust Inference and Continuous Trust Validation have been merged to reflect the direction of the market place.

1.1 How to Use Part 5

In creating the CSAP architecture, the designers wanted the result to be implementable using existing technologies and with the least development possible. Part 5 provides a perspective on the way the CSAP designers ensured those goals were met. However, the descriptions of implementation approaches may not represent the optimal approach and are not detailed enough to serve as implementation guides.

Part 5 is broken into sub-parts, each of which covers a particular aspect of implementation. Part 5 consists of three parts that cover some aspects of implementing CSAP, by no means all of them. We expect to release additional parts as developers gain experience in implementing CSAP.

- In Part 5A (this document) subtitled "Starting Out," Section 2 "CSAP Recap" and Section **Error! Reference source not found.** "Implementation Basics" provide general guidance and sets the stage. Section 4 introduces the CSAP Zero-trust Foundation. Section 5 provides more detail on the CSAP Zero trust Foundation (ZTF) and section 6 discusses going from CSAP ZTF to CSAP level 100 and beyond.
- In Part 5B (this document) subtitled "CSAP Core," Section 2 "Identity and the Authentication Service" and Section 3 "Authorization and Authorization Rule Distribution Services" discuss implementation considerations for CSAP core security components. Section 4 "The User Experience" is a lesson in a way to create a good user experience.
- In Part 5C: subtitled "Approaches," Section 2 "The Network" covers ways in which networks may be used to support CSAP functions. Section 3 "Access Controls" discusses ways access to assets and resources can be controlled. Section 4 "End to End Security" looks at ways that the CSAP architecture can be used to facilitate end-to-end security on untrusted infrastructure.

1.2 Terms

Authentication is the security mechanism used to validate an entity's identity by a trusted authority. The entity might be a user, a service, a device, an application, etc.

Authorization is the security mechanism used by a trusted authority to determine whether an entity can perform an action.

An *Asset* is the broad term we use to mean any data and metadata that is part of the process of media creation including image data, sound data, and metadata. *This is the media definition of the word "asset," and not the definition used in cybersecurity where the word asset means any data, device, or other component (hardware or software) that supports information-related activities.*

The use of *Context* is the normal definition of that word, the setting, circumstances, or environment of an event. The MovieLabs Ontology for Media Creation has a specific and different meaning for *context,* which is not used in this document.

Policies are the abstract representation of what is to be authorized.[1]

Rules are the actionable representation of a policy.

A *Device* is a piece of infrastructure in the form of a physical or virtual system that serves as a platform for the execution of software.

[1] Or, in the very specific case of Global Security Policies, what is to be denied. Global Security Policies are the only place where a "deny" construct is needed since everywhere else, CSAP is deny by default.

Mutual authentication occurs when each entity that is part of forming a trust relationship can authenticate all the others. (As will be discussed later, in this context a user and their system may each be an entity.)

mTLS (mutual TLS) is a form of TLS with additional steps that authenticate the client to the server. In TLS, the server is authenticated to the client, but out-of-band authentication is required to authenticate the client to the server, e.g., using managed device services.

We use the terms for the structure of the organizations creating a creative work that are the common parlance of Hollywood, but they map directly to the terms used elsewhere.

The Studio is the entity that owns the rights to the creative work, is responsible for funding production and has a say (usually creative) in the production process. This is the same role as a commissioning broadcaster or a network (in the way that the term is used when referring to US linear broadcasters.) View the word "Studio" as a shorthand construct for anything that fits the definition.

Depending on the context, the term *The Production* is used to mean either:

- The entity responsible for carrying out production. This may be a production company, an organization set up to produce one creative work or a department or business unit that is part of the studio.

Or

- The complete process of producing the creative work.

Vendors are companies that provide services to the production. They may also be called production service providers. Examples are a VFX company, a transportation company, and a cloud infrastructure provider.

Please do not assume that our use of these terms means that CSAP is only for Hollywood studios or only for motion picture production. CSAP is for all types of media production including scripted and unscripted television.

1.3 Visual Language Security Icons

The shapes and icons used in the diagrams in this document are part of the MovieLabs Visual Language. Rather than add a key for the security icons to each diagram, we include it here.

Further information on the MovieLabs Visual Language can be found here at www.movielabs.com/production-technology/visual-language-for-media-creation/

1.4 Choice of Examples

In this document we present examples using commercial products and services. The choice of technology vendor, for example a cloud provider, is in no way an endorsement and does not imply that one product is better than another. In fact, where possible, we have steered away from examples where there is only one vendor.

We also discuss Google's BeyondCorp, and when we do, unless we say otherwise, we are talking about the zero-trust security solution Google implemented in their own offices and about which they have published a range of papers. These papers are cited frequently in the literature. Consider this to be an academic reference.

1.5 References

1.5.1 MovieLabs Publications

The Evolution of Production Workflows, MovieLabs, 2020

The Common Security Architecture for Production, Parts 1 to 4, MovieLabs.

1.5.2 Publications from Government Organizations

Zero Trust Architecture, NIST Special Publication 800-207, https://doi.org/10.6028/NIST.SP.800-207

2 Identity and the Authentication Service

Identity management is a level 100 requirement.

