ermetic | **Whitepaper**

Misconfigurations Leading to Amazon S3 Ransomware Exposure

Hard Facts and Mitigation Techniques

A report on new cloud security research that identifies the threat of ransomware to misconfigured S3 buckets and recommends strategies for effective mitigation

Executive Summary

As more and more data moves to the cloud, cloud storage platforms are becoming an attractive target for ransomware operations. Whether you're a security practitioner, a developer, a data officer or other executive, if your business relies on data (read: just about every business), ransomware is a concern.

Amazon Web Services (AWS) documentation states that Amazon Simple Storage Service (Amazon S3) buckets are designed for 11 9's (99.999999999%) of durability – which is very reliable. While AWS provides this durability, per the Shared Responsibility Model, configurations of the buckets and the access permissions to them are the responsibility of the customer. This means that if permissions to a data resource should be abused by the identity to which they are granted, no guarantee to durability on the vendor's side would be effective or even relevant.

In our research, we mapped out scenarios in which a combination of permissions could allow an identity to perform ransomware on a bucket. We then analyzed several Amazon S3 features that could potentially mitigate the risk from such misconfigurations, to identify scenarios in which using these features would be effective. We also determined the most common risk factors that can make identities vulnerable to being compromised.

Putting theory to practice, we used the Ermetic analysis engine to analyze real environments and look for scenarios in which all the following factors were true:

- An identity had a permissions combination that enabled it to perform ransomware
- Effective mitigation features were not enabled on the S3 buckets to which the identity had access
- The identity was exposed to one or more risk factors, such as public exposure to the internet, that could lead to its being compromised

The results revealed very high potential for a ransomware attack in the environments of the organizations surveyed, via different scenarios. Key findings included:

- Overall, in every enterprise environment sampled, we found identities which, if compromised by an attacker, could be used in executing ransomware. These were identities that were configured with a risk factor as well as had permissions that enabled them to perform ransomware on at least 90% of the buckets in an AWS account.
- Over 70% of environments had machines publicly exposed to the internet that were linked to identities whose permissions could be exploited to allow the machines to perform ransomwarelevel privileges, they would be entitled to sufficient access to perform ransomware

- Over 45% of environments had third-party identities with the ability to perform ransomware by elevating their privileges to admin level. By gaining admin level privileges, they would be entitled to sufficient access to perform ransomware
- Almost 80% of environments had AWS Identity and Access Management (IAM) users with enabled access keys that had not been used for 180 days or more and that had permissions that enabled them to perform ransomware
- Almost 60% of environments had IAM users that had the risk factor of console access without a requirement for MFA at login and that had permissions that enabled them to perform ransomware

Our research focused on only "smash and grab" operations that involve the compromising of single identities. In targeted campaigns, bad actors will likely move laterally to compromise multiple identities and use their combined permissions to increase access to resources, greatly improving their ability to execute ransomware.

We conclude this report with several strategies you can easily implement to dramatically reduce potential ransomware exposure resulting from misconfigurations of S3 buckets in your environment.

Background

Ransomware is a type of denial of service attack designed to extort the victim -- usually for money -- in exchange for a return of access to the resources to which the victim was denied. For example, a type of ransomware aimed at data is based on the principle of denying access to the data using encryption and offering the decryption key and/or "service" of decrypting and returning the data to plain text in exchange for a handsome ransom.

As more and more infrastructure makes its way to the public cloud, it is expected (if not almost certain) that we will begin to increasingly encounter different kinds of attacks aimed specifically at data resources in the cloud. One of the most popular data resources on AWS (and possibly overall) is S3 buckets. Amazon S3 (or Simple Storage Service) is "an object storage service that offers industry-leading scalability, data availability, security, and performance." Specifically, S3 is advertised as, and considered, incredibly durable ("99.999999999% (11 9s) of durability").

Since S3 buckets are very fault-resistant, the access configuration of the environment is the most -- and, as we will show, highly -- probable vector through which the data stored on them can be lost or the access to it denied.

This is the vector that a malicious actor will likely attempt or be forced to leverage when aspiring to perform ransomware.

Understanding how this vector can be leveraged to perform such an attack is crucial for every security practitioner seeking to minimize their environment's attack surface for S3 bucket ransomware attacks by properly configuring their S3 bucket and access permissions to them.

Methodology

Our research involved sampling real-life cloud environments to discover all the identities with relevant permissions combinations, reviewing the configurations of the potentially exposed S3 buckets and analyzing the security posture of the identities. While such assessments are near-impossible to carry out manually, we were able to automate the process using the Ermetic cloud analytics platform to gather and compile the relevant configurations.

We then looked at three built-in AWS mechanisms for their effectiveness in mitigating the damage should an S3 bucket be attacked. (We delve further into the pros and cons of these mechanisms in the "Recommendations for Mitigating S3 Exposure to Ransomware" section.)

We also analyzed the execution vectors through which a bad actor can carry out an S3 ransomware attack. We provide a high-level description of these vectors and drill down into the specific S3 actions for which the attacker would need permissions to execute such an attack. To be clear, the presumed attack is NOT breaching S3 buckets themselves, rather, is using a compromised identity that, due to misconfigurations, is technically allowed to use a relevant set of permissions that may enable the bad actor to cause damage.

