ACCOUNT JUMPING POST INFECTION PERSISTENCY & LATERAL MOVEMENT IN AWS

Dan Amiga (dan@fire.glass) , Dor Knafo (dor@fire.glass)

Abstract— **The widespread adoption of AWS as an enterprise platform for storage, computing and services makes it a lucrative opportunity for the development of AWS focused APTs. We will cover pre-infection, post-infection and advanced persistency techniques on AWS that allows an attacker to access staging and production environments, as well as read and write data, and even reverse its way from the cloud to the the corporate datacenter.**

I. Introduction

This paper will cover several methods of infection including a new concept called "account jumping" for taking over both PaaS (e.g. ElasticBeans) and IaaS (EC2, EC2 Containers) resources, poisoned AMIs, dirty account transfer, and leveraging S3 and CloudFront for performing AWS specific credentials thefts that can easily lead to full account access.

In addition, we will focus on the post-infection phase, and look at how attackers can manipulate AWS resources (public endpoints like EC2 IPS, Elastic IPS, load balancers and more) for complete MITM attacks on services.

We will further describe how an attacker's code can be well-hidden via Lambda functions, cross zone replication configurations, and the problem with storage affinity to a specific account.

We will also examine hybrid deployments from the cloud and compromising the on-premise datacenter by leveraging and modifying connectivity methods (HW/SW VPN, Direct connect or cloud hub).

Finally, we will suggest best practices that can be used to: protect against attacks such as bastion SSH/RDP gateways; understand the value of CASB based solutions and where they fit; leverage audit and HSM capabilities in AWS; and utilize different isolation approaches to create isolation between administrators and the cloud, while still providing access to critical services.

II. Infection

Amazon Web Services (AWS) provides an easy-to-manage cloud platform to store digital assets, host servers and more. Amazon also has many settings for security controls, including a firewall to block incoming and outgoing traffic, and different identity and access management (IAM) accounts with varying levels of privileges. However, mis-configurations or bad management of credentials can allow an attacker to gain access into the cloud, and exfiltrate both company and consumer data.

A. The AWS Infection Potential

One of the strong usability points in AWS is that every action has a corresponding public API endpoint. In fact, the API is even larger in scope than the AWS Management Console itself. Every API call made to AWS is authenticated via an access key and a secret key, and then authorized via the corresponding permissions that were assigned in IAM (AWS's identity management service).

An attacker must first move into an organization's AWS account to gain access to credentials (e.g. user, password & MFA device if activated), or to an access token. Once access has been established, the attacker then attempts to perform elevation of privileges (if needed) to escape Cloud Trail's watchful eye, which is the auditing service in AWS (if activated).

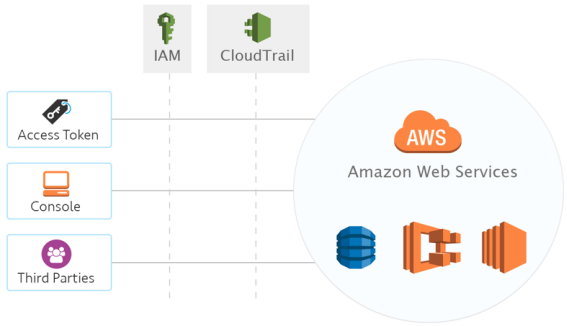

figure 1: AWS infection potential.

B. AWS Third-parties

It is imperative to analyze the trust relationship between external services in use. Whenever choosing a service to support an AWS environment, organizations must understand the risks by doing so, and take steps to protect assets.

1) Source Repositories

Source code repositories such as Github and Bitbucket are well-known targets for crawlers seeking secret keys. There are many well-documented cases online that highlight user accounts being compromised by such a leak of credentials.