Robust identity management is not new and there are compelling reasons to have implemented it long before CSAP came along. There are many security solutions from the cloud providers and from cloud-agnostic technology providers that provide identity management that can be integrated with most systems and many SaaS offerings.

In CSAP, identity management goes beyond people and includes organizations, devices, and applications and services. We will look at how that is achieved for things other than users later in this document.

Identity management isn't the sole requirement for trust: it attests that whatever has been authenticated is the trusted entity it claims to be.

When a server presents a certificate and it is checked for validity and currency, we can assume the server is what it claims to be, but security requires that measures must have been taken to assure that the server has not been tampered with. In other words that the trusted entity is trustworthy.

Fortunately, that isn't a new requirement that comes with CSAP or zero-trust in general. No server or, to generalize, no device should be left vulnerable to tampering even if it is within a security perimeter that is trusted beyond a level that is reasonable given the history of breaches of security perimeters.

2.1 Trustworthiness and Authentication

In our context, the model of trust has two components: trustworthiness and authentication.

Trustworthiness is the outcome of a process by which someone decides they trust something and are prepared to call it trusted. In our example above, the basis of my decision to trust the cardiac surgeon might include:

- Trust in state regulations that regulate cardiac surgeons.
- The reputation of the medical school where the surgeon trained.
- The reputation of the hospital and how long the surgeon has worked there.

In fact, what I have done is a risk assessment. My assessment means that the risk is within a range that I am comfortable to live with.

More generally, the decision as to whether something is trustworthy is entirely up to whoever is setting security requirements. It is outside of the scope of CSAP.

Authentication, however, is in the scope of CSAP and is foundational. It is the process that establishes through some method of identity management whether something is the trusted entity it claims to be.

To conclude the example, the last step is signing the consent form authorizing the surgery.

2.2 Authentication Service

The authentication service is a level 100 requirement.

The authentication service, which could be manifest in more than one system, is responsible for the authentication of four types of entities:

- Participants
- Infrastructure (e.g., devices)
- Applications
- Services

The authentication service makes use of services in the supporting security components.

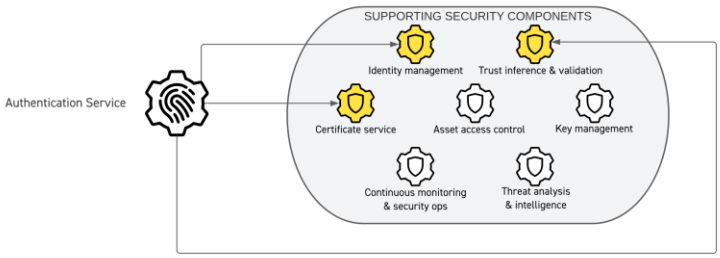

Figure 2-1 The supporting security components used by the authentication service

One of the CSAP authentication service's roles is to act as a front-end to identity management services, and to keep track of what has been authenticated along with identity related parameters such as expiration time of authentication tokens. (We use the phrase "authentication token" or simply "token" to generally denote the data that a participant or a device presents to prove it has been authenticated.)

Of course, how the authentication service is implemented depends on the nature of the supporting security components that are used, and it might be implemented entirely using supporting security components.

2.3 Participant Authentication

Participant authentication is a level 100 requirement.

In the simplest case, a single sign-on system (SSO) is used to authenticate participants as the sign-in to access any service or asset for the production. If the SSO system manages identity across multiple organizations or productions, the authentication service must only successfully authenticate those participants who are provisioned in that service, e.g., in a SaaS tenant for the production or studio.[2]

Trust inference is a level 300 requirement

[2] A participant requires both authentication and authorization to participate in a workflow, so a participant could be authenticated but not authorized to access any resources for the production. However, that does not mean a participant should be authenticated for services which their production(s) are not using.

The requirement is that the context of sign-on requests is taken into consideration and based on the context, the authentication requirements can be changed including, for example, denying any sign-on attempt from a blacklisted location.

If participant authentication is done using an identity management service that does not support trust inference, then there is likely a complex task to add trust inference with its necessary analysis of user traffic. However, trust inference under various names, is a function of many identity management solutions.

Solutions such as Okta's Adaptive Multifactor Authentication together with their ThreatInsight provide the necessary trust inference functions. ThreatInsight evaluates a sign-in request before the user authentication process using rules and data collected by ThreatInsight and the sign-in request may be denied before the credentials presented are reviewed. The Okta Adaptive Multifactor Authentication then modulates the authentication step based on the context of the sign-in request.

With many identity management solutions there may be little or nothing else for the authentication service to do for participant authentication.

2.4 Certificates

Certificate management is a level 100 requirement.

A digital certificate, also known as a public key certificate, certifies the ownership of a public key by the identity of the subject of the certificate.[3] A PKI transaction with an entity claiming to be the certificate owner can determine whether they are the certificate owner.

Certificates carry other information including the certificate authority (CA),[4] the authority that issued the certificate. The CA distributes lists of certificates that have been revoked.

Certificates must be verified, typically that is done through the certificate's CA. It is not enough to know that the entity that presented the certificate is the owner of the certificate, it is critical to validate that certificate as being current.

Certificates are used extensively in CSAP; they are one of the core methods of authentication in device and service authentication.