Permissions / Mitigations

We defined scenarios in which a malicious actor could successfully wage a one-off ransomware attack (or full-blown ransomware campaign). We then translated those scenarios into the combinations of permissions that would make the attack possible, namely:

- Accessing the data and then discarding it using straightforward delete operations
- Accessing the data and then discarding it using lifecycle configuration rules
- Accessing the data and then discarding the KMS key used to encrypt it[1]

- Escalating privileges to the bucket using a bucket policy
- Denying access to the bucket using a key policy on a KMS key used to encrypt the bucket
- Modifying ACLs to escalate privileges and gain access to the bucket

To amplify the existence of risk, for all scenarios we required inclusion of s3:ListAllMyBuckets -- which allows listing an account's buckets -- as this permission makes a potential attack even easier.

Amazon S3 Deletion Prevention Mechanisms

We looked at three AWS bucket mechanisms that can help mitigate the identified attack vectors.

MFA Delete

AWS helps make the task of permanently deleting an object extremely difficult by enabling you to require that the bucket owner "include two forms of authentication in any request to delete a version or change the versioning state of the bucket" (see the AWS MFA delete documentation). Keep in mind that performing actions on behalf of the bucket owner (usually the AWS account in which the bucket was created) means using the root user of an account. So, while MFA delete is an obviously extremely strict limitation on attackers, it is also a pretty strict limitation for anyone who may need to permanently delete objects from the bucket as part of a regular business operation.

Using MFA delete also requires versioning to be enabled and, to enable MFA delete, you must use the root user of an account with two forms of authentication.

Object Locking

Another AWS mechanism that is extremely effective in retaining data is object locks. Simply put, object locks store objects in a Write-Once-Read-Many (WORM). Object locks guarantee that for a predefined period of time (called a "Retention Period") or until configured otherwise (called a "Legal Hold"), the specific version of an object to which the lock applies will not be allowed to be deleted.

Using object locks requires versioning to be enabled. The more interesting limitation around object locks, though, is probably the fact that, currently, AWS does not allow them to be enabled or disabled after a bucket has been created. That is, even if you wish to use object locks, you will not be able to apply locks to objects in an existing bucket for which the mechanism wasn't enabled during the bucket's creation. In this situation, you could create a new bucket with object locking enabled, copy or migrate the original bucket's content to it and apply object locks.

Bucket Versioning

AWS offers a versioning mechanism that allows you to configure a bucket to maintain versions of the objects stored in it. When a bucket has versioning enabled, a deleted/written over object will not be removed permanently; rather, the bucket retains the old version of the object and simply presents/serves the new object version.

AWS does not employ a "delta" mechanism to optimize the storage it uses; it simply retains the new and old versions of objects in full. This fact has, of course, implications for the cost of employing the mechanism.

Execution Vectors

AWS does not employ a "delta" mechanism to optimize the storage it uses; it simply retains the new and old versions of objects in full. This fact has, of course, implications for the cost of employing the mechanism.

Access and Destroy

The most straightforward way for an attacker to perform ransomware on a bucket is by gaining access to, copying and deleting the information in the bucket -- or otherwise denying access to it to anyone else. Doing so allows the malicious actor to hold the data (so it can retrieve it if the victim pays the ransom) without the victimized organization having any way to obtain it.

The malicious actor can get more "creative" and efficient. For example, instead of storing the data in its own account, which would be at its own expense, the attacker can encrypt the data locally and store it on the compromised bucket in the victim's environment (assuming it also gained the ability to write objects in the bucket). For our purposes, we will assume that simply being able to obtain the data and deny access to the data by its rightful owner, even if the malicious actor has to house it later, are enough to make the attack worth the malicious actor's while.

This attack method has four types of combinations that are interesting to distinguish between: data deletion when versioning is not enabled, data deletion when it is, deleting a KMS key used to encrypt the bucket (should the bucket be encrypted with a KMS key) and deleting the information on the bucket using lifecycle rules. Each of the following methods requires the ability to access the information in the bucket first; doing so would include performing the following functions:

- Listing the objects in the buckets with the s3:ListBucket permission granted to a specific identity or, if the bucket is open for use publicly, by its ACL
- If the bucket is KMS encrypted, decrypting the information on the bucket using kms:Decrypt on the KMS key with which it is encrypted
- Reading the contents of the objects using s3:GetObject

We will now describe the methods for denying access to the objects in the buckets -- an action that complements the ability to read information from them.

Data Deletion When Versioning Is Not Enabled

When the bucket does not have versioning enabled on it, it's sufficient to be able to delete an object using s3:DeleteObject or by overwriting it with a different value -- possibly, as mentioned, ciphertext generated by encrypting the original data locally -- using s3:PutObject or by the bucket being public for overwriting due to its ACL.

Data Deletion When Versioning Is Enabled

If versioning is in fact enabled, the malicious actor would have to have access to two additional functions to effectively carry out its plan; first, it would need to be able to list all the versions using s3:ListBucketVersions or by means of the bucket being public for listing due to its ACL. It would then need to be able to perform s3:DeleteObjectVersion so it can remove all previous versions of the object in the bucket in addition to deleting/overwriting the current version.

KMS Key Deletion

If the bucket is KMS encrypted, there's another way the malicious actor can make the objects in the bucket unavailable. If it can successfully delete the KMS key with which the objects were encrypted, there is no way for anyone to decrypt the bucket. To do so, the malicious actor will need to have permissions to perform kms:ScheduleKeyDeletion on the KMS key in question.

Fortunately, deleting KMS keys doesn't happen instantaneously and, at the very least, would require scheduling the job seven days in advance. So detecting that such a job has been scheduled and canceling it using s3:CancelKeyDeletion is possible. However, this kind of monitoring would need to be done regularly. Unfortunately, if this vector is performed successfully, it could be devastating to all the buckets encrypted by the deleted KMS key whose data the attacker had first copied and saved before deleting the key. Also, of course, object locking and even MFA delete would be of no use in preventing the damage caused by deleting the key.

