UNIVERSITY OF AMSTERDAM

Profiling the abuse of exposed secrets in public repositories

Maurice Mouw
mmouw@os3.nl
University of Amsterdam

February 2020

Abstract

The goal of this research was to gain insight into how secrets are abused after they have been leaked into public repositories on several code collaboration platforms. To do the research the git project CloMo was created, consisting of infrastructure-as-code scripts that automatically provision a monitoring system and provide automation to setup an AWS environment. Using the automation provided by CloMo, honey tokens were intentionally leaked to GitHub, Bitbucket, GitLab and SourceForge. The events generated during the experiments were analyzed and classified using the monitoring platform and MITRE ATT&CK framework. Tokens were mainly found on Github depending on the method of leaking. Of the tokens used all attempts started with a discovery technique, for example *DescribeInstances*. In case the initial attempt did not return an error several attacks initiated automation attempting to modify the cloud compute infrastructure by creating large (expensive) Elastic Compute instances. The likely motivation behind setting up these instances is to mine crypto currency.

1 Introduction

In 2013 GitHub shutdown its search function because developers started posting links to sensitive files on twitter [4]. In 2016 an Uber breach in which 57 million customer records were exposed, was due to an access key being accessible via GitHub [16]. A Starbucks API key for JumpCloud was discovered by a researcher in a public GitHub repository [10].

Code collaboration and code sharing have become a common practice today. Both software developers and system administrators/engineers use code collaboration platforms like GitHub, Gitlab, Sourceforge or Bitbucket to share and collaborate on code. A common problem with code collaboration is that unintentionally or even worse intentionally passwords or (secret) keys are stored in a public code repository.

Meli et al. [13] did a large scale analysis of secret leaking on GitHub, they found that over a 100.000 repositories contained secrets and thousands of new, unique secrets are leaked every day. Many tools have been developed for scanning and finding secrets in repositories for example publicly available tools like Git-secrets, Shhgit or commercial variants like GitGuardian or TruffleHog.

The use of code collaboration and cloud platforms in which secret keys are (un)intentionally stored seems to be a given today. While many tools exists to find secrets it still happens (frequently) that secrets are published as part of a public repository. This brings some interesting questions, what happens with these keys after they are published? How fast are keys/secrets found by malicious users? How are these keys/secrets exploited after they have been found and can the usage of the abusers be profiled?

2 Research question(s)

How are leaked credentials/secrets on code collaboration platforms like GitHub, GitLab, SourceForge and Bitbucket found and abused by malicious users in a cloud platforms like Amazon Web Services?

2.1 Sub-research questions

1. **How do malicious users find secrets in public code collaboration repositories?**

2. **How can the use of leaked credentials/secrets be profiled?**

3. **How do malicious users (ab)use found credentials in code collaboration repositories?**

4. **How can the adverse affects/abuse of compromised keys be limited?**

3 Related work

As mentioned in section [1] Meli et al. [13] did a large scale systematic study in which they used GitHub's search API and GitHub's BigQuery snapshot to extract and validate hundreds of thousands of potential secrets in a nearly six month scan of real-time public GitHub commits and a public snapshot that covered 13% of the open-source repositories. They used regular expressions to find potential secrets and a set of filters to validate found secrets. The researchers estimate that 93.74% of the discovered API secrets are sensitive and 76.24% of the found asymmetric keys are sensitive. Sinha et al. [18] discuss several techniques for detecting and mitigating leakage of keys. The authors discuss a pattern based detection technique in which regular expressions in combination with static prefixes/suffixes (e.g. the AKIA prefix for Amazon Web Services API tokens) are used, this resulted in many false positives. A heuristics-driven filtering detection technique in combination with a standard password strength estimator is discussed, this resulted in a 100% recall with a precision of 91%. Finally a Source-based program slicing technique in which a algorithm was developed and used in combination with a password strength filter, this resulted in a 100% precision with a 84% recall. Atlassian created a open-source project called Spacecrab in with which they automated the creation of Amazon Web Services (AWS) API tokens with the goal of detecting malicious access attempts. Bourke and Grzelak [3] researchers from Atlassian created a tool called Spacecrab with which AWS API tokens can be created in bulk. The researchers used Spacecrab to create AWS API tokens and publish these to GitHub/Gist and Pastebin. Of the published tokens on GitHub 82.38% were found after approximately 30 minutes of publication. Only 9.33% of the AWS tokens published on pastebin were discovered and this took considerably longer with an average of approximately 25 hours.