Certificates can be created and managed by either:[5]

- A public certificate authority, generally trusted by major browsers and operating systems.
- A private certificate authority, a CA that is trusted by the enterprise.

There may be operational reasons outside of the scope of this document to favor one type of CA over another, such as cost and response time. Generally, we believe that implementing authentication using

[3] https://en.wikipedia.org/wiki/Public_key_certificate

[4] https://en.wikipedia.org/wiki/Certificate_authority

[5] It is not a requirement that a certificate authority is used however the alternative is typically self-signed certificates which may increase the risk of impersonation.

self-signed certificates is complex, but CSAP does not place a constraint on how certificates are managed.

Examples of private certificate authorities include:

- Google Cloud Platform Certificate Authority Service https://cloud.google.com/certificate-authority-service/docs
- The AWS Certificate Manager Private Certificate Authority https://aws.amazon.com/certificate-manager/private-certificate-authority/

2.5 Device Authentication

Device authentication is a level 100 requirement.

A device is a piece of infrastructure. It might be an execution platform (e.g., a workstation, a server or virtual machine), or it might be device such as a camera. In this document we will discuss the former where a device is a place where software executes and has a common operating system.

For the purposes of the discussion in this section, our "device" is a piece of compute[6] infrastructure meaning it is an execution platform that might be either physical or virtual, and that supports the execution of software whether an application or a service. Since we need that software to execute securely, there are two options.

1. The device is trustworthy
2. The application or service can run securely on untrusted infrastructure

The second option is very difficult to achieve if nothing about the compute device is trusted. DRMs for consumer media distribution are designed to run on devices that are under the control of the attacker, but even there, certain components such as the video path from content decryption to video output must be secured.

The first option requires two things: some criteria by which a compute device is declared trustworthy, and a way of authenticating that a device is the trusted device it claims to be.

The device's security posture is the first of three aspects to authenticating a device. Tools such as Microsoft Defender for Cloud and Microsoft Defender to Endpoint provide security posture management as well as threat protection and can be used to determine whether a device can be trusted according to an organization's rubric for trustworthiness.

The second aspect is the identity of the instance of the device. In Azure, for example, device identity is an object in Azure Active Directory (Azure AD.) This device object is like users and groups and gives administrators information they can use when making access or configuration decisions. The identity is acquired either through Azure AD registration used by a BYOD device or a mobile device, or through Azure AD join which is for devices owned by the organization.

[6] Note that cloud storage requires the execution of software in order to function.

Whether each instance needs its own unique identity is determined by several factors and an implementation should include a risk assessment if unique identities are not used. In some cases, the absence of individual device identity may make the system more vulnerable to class attacks.

A third aspect of device authentication is the context of the authentication request, this is trust inference.

Implicit in device authentication, is the need for device identity.

2.6 Application Authentication

Application authentication is a level 100 requirement.

Our definition of application authentication is that:

1. The application came from the developer it was supposed to come from.
2. The application has not been modified.

The goal of application authentication is two-fold. One is to ensure that there has been no malicious tampering with the application after it was deemed to be trusted, for example it does not contain malware. The second goal is protecting the integrity of a workflow by ensuring the correct version of an application is used.

A code signature can be used to achieve either goal. A code signature is a cryptographic hash[7] of the code that confirms the author and guarantees the code has not been altered or corrupted.[8] Checking the signature requires the author's public key which is contained in a certificate that can be (and should be) validated with a certificate authority.[9] Do you trust the author? Code signing does not offer any protection against malware introduced before the code was signed including any malware introduced in open-source software.

- On Windows and Mac OS X, the code signature is used to authenticate the software on first run which means that other security measures (e.g., endpoint security) must be relied on to maintain the application in the state it was in when it was authenticated.
- On Linux, the signature (or a hash) is typically checked by the package manager when the package is installed, by itself this offers less protection.

[7] A cryptographic hash function is a mathematical algorithm that maps data of an arbitrary size (often called the "message") to a bit array of a fixed size (the "hash".) It is a one-way function, that is, a function for which it is practically infeasible to invert or reverse the computation. The signer's private key is used to generate the hash and it can be verified using the signer's public key.

[8] For a more detailed explanation see the article *Introduction to Code Signing* at https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms537361(v=vs.85)

[9] See https://www.ssl.com/faqs/what-is-a-certificate-authority/ for an explanation.

For example, the AWS Signer is a code-signing service that can be used to sign Lambda functions. A code signing configuration is created and attached it to the Lambda function. It defines a list of allowed signing profiles and the policy action to take if any of the validation checks fail.

When the code package is deployed, Lambda validates the code package before accepting the deployment.

Lambda layers follow the same signed code package format as function code packages. Add a layer to a function that has code signing enabled, and Lambda checks that the layer is signed by an allowed signing profile. When code signing is enabled for a function, all layers that are added to the function must also be signed by one of the allowed signing profiles.

A more resilient approach to application authentication is to check the signature when the application is run or, at a higher level of security, while the application is running. These two checks are for protection against modification of the application after it is deployed or first run.

2.6.1 Trusting Applications

As is the case in other parts of CSAP, the role of authentication is to determine if the entity is the trusted entity it is claimed to be. Whether that is the case is not a CSAP issue. However, we note that determining whether to trust applications, whether acquired or developed in house, is a thorny matter particularly when code is used, as is the case with open-source code, that was developed elsewhere.