Managing Lifecycle Configuration

S3 buckets have a very useful mechanism for managing "Lifecycle Rules" by performing actions, such as transitioning versions of objects between storage classes, automatically expiring current versions of an object and -- and here's where it gets interesting -- permanently deleting previous versions of a bucket. Combining the ability to expire an object and then have it permanently deleted is potent. A malicious actor that gains the ability to configure lifecycle will be able (unless otherwise prevented) to configure the bucket to "self destruct" its objects. Since each step -- expiring the objects and permanently deleting the objects -- requires a lag time of at least one day from the configuration time to the attack, this action would have a "lag time" of at least two days before becoming effective, allowing an administrator to pick up on and react to it before it becomes effective.

To be able to manage lifecycle configuration on a bucket, a malicious actor would need to be entitled to the permission s3:PutLifecycleConfiguration.

Resource-Based Policy Denial of Service for KMS Keys

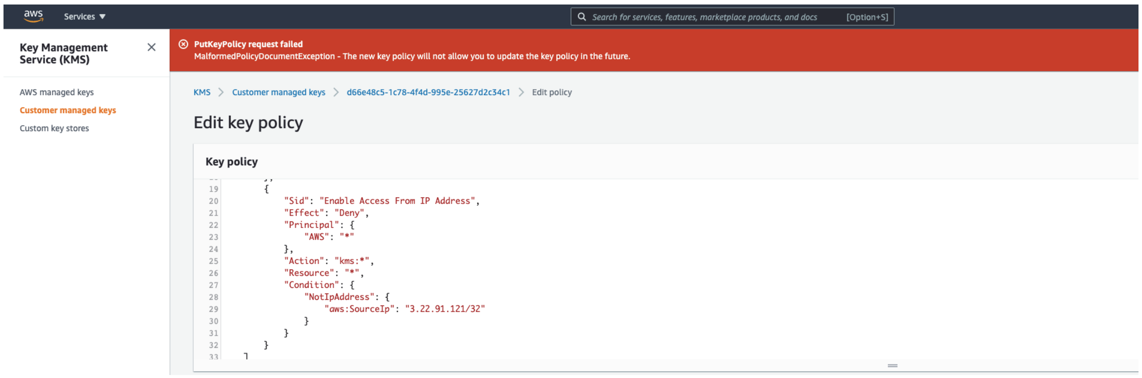
Another way through which a malicious actor can deny access to a bucket and hold it hostage is by making a simple change to a resource based policy. As shown in a presentation last year by the late Spencer Gietzen, by modifying a resource based policy of a KMS key used for encrypting a bucket (using kms:PutKeyPolicy), a malicious actor can prevent others (including the bucket owner) from gaining access to the bucket and its objects -- and, of course, prevent them from changing the policy to remediate the situation. Since you need to be able to perform the kms:Decrypt action to access the encrypted bucket, preventing access to the KMS key effectively prevents access to the information in the bucket.

Here's how the attack would work: say the malicious actor gains the ability to manipulate the resource based policy of the KMS key. The actor will use this ability to apply this policy:to apply this policy:

```
{
   "Version": "2012-10-17",
   "Id": "key-consolepolicy-3",
   "Statement": [
       {
           "Sid": "Enable Access From IP Address",
           "Effect": "Deny",
           "Principal": {
               "AWS": "*"
           },
           "Action": "kms:*",
           "Resource": "*",
           "Condition": {
               "NotIpAddress": {
                   "aws:SourceIp": "<ELASTIC_IP_CONTROLLED_BY_ATTACKER>/32"
               }
           }
       },
       {
           "Sid": "Enable Access From IP Address",
           "Effect": "Allow",
           "Principal": {
               "AWS": "*"
           },
           "Action": "kms:*",
           "Resource": "*",
           "Condition": {
               "IpAddress": {
                   "aws:SourceIp": "<ELASTIC_IP_CONTROLLED_BY_ATTACKER>/32"
               }
           }
       }
   ]
}
```

This policy change makes it impossible for anyone who can't make calls from an elastic ip that the malicious actor controls to perform any actions on the bucket (including modifying the resource based policy to regain control of it).

Note, as shown in Figure 1, that AWS prevents you from locking yourself out when applying a key policy, to prevent such lockouts made by mistake. However, this preventative measure does not prevent the setting of a new policy from the actual elastic IP address.


aws
Services ▼
Search for services, features, marketplace products, and docs [Option+S]
Key Management Service (KMS)
AWS managed keys
Customer managed keys
Custom key stores
KMS > Customer managed keys > d66e48c5-1c78-4f4d-995e-25627d2c34c1 > Edit policy

PutKeyPolicy request failed
MalformedPolicyDocumentException - The new key policy will not allow you to update the key policy in the future.

Edit key policy

Key policy

```
        {
            "Sid": "Enable Access From IP Address",
            "Effect": "Deny",
            "Principal": {
                "AWS": "*"
            },
            "Action": "kms:*",
            "Resource": "*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "3.22.91.121/32"
                }
            }
        }
    ]
```

Figure 1: Being blocked when trying to configure a key policy that would restrict oneself from making future changes to the policy

Bucket Privilege Escalation via ACLs

If an attacker can access s3:PutBucketPolicy, they can grant themselves permissions to potentially perform the "Access and Destroy" vector described earlier (see next section).