2) Cloud Monitoring Services

Cloud monitoring services such as datadog receive unfiltered information directly from the AWS environment. In some cases, this data may contain secrets that will be stored in

other services that are probably less secure than AWS. One scenario is using a third party to analyze AWS CloudTrails. AWS CloudTrail is a web service that records API calls for an account and delivers corresponding log files. For example, when using with Elastic Container Service (ECS), task definitions that pass to the container will be logged. Task definitions contain environment variables, and while it is not recommended by Amazon to store secrets in task definitions, it is common to store credentials that can be leaked less secure services, and therefore increase the odds of being stolen by an attacker. Recently, Datadog, which is a widely used cloud monitor service, has been hacked. As such, the credentials used to integrate with this service, and the credentials that have been logged by CloudTrail, are in danger.

3) Phishing and Man in the Browser

Traditional email phishing attacks are also associated with AWS. There are many techniques to enhance the reputation of a source in order to make a phishing attack more effective. One example is to host a phishing website on AWS S3 and use the static serving option. In this case, the malicious person exploits both S3 domain and Amazon favicon to trick victims into opening an attachment, clicking on a link, or divulging confidential information.

Another popular phishing method is to send phony AWS emails to admins, such as the famous E2 [Retirement Notification]: "Amazon EC2 Instance scheduled for retirement" or the common "Amazon Web Services Invoice Available". Already-infected administrator endpoints, such as those with man in the browser attacks (e.g. browser extensions), can be used to gain access to and leak credentials or access keys.

4) Metadata Server

All EC2 instances have meta-data, such as the used AMI, kernel and region. The instance metadata is available from the running instance. To view all categories of instance metadata from within a running instance, use the following URI *http://169.254.169.254/latest/meta-data/*

In many cases, an EC2 instance is used as a proxy, and can be manipulated to send a request to the instance metadata. An attacker can use the information stored in the instance metadata to gain access to the AWS environment. For example, the following categories are stored in the instance metadata and can lead to credentials theft:

- Local IP Address
- User-data
- Instance profile
- AMI

It is also possible that the AMI the server is using can be compromised to gain access to EC2 instance. A more elegant and reasonable scenario would be using the access to the instance metadata in order to get instance profile information. Instance profiles are defined by the AWS architect, who determines which permissions will be available to the EC2 instance using the profile. In this case, it is possible to steal the credentials since the access tokens are stored in the instance metadata.

5) Poisoned AMI

All AMIs should be carefully examined before being launched within an account. Any AWS account administrator can build an Amazon Machine Image (AMI) and expose it to other accounts. This opens the door to several infection points. If by mistake (or phishing, or any other form of social engineering) a malicious AMI is launched without an enterprise account, that AMI can then run well-hidden malicious code. In addition, code within an instance of an AMI in an enterprise account can be granted (by an innocent admin) permissions via instance profiles. Also, many admins only care that AMIs "just work", and do not focus on the fact that AMIs should be run through vulnerability scanning.

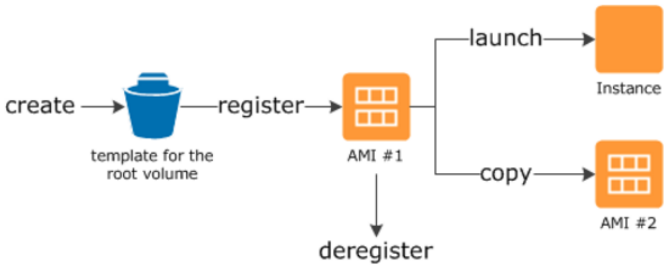

figure 2: Public AMI process.

6) Account leftovers & "Account Jumping"

In light of the constant need to save money, organizations (startups especially) often purchase an AWS account from third parties for a discount. A good example arethose that give free credits through startup accelerators and seed programs. However, often these startups tend to close shop. They are then left with accounts that have tens of thousands of USDs in AWS credits, which they put on for sale for a very large discount in some form of black market. A prospective buyer would need to iterate through all the resources in all the regions and detect any leftovers -- this is not a trivial task! Since there is no 'clean account' option, and some resources would even stay undetected in terms of billing (e.g. resources that fall into the free tier), there is a large potential for resources to stay active. Good examples are free tier lambda functions and SNS notifications that can be used to leak data, and/or access keys to the outside world.