4 Background

This paper analyzes the effect of leaking secret tokens on code collaboration platforms. This section discusses some of the technologies and concepts that are relevant to the research. First several important technologies of AWS are discussed. Next a description of honey tokens is given and relevant technologies regarding this. Finally the Mitre ATT&CK framework is discussed, which was used to analyze the results of the experiments.

4.1 AWS concepts and technologies

AWS is one of the major cloud-providers that exists today. With a majority of the market share its currently the market leader as a cloud provider [2]. AWS is adopted by companies both large and small to host its IT infrastructure or services and comes with a plethora of technologies to create, configure and manage those infrastructures and/or services. AWS provides several methods to harden and monitor cloud infrastructure and services hosted in AWS.

Amazon allows a *Cloudtrail* to be created that logs important management events. Cloudtrail logs events like listing, creating or modifying information about objects, for example the creation of an Elastic Computing (EC2) instance or Virtual Private Cloud (VPC). These logs depending on the configuration could be stored in the AWS logging facility called Cloudwatch or they could be stored into a persistent storage called an Simple Storage Service (S3) bucket. A Cloudtrail by default spans all regions in the AWS account it was created.

To limit the actions users can take within a single account policies can be defined to restrict access. AWS by default applies implicit *least privilege* access rights when creating an account via the Identity and Access Management (IAM) tooling. If no policies are linked directly or indirectly via a role/group to an IAM account allowing certain actions the user can effectively do nothing. The exception to this is the root account, a root account by default is allowed to do everything. Within a single AWS account its not possible to restrict the actions the root account can take. However, this is possible when creating an organization.

An organization allows management of AWS accounts linked to said organization. Within the context of an organization there is one AWS account that is the *management account* and all other AWS accounts linked to that organization are managed accounts. This structure allows for more stringent controls on any of the managed accounts via Service Control Policies (SCPs) and the consolidation of billing for all managed accounts. The SCPs can be assigned to managed accounts directly or via an Organizational Unit (OU). With SCPs restrictions can be set so that the actions the root account of a managed AWS account can be limited. When AWS accounts are members of an organization it is possible to create a Cloudtrail in the management account of that organization that spans all member accounts and its regions. This effectively allows monitoring of all AWS accounts from a single Cloudtrail within the management account of the organization.

4.2 Honey tokens

Honey tokens are the digital entities such as digital data created and solely analyzed which are used to capture digital thefts [12]. Honey tokens can come in many forms it could be an URL, database record, user account, fake executable or e-mail address. The goal of honey tokens is (early) detection of malicious activity within an organization or environment.

In essence credentials of an account with extremely limited rights that is intentionally put in a location where malicious users can find it could be considered a honey token. In AWS it is possible to manually create accounts with AWS API keys or use the tools like the in section 3 mentioned Spacecrab project to create these honey tokens. It is also possible to use honey tokens created with the tooling provided by the organization Thinkst. Thinkst provides a publicly available service for creating canary tokens (synonymous to honey token), these are valid users with AWS API keys created on the Thinkst AWS infrastructure [19]. When such a token is used a mail will be send to the configured mail address and details like the IP-address and user-agent regarding the use of the token will be provided.

4.3 Mitre ATT&CK

MITRE ATT&CK is a globally-accessible knowledge base of adversary tactics and techniques based on real-world observations [14]. The Mitre ATT&CK model was created by Mitre in 2013 and has gone through several updates since its creation. In 2019 Mitre published ATT&CK for Cloud. With this publication Mitre attempts to describe the tactics and techniques malicious users/attackers use against cloud environments and services and possible mitigation techniques to counter those. The model provides information about different techniques used by attackers in stages/tactics to get access to information or environments without detection.

5 Methods

To analyze the effect of leaking secrets a Lab environment with a set of experiments were defined. This section first describes the Lab and AWS environment setup to do the experiments in. Next a description is given about the experiments, the first experiment using static users and code repositories is described. Finally the second experiment using unique users and repositories is described.

5.1 CloMo

To collect, process and analyze the logging generated during the experiments the git project CloMo [15] was created. The CloMo project contains a set of automation scripts using Terraform [9] and Saltstack [17] to setup an environment that can receive logging from AWS and to provide automation to deploy the experimental setup required for the research project. To reduce costs and limit risk the lab environment was setup on a physical server in control of the author.