In that case, we want to know:

1. Did the code developer understand enough about securing coding to make the application behave in a secure manner?
2. Do any open-source libraries that are used contain malware?[10]

That list is by no means complete and equally applies to libraries that are not open-source code.

Since this document does not present best practices, we do not discuss potential solutions to those issues.

2.6.2 Challenges of Plug-Ins

The nature of the applications used in production, and the way they are used, can raise some challenges when authenticating applications and there is not a consensus approach to the issue in general

Authenticating plug-ins is a challenge. Code signatures do not help unless they are checked at some point and signatures of plug-ins are not verified by the operating system since they are not installed in the same way as applications. Ideally, they would be verified by the application but whether that is practical is going to be specific to how plug-ins are handled.

There are solutions available which offer a degree of protection. For example, Maya's Secure Plugin Loading will prevent plugins in non-trusted locations (i.e., those outside Maya's default trusted locations as well as those specified by the user) from loading without the user's knowledge: either by asking the

[10] https://hackaday.com/2018/10/31/when-good-software-goes-bad-malware-in-open-source/

user for permission to continue or denying the plugin from loading entirely. However, that does depend on the discipline of the user and/or protection of the trusted location and control over the "Secure Plugin Loading" setting.

If there are appropriate polices as to how a plug-in gets into a trusted location, which goes hand in hand with a policy to define a trusted location, this may be sufficient to authenticate the plug-in for the purpose of the ensuring it comes from a trusted source, but it does not authenticate, for example, the plug-in version nor that one plug-in is used and not another.

2.6.3 Internally Developed Code

Authenticating Internally developed code could use the same code signature method as described earlier in this section. The signing step can be part of the final step before the code is distributed. The reader is cautioned that the code signature verifies that the application is the binary that the development process created and protects against alterations since it left the development process. It does not guarantee that the code does not include malware introduced during the development process.

2.6.4 Interpreted Code

Several languages, such as Python and shell scripts, typically run interpretively meaning that each line of code is converted to machine language as it is executed. By comparison, compiled code is converted to machine language as the final step in application development. There are few tools around to sign, for example, Python code that is not compiled. While manual methods exist such as signing the .py file, checking the signature would also need to be manual.

Without some means for authenticating that the code, there is no way to determine if the code has been altered.

2.7 Service authentication

Service authentication is a level 100 requirement.

This section applies to a service that is not a SaaS service operated by operated by a third party that fully controls its security.

Earlier we said that a trusted entity must be trustworthy, which means that we could say a platform running software is trustworthy if (a) the platform is secured and free from malware and (b) the software has not been modified. We now have a chain of trust: we trust the platform because its security measures make it trustworthy, and we use the platform to authenticate the software before it is run and, perhaps, while it is running.

How a service is authenticated depends on the relationship between the execution platform and the service software.

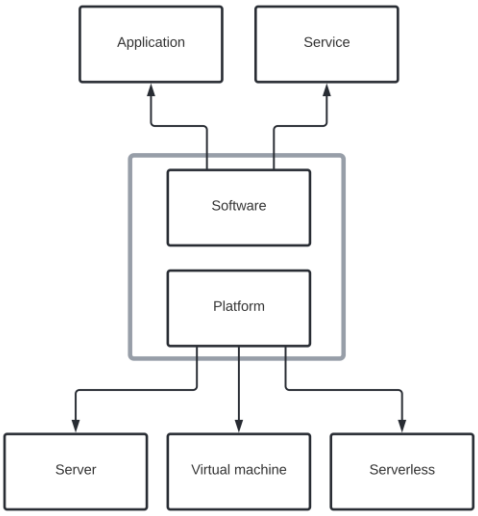

Figure 2-2 Service

How we regard the device and software combination are illustrated in these examples:

Example	**Authentication**
A virtual machine that is an immutable combination of an execution environment (an OS) and one or more software applications/services. It is fully configured prior to instantiation.	The same certificate-based authentication as a device.
A server that is an operating system running on either hardware or virtualized hardware and has the characteristic that software, subject to administrative rights, can be installed on it while it is running.	Certificate-based authentication of the device followed by authentication of the software.[11]

[11] The device should be authenticated first, if the device cannot be trusted then the process of authenticating the software may not be trustworthy.

3 Authorization and Authorization Rule Distribution Services

CSAP is workflow driven security which means that the core of its operation is turning the requests from workflow management to authorize an activity, expressed as authorization policy requests, into authorization rules that do so.

Workflow evolution is a driver for closely coupling authorization with workflow management, something that will be aided by clear communications with all collaborators.

3.1 CSAP Authorization Rules

Long lifetime authorization rules are a level 100 requirement
Medium lifetime authorization rules are a level 200 requirement
Short lifetime authorization rules are a level 300 requirement

For the purposes of defining the security levels, authorization rules are categorized as follows:

Authorization rule Category	Examples
Long lifetime	Authorization rules are created for each workflow and the lifetime is the duration of the production or a phase (e.g., post-production) of the production
Medium lifetime	Authorization rules are created for each workflow and the lifetime is approximately the duration of the workflow
Short lifetime	Authorization rules are created for each task and the lifetime is approximately the duration of the task

CSAP authorization rules are designed to map well into the whole production; for each workflow that is part of the production; and for each task that is part of a workflow.

