S3 Ransomware Part 2: Prevention and Defense

By Spencer Gietzen

Published: 2019-06-10 · Archived: 2026-04-05 20:20:17 UTC

This is part two in a two-part series on S3 Ransomware. Part One discusses the attack vector of S3 Ransomware and demonstrates a proof of concept.

Note: This post not only discusses defense mechanisms against S3 ransomware, but it also touches on important general security hygiene you should be following in your AWS accounts.

Part 1 Recap: S3 Ransomware

As demonstrated in part one of this blog, S3 ransomware is an attack that can have an extremely high impact on a company. S3 ransomware is when an attacker is able to gain access to a victim's S3 buckets, then they replace each object with a new copy of itself, but encrypted with the attacker's KMS key. The victim would no longer be able to access their own S3 objects and would need to submit to the attackers demands in order to get them back (or risk the extra time it might take to go to the authorities or AWS incident response).

S3 Ransomware Security Controls

While S3 ransomware can be fairly straightforward for an attacker to perform, with the right defenses in place, you can protect yourself. There are many different ways to defend against and prevent S3 ransomware, and this post aims to outline those various methods.

Defense and Prevention Methods

There is no single method that acts as a "silver bullet" when preventing and defending against S3 ransomware given the cost, effort, and impact each method has on its own. Instead, the following methods are meant to be a collection of good practices to follow. Not every method is a good fit for every environment, so it is important to understand them all and be able to choose the best methods to implement in your own environment.

Method #1: Follow Security Best Practice to Prevent Unauthorized Access to your Account

This is the most obvious defense, in that it essentially means "don't let unauthorized people into your environment", but that is much easier said than done. There is a laundry list of things to do for this defense, but some important ones are outlined here:

- Have all users use methods of temporary access to access the environment instead of long-lived credentials (use IAM roles instead of IAM users).
- Create pre-receive hooks in your Git repositories (more info here) to monitor for commits that may accidentally contain credentials and deny them before a developer pushes their access keys or credentials.

- Perform regular phishing/social engineering training with your team to educate employees on how to spot targeted attacks.
- Enforce multi-factor authentication (MFA) *everywhere possible* for everyone (both for the AWS web console and for AWS access keys)! This makes it more difficult (but still not impossible) for an attacker who has stolen credentials to actually use those credentials.
- Enforce long, complex passwords/passphrases, and if MFA is not being enforced *everywhere*, then enforce password expiration at a reasonable interval while disallowing repeat passwords
- Ensure that strong application security is in place for any application that has AWS access. This can help prevent something like a server-side request forgery (SSRF) attack to an EC2 instance's metadata or a local file read/remote code execution vulnerability from reading credentials from the AWS CLI or environment variables.
- Regularly audit and monitor IAM access that is delegated within your accounts/organization (with something like Security Monkey by Netflix).

Additionally, consider reading this blog post on how AWS accounts are compromised.

Method #2: Follow the Principle of Least-Privilege

This defense involves setting up your environment and delegating access in such a way that no one has more access than they should. That means that for any user (role/group/etc), they should only have the IAM permissions granted to them that they *need* to use, and those permissions should only be allowed on resources that they *need* to work with.

For example, a simple "s3:PutObject" permission granted on any resource ("*") could mean a massive ransomware attack across every bucket in your account. By limiting the resource to a specific bucket, such as "arn:aws:s3:::example_bucket", then only that specific bucket could be targeted.

The principle of least-privilege should be applied at both the IAM policy level and the resource policy level (such as an S3 bucket policy) to be most effective.

Method #3: Logging and Monitoring Activity in Your Account

You should always have AWS CloudTrail enabled in your account. Ideally, CloudTrail should cover all regions in all of your accounts, log read and write management events, log data events for your S3 buckets and Lambda functions, enable log file encryption, and enable log file validation.

It might not always make sense to log data events for *every* bucket in your account depending on your budget, because it can get fairly expensive if they are frequently accessed. If you aren't logging data events for every bucket, it is very important that the buckets you *are* logging are the ones that contain sensitive content. This way, you can view fine-grained details of who is accessing your data, how they're doing it, and where they're doing it from.

Log file validation in CloudTrail can help ensure that your log files are not being modified before you read them, so you can 100% trust what you are looking at. This is important to enable, but be sure to actually verify your logs

(with something like “aws cloudtrail validate-logs”), as you won’t be alerted of modifications to your logs, even with the setting enabled.

In addition to CloudTrail, a tool like AWS GuardDuty should be enabled to monitor for malicious activity within your account. These alerts along with your CloudTrail logs should be exported to an external SIEM for further inspection and monitoring.

If you use AWS Organizations, you should be enabling CloudTrail and GuardDuty at the Organization level and applying them to your child accounts. This way, attackers in child accounts cannot modify or disable any important settings.