III. Post Infection

Once an account has been compromised, the next obvious challenges are how to stay undetected for as long as possible, how to silently research the environment, and how to create as many persistency hooks as possible.

A. Staying Undetected

AWS has done a great job creating multiple logging and auditing mechanisms. Primarily:

1. AWS CloudTrail is a web service that records API calls for am account and delivers corresponding log files.
2. AWS Config, which is an integrated AWS service that enables automatic enforcement and verification of AWS resource modifications.
3. AWS Cloud watch, which is an integrated monitoring

service for AWS that enables organizations to collect, monitor, set alarms, and automatically react to changes in AWS resources.

We are suggesting various techniques that enable an attacker to stay under the radar of the above security mechanisms, and alter resources, query data and inject new persistency hooks into the environment.

B. *AWS CloudTrail*

CloudTrail should always be activated on a global scope. Otherwise, admins have no visibility to actions undergoing inside the account. Since it's a paid service and due to the fact that it is **not turned on by default,** there is a likelihood that certain accounts will not have the service configured. However, for the purpose of this paper, we assume a more advanced and security focused admin is managing the account, and that CloudTrail is activated. Here is how an attacker can alter the normal behavior of a CloudTrail enabled account:

The first option is to delete or stop the CloudTrail configuration by using the API Calls:

```
$ aws cloudtrail delete-trail --name <trail>
$ aws cloudtrail stop-logging --name <trail>
```

These methods are loud, and it reasonable that an admin will be actively monitoring them. On the other hand, CloudTrail will stop logging API calls immediately, and the admin will need to determine what the attacker did with the stolen credentials.

CloudTrail by default applies to all regions. As such, it creates the same trail in each region, and delivers log files for all regions. An attacker can disable this setting so that only the home region (i.e. the region where the CloudTrail was created) will continue logging:

```
$ aws cloudtrail update-trail --name <trail> --no-is-multi-
    region-trail --no-include-global-service-events
```

Notice that the flag "no-include-global-service-events" will disable the logging from a global service such as IAM and AWS STS.

CloudTrail typically delivers log file within 15 minutes of an API call. In addition, the service publishes log files multiple times an hour; usually about every five minutes. These log files contain API calls from all of the account's services that support CloudTrail.

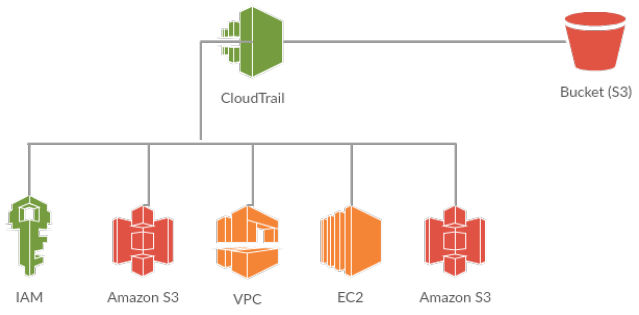

figure 3: AWS CloudTrail high-level structure.

To make the life a bit harder for the admin, an attacker can update the S3 bucket lifecycle configuration to delete the files after one day only. In this case, we did not prevent logging completely, but it is stealthier than the previous options.

Another way to manipulate the S3 bucket is to use event-driven compute service, where AWS Lambda runs code in response to events, such as changes in data in an Amazon S3 bucket or DynamoDB table. A good option is to set up a lambda to immediately delete every log file written to this S3 bucket. Lambda function is invoked directly by S3, and it will win any race against other code attempting to consume files written to the bucket, making them invisible.

Lambda free tier includes 1 million requests per month and 400,000 GB seconds of compute time per month. Using this lambda function to hide trails for a whole month will require 43800 / 5 = 8760 requests, 0.00876% of the free tier, and has no effect on the monthly bill.

The AWS Key Management Service (KMS) is a managed service that makes it easy to create and control encryption keys used to encrypt data. Encrypting CloudTrail log files with KMS builds on the S3 feature called server-side encryption (SSE). When configuring CloudTrail to use SSE-KMS to encrypt log files, CloudTrail and S3 use KMS's customer master key (CMK) when certain action are performed with those services.

