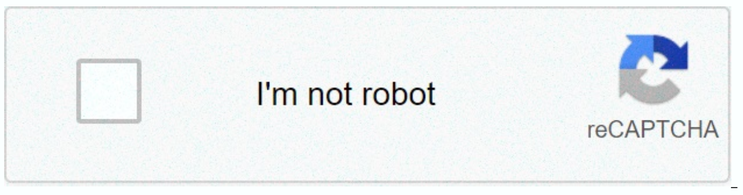


Continue

# Aws pentesting cheat sheet

In our last AWS penetration testing post, we explored what pentesters can do after compromising cloud server credentials. In this installment, we'll look at Amazon Web Service (AWS) instances from situations without credentials and in particular, potential security vulnerabilities in the AWS S3 Simple Storage bucket. As we walk through the AWS S3 methodology, we'll deploy it to Alexa's top 10,000 sites to identify the most popular AWS users for this vulnerability. Any sites with open S3 permissions have been contacted by Rhino Security Labs in advance for repairs. Amazon Simple Storage Service (S3) is an AWS service for users to store data in a secure way. S3 Bucket permissions are secure by default, which means that once created, only bucket owners and objects have access to resources on S3 servers as described in the S3 FAQ. You can add additional access controls to your bucket by using Identity and Access Management (IAM) policies, making S3 a powerful resource for restricting authorized users. Getting new users and bergi through security policies can be tricky and time consuming, causing less experienced companies to open permissions they don't want. IAM can draw many parallels with Windows group policy, granting very specific access permissions and controls to groups and their users. Like Windows policy errors, lightweight identity management on S3 buckets can vary in impact - from small information leaks to full data breaches. Some sites, for example, use S3 as a platform to serve assets such as images and Javascript. Others drive full server backups to the cloud. As with any security vulnerability, the risk is not only the presence of problems, but the context as well. As a penetration tester, a quick check for the presence and configuration of the S3 bucket can be an easy win and always worth checking out. Many AWS applications are not configured behind a web application firewall (WAF), making it easy to identify server regions using nslookup. If the server is behind WAF, you may need an alternative method to determine the target IP address. Using nslookup from flaws.cloud IP address reveals it is located in us-west-2. The entire CTF flAWS, which includes AWS security can be found here: Once the region is specified, you can then start the common query and enumerate the bucket name. It's not actually necessary to define a region in advance, but it will save you time later to querie AWS. We recommend a combination of subdomains, domains, and top-level domains to determine if your target has buckets in S3. For example, if we search for your S3 bucket www.rhinosecuritylabs.com, we can try the bucket name rhinosecuritylabs.com, and To determine the validity of the bucket name, we can navigate to the automatically assigned S3 URL provided by Amazon, which is the format use the following command line: sudo aws s3 ls s3://$bucketname/ --region $region If the command returns a directory listing, you can successfully find a bucket with unlimited access permissions. Image of S3 flaws.cloud directory list in the browser. Each S3 bucket is provided with a URL by Amazon from the bucketname.s3.amazonaws.com Some S3 buckets are used to host static assets, such as images and Javascript libraries. However, even for these buckets with less sensitive assets, an open upload policy can allow attackers to upload custom Javascript libraries, allowing them to serve malicious Javascript (such as BeEF Hook) to all users of the app. Of course, many of the more sensitive uses for the S3 exist. In our research, we found buckets that allow downloads of system backups, source code, and more - an important risk for leaking companies. Even log files available for download can reveal usernames, passwords, database queries, and more. Domains in alexa top 10,000 store g-zip log files and are stored on S3 servers. More log files from alexa top 10,000. In this case, the production API log is stored without authentication. The site's .git subdirectory, stored in Amazon S3, indicates extraction of source code is possible. All server backups are stored in S3 for Alexa Top 10,000 sites. Using the Alexa Top 10,000 site, specify which sites have S3 buckets that allow directory lists, downloads and uploads to be enabled. This is done programmatically by searching each region using the AWS CLI, issuing commands: aws s3 ls s3://$bucketname/ --region $region $bucketname will only be the name of the domain being tested, as well as any subdomains we previously found (foo.com, www.foo.com, bar.foo.com, etc.). If a bucket exists and no permission errors are lifted, this indicates open list permissions. The download command was successfully specified by retrieving a nonzero size file and copying it locally to disk. If the file is successfully transferred, it shows open download permissions. Successful uploads are shown (by unknown others) through a file called testupload.txt. This file appeared on several S3 buckets tested, indicating other users performed similar enumeration tests in November 2016. Similarly, this information is provided to companies that are found to be vulnerable. The testupload .txt uploaded to several open S3 servers, shows someone had tested upload permissions before Rhino Security Labs conducted this research. Of the 10,000 sites audited, 