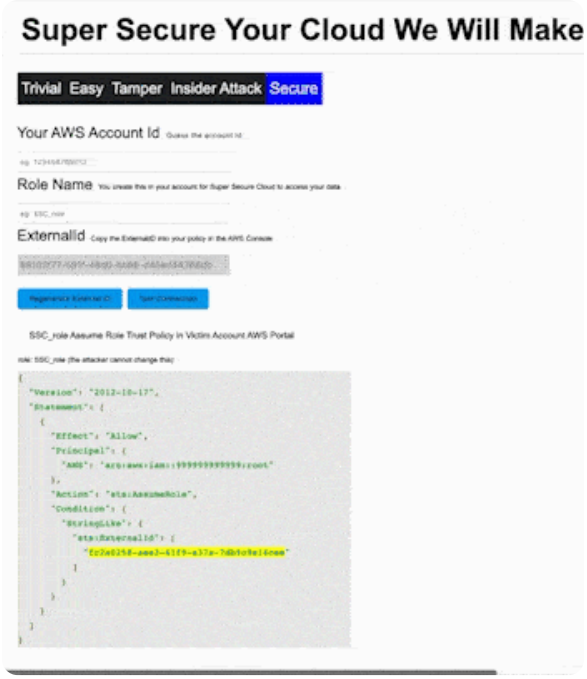
CLOUD IN SECURITY INSIGHTS

AWS IAM Assume Role Vulnerabilities Found in Many Top Vendors

by Kesten Broughton on June 12, 2020



In this first blog in our series on cross-account-trust we will present the results from 90 vendors showing that 37% had not implemented the ExternalId correctly to protect against confused-deputy attacks. A further 15% of vendors implemented the AWS account integration in the UI correctly, but the ExternalId parameter was not properly validated on the backend, making those sites also vulnerable. We will finish by discussing the new attack surfaces exposed by the AWS cross-account-assume-role trust. We conclude that vendors and clients should critically examine whether role trust is the best trust mechanism for their multi-tenant SaaS solution.

Most companies operating in the cloud use one or more third party vendors for logging,

security, data integrations, or other services. Unlike Google Cloud or Azure, which grant access via service accounts based on keys, AWS encourages customers to grant access to third parties via the cross-account-assume-role mechanism and the use of an ExternalID. The cross-account-assume-role mechanism provides a means for secure access without sharing any secrets (like a private key) between the two parties, and the ExternalId feature is designed to avoid "confused deputy" issues. The "confused-deputy" refers to when an agent with delegated authority (vendor portal) cannot differentiate between a legitimate target of a privileged action and an incorrect target (attacker vs victim AWS account). For a more thorough explanation of the ExternalId confused deputy problem, see the **AWS documentation**. We will only summarize it briefly here.

GCP, Azure, and AWS all offer user accounts, roles, and role-based-access-controls (RBAC). User accounts typically have both a means for web authentication but also identity keys for use with software and CLI tools in the form of identity keys and associated secret keys. These user or service accounts can have privileges attached directly to them. However, it is a best practice to only grant the user credential the ability to assume roles and then manage the roles for fine-grain access control. In AWS, using an assumed role is a 2-step process. The first step returns temporary role credentials (access key, secret key, and session_token) and the second command uses these credentials to sign a request and make a second API call. All calls are made on a vendor server.

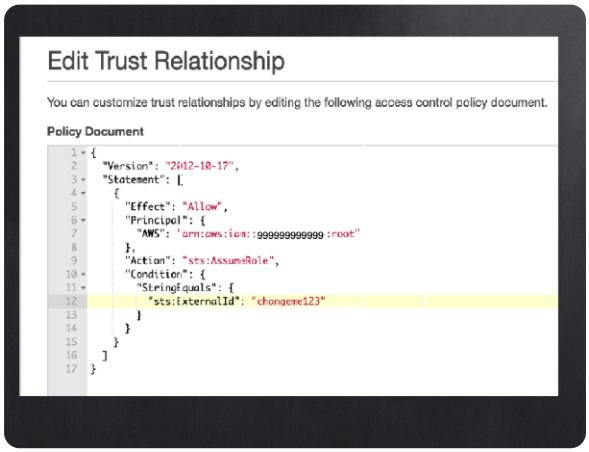
```
aws_vendor_principal> aws sts assume-role
-role-name myrole -role-session-name test
tmp_aws_client_principal> aws subcommand
```

In GCP, the same can be done similarly or combined into a single call.