The Lab environment setup with CloMo consists of 4 Virtual Machines (VM), the first VM created is installed and configured with Saltstack to automate the configuration of the other nodes within the environment and the nodes deployed in AWS during the experiment. The second VM is installed with Docker and setup to host the Thinkst canary token Docker container. The third VM is installed and configured with Logstash [6], Elasticsearch [5] and Grafana [8] to collect, process and analyze logs gathered during the experiments. Jenkins [11] was installed on the fourth VM to allow the setup of pipelines to automate most of the prerequisites required for the experiments.

A pipeline was setup in Jenkins to automate much of the required work for the experiments as shown in figure 1. The pipeline creates and configures the AWS instances using Saltstack and Terraform as the first step. It updates the prefix in Logstash to identify each iteration of an experiment as the second step. It then collects dummy repositories from a private GitLab server. It collects the required honey tokens as the fourth step. The passwords and AWS tokens were provided in a file as part of a Git repository. The tokens are then inserted into the dummy repositories and pushed to their respective public repositories on GitLab, GitHub, Bitbucket and SourceForge.

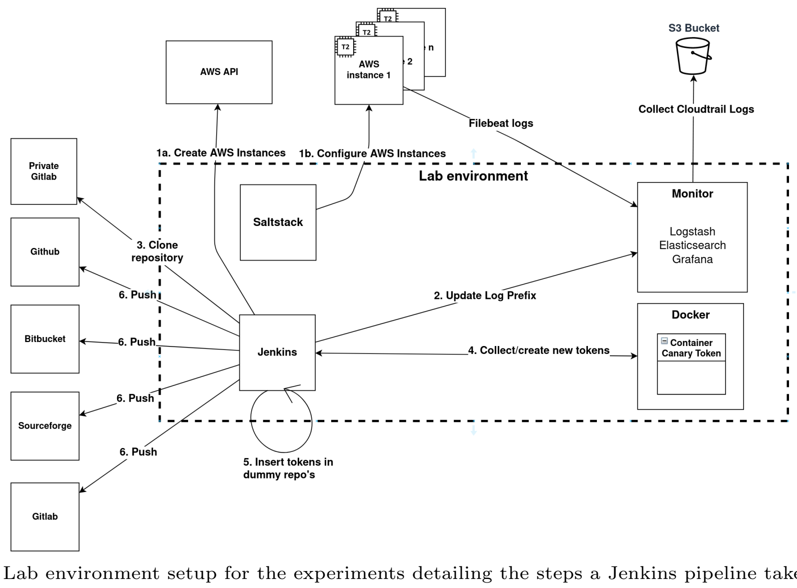

Figure 1: Overview of the Lab environment setup for the experiments detailing the steps a Jenkins pipeline takes to setup an experiment.

An additional set of Infrastructure-as-Code (IaC) scripts were setup using Terraform to create the required AWS accounts, IAM users, network objects (VPC, Subnet, SecurityGroup, etc.), S3 buckets and Cloudtrail logging in the Amazon account(s) setup for this experiment. However, these are not part of the pipeline but only need to be executed once to create the proper environment to do the experiments in. To reduce risk and provide more stringent controls a organization was setup in AWS from which the created management account could apply SCPs on any managed AWS accounts created with Terraform. The created policies allowed limitations to be set on managed account(s) for example disabling any actions the root user of a managed account could take as shown in the example code snippet below.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictEC2ForRoot",
            "Effect": "Deny",
            "Action": [
                "ec2:*",
                "health:*",
                "iam:*",
                "importexport:*",
                "networkmanager:*",
                "organizations:*",
                "pricing:*",
                "s3:GetAccountPublic*",
                "s3:ListAllMyBuckets",
                "s3:PutAccountPublic*",
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                "StringLike": {
                    "aws:PrincipalArn": [
                        "arn:aws:iam::*:root"
                    ]
                },
                "ArnNotLike": {
                    "aws:PrincipalARN": [
                        "arn:aws:iam::*:[role]"
                    ]
                }
            }
        }
    ]
}
```

5.2 Experiments

Two experiments were defined for this research project. Considering the MITRE ATT&CK framework, the experiments make initial access to AWS accounts easy by publishing honey tokens on the public facing repositories in GitHub, GitLab, Bitbucket and SourceForge. These tokens provide access to valid IAM accounts setup in a managed AWS account to monitor actions taken by malicious users via Cloudtrail after they have established access to a valid account as shown in figure 2.

5.2.1 Experiment 1 - static users and code repositories

For the first experiment one managed AWS account was setup with four Identity Access Management (IAM) users. These IAM users had read access to several AWS services in the eu-west-1 region (e.g. EC2, S3, Lambda, VPC). Each IAM account was used for leaking its AWS API honey tokens on only one of the code collaboration platforms. This allowed for identifying on which platform a given token was found. For each iteration and code collaboration platform a unique set of AWS and Canary tokens were generated. In total 4 valid AWS tokens and 4 Canary tokens were published per iteration.

The first set of experiments consisted of 10 iterations. In the first 5 iterations the tokens were published for 2 hours before rotating the honey and canary tokens. The following 5 iterations the honey and canary tokens were published for approximately 24 hours before rotating them. The AWS API honey tokens were embedded in the main Terraform file using the keywords required by Terraform to identify the keys as shown in the code snippet below.