However, even without the ability to modify the bucket policy, an attacker can exploit yet another vector to obtain access to buckets: leveraging buckets that are publicly accessible via ACLs (a legacy mechanism designed to allow public access to buckets). If a bucket is currently accessible via ACLs or a malicious actor can modify the bucket's ACLs to make the bucket accessible, the actor has a viable path for accessing the material on the bucket and deleting the objects. As there are several mechanisms that determine if an ACL is effective or not on a bucket, there are several attributes to process to determine if a malicious actor can gain access by using the ACL. Let's review them.

First, an ACL can allow an anonymous entity to configure the bucket ACL and, in doing so, provide itself more permissions by making it public. This may also be accomplished by the action s3:PutBucketAcl.

An ACL can also be placed on an object using the s3:PutObjectAcl action, which can allow an anonymous user to gain access to the contents of an object, replacing s3:GetObject.

An account can place a public block on all ACLs (which is usually not a bad idea). This configuration can also be overridden by performing the s3:PutAccountPublicAccessBlock action.

You can configure a bucket to block all or new ACLs placed on a bucket. Also, any public access (including access points, which we do not cover here) can also be blocked using this mechanism (also usually not a bad idea). This configuration can be overridden by performing the s3:PutBucketPublicAccessBlock action.

An appropriate bucket policy can prevent malicious actors from gaining access to the buckets or performing certain actions on them by using appropriate "Deny" statements. However, an attacker is also able to remove such statements using the s3:DeleteBucketPolicy.

Therefore, the following combination of permissions (or lack of preventative blocks) can be utilized to manipulate a bucket's ACLs to allow a malicious actor with enough permissions to perform an effective ransomware attack (each bullet listed below must be satisfied):

- Permission to s3:PutBucketAcl or having the bucket available for public configuration via an ACL
- Permission to s3:GetObject or s3:PutObjectAcl (which could allow setting public access to objects)

- If the bucket is KMS encrypted, decrypting the information on the bucket using kms:Decrypt on the KMS key with which it is encrypted
- No public block on the account configuration or having permission to s3:PutAccountPublicAccessBlock
- No block for any and/or new ACLs and/or all public access, or having permission to s3:PutBucketPublicAccessBlock
- No deny statements for relevant actions on the bucket policy that prevent public access should an ACL be in place and/or no bucket policy at all, or having permission to s3:DeleteBucketPolicy

It should be noted that, as seen in the AWS ACL permissions table, for any identity other than the bucket owner (the AWS account), write permissions granted by an ACL “only” provide the grantee with s3:PutObject permissions. This means that versioning is supposed to be a sufficient solution for addressing this kind of exposure.

Bucket Privilege Escalation via Bucket Policy

In a more straightforward approach to bucket privilege escalation, a malicious actor can, by placing a bucket policy using s3:PutBucketPolicy permissions, allow itself access to any action using even external identities.

If the bucket is encrypted using a KMS key, the malicious identity will also need to be able to carry out kms:Decrypt on that key to access the data on the bucket (so that, going forward, it can extort ransom). Note that, in this scenario, a service control policy (SCP) that denies access to relevant actions for getting the objects, such as s3:GetObject and s3:GetObjectVersion, can be effective in mitigating the threat since it will apply to identities within the account where it is applied. So, unless it also has access to an external identity that has the ability to use kms:Decrypt (on which the SCP won’t apply), the malicious actor will not be able to access the bucket’s objects.

Privilege Escalation to Admin

We also considered identities that can escalate their account privileges to the administrator level, such as IAM roles and IAM users, to be a threat to all buckets in an account. For an excellent review on privilege escalation in an AWS environment, see this Rhino Labs post.

Note that if the bucket is encrypted using a KMS key from the account where the malicious identity can become an admin, this vector can be more severe. This is because a malicious actor elevated to admin will also have full access to the key encrypting the bucket, so could implement the vector of resource based policy denial of service for the KMS key. This vector will be resistant to MFA delete and object locking mitigation controls as they cannot assist when the encryption key is taken hostage. However, since buckets aren't always encrypted -- and even when they are it's not necessarily with a KMS key within the account -- we've considered MFA delete and object locking as effective mitigation controls for this vector.

Permissions Combinations and Native Mitigation Effectiveness

Table 1 summarizes the S3 attack methods discussed here, the permissions combinations required to implement them, if an attack has a lag time and if using MFA delete, object locking or bucket versioning would be effective mitigating actions.

Scenario	Permissions Combination	Lag Time	Mitigation Controls		
			MFA Delete	Object Locking	Bucket Versioning
Accessing the data on the bucket, then deleting it using delete operations along with any versions	• s3:ListBucket / public list accessible via ACL • kms:Decrypt* • s3:GetObject • s3:DeleteObjectVersion • s3:ListBucketVersions / public list accessible via ACL	**None**	✓	✓	✗
Escalating privileges to the bucket using a bucket policy***	• s3:PutBucketPolicy • kms:Decrypt*	**None**	✓	✓	✗
Accessing the data on the bucket, then deleting it using delete operations	• s3:ListBucket / public list accessible via ACL • kms:Decrypt* • s3:GetObject • (s3:DeleteObject / s3:PutObject) / public write accessible via ACL	**None**	✓	✓	✓
Modifying ACLs to escalate privileges and gain access to the bucket	• s3:PutBucketAcl / public config accessible via ACL • s3:GetObject / s3:PutObjectAcl • kms:Decrypt* • No public block on account / s3:PutAccountPublicAccessBlock • (No block for any / new ACLs or All public access) / s3:PutBucketPublicAccessBlock • (No deny statements for relevant actions on the bucket policy / no bucket policy) / s3:DeleteBucketPolicy	**None**	✓	✓	✓
Denying access to the bucket using a key policy on a KMS key with which the bucket is encrypted	• kms:PutKeyPolicy** • s3:GetEncryptionConfiguration / kms:ListKeys	**None**	✗	✗	✗
Accessing the data and then discarding it using Lifecycle configuration rules	• s3:ListBucket / public list accessible via ACL • kms:Decrypt* • s3:GetObject • s3:PutLifecycleConfiguration	**2 days**	✓	✓	✗
Accessing the data and deleting the KMS key used to encrypt it	• s3:ListBucket / public list • kms:Decrypt* • s3:GetObject • kms:ScheduleKeyDeletion**	**7 days**	✗	✗	✗
Admin-level privilege escalation	• Privilege Escalation Review	**None**	✓	✓	✗