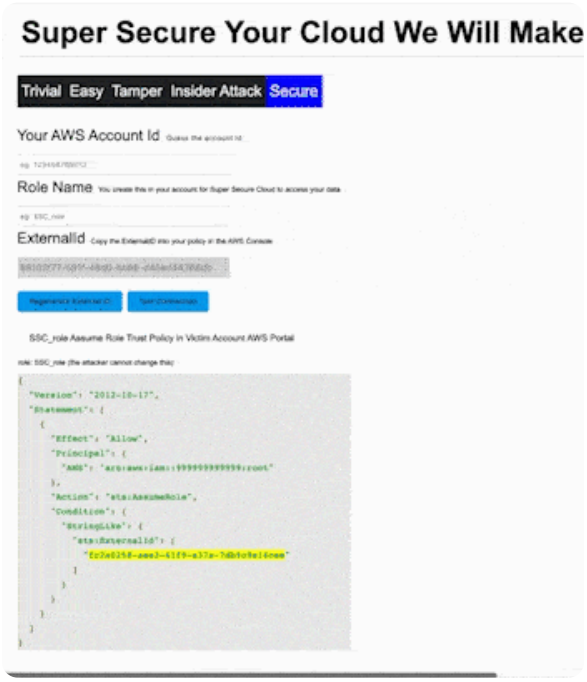
```
gcp_client_principal> gcloud -impersonate-
service-account=$service_account_email
subcommand
```

The command prompt is labeled `gcp_client_principal` to note the fact that this call is made using credentials stored locally in the ~/.gcloud directory. The critical difference is that while **Azure** and **GCP** always anchor their cross-account trust with a key-based service-account/user credential supplied by the customer, that approach is optional in AWS. AWS also allows – and strongly recommends – an Amazon Resource Name (ARN) reference to a vendor account/principal without the exchange of true secret material. There are definitely some advantages to this mechanism and we will discuss the narrow use cases where we feel this is superior in a future blog post, but role-trust without a key-based anchor reduces the ability to apply out-of-cloud-band layers of security and opens up the ExternalId Confused Deputy problem.

The trouble arises when an AWS trust policy like the following in a client's AWS account that is potentially guessable (as highlighted in yellow with the example below), is combined with the use of an ExternalId that can be created or modified by any user in the 3rd party Portal.



To help guard against guessability, the ExternalId should be a long, unguessable value like a UUID (Universally Unique Identifier). But most importantly, only the 3rd party (the trusted party) must create ExternalIds as part of the flow of creating the cross-account trust. No customer should be able to create or select their own ExternalId. Moreover, the 3rd party provider should verify that the customer has *actually utilized the ExternalId as a condition in the trust policy* before storing it and finalizing the cross-account trust. To be clear, the user is always in full control of their AWS account and can modify the trust policy document above as they please. But testing the connection from the vendor portal should only work if the vendor-supplied ExternalId matches a value only the vendor controls. The first screenshot below represents a vulnerable vendor portal.