```
provider "aws" {
  region     = "eu-west-1"
  access_key = "<AWS API TOKEN ID>"
  secret_key = "<AWS API TOKEN SECRET>"
}
```

The canary tokens were embedded in a set of Cloud-init files as shown in the code snipped below, these files also contained simple MD5 hashed password (e.g. Secret13) for one account that allowed access to one of the four EC2 nodes in AWS. This to uniquely identify from which platform a given password was used. All code including the tokens were published in a single commit. A single Jenkins pipeline was setup for the experiment in which the AWS instances were created, the different repositories were retrieved, the tokens were updated and code repositories were pushed.

```
echo access_key = <CANARY ID> >> s3_bucket.conf
echo secret_key = <CANARY SECRET> >> s3_bucket.conf
```

5.2.2 Experiment 2 - unique users and repositories

In the second experiment 5 unique accounts were setup with one public repository for each of the code collaboration platforms. A unique IAM user was setup for each code collaboration platform for each account totaling 20 IAM users. The IAM accounts in this experiment had read access to multiple services in all AWS regions. Two iterations were executed per unique account resulting in a total of 10 iterations. The leaked honey tokens were active for 72 hours in each of the repositories before deactivating the keys. In these runs the honey tokens were added in separate commits using the AWS format. This is the format used to export the AWS token as environment variables as shown in the code snippet below. The honey tokens in this case only consisted of AWS API tokens and clear-text passwords.