Depending on your budget and other factors, consider enabling other types of logs and monitoring tools for your other resources. This might include something like Elastic Load Balancer access logs or host-based logs for your EC2 instances, or possibly enabling something like AWS Inspector or AWS Config.

Method #4: S3 Object Versioning and MFA Delete

This is perhaps the most important, but potentially very expensive, defense method to use against S3 ransomware specifically. S3 Object Versioning allows S3 objects to be “versioned”, which means that if a file is modified, then both copies are kept in the bucket as a sort of “history”. The same thing happens if a file is uploaded with the same name as a file that already exists in the bucket. An example scenario here would be a versioned bucket that CloudTrail logs are being stored in. If an attacker modified a log file to remove traces of their activity, then the defender could compare the old version of the file and the current version to see *exactly* what the attacker removed.

S3 Object Versioning is not enough on its own though, because in theory, an attacker could just disable the versioning and overwrite/delete any existing versions that are in the bucket without the worry of a new version being created. To combat this, AWS offers the feature of multi-factor authentication delete in S3 buckets. Having MFA delete enabled forces MFA to be used to do either of the following two things:

1. Change the versioning state of the specified S3 bucket (i.e. disable versioning)
2. Permanently delete an object version

If both versioning and MFA delete are enabled on a bucket, that means an attacker would need to compromise the root user and their MFA device to disable versioning and MFA delete on the bucket. This is possible in theory, but in practice is very unlikely.

Method #5: Bucket Policies and ACLs

It is imperative to not make your buckets accessible by the public. Buckets and the objects in them can be made publicly accessible through a variety of different ways, but most often it is the bucket ACL or bucket policy that is the culprit.

You should avoid using the bucket ACL entirely in most cases, but sometimes it’s not possible due to a few different reasons. It is an old, “managed” way of managing access to your bucket and it does not allow fine-grained access control. If you grant someone access to “List objects” in an ACL, they can get a list of all the

objects in the bucket in a single API call *and* can read any of those files. If you grant someone access to “Write objects” in an ACL, that means they can create, overwrite, *and* delete objects in the bucket. Those two are reason enough to avoid using ACLs.

You should be using bucket policies instead of ACLs because it allows the most fine-grained permissions management. Instead of granting a user list *and* read permissions in the ACL, you could grant them only one of the two permissions in the bucket policy. You can even impose further restrictions, such as read access to only a few objects in the bucket. For example, if you have a website that is reading S3 objects directly from your bucket, it does not need permission to list every object in the bucket, because it should already know what it is fetching. In this case, you could just grant it “s3:GetObject” in the bucket policy instead of list and read in the ACL. Instead of granting create/overwrite *and* delete object permissions through the ACL, you could grant one or the other and impose further restrictions, just like above.

Another feature of S3 bucket policies is that they have the ability to enforce encryption of a specific type. Examples of this are forcing any uploaded file to be encrypted with AES256, or forcing any uploaded file to be encrypted with a specific AWS KMS key. This can be used to prevent S3 ransomware because, ideally, the attacker won’t have access to modify the bucket’s policy.

You could set your bucket policy to only allow objects to be uploaded with your specific KMS key. In that case, the attacker would not be able to use another KMS key that you haven’t specified in the policy, which is necessary to ransomware the bucket. They would then get an access denied error message, ultimately preventing the attack.

Here is an example S3 bucket policy that forces file uploads to use a certain KMS key for encryption:

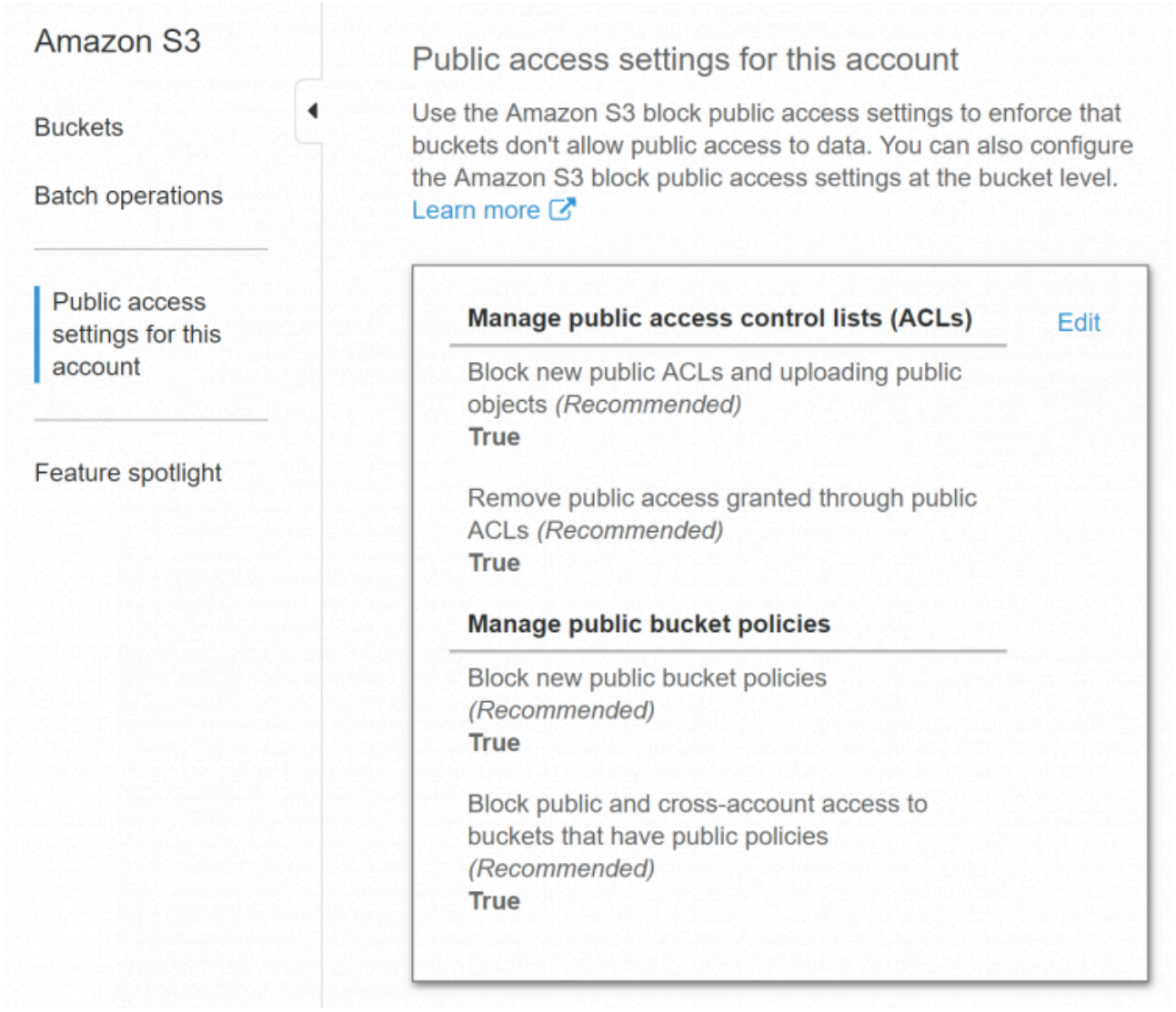
```
{"Version":"2012-10-17","Statement":[{"Effect":"Deny","Principal":"*","Action":"s3:PutObject","Resou
```

More information on S3 actions, resources, and condition keys can be found here.

Method #6: Account-Wide S3 Public Access Settings

If your concern is that your developers might grant public accessibility to your buckets, then you can utilize the account-wide option to block public ACLs and public policies from taking effect. Note that this applies to every bucket in your account and might cause problems if you rely on cross-account/public access to specific buckets in your account.

If you can verify that it is alright to block public access to all your buckets, then you should consider enabling the account-wide S3 public-access settings for your account.



The screenshot above shows the public access settings for an example account in the AWS web console. In this example, all existing, public bucket ACLs and all existing, public bucket policies will be blocked, meaning that they won't actually grant public access because of this higher-tier public access setting. Additionally, new ACLs and policies that grant public access will be blocked as well.

Method #7: Backups

Finally, of course, you *need* to back up your data! Whether that is done with a MFA versioned bucket, data replicated across buckets or accounts, or even locally, it is extremely important. With sufficiently backed up data, you can just ignore the attacker (after your incident response plan has been put into action, of course) and restore your backups, then it was like you were never ransomwared at all.

Recovering from a Successful Ransomware Attack

If you've been targeted with an S3 ransomware attack, how you respond will likely depend on a few different factors. AWS Security is aware of the risk of the attack vector, but it is uncertain what their role could be in helping you—if any. If you can accept the extra time it will take, it would likely be best to contact the authorities,

though the delay involved in that may be too much to accept for your business. For that reason, it is best to implement a strong defense to this attack so that you don't end up in that situation needing to weigh your potential options and their risks.

Incident Response Plan

If you have been targeted, then the first step would be to enact your incident response plan. Doing this will lower the blast radius of the attacker, get them out of the environment, and determine what they have gained access to and attacked. This is one reason why strong logging and monitoring in an AWS environment are both essential. Knowing what they attackers has accessed will help you determine what your next steps need to be.

Script for Checking Bucket Configurations

We wrote a script that checks the important settings on all buckets in an AWS account. This includes checks for object versioning and MFA delete on each bucket. You can find this script on our GitHub. It also has an option to enable object versioning for any buckets that don't have it enabled already.