There are no functional differences between the authorization rules in each category, the difference is in the lifetime and, therefore, the number of authorization rules generated.

An authorization rule describes what is authorized in terms of:

- Participant (user, group, etc.)
- Device (physical or virtual)
- Action (the task to be done)
- Application (software with a user interface or with an API)
- Time frame (start time, start event, end time, end event)
- Asset (with CRUD-like permissions)

An authorization rule is made up some or all these components as is necessary in a context.

An authorization rule should have some way of determining its provenance, meaning the authorization service that created it. This assists the ARDS determining where a rule came from.

3.2 Authorization service

Authorization service is a level 100 requirement.

The authorization rule lifecycle is shown in this diagram.

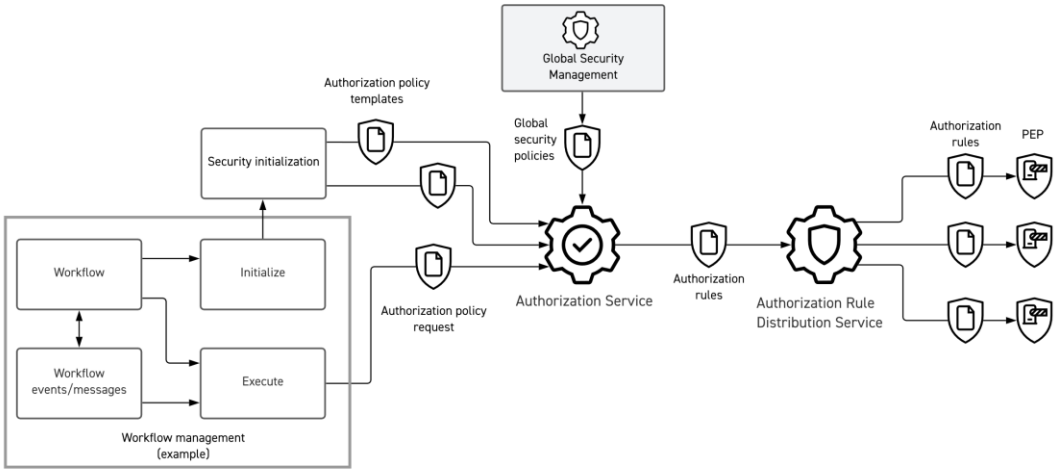

Figure 3-1 Lifecycle of a CSAP authorization rule

An authorization rule starts life at the interface between the workflow management function and the CSAP authorization service when workflow management creates an authorization policy request.

Authorization policy templates are created by security initialization when workflows are set up or changed. The authorization service selects an authorization policy template appropriate for the authorization policy request from workflow management. While it is not a requirement, the CSAP designers anticipate that authorization policy templates will be specific to how or where the resulting authorization rule will be acted upon.

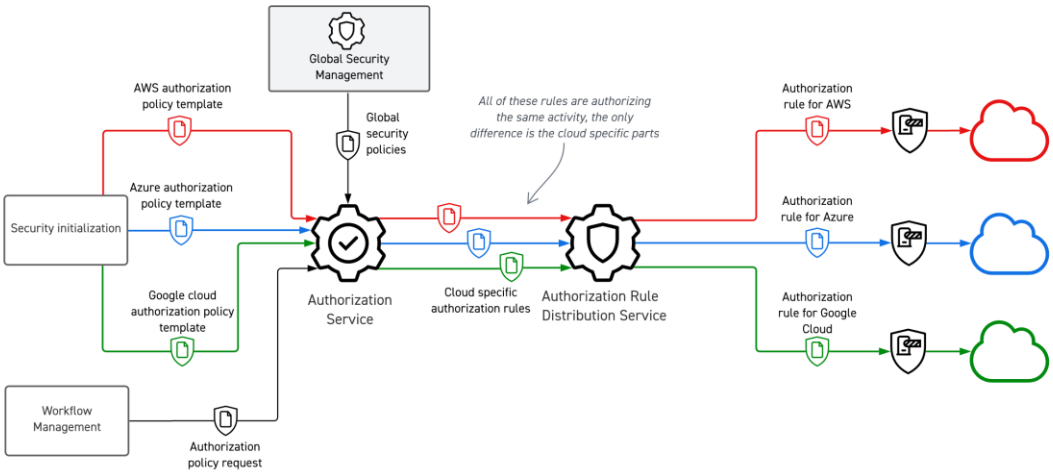

Figure 3-2 Example of authorization rules specific to the application

The authorization rule describes what is authorized and it is used in a system where nothing is authorized by default. This is different from, for example, a policy in a firewall which may define either what is authorized or what is to be blocked.

3.2.1 Global Policies

Processing global policies is a level 100 requirement.

The authorization service must include global security policies where they are used when authorization rules are created. Global policies are best thought of as a filter that defines what cannot be authorized, potentially invalidating the intended authorization rule.

Global security policies are policies that may apply to all workflows, and, unlike authorization rules, they are rules that deny an activity based on one or more components of that activity. Examples include:

- Forbidding the use of any VFX vendor for a particular scene in the creative work.
- Preventing the use of a system because of a security breach.