Each time CloudTrail puts a log file into the S3 bucket, S3 sends a *"GeneratDataKey"* request to KMS on behalf of CloudTrail. In response to this request, KMS generates a unique data key, and then sends S3 two copies of the data key one in plaintext and one that is encrypted with the specific CMK. S3 uses the plaintext data key to encrypt the CloudTrail log file. S3 stores the encrypted data key as metadata with the encrypted CloudTrail log file.

Each time an organization receives an encrypted CloudTrail log file from the S3 bucket, S3 sends a *"Decypt"* request to KMS on behalf to decrypt the log file's encrypted data key. In response to this request KMS uses the CMK to decrypt the data key, and then sends the plaintext key to S3 that uses the key to decrypt the CloudTrail log.

Here is how all fits together:

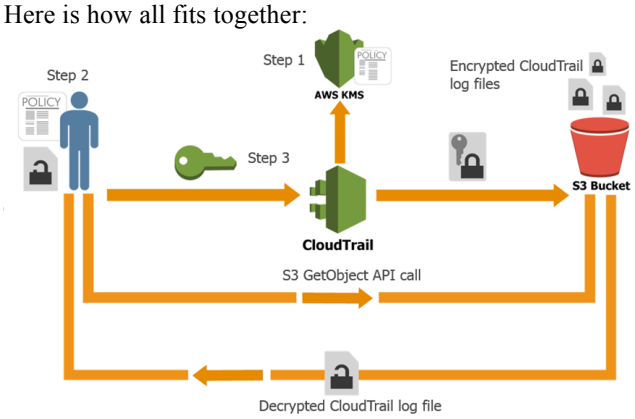

figure 4: AWS KMS integration with AWS CloudTrail.

An attacker can leverage this knowledge to enable encryption of the log file using a KMS with a policy that does not allow

decryption for all users, but allows to *"GeneratDataKey"* only to CloudTrail, so that CloudTrail will accept the key's policy:

$ aws kms create-key --policy <kmspolicy> --bypass-policy-lookout-safety-check

The *"bypass-policy-lookout-safety-check"* flag will prevent the principal making the request from making a subsequent put-key-policy request to CMK, so that the CMK becomes unmanageable. Encrypt the trails using the newly created key:

$ aws cloudtrail update-trail --name <trail> --kms-key-id <key-id>

Disable and schedule the key for deletion so that S3 will fail to call *"GenerateDataKey"* for this key and as a result CloudTrail will be failed to log:

$ aws kms disable-key --key-id <key-id>
$ aws kms schedule-key-deletion --key-id <key-id> --pending-window-in-days 7

The key deletion cannot happen immediately, but the trails will not be written regardless. Inspecting the trails in the AWS web interface will not show any sign of failure, either. However, checking the trail status via cli will show "LastDeliveryError" as "KMS.DisableException".

$ aws cloudtrail get-trail-status --name <trail>.

IV. Persistency

After the attacker gains control of the AWS environment, one course of action is to plant backdoors to get the access back, even if the admin must delete the stolen access token.

1) Security Token Service

The AWS Security Token Service (STS) is a web service that enables organizations to request temporary, limited-privilege credentials for AWS Identity and Access Management (IAM) users, or for users that have been authenticated (federated users).

When the access token that used to create the session token is deleted, the session token continues working until its expiration time. Acceptable durations for IAM user sessions range from 900 seconds (15 minutes) to 129600 seconds (36 hours), with 43200 seconds (12 hours) as the default. Sessions for AWS account owners are restricted to a maximum of 3600 seconds (one hour). If the duration is longer than one hour, the session for an AWS account owner defaults to one hour. The following will generate a session token that will last for 36 hours:

$ aws sts get-session-token --duration-seconds 129600

There are some restrictions for using a session token. For example, it is not possible to call any IAM APIs unless MFA authentication information is included in the request, and it is not possible to call any STS API except AssumeRole. These are easy to bypass if the original token has "iam:passRole" permissions, to spin up a compute with iam permissions.