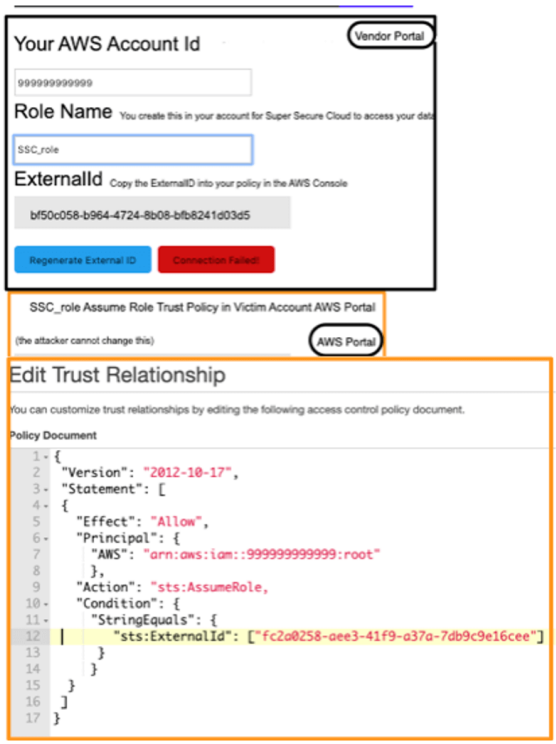


To try out the attack on a simulated vendor page, click **here**.

The insecure vendor portal above suggests a default ExternalId which most customers will use. Then, if an attacker creates a SecureCloud account and submits another SecureCloud customer's IAM ARN (Amazon Resource Name), they will gain access to whatever the portal offers. It is not difficult to enumerate ARNs in an account (in this case we used Pacu), and enumeration is not even required in the trivial case because the vendor tells all customers to use the role "SSC_role". If SecureCloud is read-only for metadata and billing, then this is a low to medium issue per the risk ranking below. If the vendor allows S3 reads or writes to any resources, it is likely a critical.

The screenshot below shows a safe vendor portal. Note that the ExternalId box is not

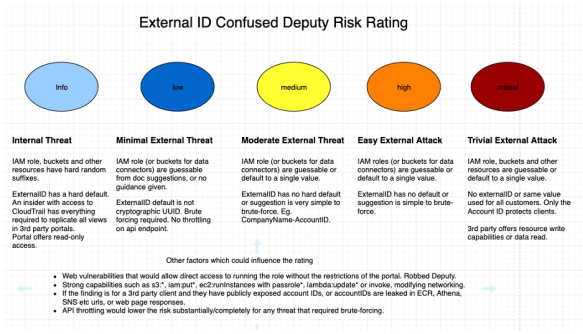
editable, though it can be regenerated.



We should also point out the way that AWS handles the Principal (arn:aws::iam:999999999999:/root) in the above policy. Some would assume that this grants a special high-privilege authority “root” in the 999999999999 account access to your AWS account. However, that reference is not limited to the root principal of the other account but is really a *delegation* to that other account so that the root or another powerful principal on the other side can then decide who from that other account can access the resource (the same is true in S3 and other resource policies). So, in fact, “root” means that the whole account is trusted, and that trust can be delegated on the other side to any principal in 999999999999 with `sts:AssumeRole` on Resource “*”. Of course,

it can be limited to specific resources – but the decision is left to the trusted account. Having `sts:AssumeRole` on "*" is high privilege, but many Cloud Security services do not alert for it being granted, and we often see it mistakenly granted to services which are not intended to be admins. The trust policy could instead use a specific role like `arn:aws::iam:999999999999:/role/role-to-asume-customer-accounts`, but of course high privilege principals in the trusted account could still assume this specific role. Instead, the vendor could request that `arn:aws::iam:999999999999:/user/user-to-assume-costomer-accounts` be granted access to the customer account, which would restrict access to a set of vendor-managed IAM user credentials which could be kept out-of-band as a user – unlike a role – cannot be "assumed".

Praetorian uses the following guide for assigning risk to 3rd party confused deputy findings.



Results and Methodology

Risk	info	low	medium	high	critical	Total at risk
UI %	5	10	8	5	5	37%
Tamper%	2	4	5	1	1	15%
Total %	7	12	13	6	6	52%

Percent of Role-Based Trusts Vulnerable by Risk Level with and without Tampering (90 vendors total) based on **CVSS**

Vendors who only offered key-based access to client calls were excluded from the calculations. The vendors who made errors in the UI (User Interface) either did not understand the ExternalId concept or had a lapse such as getting the "create" form correct, but then allowing modification in the "update" form. Many vendors who understood the purpose of the ExternalId failed on execution. Typically, this was due to including ExternalId in POST/PUT requests in the create or edit forms for integrating AWS. The UI showed the ExternalId as immutable but the HTTP request to the API accepted our tampered value. Combining either flaw is represented in the Total row.