If an authorization rule is created that is not allowed by global policies, the authorization service would be expected to send an appropriate error message back to the workflow management along with any information that says why the policy request could not be fulfilled.

Global policies are the only part of CSAP that uses a “deny” construct, but they are only used between global security management and the authorization service. This means that global security management takes precedence over workflow management requests and the authorization service should be implemented accordingly.

3.2.2 Authorization Rule Revocation

When implementing CSAP, consideration must be given to a mechanism to revoke authorization. The CSAP authorization mechanism includes provision for authorization rules to include a start event, the point at which they go into effect, and an end event when they expire. However, changing circumstances may mean that an authorization rule is no longer valid. For example, an artist is reassigned[12] to another task and any authorization rules authorizing their old task must be revoked.

To revoke authorization is to revoke the authorization policy request that created the authorization rules necessary to fulfill that request.

Revocation requires a breadcrumb trail from the authorization service that created the authorization rule to the PEP enforcing it. This should not be particularly difficult; authorization services keep track of which ARDS authorization rules are sent to, and the ARDS keeps track of which PEP the authorization rule was sent to.

It does mean a unique system-wide identifier is associated with each authorization rule and, in order for workflow management to rescind an authorization policy request, workflow management must be able to identify the authorization rules created in response to each request.

3.3 Authorization Rule Distribution Service (ARDS)

ARDS is a level 100 requirement.

The ARDS is responsible for delivering authorization rules to the policy enforcement points and keeping track of which PEP has each authorization rule (see section 3.2.2.)

Sending every authorization rule to every PEP is not optimal because each PEP would accumulate a lot of rules that are not relevant and that it cannot act on, but there is nothing in CSAP to forbid that implementation.

3.4 Integrated Implementation

The distinction between the authorization service and the ARDS is architectural and there is no requirement that they are implemented as distinct services. In many cases, implementation will be simplified by combining them. The only constraint CSAP imposes is that each PEP is only connected to a single ARDS (see section 3.5 Distributed Implementation.)

[12] We use reassignment in this example because if the artist left the production, presumably, authentication would fail.

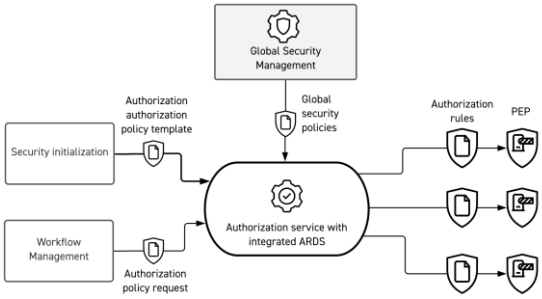

Figure 3-3 Authorization service with integrated ARDS

Similarly, the policy enforcement point is an architectural construct and can be part of whatever it is serving. The only requirement is that it uses authorization rules but those will typically be tailored accordingly in any event.

Furthermore, the ARDS and the policy enforcement point(s), or the authorization service, ARDS and PEPs can be implemented as one thing or can be entirely delegated to the infrastructure. If a cloud provider's services provide the functions needed for the authorization service, the input will be the authorization policy templates, the authorization policy requests and global security policies.

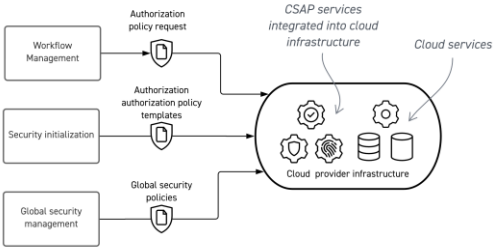

Figure 3-4 CSAP components integrated in cloud infrastructure

3.5 Distributed Implementation

CSAP is a distributed architecture. One implementation choice would be to instantiate an authorization service for each point of workflow management in the production. Similarly, it may make sense to instantiate an ARDS for each part of the infrastructure, for example for the infrastructure that uses one cloud provider.

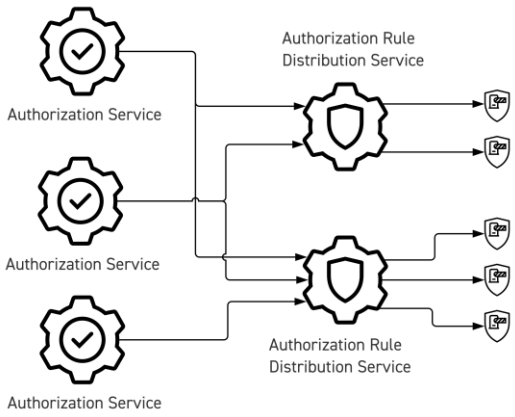

Figure 3-5 Relationships in a distribute implementation

As the diagram above shows, the relationship between authorization services and ARDS is many to many. CSAP requires that each policy enforcement point is connected to only one ARDS.

The diagram below is an example of how this might be implemented when a production's workflows use three cloud providers' infrastructure. In this case there is a single authorization service and an ARDS for each cloud provider. Note that the authorization policy templates and the resultant authorization rules are specific to each cloud provider even if when the same activity is being authorized on each.