```
aws_access_key_id="<AWS API TOKEN ID>"
aws_secret_access_key="<AWS API TOKEN SECRET>"
```

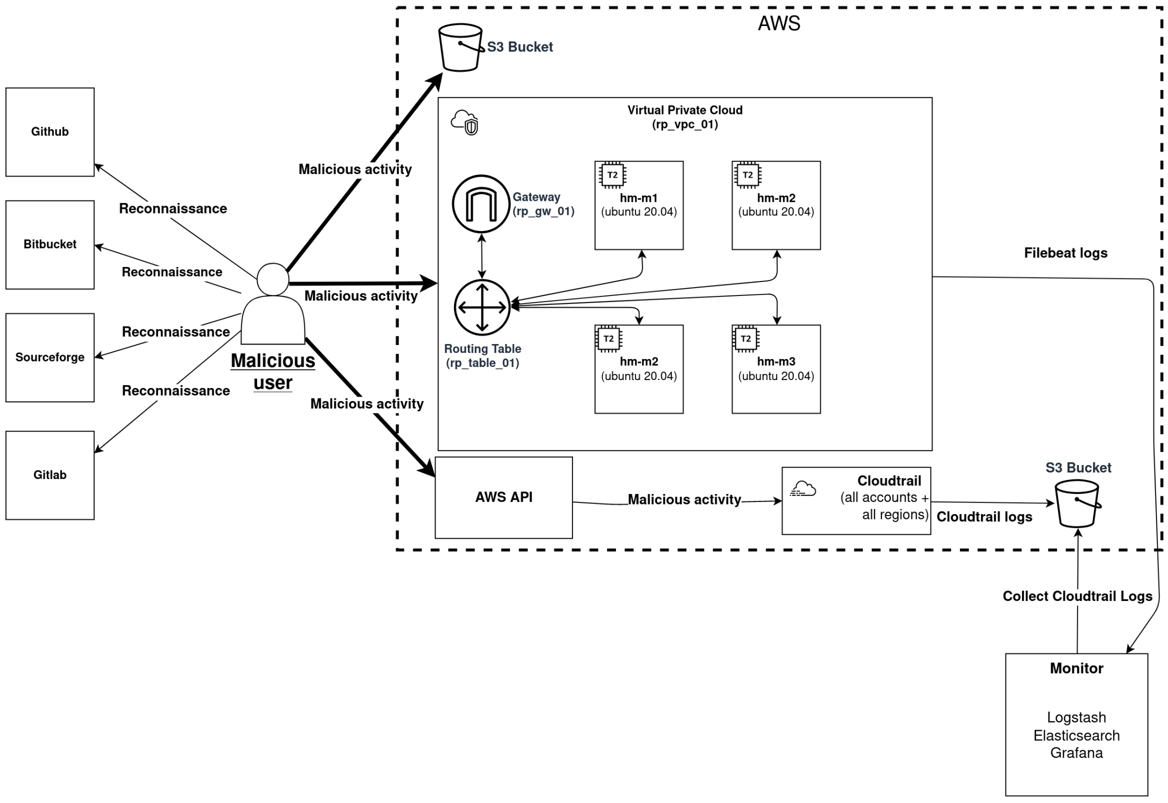

Figure 2: Overview of the AWS experimental environment.

6 Results

This section the results of the experiments are described. First the results of the first experiment are described which used static users and repositories . Followed by the results of the second experiment in which unique users and code repositories were setup.

6.1 Static users and code repositories

For each iteration of the first experiment, the code containing honey tokens and passwords were committed to the same repository on each of the code collaboration platforms. The following tables provide an overview results of the first experiment.

	GH	**GL**	**BB**	**SF**
API tokens found	2/10	0/10	0/10	0/10
Canary tokens found	2/10	0/10	0/10	1/10
Percentage found	20%	0%	0%	5%

Table 1: Overview of tokens found by malicious users in the first experiment, regardless of the amount of times a single token was (ab)used. GH = GitHub, GL = GitLab, BB = Bitbucket, SF = SourceForge

Each iteration contained one usable AWS API honey token and one Canary token resulting in a total of 10 tokens per code collaboration platform for all iterations combined. Table 1 details the amount of tokens found in the first 10 experiments by malicious users. None of the MD5 hashed passwords were used in the EC2 instances. None of the AWS API or Canary tokens were found on GitLab or Bitbucket, while only 4 of the 20 tokens placed in the public GitHub repository were found and 1 of the 20 tokens placed in Sourceforge. However, all AWS API tokens created for the experimental AWS environment on GitHub or Bitbucket triggered the automated quarantine policy within a minute of committing the token. This effectively denied any actions on such a quarantined account.

Table 2 details the fastest and slowest time after which tokens were found and the amount of times they were used. The identification was done based on the time gaps between the attempts a honey token was used. The fastest a token was (ab)used is 45 minutes and 26 seconds, this was a token placed on Github. The longest it took for a token to be found was approximately 56 and a half hours, this was a canary token placed on Sourceforge in iteration 8. The AWS API honey token placed in the Sourceforge repository that allowed access to the AWS environment was already rotated and unusable.

ITR	**FA**	**SL**	**HON**	**CAN**
1	01:44:56	01:27:33	0	3
2	00:45:26	01:37:33	4	0
3	N.A.	N.A.	0	0
4	N.A.	N.A.	0	0
5	N.A.	N.A.	0	0
6	01:18:34	03:04:56	3	0
7	N.A.	N.A.	0	0
8	54:03:57	56:24:21	0	3
9	N.A.	N.A.	0	0
10	04:04:59	N.A.	0	1

Table 2: Detailed overview of the time (HH:MM:SS) in which tokens were found and the amount of times they were used per iteration. ITR = Iteration, FA = Fastest, SL = Slowest, HON = Total usage attempts AWS API honey tokens, CAN = Total usage attempts Canary tokens

Action	**T*U**	**MAT**
DescribeInstances	7	CID
GetAccountAuthorizationDetails	2	AD
N.A	5	N.A

Table 3: Overview of actions taken by abusers with found tokens. T*U = Times used, MAT = Mitre ATT&CK Technique, CID = Cloud Infrastructure Discovery, AD = Account Discovery

Table 3 details the type of actions that were taken, how many times those were taken and what type of technique this is considering the MITRE ATT&CK framework. From the tokens that were abused most attempts were made using the AWS Software Development Kit (SDK) for JavaScript version 2 (aws-sdk-js), this is one of the official tools provided by AWS for interfacing with the APIs. All attempts made that could be recognized started with a *DescribeInstances* regardless of the user agent. In some cases this was followed by a *GetAccountAuthorizationDetails*. The regions in which this action was executed differed but most of them were in the AWS region us-east-1.

6.2 Unique users and code repositories

The following tables show the results of the second set of experiments in which unique users with unique repositories were setup for each iteration. In total 10 AWS API honey tokens and 10 clear-text passwords were committed to public repositories and this was done in separate commits after committing the initial code base.

	GH	**GL**	**BB**	**SF**
API tokens found	10/10	0/10	0/10	0/10
Percentage found	100%	0%	0%	0%

Table 4: Overview of honey tokens found by malicious users in the second experiment, regardless of the amount of times a single token was (ab)used.
GH = GitHub, GL = GitLab, BB = Bitbucket, SF = SourceForge

Table 4 details the amount of times each of the leaked honey tokens were used. All honey tokens placed on GitHub were found and none of the tokens place on GitLab, Bitbucket or SourceForge were found. None of the clear-text passwords were ever used.

REPO	**FA**	**SL**	**HON**
1	00:14:19	11:22:01	3
2	00:04:27	26:41:13	4
3	00:08:16	01:30:15	3
4	00:02:56	06:52:58	2
5	14:45:36	26:37:01	9