If it was determined that the ExternalId could be modified, the tester would create a second vendor account to ensure other backstop measures were not employed. For example, it was common for vendors to disallow integration of an AWS account ID once another customer had used it, which effectively blocked the attack (assuming no bypass was found).

The vendors we tested ranged from new startups to billion-dollar companies. Praetorian has attempted to contact all vendors found to have the vulnerability, and is coordinating with cert.org and AWS to continue responsible disclosure. We are not

disclosing any vendor names to afford time for remediation, but we encourage all customers and vendors to review their AWS integrations and guidance is provided in Part II.

Risk

Value of the target

What's better than hacking a bank? Hacking a security company with 5 banks as customers. This thinking can be seen in recent trends targeting cloud service providers. The standards for such trusted providers should be raised accordingly.

Insurer Beazley PLC **estimates** that 24% of the ransomware claims that it received during the third quarter of 2018 were caused by a vendor or managed service provider.

Revelations in mid-2019 about the Advanced Persistent Threat APT10 CloudHopper campaign show just how focused advanced hackers are on targeting 3rd party providers, especially Managed Service Providers with access to hundreds or thousands of customer accounts. However, the Center for Internet Security (CIS), Cyber Security Alliance (CSA) and the cloud providers talk almost exclusively about the shared responsibility model between the cloud provider and the customer. The reality is that nearly all companies with more than 100 employees have granted 3rd party access to their infrastructure. Most of Praetorian's cloud customers use 5 or more vendors to support their infrastructure and services.

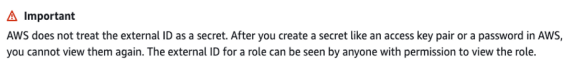
Difficulty of the attack

In order to conduct the attack, the following is required:

1. A vendor which is susceptible
2. The AWS account ID of a vendor customer
3. The role name used in the customer's account to grant trust
4. The ExternalId obtained via CloudTrail logs or brute-force

Our research has shown that the first item is not hard to come by. Obtaining the account ID and role come from two main sources. Many times, both of these are leaked in documentation, source repositories, screenshots or slide-shares. If not, the tool Pacu by Spencer Gietzen of RhinoLabs can be used to brute force AWS roles given an account ID as described in **Assume the Worst: Enumerating AWS Roles through 'AssumeRole'.** The title is apt – one should not assume that their AWS account ID nor IAM role is a secret. Of particular interest is the fact that Pacu is run with any attacker's valid AWS credentials and the logs of failed attempts show up in the attacker's CloudTrail, not the victim's.

As for the ExternalId, AWS clearly states in their documentation that they do not consider the ExternalId to be a secret. From their **documentation**:



The ExternalID is logged in CloudTrail in clear-text and most vendors follow this lead. It can be expected to be in logs, browser history, etc. None of the portals that Praetorian investigated had throttling for the account integration page, so brute-forcing probability depends on the how complex the suggested or provided externalId is.

Vendors fall into three main classes regarding ExternalId:

1. No ExternalId, or a recommended universal value
2. User chosen or an easily enumerable value like a vendor account ID or timestamp
3. Hard UUIDs which cannot be brute-forced (but may be leaked or stolen)

Compromise Impact

At a minimum, any vendor with the ExternalId vulnerability offers an attacker privilege escalation from the ability to read CloudTrail in a victim AWS account to the privileges of the vendor portal. This is a “low” or “informational” risk finding, but it would be a convenient way to achieve persistence and it would be very difficult for a customer to detect. The vendor dashboard would give them persistent and undetectable recon with valuable security information already organized for the attacker’s use.

The worst-case scenario would be devops, MSPs, data analytics, or security vendors offering auto-remediation where the vendor-trusted role has write capabilities (sometimes even admin). Data analytics and backup

solutions could yield complete data compromise or corruption. In these case, the ExternalId vulnerability is a high or critical risk issue. One of the main reasons vendors offer editable ExternalIds is to make it simpler for enterprise customers with 50+ AWS accounts to share a single ExternalId across each trusted role in each account. However, this would mean that compromising the ExternalId for a low-security AWS account could lead to access to the AWS account with the "crown jewels", the company's most critical information. In a pen-test engagement, crossing from one AWS account to another is typically the hardest barrier to surmount. The trusted role name will likely be the same across accounts within an enterprise due to copy-pasting or scripting: if the ExternalId is also replicated, it becomes trivial to cross AWS account boundaries via vulnerable vendors.