Note: All references to KMS encryption refer to encryption by a customer-managed key.
*** If the bucket is KMS encrypted, the allowance is required on the KMS key.**
**** Works only if the bucket is KMS encrypted. The allowance is required on the KMS key.**
***** If the bucket is encrypted by a KMS key, mitigation is also possible using a service control policy (SCP).**

Permissions combinations for performing ransomware and the effectiveness of mitigation features

Detecting Ransomware Exposure in an AWS Environment

Let’s imagine you are tasked with detecting where ransomware exposure of S3 buckets exists in an AWS environment. You first need to find out if the environment contains identities that can perform a combination of actions on one or more buckets that, should the identity be compromised in some way, could then be exposed to a ransomware attack.

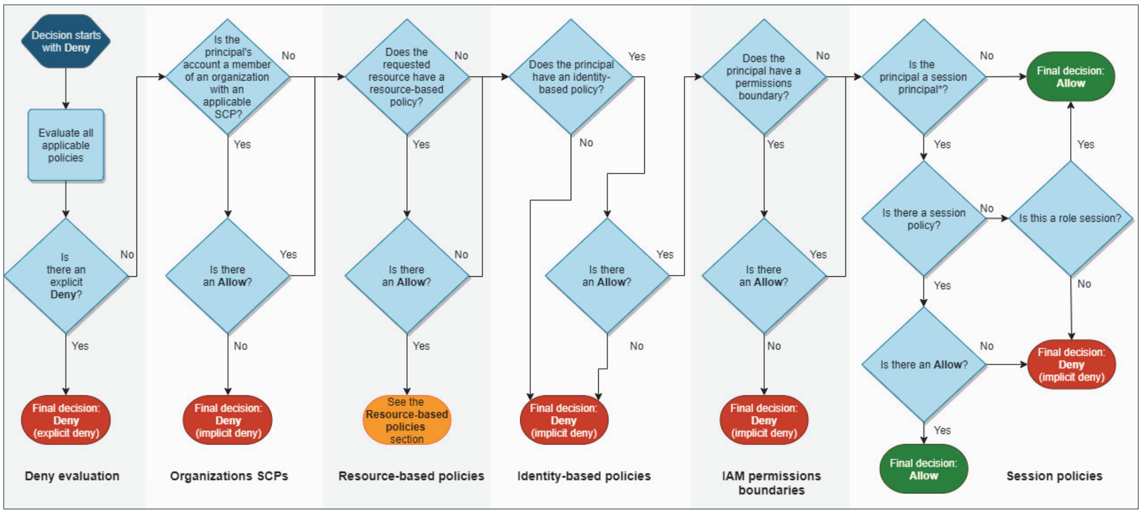
Detecting such identities is a Herculean task; however, things get even more complicated. To make the assessment truly comprehensive you need to assess which identities that can allow a malicious actor to perform the action are actually exposed to the risk of being compromised. You would then need to combine all this information to determine which identity at risk has the ability to perform ransomware on which S3 buckets -- and then assess where to apply mitigating actions, preferably in a prioritized manner.

Unfortunately, gathering and processing all this information is easier said than done. Doing so requires including many sources and extensive computations for the assessment to be correct.

The Ermetic platform does precisely this. It gathers and processes information about both the cloud security posture of the cloud environment (CSPM) and the entitlements of all identities in the environment (CIEM). This combination of capabilities for collection and processing enabled us to run a simple analysis on dozens of AWS accounts of participating organizations -- and receive output that showed just how much and what type of exposure exists in them. The aggregated results are presented in the next part of this document. Before reviewing them, let’s explain what we looked at.

Identities Entitlements

Understanding which identity has permission to do what action is extremely complicated. You get a sense of this from the complexity of the iconic AWS diagram in Figure 2, which shows the evaluation process that AWS uses to determine if to approve or deny an action on a resource.



Decision starts with Deny
Evaluate all applicable policies
Is there an explicit Deny?
No
Yes
Final decision: Deny (explicit deny)
Deny evaluation
Is the principal's account a member of an organization with an applicable SCP?
No
Yes
Is there an Allow?
Yes
No
Final decision: Deny (implicit deny)
Organizations SCPs
Does the requested resource have a resource-based policy?
No
Yes
Is there an Allow?
No
Yes
See the Resource-based policies section
Resource-based policies
Does the principal have an identity-based policy?
Yes
No
Is there an Allow?
Yes
No
Final decision: Deny (implicit deny)
Identity-based policies
Does the principal have a permissions boundary?
No
Yes
Is there an Allow?
Yes
No
Final decision: Deny (implicit deny)
IAM permissions boundaries
Is the principal a session principal*?
No
Yes
Final decision: Allow
Is there a session policy?
No
Yes
Is this a role session?
Yes
No
Is there an Allow?
No
Yes
Final decision: Deny (implicit deny)
Final decision: Allow
Session policies

Figure 2: AWS policy evaluation logic (source: AWS)

There is simply so much to take into account - IAM policies, resource-based policies, permission boundaries and service control policies are just some of the items to be considered.