Table 5: Detailed overview of the time (HH:MM:SS) in which tokens were found and the amount of times they were used per iteration.
REPO = Public Code Repository, FA = Fastest, SL = Slowest, HON = Total usage attempts AWS API honey tokens, TOT CAN = Total usage attempts Canary tokens

Table 5 shows the fastest and slowest time after which tokens were used and the amount of times they were found by uniquely identifiable malicious users. The identification was done based on the time gaps between the attempts a honey token was used. During this experiment the fastest a honey token was found and used, was in 2 minutes and 56 seconds for one of the repositories. The time in which the honey tokens were used varied some tokens were used in minutes while others took over a day before any attempts were made. The longest duration before a token was used was 26 hours and 41 minutes. While some tokens were used only once by a unique malicious user, others were used by multiple unique users, the most being 9 times.

After a key was found different types of Discovery technique were used. There were few attempts to get more information regarding the users, roles and policies that were configured. There were several attempts to get more information about MFA devices and access-keys. Table 6 provides an overview of most used account discovery attempts that were made in the AWS environment.

Action	**T*U**	**MAT**
GetUser	5	AD
GetCallerIdentity	4	AD
ListUsers	2	AD
ListRoles	2	AD
ListMFADevices	2	AD
ListGroups	2	AD

Table 6: Overview of the Account Discovery actions taken most by abusers with found tokens.
T*U = Times used, MAT = Mitre ATT&CK Technique, AD = Account Discovery

There were multiple attempts to get more information about the infrastructure by using Cloud Infrastructure Discovery (CID) techniques. Table 7 details some of the CID techniques used most frequently. Most attempts to gain information about the AWS environment started with *DescribeInstances*. One of these attempts used automation to iterate over all the AWS regions requesting information about the VPCs, Subnets, SecurityGroups, addresses, hosts among other objects. In this case Boto3 was used which is based on the AWS SDK for Python.

Action	**T*U**	**MAT**
DescribeInstances	43	CID
DescribeSubnets	25	CID
DescribeVpcs	25	CID
DescribeRouteTables	23	CID
DescribeNetworkAcls	23	CID
DescribeHosts	22	CID
RunInstances	1076	MCCI

Table 7: Overview of the actions taken most by abusers with found tokens.
T*U = Times used, MAT = Mitre ATT&CK Technique, CID = Cloud Infrastructure Discovery, MCCI = Modify Cloud Compute Infrastructure

After an initial Discovery technique there were occurrences where automation was activated if those discovery attempts did not return an error/access denied. In total automation was used four times with the aws-sdk-js version 2. These attempts tried to created dozens or hundreds of large EC2 instances over a span of minutes in multiple AWS regions. A total of 1076 attempts were made to modify the cloud compute infrastructure as shown in Table 7.

7 Discussion

In this section the results are discussed. First the some of the aspects of searching in the different code collaboration platform is discussed. Next the difference in committing to code repositories is discussed, namely on GitHub. The profile of repositories in which secrets were leaked is discussed and finally the effect of the AWS quarantine policy is discussed.

7.1 Searching in Code collaboration platforms

From the code collaboration platforms on which secrets were leaked during the experiments only GitHub provides a REST API with extensive search capabilities on public repositories. An example of this is the ability to search commits done to public repositories on a specific date. There are limitations to this, for example the results per page is set to a maximum of 100 and the maximum amount of requests for non enterprise users is set to 5.000 per hour [7]. This still allows the retrieval of 500.000 commits per hour done to public repositories.

Both GitLab and Bitbucket provide (REST) APIs for searching. The API provided by GitLab limits searches to a specific repository, there are advanced search functionalities available in GitLab however, this is a paid feature. The API provided by Bitbucket seems to limit anonymous requests to 60 per hour, when logged in this is increased 1000 requests per hour. However, to browse commits the API also seems to require to specify a specific repository [1]. Considering these restrictions, its harder to search public repositories on both platforms. Sourceforge does not provide an API with search capabilities that any of the other repositories do provide. This helps explain why published tokens are frequently found on GitHub and not on any of the other platforms.

7.2 Committing to Code repositories