2) Virtual "Private" Cloud -

Backdoor a VPC can be done using a publicly accessible endpoint to the internet and a Lambda function. A public AWS endpoint can be a SQS or AWS Gateway API used to receive commands from the internet. The public endpoint can integrate with Lambda function that resides inside a VPC. Although the VPC that we would like to backdoor has a security group that denies all inbound and outbound traffic, using the public endpoint to receive commands triggers the lambda function from inside the VPC.

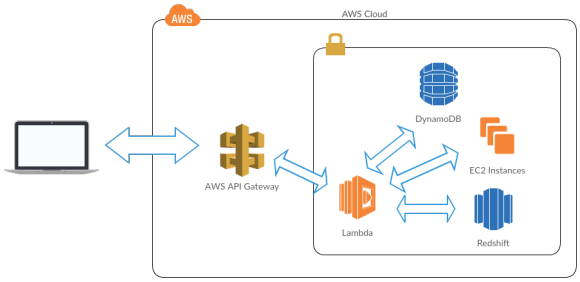

figure 5: Example of VPC backdoor architecture.

In this case, the SQS receives commands from the internet, an AWS Lambda that polls from the queue, and then sends to the lambda inside the VPC the incoming command. When the lambda inside the VPC gets the command, it executes it inside the VPC.

V. Conclusion

The widespread adoption of AWS as an enterprise platform for storage, computing and services makes it a lucrative opportunity for the development of AWS focused APTs. This paper covered how an AWS environment can be infected, and discussed post-infection first steps, and advance persistency techniques that can allow an attacker to access staging and production, remain undetected for years, and even backdoor a VPC.

References

Aslam, Bilal. (2016). *Our Nightmare on Amazon ECS.* Retrieved from http://www.appuri.com/blog/-our-docker-nightmare-on-amazon-ecs

Amazon Web Services, Inc. (2016). *Instance Metadata and User Data.* Retrieved from http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

Amazon Web Services, Inc. (2016). Using API Gateway with VPC endpoints via AWS Lambda. Retrieved from https://aws.amazon.com/blogs/compute/using-api-gateway-with-vpc-endpoints-via-aws-lambda

Amazon Web Services, Inc. (2016). *Using Instance Profiles*. Retrieved from http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html

Amazon Web Services, Inc. (2016). *Task Definition Parameters*. Retrieved from http://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definition_parameters.html

Balduzzi, M., Zaddach, J., Balzarotti, D., Kirda, E., and Louiero, S. (2012). A Security Analysis of Amazon's Elastic Compute Cloud Service. *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*.

Barron, C., Yu, H., and Zhan, J. (2013). Cloud Computing Security Case Studies and Research. *Proceedings of the World Congress on Engineering 2013*. Vol II, WCE 2013, July 3 - 5, 2013.

Becherer, Andrew (2016). *2016-07-08 Security Notice.* Retrieved from https://www.datadoghq.com/blog/2016-07-08-security-notice

Grzelak, D. (2016). *Backdooring an AWS Account.* Retrieved from https://danielgrzelak.com/backdooring-an-aws-account-da007d36f8f9

Grzelak, D. (2016). *Exploring an AWS Account Post-Compromise.* Retrieved from https://danielgrzelak.com/exploring-an-aws-account-after-pwning-it-ff629c2aae39

Grzelak, D. (2016). *Disrupting AWS Logging.* Retrieved from https://danielgrzelak.com/disrupting-aws-logging-a42e437d6594

Padhy, R.P., Patra, M. R., and Satapathy, S.C. (2011). Cloud Computing: Security Issues and Research Challenges. *IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS)*. Vol. 1, No. 2, December 2011.

Riancho, A and Shah, J. (2014). Amazon AWS Services' Security Basics – Escalating Privileges from EC2. *Black Hat Webcast.* Webcast retrieved from https://www.blackhat.com/html/webcast/11202014-amazon-web-services-security-basics-escalating-privileges-from-ec2.html