It's also important to remember that identities can assume IAM roles (which, in turn, can allow the identities to assume other roles). This capability can significantly expand what that role can be allowed to do. The reason this is so important to point out is that even some native AWS tools used to process entitlements, such as AWS policy simulator, don't take this into account.

The Ermetic analysis engine processes this comprehensive and complex information and doesn't just stop at understanding the effective permissions to actions that each identity can perform. It also fully examines AWS CloudTrail logs to determine what permissions each identity actually uses so as to determine what actions each identity should be able to perform, and which permissions are excessive. This fuller approach allows us to determine not just where exposure exists but where it's simply not necessary and can be easily mitigated.

We looked for the existence of identities with the effective permissions to perform one or more of the combinations described above along with the ability to list the buckets within an AWS account using s3:ListAllMyBuckets. Such identities, upon being compromised, can allow a malicious actor to fairly easily detect all the buckets it might be able to access and utilize enough of that access to perform a ransomware attack.

Buckets Posture

As mentioned before - certain mechanisms can make a bucket "resistant" to some of the methods described (see Table 1). To understand if exposure is present, we need to combine information about a bucket's existence with configuration information, such as if versioning, object locks and/or MFA delete are enabled on the bucket.

Also, as described before, we need to process information regarding the ACL and ACL-related configuration of the bucket.

Identities Security Posture

Next, we reviewed various risk factors for identities, to detect scenarios in which it would be possible for an identity to not only perform a ransomware attack using the permissions to which it is entitled but would also be somewhat likely to be compromised.

The list below of potential risk factors for identities in an AWS environment is, of course, not comprehensive; rather, it reflects risk factors that are straightforward to exploit and, using readily available controls, could be easily mitigated.

IAM Users

The main risk factors for IAM users is derived from enabled access keys -- as discussed in our post on TeamTNT malware. As per best practice, we considered a risk factor to be access keys not rotated for 90 days. As a more severe risk factor, we considered active access keys not used in 180 days. We also considered IAM users that had console access without MFA enabled to be at risk.

As a matter of interest, we looked for users that were entirely inactive for 90 days or had credentials not used for 90 days; however, we did not consider such situations to be a risk factor.

IAM Roles

We considered IAM roles to be at risk if allowed for use by a third party as this, of course, puts the role outside the organization's control.

We also looked for IAM roles that were inactive for more than 90 days (as per cloud security best practice, such roles should have been disabled); however, we did not consider such situations to be a risk factor.

EC2 Instances

We determined Amazon Elastic Compute Cloud (Amazon EC2) instances with public exposure to the internet to be a risk factor. This is because a malicious actor is much more likely to exploit an Amazon EC2 that is publicly available than one that is not.

EC2 instances gain access to resources in a cloud environment using IAM roles they are allowed to assume. For this reason, we looked for EC2s that had an IAM Role enabled that allowed them to perform one of the permissions combinations in the above table and whose network configuration made them publicly available on the internet.

Results

As mentioned, we looked for identities that had the ability -- by means of their permissions, a lack of effective mitigation and exposure to a risk factor -- to perform ransomware on at least 90% of the S3 buckets in an AWS account of the AWS accounts we sampled.

Our findings were as follows:

- Every environment we sampled had at least one AWS account in which an identity -- and often many more than one -- met the above criteria

- In over 70% of environments, we found EC2 instances that met the above criteria, with the risk factor being public exposure to the internet. Moreover, the permissions that granted access to the buckets were excessive. That is, the risk of these identities being compromised could have been significantly reduced without hurting business operations by simply removing the unnecessary permissions.

- In over 45% of environments, we found IAM roles available for third-party use that were allowed to elevate their privileges to admin. This finding is incredible -- and pretty horrific for cloud security reasons beyond ransomware. With respect to this study, the finding means that the S3 buckets in the environment were exposed to ransomware.

- In over 95% (!) of environments, we found IAM users that met the above criteria, with the risk factor being access keys that were enabled but unrotated for 90 days

- In almost 80% of environments, we found IAM users that met the above criteria, with the risk factor being access keys enabled but inactive for more than 180 days
- In almost 60% of environments, we found IAM users that met the above criteria, with the risk factor being console access that was enabled but without a requirement to use MFA at login
- Over 96% of environments had inactive IAM roles and almost 80% of environments had inactive IAM users that met the above criteria

Keep in mind that by using lateral movement a malicious actor could potentially access more than one identity and leverage many other risk factors not as straightforward as those presented here.

If the sample we used is any indication, millions of enterprises currently using S3 as reliable data storage are in danger of ransomware attack due to misconfigurations. The high possibility of exposure to even simple ransomware operations is a clear call to action for cloud security stakeholders to take mitigating steps.

Recommendations for Mitigating S3 Exposure to Ransomware

You can take several courses of action today to minimize your organization's exposure to ransomware operations on S3 buckets due to misconfigurations and improve your ability to recover from such an event. We detail these strategies here.

Least Privilege Access as a Strategy

The most effective way to prevent malicious actors from performing any kind of action in your environment is to simply not grant permission where it's not necessary. By designing and granting the identities in your environment the bare minimum of permissions they need to perform their jobs, you also bring to a minimum the blast radius from a hack. This approach is critical for not just mitigating the risk of ransomware but is an excellent, holistic way to minimize the fallout from cybersecurity breaches to your environment. Since very few identities actually need permissions that allow them to perform such an attack across a wide range of buckets, a sensible permissions strategy effectively enforced is a great solution to the ransomware exposure problem.