In addition to an API for searching, the contents and size of a commit also seem to influence the results. In the first experiment the tokens were committed as part of larger code bases in a single commit. In addition to the keywords identifying tokens the format could also make a difference as the first experiment using the Terraform format was not found often. The second experiment used the shell environment variable method used by AWS to identify the tokens and these were published in a commit only containing those values. This seemed to increase ease of finding the tokens.

7.3 Repository profile

The experiments did not focus on the profile of the code bases in which the honey tokens were leaked. The abuse of tokens seems random and its likely that a high profile repository that is setup by a large enterprise might attract more and different type of attackers. A targeted attack would most likely result in more/different types of techniques to do reconnaissance, attempts to consolidate and obfuscate gained access to an AWS environment.

7.4 AWS Quarantine policy

When AWS tokens linked to an IAM account are pushed to a public repository on GitHub or Bitbucket a quarantine policy is automatically applied to that IAM account. In the first experiment the policy was not removed on accounts on which it was applied. The negative result when tokens were tested by malicious users seemed to deter them from doing any further investigation on what the tokens could do in AWS. To get better data the quarantine policy was removed in the second experiment. For two iterations of the second experiment this resulted in a malicious user trying to access the account before the quarantine policy was removed. Again the attackers did not make any other attempts after the initial attempt failed.

8 Conclusion

The goal of this research was to identify how leaked credentials/secrets on code collaboration platforms like GitHub, GitLab, SourceForge and Bitbucket are found and abused by malicious users in a cloud platforms like AWS. In order to identify how secrets are found and abused a set of experiments were defined in which secrets were intentionally leaked to code collaboration platforms.

To monitor if the tokens were found a open-source project was setup called CloMo. CloMo contains a fully automated setup using Terraform and Saltstack for Logstash, Elasticsearch, Grafana and Jenkins. CloMo was used to setup the lab environment and AWS environment in which the experiments were performed. The lab environment was used to collect and analyze the Cloudtrail and Filebeat logs from the AWS environment. Jenkins was equipped with several pipelines to automate the creation of the AWS environment and publication of the honey tokens to public code repositories for the experiments.

Of the honey tokens stored in public repositories on GitHub, GitLab, Bitbucket and SourceForge only those on GitHub were frequently discovered by malicious users. The successful reconnaissance step on GitHub could be contributed to the REST API GitHub provides for searching through commits done to public repositories. Malicious users can find tokens within minutes after publishing the tokens. When a honey token was found almost all attempts started with a discovery step using the *DescribeInstances* action, in case an error was returned this was sometimes followed by an additional Discovery action like *GetAccountAuthorizationDetails*.

In case no error occurred the malicious user in some cases attempted to create EC2 instances with large resources, for example the r4.large, m3.large or m5n.large instance types. This process seemed to be automated as there was a large number of attempts in a short period of time using the AWS SDK for JavaScript in combination with Tor. The most likely reason for this is to use the instances for crypto mining. Actions to consolidate access to an account by for example changing the password or setting up lambda's that created new users or tokens were not made.

In the first set of experiments all commits done to Bitbucket and GitHub automatically triggered the secret scanning features enabled on all public repositories on either platform. This in turn automatically enabled a quarantine policy on the IAM user in AWS, effectively disabling any actions that IAM user could take. When a malicious user attempted to abuse found tokens and the response resulted in an error containing an *Access Denied* or *Client.UnauthorizedOperation* due to the quarantine policy no further actions were taken.

A good practice to reduce risk would be to create IAM users with very specific policies enabled, allowing it to do only those things the account was setup for. Regardless if an AWS environment is used by a single person or multiple people, using an organization and setting up at least 2 AWS accounts can help limit the actions a malicious can take when setup with proper SCPs. A proper SCP on a managed account can limit the risks in case a (root) AWS account is compromised. It also provides monitoring capabilities from the management account across all managed accounts via Cloudtrail. Additional SCPs could be setup to limit actions taken on managed account for example blocking the removal of a Cloudtrail or blocking the use/creation of instances or services in specific regions.