107 buckets (1.07%) found with list permissions belonging to 68 unique domains. These buckets were found hosted in 10 of the 11 regions, with fips-gov-us-west-1 (36% buckets) and US-east-1 (37% buckets) being the most popular. This bucket contains everything from nothing at all, to git repository, up to the company's Redis server backup. Of these 107 buckets, 61 (57%) have open download permissions to anyone viewing viewing 13 (12%) have open upload permissions, and 8% have all three. The amount of security controls AWS provides customers to manage their assets is staggering. Although the structure is locked by default, a large number of companies are still loosening their S3 settings, allowing unauthorized access to their data. In this study, sensitive company data, backup log files, and other data were identified without the need for authentication. If you are using S3, proper IAM control is essential to prevent the type of information leakage shown here. Applying access logging can also help identify when/where your credentials are used to access your resources. >> Worried about this security vulnerability in your environment, or want more information about the risks highlighted above? Contact. >> Also, download Buckethead from or Github. Buckethead is a tool developed by Rhino Security researchers to more easily find AWS buckets. Introduction When testing penetration of the Amazon Web Services (AWS) environment there are different perspectives assessments can consider, some very similar to external infrastructure/web application assessment and some different. I will separate the same things from the different things from traditional penetration testing taking into account the following types of cloud testing and then breaking each into types of tests that can take place: Testing in the Cloud: testing traditional systems hosted only in a cloud environment. For example, this could be a virtual system that has been moved from on-premises to the cloud (e.g. lift and shift) or it could be a web application hosted in the cloud where only the application itself is considered in scope for assessment and not supporting infrastructure. Testing in the Cloud: an in-cloud testing system that is not publicly exposed, this for example can test application hosting servers or for example, a testing system hosted in the cloud but has a firewall that prevents direct access and is otherwise accessed via a bastion host (e.g. a private VPC). In addition we will consider the risks associated with compromised applications that allow attackers to access backend infrastructure. Testing the Cloud Console: testing the configuration of the cloud console (sometimes referred to as a portal) itself, such as viewing user accounts that have been set up, their permissions,

preconfigured access control lists, etc. This is effectively a configuration review and can be driven by compliance – however there are still a few things in this category to consider as penetration testers if access to the cloud console is gained during penetration tests. It's also most likely an efficient way to determine the potential path of privilege escalation. I'll finish with a little AWS security glossary that will act as a small introduction to the vast world of AWS terminology and demonstrate some built-in features that surely deserve you to realize they exist. Don't forget that you'll need permission from Amazon as well as companies that have applications to perform test activities. Penetration Testing of Amazon-hosted services has clearly been done before, therefore Amazon has a well-documented process for notifying them when activity testing will take place, which systems you authorize to conduct penetration testing against, and appropriate terms of service and rules of engagement. More information here: AWS only supports penetration testing of a small number of its services, such as EC2, RDS, CloudFront, Lamba, &c. However, that does not mean that security testing cannot be performed against other systems, it simply means that active penetration testing and vulnerability testing type activities cannot be performed. That will limit you to console configuration reviews. Terms and conditions apply; I'm not a lawyer; read small print; battery is not included. Testing on Cloud Test systems and applications that are only picked up and shifted to the cloud will likely be no different from testing on-premises hosted applications. The difference will start coming in when cloud installation maturities increase until AWS-specific functionality is included, such as S3 buckets. S3 Bucket Troubles S3 stands for Simple Storage Service. It is designed as a web service to store and retrieve a certain amount of data from anywhere on the web. In short, highly scalable, reliable, fast, inexpensive data storage. By default, access to them is fairly limited, but there are several ways to get a little messy and potentially allow attackers to access or store arbitrary data in your bucket. The S3 bucket is in a global name space, so it must be given a unique name. S3 buckets can be used for arbitrary data storage but many people use them for website hosting and web content. Being able to count S3 buckets is just a fact of how it works, but being able to list their content, see all the content, and write to them is a configurable option. So the problem around S3 is effectively: The world's readable bucket Buckets are readable Worlds. Access to S3 buckets can be controlled in several ways: IAM policy, S3 bucket policy, and S3 ACL. S3 ACL is a legacy. There is Amazon's documentation of their protection here: . World-readable S3 buckets can be accessed such as: aws s3 ls s3://bucket.name If you haven't used awscli before you can set it up free tier accounts on AWS ( and then set up user accounts Identity and Access Management (IAM) is here: If you set up a user account under a user ( then you can view or create accessKey and secretKey here: sudo easy_install awscli aws configure AWS Access Key ID [None]: Your-AWS-Access AWS Secret Access Key [None]: Your-AWS-SecretKey Default region name [None]: us-west-2 Default output format [None]: The S3 World-readable bucket can actually refer to two issues, the first being the S3 bucket with read permissions granted to Everyone which in fact means everyone on the internet, not everyone in your organization. Additionally, granting read permissions to All AWS Users means everyone with an AWS account, not everyone on your user list. Finally, you can't restrict users from listing specific buckets. They can list them all or they can't. Keys that can list a single bucket can list all buckets for that account. So make sure their names don't contain anything secret. It is also possible for anonymous users to calculate that a bucket exists remotely by guessing its name. Not the biggest vulnerability to be aware of but it should be noted that it exists. If you want to see practical examples of this kind of problem then check out flaws.cloud! There are examples of this happening in the wild as well: Losing Your Key Access to AWS services is done by keys (accessKey and SecretKey). Permission is very detailed. Of course, the permissions you get to AWS differ based on the key you use, as you would expect. One vulnerability to consider is losing your key. If any AWS Keys (accessKey and secretKey) are disclosed then bad things can happen – attackers will obviously get all the privileges that the key offers. So I'm going through several ways in which AWS keys can be compromised, some quite clear and some using cool AWS-specific functionality. Starting from the beginning... Inadvertently do them If you work with versioning systems like GIT there is the potential that the key will commit to the repository and it's not as simple as simply releasing them, or overwriting them with another commit. I won't explain how to remove a key from a commit because my recommendation is: If you've accidentally revealed the key, revoke it and issue a new key. If the key is disclosed to a party that is now deemed unworthy of trust, revoke and issue a new key. If you write it in a post-it note, revoke it and issue a new key. It might look like for some of you? This is the case: 10,000 GitHub users accidentally revealed their secret AWS access key Extracting keys from EC2 EC2 instances is computing services as part of AWS that are similar to virtual private servers. While an attacker is unlikely to have raw access (e.g. console access) to an EC2 instance there is a possibility that the attacker may steal console access and then use that access to steal an AWS key. Most organizations will realize that if an instance (or vps) is compromised to this level the significant impact has been caused by the attacker and will likely perform some actions such as leveling the instance and rebuilding it to prevent a backdoor, etc., from being used – however you should also consider that the key may have been compromised. This could work, for example, in the case of web applications where remote code execution vulnerabilities have been caused by vulnerabilities. Attackers can access AWS Meta Services to pull keys. The Instance Metadata Service has documentation here: In short, an attacker, from a compromised instance, can access: that will return the role and: will return the key. More information is available here: Yep, which also happens: Abusing AWS metadata services using SSRF vulnerability testing in the Cloud By testing in the cloud I mean effectively having a system perspective running within an Amazon Virtual Private Cloud (VPC) or equivalent. Something similar to if Penetration Tester walks into the office and connects the laptop to an available network port and starts scanning the machine, or if the attacker to harm the client's device with a phishing attack and starts trying to trigger it into the company's network. In the context of a cloud environment an attack could potentially occur if a cloud server, or EC2 instance, is compromised by an attacker – how they will attempt to attack additional machines within the VPC. As with any network environment security testing there are several different approaches. One of the simplest approaches is to perform vulnerability scanning in VPC, which can be achieved by applying vulnerability scanning tools, this can be obtained through Amazon Machine Images (AMI), where there are scanners available from Tenable (Nessus) and Qualys, for example. Alternatively, Penetration Testers can be granted secure access to instances within a VPC, or interconnected to a VPC, to allow them to access the network for their testing. Another approach is to set up VPN tunnels into AWS environments to enable penetration testing activities in a more direct