This screenshot shows some of the example output of running the script against a vulnerable AWS account.

```
PS C:\> py .\s3-ransomware-bucket-check.py -p default
Finding buckets...
Checking configuration of 14 buckets...

Scan complete, successful results output to ./default_ransomware_bucket_scan_1559930832.csv
```

When opening the CSV file where the results were output, you will see something like the following screenshot (S3 bucket names are censored):

	A	B	C	D	E
1	Bucket Name	Object Versioning	MFA Delete	Note	Recommendation
2	[redacted]	Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
3		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
4		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
5		Enabled	Disabled	Bucket is protected against ransomware attacks, but a	Enable MFA delete
6		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
7		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
8		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
9		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
10		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
11		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
12		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
13		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
14		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete
15		Suspended	Disabled	Bucket is VULNERABLE to ransomware attacks	Enable object versioning and MFA delete

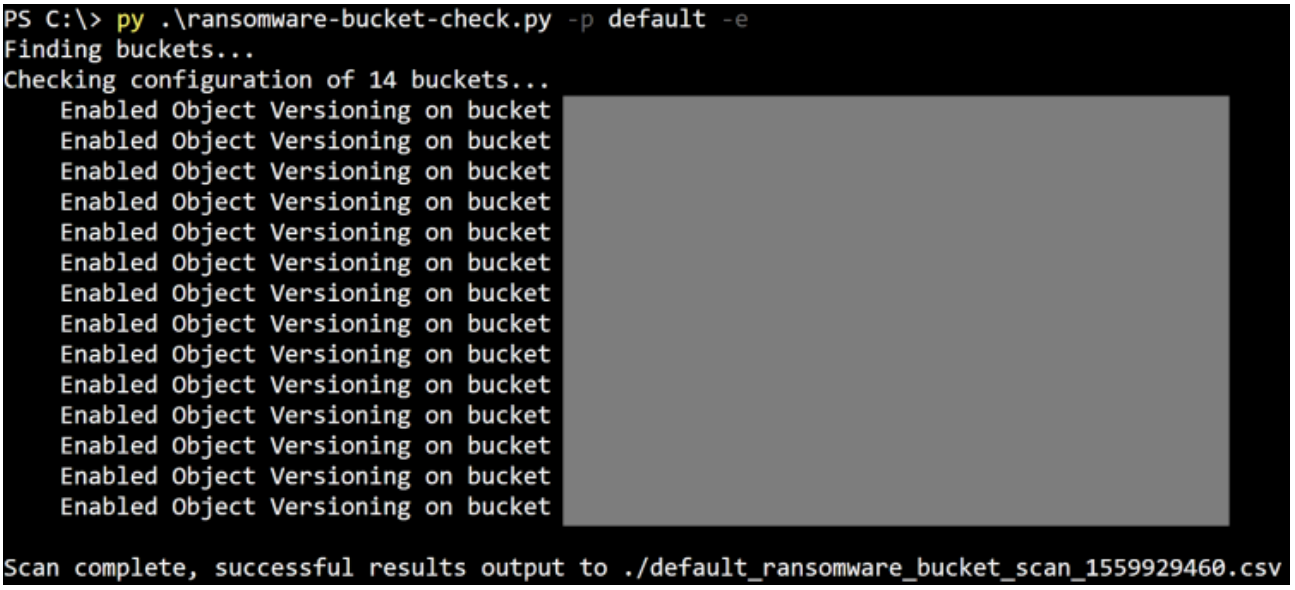
There are a few arguments to be aware of when running this script:

```
-p/--profile: The AWS CLI profile to use for authentication with AWS (~/.aws/credentials).

-b/--buckets: A comma-separated list of S3 buckets to check. These should be owned by you, or at lea

-e/--enable-versioning: If this argument is passed in, the script will attempt to enable object vers
```

The following screenshot shows the usage of the enable versioning argument (S3 bucket names are censored again).



The CSV file will report “Enabled” for object versioning for any bucket that had its versioning successfully enabled. If the script fails to enable versioning (because of something like a permissions error), it will move onto the next bucket and mark the failed bucket with its original versioning setting (“Disabled”/”Suspended”).

Conclusion

S3 ransomware can be fairly straightforward for an attacker to perform, but there are a variety of both easy and difficult defense mechanisms that defenders can put in place. At the lowest level, it is simple for a defender to enable versioning and MFA delete on an S3 bucket, which would effectively prevent ransomware in a majority of cases.

It is extremely important to implement defense mechanisms in your AWS environment and sensitive S3 buckets. While it might not be necessary to execute every method outlined above, it is imperative to determine which attack vectors and entry points you are most susceptible to, and therefore need to protect against.

While the above outlined methods will be very useful in helping prevent against S3 ransomware, there are always more methods of defense for prevention/detection out there. We encourage any readers to contact us with other ideas so we can add them to this post (credited to you) and share them with other people who are trying to defend their own environment.