Using a CIEM platform and, specifically, one that also has CSPM abilities, such as the Ermetic platform, can be of great value in implementing the least privilege approach. In the absence of such a platform, here are several actions for restricting privileges in your AWS environment that will reduce the chances of it being the target of a successful ransomware campaign.

Deny Sensitive Actions

By using a resource-based policy on a bucket and/or a Service Control Policy on an account, you can easily deny certain actions and specifically allow them only when absolutely necessary. Of course, you will have to make sure that identities that actively need to use these permissions can still do so. Considering that the utility of these actions should not be widely available, this mapping should be fairly straightforward (though, again, without a CIEM platform, at least a bit challenging).

On S3 buckets you will want to limit the ability to configure buckets – or figure out their configuration – by limiting actions such as:

- s3:PutBucketPolicy
- s3:DeleteBucketPolicy
- s3:PutLifecycleConfiguration
- s3:PutBucketAcl
- s3:GetEncryptionConfiguration
- s3:PutEncryptionConfiguration (this permission is not explored in our scenarios but can be used maliciously so should not be widely accessible)

On top of that, deleting data from a bucket is usually not something done by a variety of identities. So an action such as s3:DeleteObject should be granted very carefully and, to a greater extent, the action s3:DeleteObjectVersion, as even fewer identities actually need to manage previous versions of the objects when versioning is enabled. Finally, although seemingly counterintuitive, the ability to list the contents of a bucket using permissions such as s3:ListBucket and to list the buckets within an environment using s3:ListAllMyBuckets can be cardinal to a bad actor looking to access data. In any case, such permissions should not be widely assigned (after all, how often does a service running legitimately actually need to list all the buckets in an environment?).

KMS keys used to encrypt buckets are very sensitive resources for which access management is critical. Similarly to S3, the ability to place a resource policy on the key using kms:PutKeyPolicy should be limited. On top of that, scheduling the key deletion (which is an action very seldom actually performed) using kms:ScheduleKeyDeletion should also be very severely restricted. It goes without saying that actually decrypting the information that the key encrypts using kms:Decrypt is very sensitive. Finally, similarly to listing buckets information, having the ability to list keys in an environment, using kms:ListKeys, should also be limited as it seldom needs to actually be used.

Find and "Privatize" All Public Buckets

Finding and removing ACLs that make buckets public is quite easy. To make things easier, you can also configure "Block public settings" at the account level to prevent all public access using ACLs.

It's not only ACLs that can make buckets public. A bucket policy with a statement granting access such as:

```
"Principal": { "AWS": "*" }
```

may cause more than just the identities in the account to have access to the bucket as it literally applies to all AWS identities in any account.

There are several ways to automatically detect if a bucket is public (for example, this post by Auth0 shows how you can find public buckets via ACLs). Tools like Ermetic can automatically show you buckets that are public due to ACLs or their bucket policy. Of course, the hardest part is making the necessary business-internal changes to ensure that no buckets need to have their contents public and, if they do, providing an alternative way to share it.

Separation of Duties

As we've seen, the ability to perform a ransomware attack would usually require performing actions of two different calibers: for example, accessing information, and deleting versions or configuring bucket lifecycle rules. It's usually best to have different identities perform different actions so that, should one be compromised, the damage is limited. Scenarios in which a combination of permissions allows a single identity to perform ransomware are powerful examples of why separation of duties is good practice.

Remove Inactive Roles and Inactive Users

Finally, security practitioners often overlook identities that are no longer needed and are therefore no longer in use. The existence of such identities is, in fact, a violation of the least privilege principle – removing them is an easy win on the way to least privilege implementation. You can use out-of-the-box tools such as AWS's Access Advisor to find identities that are currently unused – as shown in this AWS blog. AWS also lets you find unused user credentials, which you can disable to improve your security posture.

Removal of Risk Factors

Improving the security posture of your cloud environment is a long and hard battle, the many approaches to which exceed the scope of this work. However, certain “easy wins” that are rather simple to implement can reduce dramatically the chance of identities in your environment being exploited by ransomware – or other malicious attacks.

Probably the most notable framework for preventing security risk factors is CIS Benchmarks which, among other things, includes rotating access keys, enabling MFA for users and disabling unused credentials. As obvious as these practices seem, we often see risk in real life environments simply because security practitioners failed to look for and remediate these simple steps. What we found during this research was no exception.

We also highly recommend that you take public exposure to the internet very seriously – such as EC2 instances or ECS Services running task definitions. Obviously, some resources need to be exposed to the internet. However, there’s little reason for those resources to have highly permissive privileges. In our research, we found examples of public resources that can wreak havoc on S3 buckets in accounts that lacked justification to allow such excessive access.

Finally, we making good use of the tools that Amazon supplies out-of-the-box such as Amazon Elastic Container Registry (Amazon ECR) vulnerability scanning when pushing images is also a practice that should be carried out on a regular basis to make sure unnecessary vulnerabilities are not left unmitigated.

Logging and Monitoring

Another crucial layer of security is being able to be notified when certain events occur, such as when sensitive actions are triggered on a bucket. Sometimes this can mean the difference between a malicious campaign being successful or not.

As described in Rhino Security Labs’ review of possible mitigations, having AWS CloudTrail enabled on every bucket with logging management and data events can be quite expensive – but it’s still necessary to have on the buckets that matter the most (that is, those that contain sensitive and/or mission-critical information). Also, as a review by SummitRoute points out, you can use advanced event selectors for more granular control of data event logging. You can stream events to a SIEM or, more simply, create Amazon CloudWatch alerts for sensitive events. The most important events to monitor are those you can effectively respond to.