environment. The number of instances, the complexity of the environment, and the desired level of warranty are likely to which is the best best Testing the Cloud Console Finally, by testing the cloud itself, I am referring to connecting to the AWS Console to see the security of the console configuration. Really this is a configuration review type activity but it's good to take a quick look here, as the attacker potentially compromises the account with access and provides an introduction to see some of the existing vulnerabilities and security features that Amazon provides. The first thing to say, with respect to attackers who compromise accounts with access to the AWS Console is that accounts can be configured to require multifactor authentication (MFA; sometimes referred to as two-factor authentication or 2FA). There are quite a lot of MFA options for AWS, it can also be used to control access to the console and access to AWS APIs. More details are available here: the AWS Shadow Admins allows detailed configuration of Identity and Access Management (IAM). You can restrict permissions that users must restrict to a fairly good level, but there are certain permissions that could potentially allow for a greater level of access than you might realize. There are several writings and detailed tools available, so I would recommend reviewing them as well if this concerns you: CyberArk writes: Tool: . Rhino Security Labs' Write up: Tool: In short, there are many permission sets that given certain restrictions can allow the escalation of privileges in an AWS environment, CyberArk provides 10 of these examples and RhinoLabs adds a few more. A simple example you might encounter is where users can modify the version of the policy that applies to their account by having the permission iam:SetDefaultPolicyVersion. If you consider a situation where the administrator is setting up an account and then after everything is working harden that account, it would not be a stretch to expect the administrator to grant more permissions than was necessary at first and then set that account permissions down to follow the principle of rental privileges. If an attacker's accessible policy has a higher permission level but they have access to set the default policy version, they can easily take a chance on which version to use. This kind of issue is obviously very contextual for the specific settings you have and RhinoLabs gives you 17 possible privilege escalation methods – so it's certainly worth reviewing! Finally, there are tools to help audit the configuration of your AWS Console, such as Scout2: AWS Security Features AWS has a lot of security features, and I found it found it when I talk to people who use AWS who don't know the various monitoring and security features built. There are alternatives to them all of course, and they are not all free – but at least need to check the basics to see what your options are! So here are some examples/ AWS Inspector – – is an automated security assessment service for applications deployed within AWS. If you want to give it a try, there's a 90-day trial of up to 250 agent ratings. IT found problems such as insecure protocols, software running without DEP, software without cookie stacks, root processes – but requiring agents to be installed for use. AWS CloudWatch – – is a system monitoring service. It has basic or detailed monitoring, where the latter have a cost. It monitors the system and watches metrics such as EBS read/write latency, storage availability in RDS, EC2 instance CPU utilization. AWS CloudTrail – – is a system that lets you log what's happening in your AWS environment, including actions taken in consoles, command-line tools, and services. This is useful not only for security purposes but also things like monitoring changes to the utilization of your resources over time, for example. AWS Athena – – is a service that allows you to request data stored in S3 buckets using SQL. So it's great to pair with CloudTrail to effectively search for your log dumps in a more effective, or complex way, so that's what CloudTrail offers itself. AWS TrustedAdvisor – – is a system that provides recommendations in several categories, but is certainly tiered so that there is a core and full depending on whether you are using a standard, business, or enterprise plan (the latter two get full access). It provides recommendations on how to improve your environment in categories such as: security, cost optimisation, performance, and fault tolerance. Well they actually expose all the checks that provide every level and category ( , so for example under security we have things like: security groups with unlimited access, unlimited access to the system to specific network ports, missing MFA in root accounts, IAM password policies, etc. AWS Artifact – – providing access to compliance reports for requirements such as ISO 9001, ISO27001, PCI DSS, and more. AWS Shield – – managed DDOS protection, which provides always protection for DDOS attacks by and then automatically puts the mitigation in the channel on the system. There is also AWS Shield Advanced for a higher level of protection. One thing to look at for sure is Cost Protection which returns service credits for scaling fees due to DDOS attacks. Related Related Introduction to Cloud Computing: Many Clouds in the Sky Introduction to Azure Azure PenTesting