The CloudTrail logs of the vendor contain all the role ARNs and ExternalIds of all the vendor's customers. Therefore, compromise of the vendor account's CloudTrail would remove the need to brute-force any customer values. In fact, if the vendor is using their product in the AWS account which runs the vendor's web services, it is the easiest vendor customer to attack (vendor-is-also-a-client-of-vendor), as the account ID is known, leaving only the role and the ExternalId to guess. If the vendor offers the ability to access CloudTrail, CloudWatch or S3, then compromise of the vendor account would lead swiftly to compromise of all customer accounts as all required attack elements are stored in the CloudTrail logs.

Finally, there are several factors which

increase the risk or impact of the ExternalId confused deputy. We will leave the details for an upcoming blog, but in short, Server Side Request Forgery (SSRF), or other common web vulnerabilities could circumvent the restrictions of the vendor portal and afford the attacker the complete set of permissions of the trusted role.

Single-Account Restriction Bypasses

If a vendor chooses to protect against the confused deputy by preventing an AWS account from being registered a second time instead of properly protecting the ExternalId, Cross-Site Request Forgery (CSRF) vulnerabilities on the delete account integration page could allow an attacker to remove the account making it available once again. A more likely scenario is that the AWS account is set up for a trial and either the trial ends or the AWS account is removed in the portal without deleting the IAM role in the customer's AWS account. The vendor does not have the ability to remove the role which the customer created in the customer's AWS account, so it is often left behind. In either case, an attacker would then be able to gain access by the methods above.

Solution

Key or role-based trust is not a one-size-fits-all solution. Both can be done securely, but there are several pitfalls to avoid.

For vendors providing AWS integrations the

following are recommended as best practices.

1. Generate a cryptographic Universally Unique Identifier (UUID) for the ExternalId
2. Do not allow the customer to modify the UUID in the UI
3. Do not include the ExternalId in any PUT or POST requests which may accidently be consumed and used to modify the ExternalId on the server side
4. Treat the ExternalId as you would a private key. Do not log it in clear text, or display it to the user after the initial connection test succeeds
5. Ensure that the assume-role call fails when no ExtrnalId is supplied

AWS could improve its messaging and reduce the risk with the following actions.

> **Improve documentation including guidance for automation-friendly provisioning of ExternalIds**
>
> **Stop logging the ExternalId in clear text**
>
> **Make it clear to vendors that ExternalId should be treated like a secret in the vendor account**
>
> **Add support to CloudFormation 3rd Party Registry to optionally confirm correct usage of ExteralId via a "trust" handler additional to the CRUD handlers**

Is AWS IAM role-based trust always better than key-based trust?

The argument for role-based trust is made by AWS and repeated by numerous vendors. Below is an **AWS blog from 2016** which strongly argues for role-based over key-based based access. My comments are prefixed with "Praetorian>".

Using cross-account roles addresses and mitigates a number of risks, so it's worth taking a closer look at how cross-account roles help address the security questions we listed earlier.

Are keys being managed securely? The Role allows the partner to get temporary credentials when they need to use them. Unlike Access and Secret Keys, these don't need to be stored, so vendor doesn't need to be concerned with managing keys.

> **Praetorian> The vendor still needs to store the client's role-ARN and ExternalId and should treat them like secrets for defense-in-depth protection. If the vendor supports multi-cloud, they already need a solution for Azure and GCP key storage.**

Are keys being rotated frequently?
Credentials generated by STS expire after an hour. Many of our software development kits (SDKs) have credential providers that handle this automatically, so neither the APN Partner nor the customer needs to manage credential rotation manually.

> **Praetorian> If the vendor has not correctly implemented ExternalId or the webserver is ever vulnerable to SSRF then the (role-ARN, externalId) pair acts in lieu of a key. Arguments to rotate the pair are at least as strong as arguments to rotate a key. The best practice would be for the vendor to require the client to**

> provide a user credential and a trust role which the user can assume. The trust policy would have a SourceVPC condition block. Then you get the benefit of short term credentials as the user credential would be useless outside of the specified VPC and the role credentials expire. Some vendors do use this model. If a breach occurs for access to the database holding the access keys, the risk of future use of any keys would force the vendors to notify all customers to rotate keys. If a breach occurs for the vendor role, the vendor could argue that clients do not have to be notified because of 1 hour session expiration, because the vendor might not be aware that the role can be renewed perpetually. Although vendors may consider this a positive, the fact that access-keys force transparency may actually be better for customers.

Can you control who has access to the customer's AWS account? The role in the customer's account can be assumed only by an authenticated AWS identity in the partner's account. The customer knows that only the APN Partner is accessing their resources, and the APN Partner can focus solely on managing and protecting the IAM roles and users in their own account.

> Praetorian> The temporary keys returned by the metadata service can be called from anywhere. Although GuardDuty will alert if the call does not come from an EC2 instance, a better restriction would be to include the "aws:SourceVPC" in a second condition block. This could prevent the key from being used unless the call was made from a list of VPCs owned by the vendor.

Is the access policy associated with the key too permissive? A role can't have root key permissions, and since the cross-account

role's trust policy specifies the partner's account, it is more likely that the permissions in the role's access policy will reflect the partner's requirements. APN Partners can encode cross-account IAM roles in AWS CloudFormation templates to make sure that customers are giving them exactly the permissions they need.

> **Praetorian> It is unlikely that a customer would pass root credentials and this could be checked for and rejected by the vendor. A Policy Document for the user can be created with CloudFormation just as easily as it is for roles, as CloudFormation could handle creating the customer user account as well.**

None of the arguments above are necessarily worth giving up a key-based paradigm which every UI developer already understands. However, when vendors supply options for both key-based and role-based AWS access, many vendors strongly recommend role-based such as the following:

- **IAM User** - Authentication using IAM credentials, access and secret keys.

  Warning: Using IAM User credentials is not recommended as they are less secure than using IAM roles.

We have never seen an article that pushes back on the superiority of unanchored role trust. I think it would be healthy to have that conversation. I would note that GCP and Azure borrow a lot of paradigms from AWS, but the role-based trust is one they both chose not to follow. To be clear, we do not make the claim that role-trust is inferior to key-based trust in all cases and hope to elaborate on this in the future. However, we do not see it as the clear winner in all contexts (as it is often

presented), and think more nuanced guidance from AWS and better documentation is warranted. For example, if a vendor was not aware that attaching the `sts:AssumeRole` to the webserver was an obvious antipattern, or if the webserver was running in a large monolithic AWS account, then they should reconsider offering role-trust as an option. In addition, AWS should detail best practices for vendors to follow that describes a reference architecture for the vendor web and API servers.

Un-anchored role trust with ExternalId can certainly be done in a secure manner. However, after seeing how common it is for vendors to get un-anchored role-based trust wrong and that the arguments in the AWS security blog were not very strong, Praetorian recommends that each vendor critically examine their own security context before strongly recommending role-based access (or even supporting it). Clients should test the on-boarding process themselves and not assume perfect security of their trusted vendors. If you are a vendor concerned about your implementation, contact your AWS partner of Account Representative. In the next blog in this series, I will look at how the security and cloud industry got here despite the fact that the majority of vendors tested were Cloud Security service providers and many of them had been pen-tested for years.

While the conclusions reached in this blog post are our own, Praetorian would like to thank the Amazon Security and Partner Teams for their valuable feedback and assistance with contacting vendors to make corrections and improvements. We would also like to thank

Scott Piper and the maintainers of the **CloudMapper** (weboftrust) as it was used extensively in discovering new vendors on engagements.

CLOUD IN SECURITY INSIGHTS

About the Author

FOLLOW

Kesten Broughton

Kesten is the Cloud Services lead at Praetorian focused on novel attacks and remediations.

View More Articles by Kesten

If you like this, you might like . .