Relying solely on the secret scanning feature and protection provided by GitHub or Bitbucket could still result in compromised accounts. When publishing tokens to a private Bitbucket repository and switching it to a public repository the quarantine policy in AWS will not be triggered. When a token ID and token secret are published in separate commits they will not trigger any secret scanning done by the GitHub or Bitbucket, while they are found by malicious users. Using tools like Shh-git or Git-secrets in combination with Git (pre-commit) hooks could prevent a secret from every being committed to a repository. This ensures that a secret is not accidentally committed to a public repository even if they are split over multiple files. In practice making secrets available to other users of software is not always avoidable, in those cases using a tool like Vault for storing and using secrets like API tokens or passwords is a better practice then storing them in files.

9 Future work

Code collaboration platforms with public repositories have existed for several years now. The abuse of (accidentally) leaked secrets can be traced back to shortly after people starting publishing code in public repositories. It would be interesting to redo the experiments after a period of time (e.g. 1 year) to see if secrets are still found, if GitHub is still the platform on which most secrets are found and how they are abused after they are found. Leaking secrets to more/different platforms could also provide interesting results.

This research did not focus on the profiling of the code bases/repositories secrets were leaked in. Using a code base that has had more exposure (e.g. forking a well-known project) might provide very different results and could be interesting as an extension to the experimental setup provided by CloMo. Creating the profile of a fake company could also be an interesting approach. This could be done by setting up websites, multiple repositories and other items that increase the profile of the published code to see if this attracts different types of attackers.

Finally the policies currently provided in CloMo could be modified with less stringent policies on the created

IAM users. This could provide more interesting data on how an attacker tries to use the gained access. The risks of this could be high costs at the end of the experiment if the policies are not carefully setup and reviewed.

References

[1] Atlassian. *API request limits.* https://support.atlassian.com/bitbucket-cloud/docs/api-request-limits/.

[2] A. R. Bala, B. Gill, D. Smith, D. Wright, and K. Ji. Magic quadrant for cloud infrastructure and platform services. https://www.gartner.com/document/3989743, Sep 01, 2020.

[3] D. Bourke and D. Grzelak. Breach detection at scale with aws honey tokens. In *Black-hat Asia*, https://www.blackhat.com/asia-18/briefings.htmlbreach-detection-at-scale-with-aws-honey-tokens, 20-23 March, 2018 Mar 23, 2018.

[4] D. Bradbury. Github users warned over security risk. *The Guardian*, Jan 25, 2013.

[5] Elastic. *Elasticsearch.* https://www.elastic.co/elasticsearch/, Jan 14, 2021.

[6] Elastic. *Logstash.* https://www.elastic.co/logstash, Jan 14, 2021.

[7] Github. *Resouces in the REST API.* https://docs.github.com/en/rest/overview/resources-in-the-rest-apipagination.

[8] Grafana Labs, https://grafana.com/grafana/. *Grafana*, Jan 14, 2021.

[9] HashiCorp. *Terraform.* https://www.terraform.io/, Jan 06, 2021.

[10] A. Hashim. Starbuck exposed an api key in github public repository. Jan 4, 2020.

[11] Jenkins. *Jenkins.* https://www.jenkins.io/, Jan 13, 2021.

[12] N. Kambow, L. Kaur, and . Passi. Honeypots: The need of network security. *International Journal of Computer Science and Information Technologies, Vol. 5 (5)*, 5, 2014.

[13] M. Meli, M. R. McNiece, and B. Reaves. How bad can it git? characterizing secret leakage.

[14] The MITRE Corporation, https://attack.mitre.org/versions/v8/matrices-/enterprise/cloud/. *Cloud Matrix*, Oct 27, 2020.

[15] M. Mouw. *CloMo.* https://github.com/Mandorath/CloMo, Feb 7, 2021.

[16] E. Newcomer. Uber paid hackers to delete stolen data on 57 million people. *Bloomberg News*, Nov 21, 2017.

[17] Saltstack. *Salt Project.* https://saltproject.io/, Nov 18, 2020.

[18] V. S. Sinha, D. Saha, P. Dhoolia, R. Padhye, and S. Mani. Detecting and mitigating secret-key leaks in source code repositories. pages 396–400, https://ieeexplore.ieee.org/document/7180102, May, 2015. IEEE.

[19] Thinkst. *Canarytokens' new member: AWS API key Canarytoken.* https://blog.thinkst.com/2017/09/canarytokens-new-member-aws-api-key.html, 15 sep, 2017.

List of Figures

List of Tables