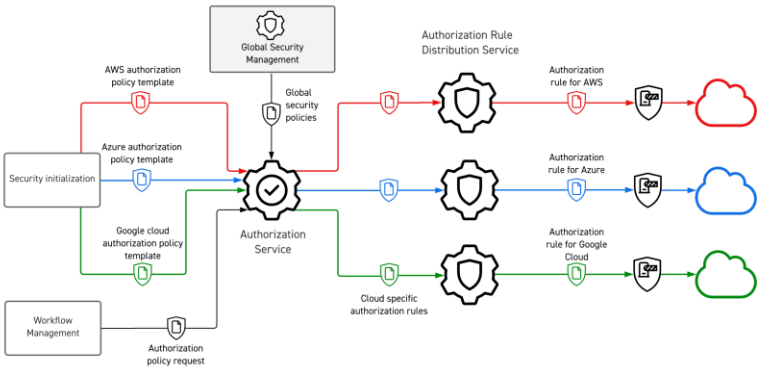

Figure 3-6 Infrastructure specific ARDS implementation.

3.6 Policy Enforcement Points

Policy enforcement points are a level 100 requirement.

A concept of CSAP, and of other zero-trust architectures, is the policy enforcement point. Authentication, authorization and authorization rules are the control plane of CSAP, and policy enforcement points are the data plane.

Policy enforcement points (PEPs) are the components where authorization rules are turned into action. They are deployed at the points in the infrastructure where authorization rules will authorize activity, for example where something is used.

There will be many different types of policy enforcement points in CSAP, and they act in different ways. Some might be acting directly, for example a PEP that controls access to an encrypted asset by decrypting the asset for authorized activity is engaged in every access to the asset. Others might be acting indirectly, for example a PEP that controls access to assets protected by an ACL might use an API to set security controls native to the protected resource and is not engaged in individual accesses.

It is hoped that most policy enforcement points can be lightweight

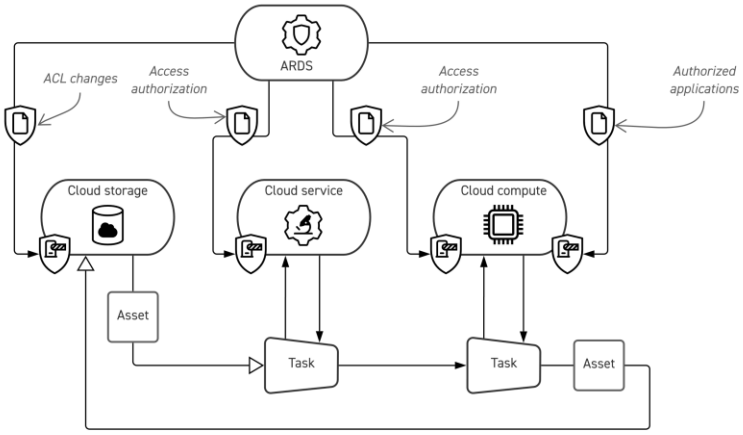

Figure 3-7 Policy enforcement points in an example infrastructure

There are four PEPs in the figure above illustrating different types of PEP and each receives authorization rules from the ARDS. From left to right these PEPs are:

1. Setting the ACL (access control lists) for the cloud storage
2. Controlling access to the service
3. Controlling access to the cloud compute
4. Controlling the applications that can be run on the cloud compute

This next diagram shows a PEP associated with a microservice.

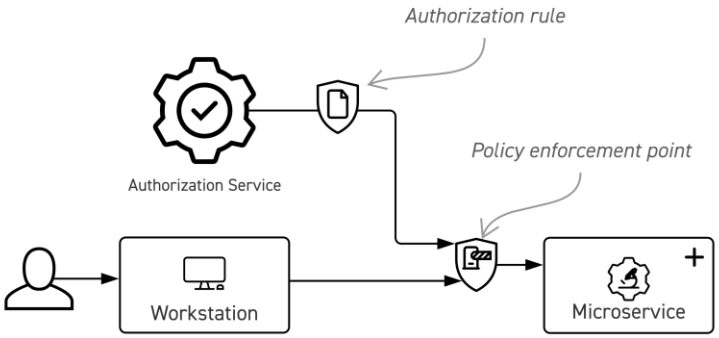

Figure 3-8 Example of policy enforcement point

In this case, the PEP is a proxy between the microservice and the rest of the world, and the PEP allows the microservice to run unmodified. Equally, provided the service can be trusted, the PEP could be integrated into the service.

The PEP ensures that connections to the microservice are from authenticated entities and are authorized by an authorization rule, it also authenticates the microservice to the entity that is connecting. In CSAP, authentication should always be mutual.

4 The User Experience

Uninformative error messages (or the lack of error messages) when something doesn't work frustrate users. Determining what went wrong and how to fix it can take even seasoned computer professionals (such as the authors) an unreasonable amount of time. Resorting to an Internet search, or even a search of the vendor's website, will often provide many unofficial answers some of which work, some of which don't and some of which are dangerous.

The failure to perform a task requires that users are presented with the information necessary to remediate the failure if it should not have happened.

Escobedo, Żyźniewski, Saltonstall[13] described the Explanation Engine in Google's BeyondCorp.