So, for example, if you get alerted on a deletion job being scheduled for a KMS key (that is, by a notification triggered by ScheduleKeyDeletion), you have at least seven days to respond before the key is effectively deleted. Also, as mentioned, the effect of a lifecycle configuration (applied via the PutBucketLifecycle event) takes at least two days – usually more than enough time for an effective response.

There are also events that can be indicators of reconnaissance activity, such as ListBuckets or ListKeys. Also, ListObjects, ListObjectsV2 and ListObjectVersions may indicate that someone is collecting information prior to an attack on a bucket.

You can also track events such as DeleteObject and DeleteObjectVersion. However, know that if they are used effectively and you find out about them during a ransomware attack, it will probably be too late. The same goes for PutBucketPolicy and PutKeyPolicy.

Deletion Prevention

AWS has two mechanisms that may enable you to effectively prevent objects or versions from deletion. These mechanisms may not be appropriate in all scenarios but, if applicable, can be quite useful.

You can use object locking for a retention period or a legal hold. You can set a default retention period for objects and can also set it in different modes. To make the most of this mechanism, you can use compliance mode, which makes it impossible for the object to be deleted until the period ends.

The key thing to understand is that the challenge with object locks (as is the challenge with versioning management as a whole) usually revolves around management of the duration, or the time in which versions would be locked. You could of course theoretically retain versions for an extremely long period of time but that will have huge costs, especially if objects in the bucket are updated regularly.

Another thing to keep in mind is that you must enable object locks on a bucket when creating it as this configuration can only be done at bucket creation. If you want to use object locks on an existing bucket without it being configured, you need to migrate the bucket to a new one created with object locks enabled. Also, if you enable object locks on a bucket, you should protect it against a denial of wallet attack. If someone with malicious intent locks objects in compliance mode for an exceptionally long retention period, you will not be able to delete the object (see SummitRoute's analysis and example of how to protect against denial of wallet).

Alternatively, you can use MFA delete on the bucket; this will require using the root user, with its MFA token used for authentication, to permanently delete objects of versions. This of course is extremely effective against malicious deletions – however, a bit "too" effective. In most scenarios, requiring the use of the root user with its MFA token is simply too much to ask for just to delete a version of an object. Even using the root user to simply make this configuration is possibly too much of a hassle, especially when there is a significant number of buckets to protect. You can achieve similar protection using a condition on specific actions (such as those we've listed above) that would require MFA authentication to perform them.

For example, you can include the following statement in a bucket resource policy, using the aws:MultiFactorAuthPresent key:

```
{
    "Effect": "Deny",
    "Action": ["s3:DeleteObject", "s3:DeleteObjectVersion"],
    "Condition": {
        "Bool": {
            "aws:MultiFactorAuthPresent": "false"
        }
    }
}
```

This is much easier to configure and still allows identities other than the root user to perform the delete operation. However, this approach is not applicable in the many scenarios in which the identity responsible for the deletion is a service (e.g., AWS Lambda function) rather than a human and/or using MFA is operationally not possible.

Therefore, we were not surprised to discover that of the thousands of buckets reviewed in our research sampling, none had MFA delete enabled.

You can find additional information about immutable storage in AWS here.

Bucket Replication

AWS offers a built-in mechanism for replicating buckets to different S3 buckets for backup purposes. As mentioned, one valuable use of bucket replication is to mitigate malicious delete operations. The mechanism is easy to use and an extremely effective solution should the original bucket get compromised. However, you will want to be aware of a few things when using this mechanism in the context of mitigating ransomware.

First, there's the cost. Replication requires versioning to be enabled on a bucket. Unless you must have versioning for business purposes, the effective cost of using this solution is not only for another bucket but also for managing version retention on the original and backup buckets. If you choose to replicate the bucket to a different region, remember that data transfer has a significant cost unto itself.

Furthermore, to truly make the most of this mechanism, you need to make sure that the bucket to which you are replicating the data is inherently more secure than the original. If exposed to the exact same threats as the original, the new bucket will be of little use (and be yet one more target holding sensitive information). In our research, the identities found vulnerable to compromise had exposure to 90% or more of the buckets in certain accounts. Fortunately, when used only as replication targets, buckets are much easier to secure – because they will be used for very specific actions and by very specific identities. It's more straightforward to apply and monitor a restrictive resource policy to buckets. You can even replicate to a bucket in a completely different account – though keep in mind that placing all replications in one account is not recommended as doing so would create (as also noted by SummitRoute) a security single point of failure.

Conclusion

The complexity of public cloud infrastructure makes understanding the risk to your organization extremely difficult. Ermetic is an award-winning identity first cloud security solution that offers leading cloud infrastructure entitlement management and cloud security posture management in one platform, through a single pane of glass. Organizations use Ermetic to reduce their cloud attack surface and blast radius while reducing time and costs -- and amplifying cloud security expertise for security and engineering teams.

Ermetic is the only solution in its category to offer full-stack insight into entitlements across identities, compute resources, data stores and the network. This comprehensive view enables you to investigate deeply what is going on in your cloud infrastructure at any given time. Ermetic helps you secure your cloud infrastructure effectively and automate least privilege based on actual use.

About Ermetic. Ermetic is an identity-first cloud native application protection platform (CNAPP) that unifies and automates asset discovery, risk analysis, runtime threat detection, compliance and remediation for AWS, Azure and GCP.

Contact us!

To learn more or schedule a demo: **info@ermetic.com**