Beyond querying and surfacing ACLs, the portal also needs to present this information to users in a useful way. We built an explanation engine to provide troubleshooting details in response to parameters of deny requests. It operates by recursively traversing a tree of subsystems that provide authorization decisions. For example, the Access Proxy ACL might require a device to be fully trusted in order to access a particular URL. Upon retrieving this ACL, the engine contacts our device inference pipeline to retrieve the conditions necessary to access the corporate resource. We then propagate this information to our front end and translate it into plain language, so the user can visit the portal to find out what's wrong with their current state and how to fix the problem.

The explanation engine can supply users with helpful information, but caution is necessary in determining the detail presented. The data collected as to why access failed may be sensitive. Escobedo, Żyźniewski, Saltonstall point out that where the data reveals problematic ACLs of protected systems and discloses information about the state of the user's account and the device, that is not information that should be presented to an attacker.

Thus, if necessary, BeyondCorp can replace the data with a less sensitive variant.

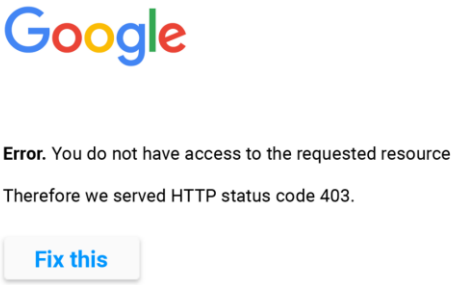

Figure 4-1 Less sensitive message to user (image: Google)

The button routes the user to the portal, also described by Escobedo, Żyźniewski, Saltonstall:

[13] BeyondCorp: the user experience, VM Escobedo, F Zyzniewski, M Saltonstall - 2017 - research.google, https://scholar.google.com/scholar?lr&ie=UTF-8&oe=UTF-8&q=BeyondCorp%3A+The+User+Experience+Escobedo+Zyzniewski+Beyer+Saltonstall

Explaining Complicated Failures: The Portal

For simple cases, like those described above, we could empower users to self-remediate using quick customizations to our error pages or the Chrome extension. However, in cases of legitimate denials of access, we knew that users and support teams would want or need to know why they were denied. The complex, multi-layered ACL logic in our back-end infrastructure can make understanding the logic behind a specific decision difficult for users and support teams alike. It might take even a seasoned SRE multiple minutes of querying many internal services to identify the cause of a single 403 error page. Given the volume of 403 error pages served by our Access Proxy daily (~12M for HTTP/S alone), human involvement in troubleshooting is unscalable and impractical.

The portal determines from the backend system the reason for the error, and, in the case of the error message in Figure 4-1, presents the user with a message showing what the problem was (to the extent that the data is not sensitive) and how they can fix it (Figure 4-2.)

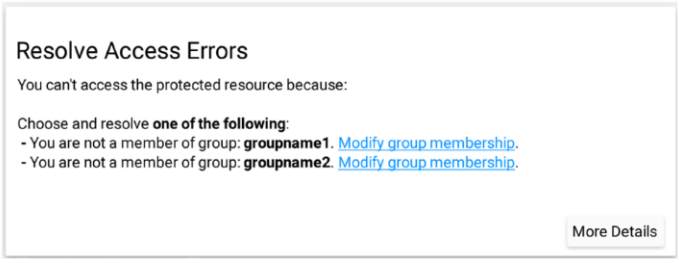

Figure 4-2 Employee-facing guidance on troubleshooting an access denied error (Source: Google)

Service desk-facing guidance includes more detailed information.

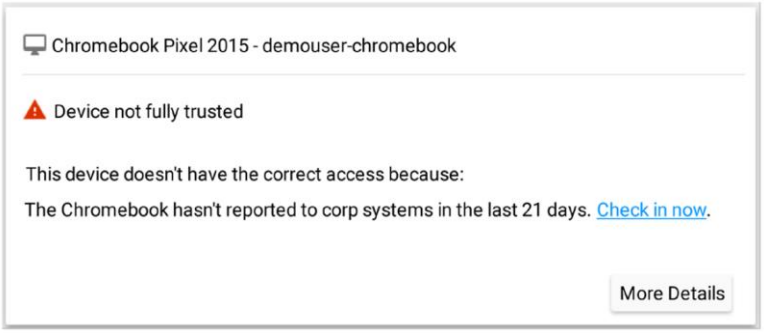

Figure 4-3 Service-desk-facing guidance on troubleshooting an access denied error (Source: Google)

The outcome contributes to the CSAP goal that security should not inhibit creative work. When security prevents someone from doing something, presentation of the error in a way that helps them find a resolution (assuming of course that the activity is authorized) will significantly contribute to meeting that goal.

5 Conclusion

CSAP is designed to use existing security technologies as much as possible although configuration for CSAP will be required. However, implementors should not be discouraged if they cannot immediately implement level 300 across their systems. CSAP is forward-looking. The attention of cybersecurity methodology and services has shifted substantially to zero-trust since the original MovieLabs' production security white paper was published at the end of 2019. In May 2021, US President Biden issued a directive[14] initiating a sweeping Government-wide effort to migrate the Federal Government to a zero trust architecture, and to realize the security benefits of cloud-based infrastructure while mitigating associated risks.

[14] https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity