

# 2023 Honey potting in the Cloud Report

Attacker Tactics and Techniques Revealed





# Inside This Report

<b>Foreword</b>	<b><u>3</u></b>
<b>About the Orca Research Pod</b>	<b><u>4</u></b>
<b>Executive Summary</b>	<b><u>6</u></b>
<b>Methodology</b>	<b><u>10</u></b>
<b>Research Findings</b>	<b><u>11</u></b>
1. GitHub Honeypot	<b><u>12</u></b>
2. AWS S3 Bucket Honeypot	<b><u>17</u></b>
3. SSH Honeypot	<b><u>22</u></b>
4. HTTP Honeypot	<b><u>24</u></b>
5. DockerHub Honeypot	<b><u>25</u></b>
6. ECR Honeypot	<b><u>26</u></b>
7. Elasticsearch Honeypot	<b><u>27</u></b>
8. Amazon EBS (AMI) Honeypot	<b><u>28</u></b>
9. Redis Honeypot	<b><u>29</u></b>
<b>Summary</b>	<b><u>31</u></b>
<b>Key Recommendations</b>	<b><u>36</u></b>
<b>About Orca Security</b>	<b><u>41</u></b>



# Foreword

*"Know thy enemy and know yourself; in a hundred battles, you will never be defeated."*

**Chinese general Sun Tzu.**

**In an era where cloud computing has become an integral part of modern business operations**, ensuring the security of cloud environments is of paramount importance. Cybercriminals are relentlessly seeking to exploit vulnerabilities and misconfigurations to gain unauthorized access to valuable resources. The more security teams can understand attacker tactics and techniques, the more effective they will be at defending themselves.

For this purpose, the [Orca Research Pod](#) launched a honeypot research project to simulate misconfigured resources in the cloud, and then monitor whether bad actors would take the bait while shedding light on the latest attack vectors and providing essential insights for fortifying cloud defenses.





# About the Orca Research Pod



The [Orca Research Pod](#) is a group of cloud security researchers that discover and analyze cloud risks and vulnerabilities to strengthen the Orca Cloud Security Platform and promote cloud security best practices. In addition, the Orca research team discovers and helps resolve vulnerabilities in cloud provider platforms so organizations can rely on a safe infrastructure in the cloud.



14+ vulnerabilities  
discovered on  
AWS, Azure, and  
Google Cloud



2021

1. [AWS Superglue](#)
2. [Azure AutoWarp](#)
3. [AWS BreakingFormation](#)
4. [Databricks](#)



2022

1. [Azure SynLapse](#)
2. [Azure FabriXxs](#)
3. [Azure CosMiss](#)
4. [Azure Digital Twins SSRF](#)
5. [Azure Functions App SSRF](#)
6. [Azure API Management SSRF](#)
7. [Azure Machine Learning SSRF](#)
8. [Azure Super FabriXss](#)



2023

1. [Azure Storage Account Keys Exploitation Path](#)
2. [Two Azure PostMessage IFrame Vulnerabilities](#)



## Bar Kaduri

Threat Research Team Leader, Orca Security

[LinkedIn](#)



## Tohar Braun

Research Technical Lead, Orca Security

[LinkedIn](#)



### The goal of our honeypot research was to find out the following:

- > Which of the popular cloud services are most frequently targeted by attackers?
- > How long does it take for attackers to access public or easily accessible resources?
- > How long does it take for attackers to find and use leaked secrets?
- > What are common attack routes and methods?
- > How can we leverage this information to increase defenses?



This research aims to equip cloud security professionals, DevOps, DevSecOps, CISOs, and development leaders with valuable insights and practical recommendations for safeguarding their cloud environments, and in doing so, help to secure the cloud for everyone.



## Executive Summary

# 1

## Discovery is *fast*

In some ways, our study confirmed what is already widely known: attackers are constantly scanning the Internet for lucrative opportunities. What did surprise us however was how fast this was happening:



On GitHub, attackers weaponized our leaked keys within minutes. It only took **2 minutes** until one of our GitHub honeypot keys was used.



The first access to our HTTP honeypot was within **3 minutes**.



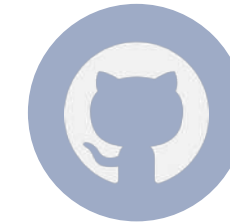
We saw access to our SSH honeypot within **4 minutes**. There were no attempts to use the key we planted, but we saw hundreds of attempts to install **malware** and **cryptominers** on our server.



Our S3 Buckets were accessed in **one hour** and the keys were used within **8 hours**.



## It took attackers..



**2 minutes**

to exploit keys exposed on GitHub



**3 minutes**

to access our HTTP Honeypot



**4 minutes**

to access our SSH Honeypot



**1 hour**

to access our S3 Buckets





## Executive Summary

# 2

## Attackers target each resource *differently*

The more popular the resource, the easier it is to access, and the more likely it is to contain sensitive information → **the more attackers will do (automatic) reconnaissance.**



Certain assets, such as SSH, are highly targeted for malware and cryptomining. We saw hundreds of attempts by attackers to install malware and cryptominers on our SSH honeypot.

Even though public assets on some resources are discovered much faster than others, it's clear that wherever you store public data, it will be compromised at some point - whether it's in minutes, hours, days, or months.



### Time to Asset Access



**2 mins**  
GitHub



**3 mins**  
HTTP



**4 mins**  
SSH



**1 hour**  
S3 Bucket



**2 hours**  
Elasticsearch



**2.5 hours**  
Redis



**4 months**  
AWS ECR

No attempts to access:  
DockerHub or Amazon EBS

### Time to Key Usage



**2 mins**  
GitHub



**8 hours**  
S3 Bucket



**4 months**  
AWS ECR

No attempts to use the keys on:  
HTTP, SSH, Elasticsearch, or Redis



## Executive Summary

# 3

## Automated key protection *cannot* be relied on

Secrets are automatically locked down on GitHub but **not** on any of the other resources, such as ECR and S3 Buckets.



Except for the breached keys from AWS in GitHub, **no keys were reported as breached**, despite the fact that some of them were used by unauthorized users.



Even if key permissions are locked down (as they were on GitHub), the key is **not entirely blocked**. Although the policy denies most permissions, an attacker can potentially still perform malicious actions on some services, such as RDS, EKS, and Elasticsearch.



This means that defenders need to be extra careful not to include secrets on S3 Buckets, and to a lesser extent Elastic Container Registry, since we also saw relatively fast key usage from these resources.



AWS keys were automatically locked down on GitHub, but..



Even though permissions of the leaked keys were locked down as soon as the git push occurred..



Using the [AWS Compromised Key](#) policy, which denies access to destructive actions..



**Even with this policy applied**, if the leaked credentials have a lot of permissions, an attacker can still do damage.





## Executive Summary

# 4

## No region is safe

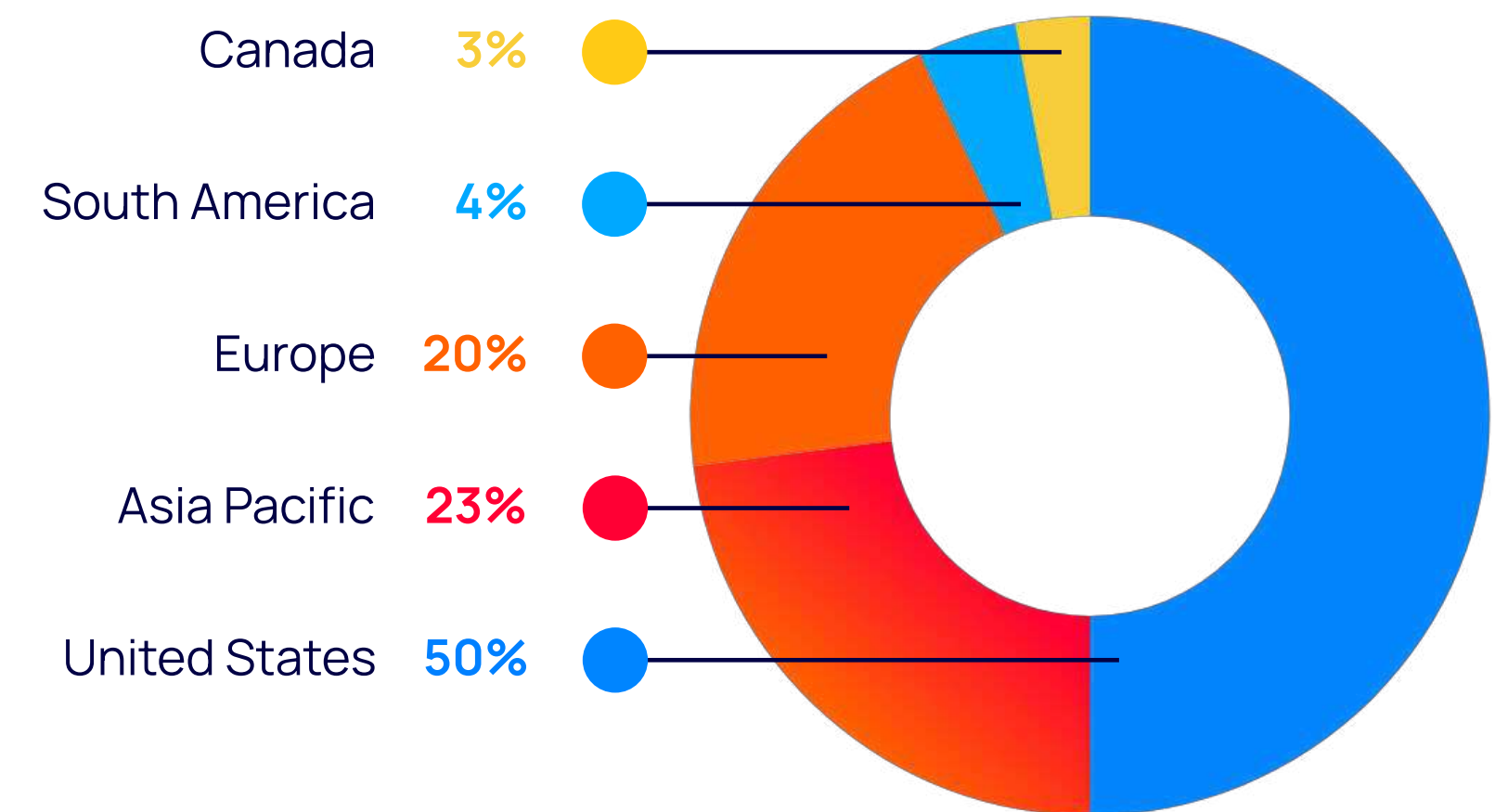
Although we saw 50% of the AWS key exploitation in the US, usage also occurred in almost every other region, including Canada, Asia Pacific, Europe, and South America.



**As more and more organizations around the world adopt the cloud, attackers are not just sticking to North America, but are opportunistically targeting every possible region.**



### AWS regions where API calls were made with leaked keys on GitHub:





# Research Methodology

**The purpose of this research** was to achieve a better understanding of how quickly attackers find assets and use secrets in each scenario. Armed with this information, security teams can establish the right protections to keep assets from being exposed, and perform the most effective remediations when an exposed asset has been found.

## In our research we measured the following:

- **Probability of asset access:** How likely is it that an accidentally exposed asset will be accessed and how quickly? Does this likelihood differ depending on the resource environment? We tracked this by monitoring traffic to the services using t-pot<sup>1</sup> or native access logs.
- **Tactics applied in asset access:** If assets are accessed, what types of commands are used most often? What does this tell us about attacker tactics? We were also able to track this by monitoring traffic to the services using t-pot<sup>1</sup> or native access logs.
- **Probability of secret usage:** How likely is it that an [exposed secret](#) in that asset will be used? By using canary AWS tokens (valid access tokens that act as tripwires), we could see when, where, and how they were used without providing the attacker access to anything that was actually of interest.
- **Tactics applied in secret usage:** If exposed secrets are used, what type of tactics do attackers use the most? What does this tell us about their strategies?

<sup>1</sup> <https://github.com/telekom-security/tpotce>



## Data collection

Our research was conducted between January and May 2023. To set up our 'honeypots' and simulate misconfigured resources, we basically broke all security best practices (don't try this at home!):

We started by creating a number of resources in different environments that allowed public or easy access. Next, we placed a secret - in this case AWS keys - in our honeypots. And then we observed as attackers took the bait..



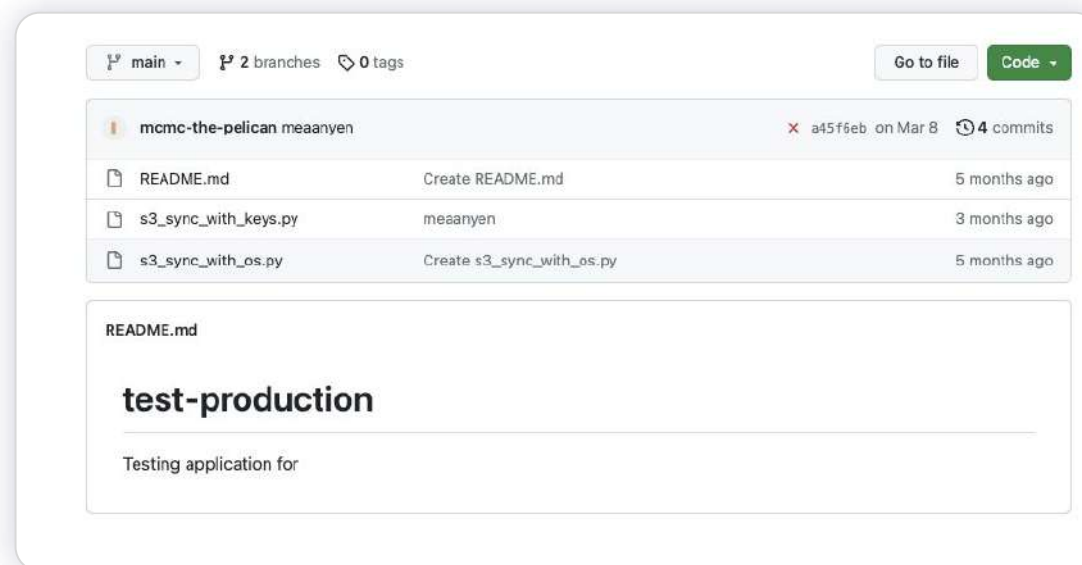
# Research Findings



# 1

## GitHub HoneyPot - Setup

1



We created a [public repository](#) with two Python files.

2

[One](#) of the Python files contained an AWS access key (secret and access keys).

3

The second one contained a bucket (s3://switanok-zustricz) with an access key in it.

Although GitHub doesn't provide access logs to public repos, we could tell repo access by tracking usage of the keys in the repos.



## Why is there significant leakage risk on GitHub?

GitHub is a source control system that stores intellectual property such as software source code, build scripts, and Infrastructure as Code scripts.

It is not uncommon for organizations to accidentally leak secrets, database passwords and other sensitive data in code commits. This is especially problematic as it's relatively easy for attackers to discover public GitHub repositories and commits.

In view of this potential risk, we wanted to find out how quickly attackers would discover and weaponize leaked secrets in GitHub commits.

50%

of organizations [store sensitive data](#) in at least one Git repository





# GitHub Honey-pot - Key Usage

Usage of the AWS key occurred **quickly and from many sources**. It took only 2 minutes for an attacker to use the leaked key.



The good news is that the leaked keys were quarantined by AWS as soon as the git push occurred; The [AWS Compromised Key Quarantine](#) policy for the leaked key was added at the exact same time as the git push was committed.

However, even with this policy applied, this does not mean that the key is entirely blocked. Although the policy denies most of the EC2, S3 Bucket, Lambda, and IAM service permissions, attackers can still access all other permissions a user might have, such as Amazon Relational Database Service (RDS), Amazon Elastic Kubernetes Service (EKS), and Opensearch.



In other words, if the leaked credentials have a lot of permissions, an attacker can still do damage.



It only took:

**2 minutes**  
before keys were exploited



## Beware of the Git History

We decided to see what would happen if we created a new commit that removed the secret while leaving the original commit in the Git history. We observed that the keys that were published in old commits were rescanned after a newer commit, indicating that attackers are searching the Git History for keys as well. Even though fewer actors discovered and used the key from the history than from the original commit, it is important to make sure that any keys are not only removed from the newest commit, but also from the commit in the history.





# GitHub Honeypot - Key Tactics

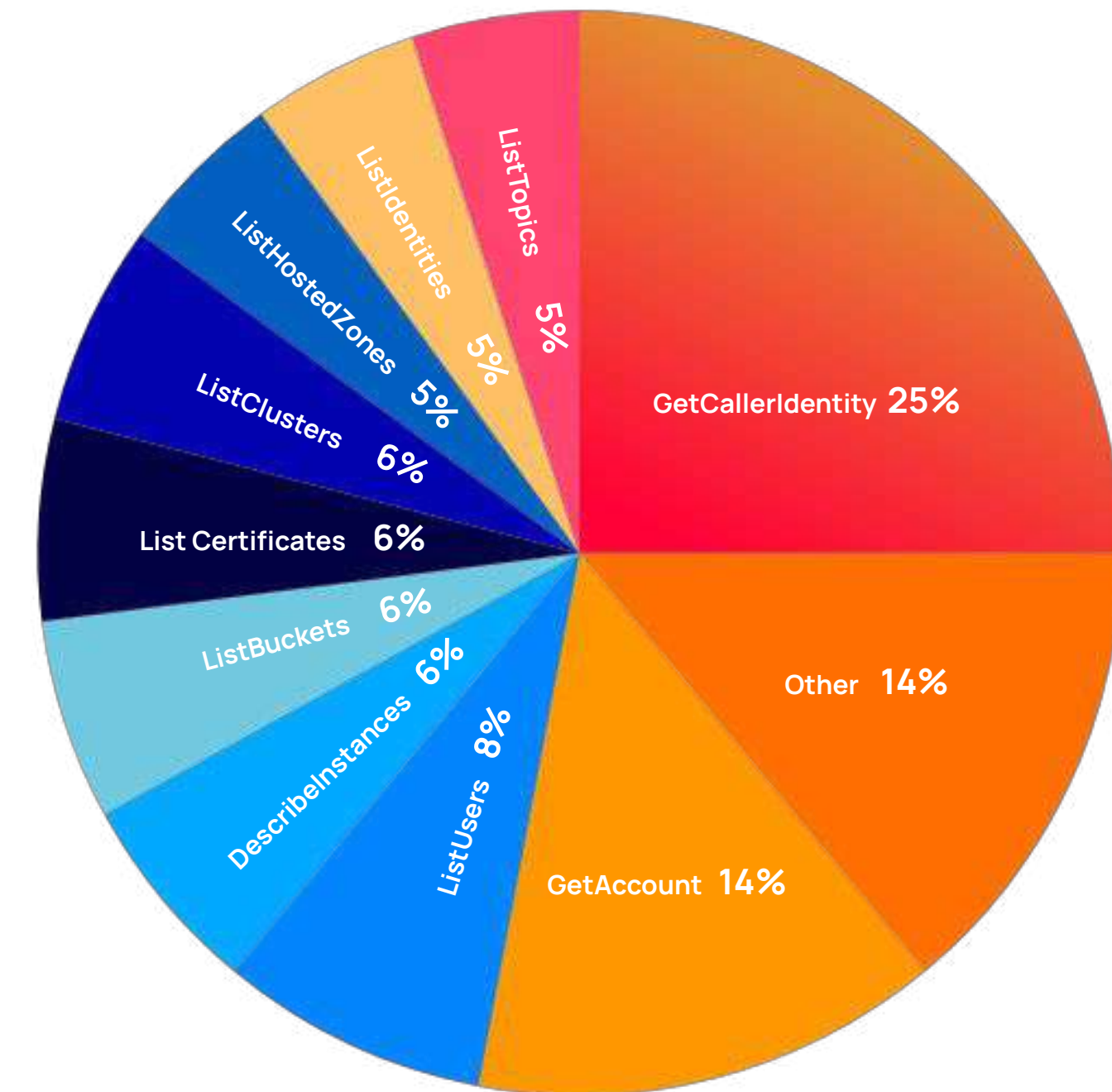
The majority of key usage was around reconnaissance: attackers were trying to find out whether the key provided access to any resources that could be of interest.



- **“GetCallerIdentity” (25%)** was the most used API call, followed by **“GetAccount” (14%)**, which suggests that actors are trying to test the validity of the secret and gather more information about the owner of the exposed key.
- Next **“ListUsers” (8%)**, and **“DescribeInstances” (6%)** are used most commonly, which point to reconnaissance commands where the actor is trying to find out what the key provides access to.
- **“ListHostedZones” (5%)** is of further interest because it can enable an attacker to further enumerate their target’s footprint and look for additional access.



## API calls made with the keys leaked on GitHub:

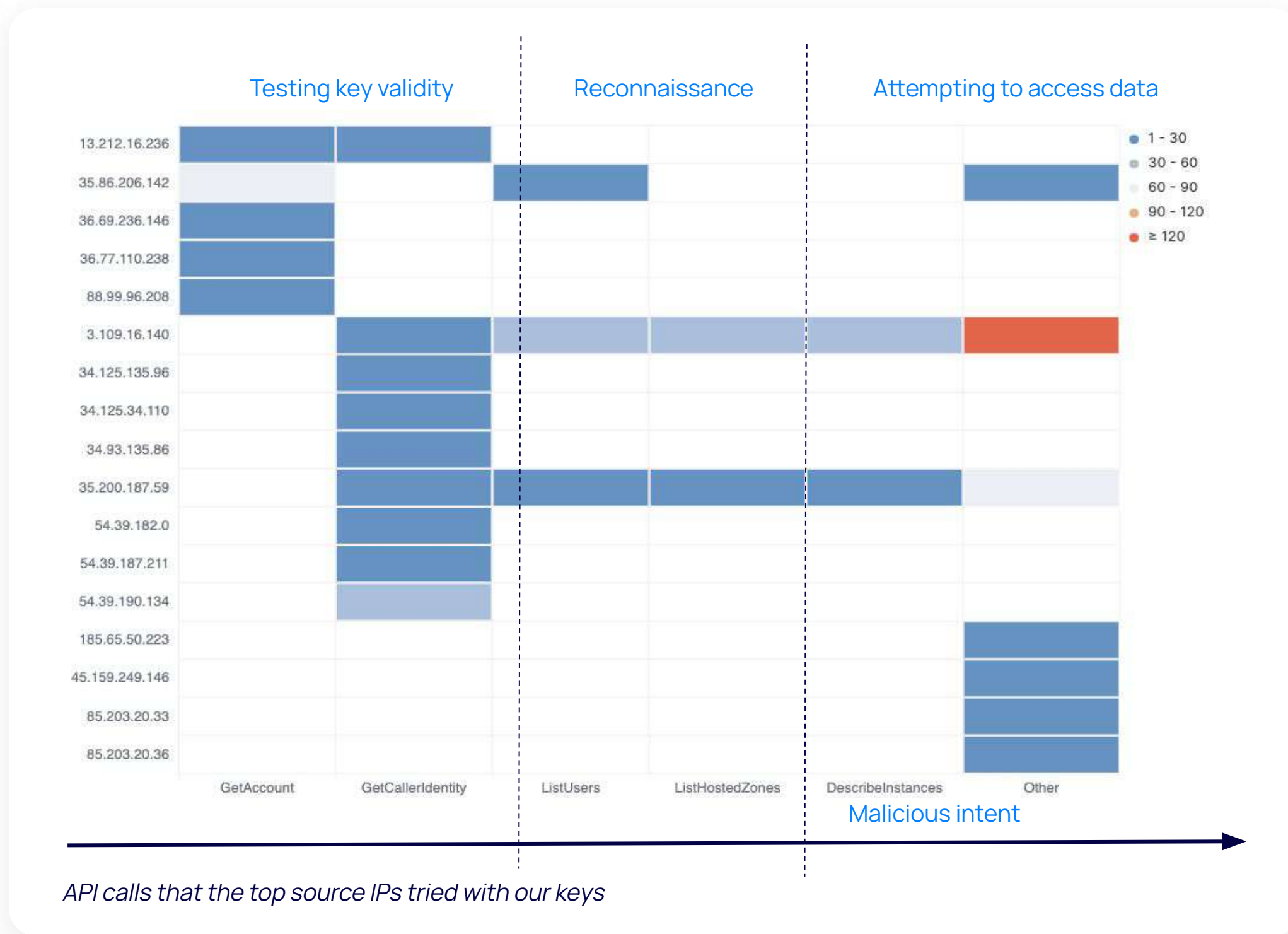






# GitHub Honeypot - Attacker Close Up

A closer look at the individual actors shows that most start initial reconnaissance and then give up, but a small number of actors are very persistent.



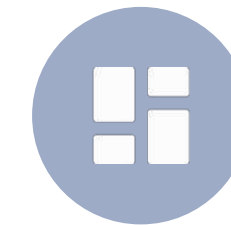
## Observations



IP addresses 3.109.16.140 and 35.200.187.59 were behind the most usage attempts and tried to access data with our key. We suspect they're the same actor. On the day of the key publication, they tried the same calls only milliseconds apart. Both IPs are located in Mumbai, yet, the attacker tried to use API calls in almost all regions, and mainly in US-East-1. The first scan from this actor came about 5 minutes after publication.



IP address 88.99.96.208 is a scanner based in Germany, it initiated one GetAccount call for each AWS region, but then gave up.



IP address 54.39.190.134 is a GitGuardian scanner (a code security platform) scanning periodically up to 7 days after publication.

# GitHub Honeypot - Regions

Although we saw half of the AWS key exploitation in the US, usage also occurred in almost every other region.

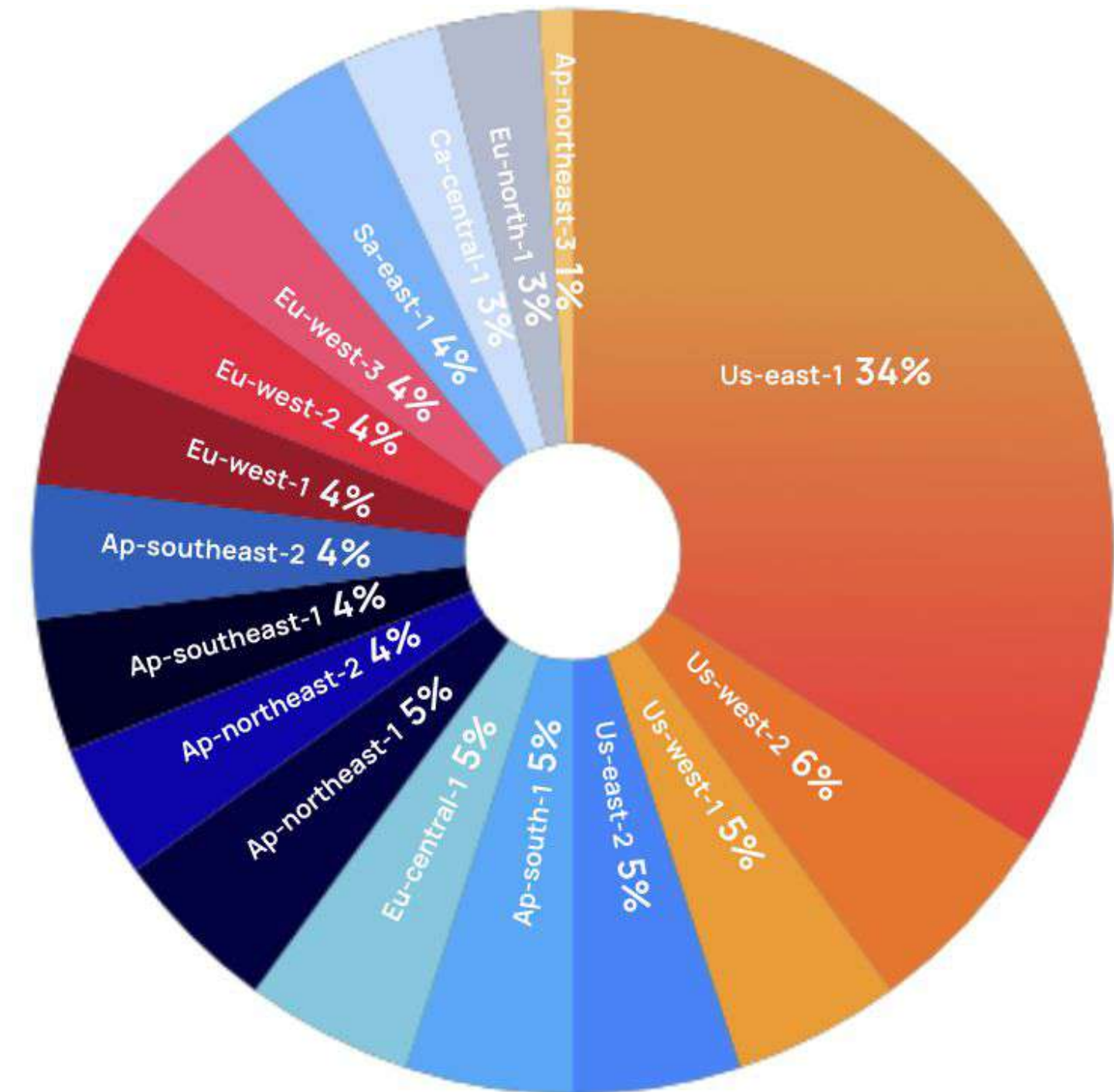


From the chart on the right we can see that, as can be expected, most of the attackers tried to use the key in us-east-1, which is the most used AWS region. However, we did not expect to see key usage in almost every other region as well, including **Asia Pacific (23%)**, **Europe (20%)**, **South America (4%)**, and **Canada (3%)**.

So, as we can see, no region is out of target for attackers.



### AWS regions where API calls were made with leaked keys on GitHub:

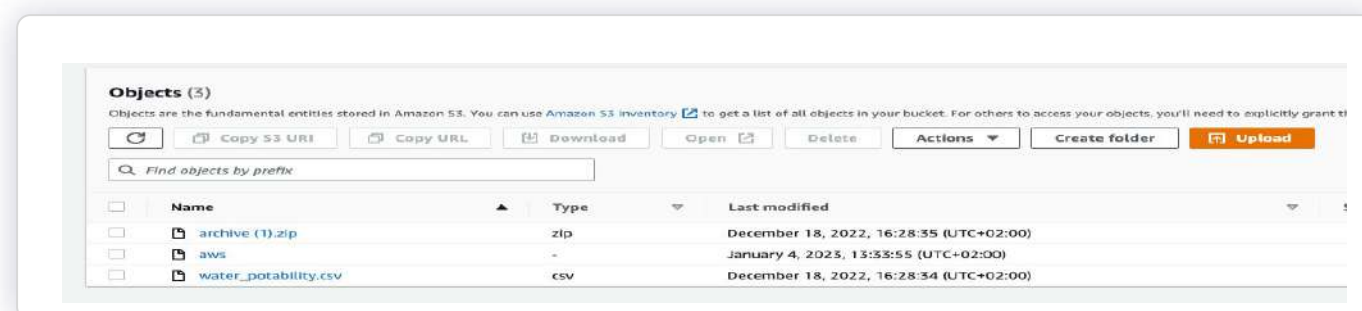




# 2

## AWS S3 Bucket Honeypot - Setup

1 We created **13 public buckets** using commonly used names.



2 Then we granted them **full List and Get permissions**.

3 Next, we put **AWS access keys** as a canary token in each bucket, so we would get a notification when the token was used as well as metrics on how it was used.

```
aws_access_key_id = AKIAR3BRNOQR7EE4CJPQ
aws_secret_access_key = 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
```

### Our S3 Bucket Honeypot Names

- ben-application-mirroring
- org.com
- static-assets-com
- prod-sandra-sadeh-simon
- san-gui-images
- sergei-bucket
- shen-test
- slava-images
- sophie-tests
- roy-prod-duplication
- staging-production-assets
- static-prod-bucket
- switanok-zustric



### Bucket naming

- Because there is no easy way to find names of new or existing S3 buckets without having the appropriate permissions to begin with, attackers have to find accessible buckets by cycling through possible names until they discover accessible ones (known as brute-forcing).
- To attract our attackers, we used bucket names that included variations of names that we know public bucket scanners are already actively searching for, and are included on common bucket names lists used by attackers.

### Public access

- Our public S3 Buckets allowed anybody to list the objects stored in the bucket and read the contents of those objects.
- This is what we wanted in our honeypot, but precisely what we *wouldn't want* on a storage bucket that contains sensitive information.







# AWS S3 Bucket Honeypot - Access

While there are actors who actively scan for public buckets with easily guessable names, only one of our buckets was actually discovered this way.

The bucket **org.com** was the first to be accessed - within two days. We suspect this is because it had a website structure and it may have attracted possible non-malicious access.

However, when none of the other buckets were accessed, we concluded that this was probably due to the lack of digital 'breadcrumbs' to our buckets, and decided to **publish 9 of the 13 bucket names** on forums and sites to try to attract more potential attackers.

And voilà! After we published the names, we saw **the first access within one hour**. Within three hours, six of the buckets were accessed.



It took attackers..



1 hour  
to **access** S3 Buckets

Count on even faster access for legitimate buckets:

While we had to leave breadcrumbs to our (fake) buckets before we saw access, we would expect there to be many more digital breadcrumbs to legitimate buckets (such as references to bucket names, IDs, and links) and therefore also expect them to be accessed *even faster* than in our tests.





# AWS S3 Bucket Honeypot - Publication

We first posted to Pastebin in different geographies and languages, which led to some access. Initially, we saw the most access from the Chinese and Ukrainian postings, but no access from our Russian postings.

Next, we used Twitter (posted with appropriate hashtags), GitHub (public repository with script that included the S3 bucket name), and Reddit (posted to r/Hacking\_Tutorials). This led to a large number of logins on the shen-test bucket.

Bucket Name	# of Initial Logins	First Publication	Post Views	# of Additional Logins	Second Publication	Post Views	# of Additional Logins	# of Total Logins
shen-test	0	Pastebin (CN)	13	5	Twitter	126	50	55
org.com	23	Not published	N/A	N/A	N/A	N/A	N/A	23
roy-prod-duplications	0	Pastebin (EN)	21	0	Reddit	34	7	7
sergei-bucket	0	Pastebin (UA)	8	6	N/A	N/A	N/A	6
switanok-zustricz	0	Pastebin (UA)	8	3	GitHub	N/A	N/A	3
sophie-tests	0	Pastebin (RU)	13	0	Twitter	25	3	3
slava-imeges	0	Pastebin (RU)	13	0	Twitter	25	3	3



The most accessed bucket was shen-test. Shen-test was first published on the Chinese Pastebin, but got far more access attempts after a post on Twitter (in English).

The S3 buckets, for which we published breadcrumbs on the Russian Pastebin weren't accessed until we also posted them on Twitter (in English).

```

text 0.11 KB | None | 👍 0 🗨️ 0
1. Знайден перелік не захищених S3
2. бакетів:
3. s3://switanok-zustricz
4. s3://sergei-bucket

```



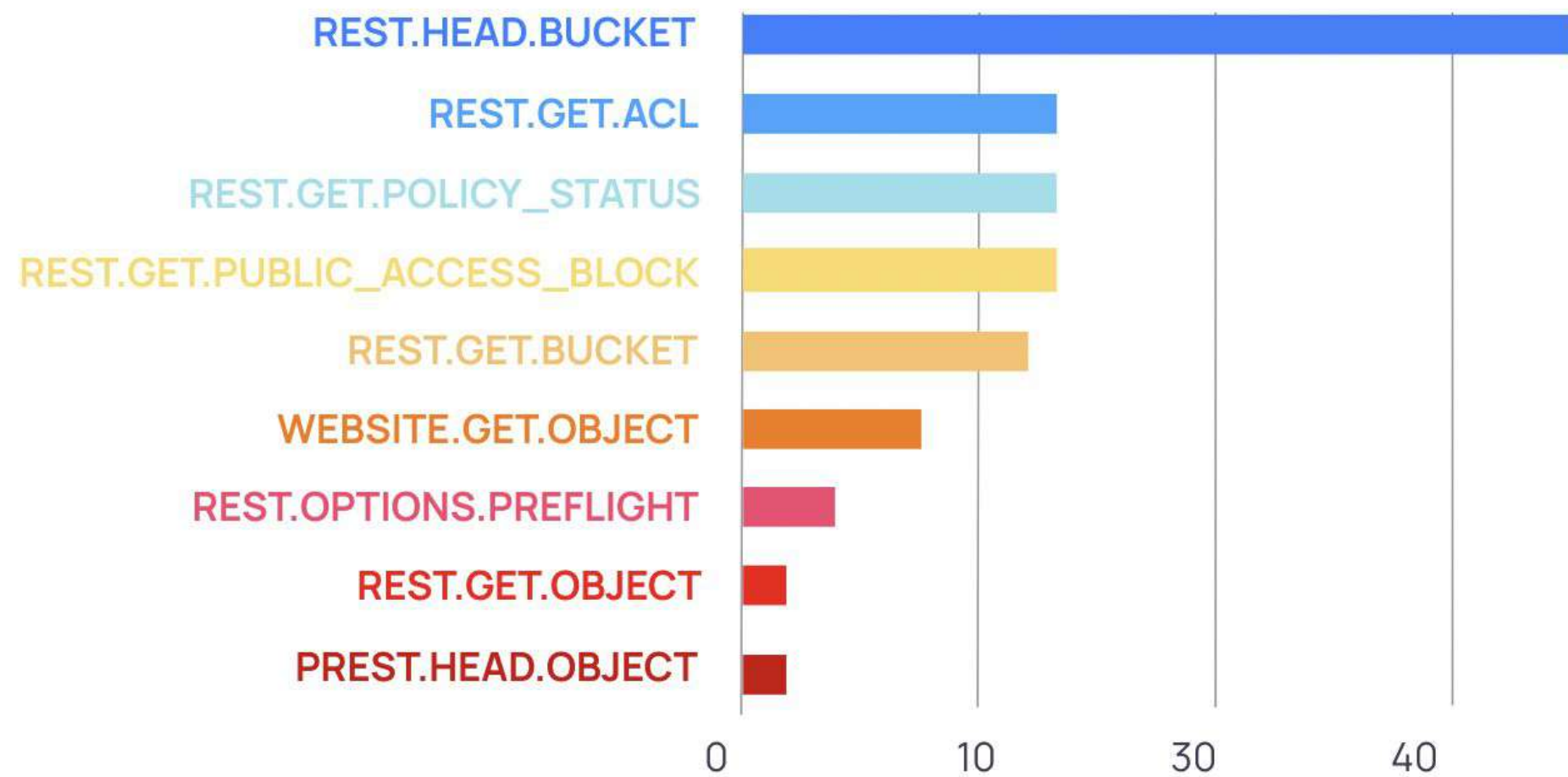


# AWS S3 Bucket Honeypot - Tactics

What actions did attackers take once they discovered the buckets?

By far the most used action was HEAD-BUCKET. This action is used to determine if a bucket exists and whether you have permission to access it. Other often used commands were GET\_ACL (to read the access control list of the resource), GET\_Policy\_Status (to find out whether the bucket is public), and GET\_Public\_Access\_Block (to retrieve the PublicAccessBlock configuration of the bucket).

## Commands used to access the S3 buckets



## Scoping out the target

We can tell from the actions attackers took that they were trying to find out whether they could access the bucket and if it contained anything interesting:

- Is this bucket public?
- What permissions do I have?
- Which users can access the bucket?
- Is there a public access block on the bucket?
- Which instances are running in this bucket?





# AWS S3 Bucket Honeypot - Key Usage

After the buckets were accessed, it took **8 hours** for attackers to start using the key. The diagram on the right shows the actions the attackers tried to take with the leaked keys.

It took attackers:



**8 hours** to **exploit keys** in S3 Buckets

## How did attackers try to use the keys they found?

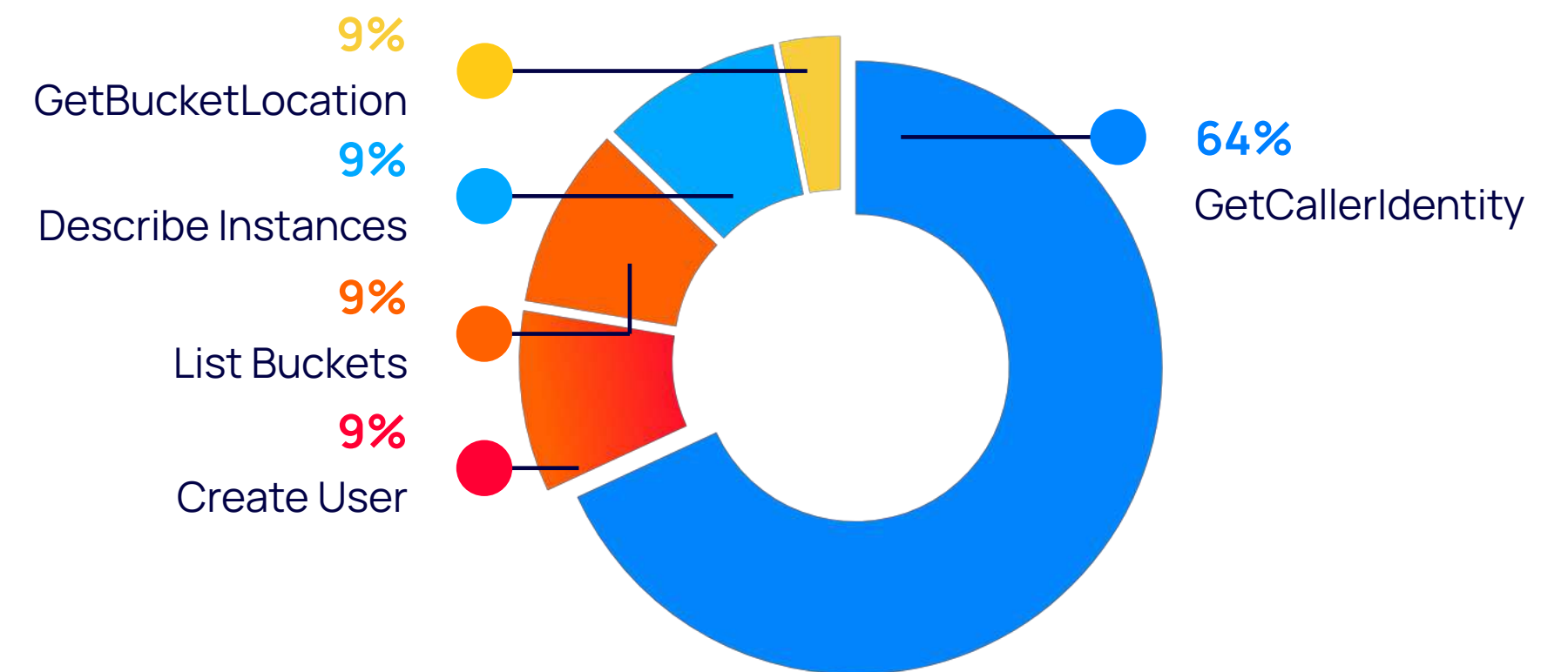
We can see that more than half used 'GetCallerIdentity' (64%), followed by 'GetBucketLocation' (9%), 'Describe Instances' (9%), and ListBuckets (9%). As can be expected, the majority are reconnaissance type actions.

We also saw more malicious attempts, such as 'CreateUser' (9%), where an actor is trying to gain persistency in the account.

If these keys had been 'real' keys, no doubt the attackers could have leveraged any information found in the buckets and caused some real damage.



## Commands used with the keys exposed on S3 buckets









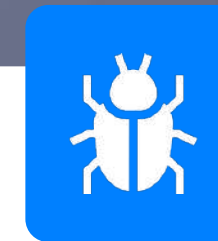
# SSH Honeypot - Tactics

The exposed key was located in /home/< user >/.aws/credentials. However, we did not detect any usage of the key.

From the CLI commands that were used, it seems like attackers were more interested in the SSH compute resources to for instance deploy cryptominers, rather than exploiting the key.

## Top CLI Commands

Command Line Input	Count
shell	10,514
system	9,324
sh	5,259
enable	5,240
while read i	5,022
dd bs=52 count=1 if=.s    cat .s    while read i; do echo \$i; done < .s	4,555
rm .s; exit	4,213
cat /bin/echo  while read i; do echo \$i; done < /proc/self/exe;	466
cat /bin/echo;	298
cat /bin/echo	178



## Attempts to run Malware

We saw hundreds of attempts by attackers to install malware on our SSH honeypot - mainly Mirai variants. Mirai is a notorious botnet, for which the source code was published in 2016. It frequently appears with new variants. If this malware is installed successfully, the machine becomes a bot that could be controlled remotely and used for many purposes, for example as a bot in a DDoS attack.

### Mirai variants detected:

```
771229b5b05e22d4f43e728b38c1e6f08fe7157e3c6dcade0e9af065f710f22d
77a2c317ca9d43acc056cf8217a8c838d23af63965b33dc931877360d5919b8d
C5bd2146ebbe575a47a666e07b99eb526d46d74e0d7758bf0bf5cb5b3adaa55a
36bc49ede8e0f4a54449602ca2bc681f96b14869841a243ddf7d94fb6f28749
A04ac6d98ad989312783d4fe3456c53730b212c79a426fb215708b6c6daa3de3
```



## Attempts to run Cryptominers

Another very known malware threat of recent years are cryptominers. Many opportunistic attackers are using the compute resource of the machine they exploited to mine cryptocurrency. We also saw hundreds of attempts to install cryptominers on our machine.

### Miners detected:

```
B9e643a8e78d2ce745f7e73eb505c8a0cc49842803077809b2267817979d10b0
28516b0407f1bef5de782d7bf916a9a7cf692ef66261768efae4423e93efe280
3a43e9ceededc2d3b8bae8f8fcff8c539047cdacdd315ebef3adc6651117325e
94f2e4d8d4436874785cd14e6e6d403507b8750852f7f2040352069a75da4c00
```





# 4

## HTTP Honeypot

For our HTTP honeypot, we created a machine with an open port of 80 and a simple webpage. The secret key was under the following path: "http://<IP>/credentials.html". We created a web server that returned an AWS key in a webpage when accessing it via tcp/80 or tcp/8080.

The first access to the machine was **within 3 minutes**. Most of the access we saw was reconnaissance on the website and some were attempts to find potentially vulnerable web pages. We never saw usage of the key that we exposed.

### Top URIs accessed

URI	Count
/	7,114
/.env	3,433
/users/sign_in	2,056
/assets/favicon-7901bd695fb93edb07975966062049829afb...	777
/robots.txt	388
/.well-known/security.txt	325
/sitemap.xml	322
/favicon.ico	261
/assets/touch-icon-iphone-5a9cee0e8a51212e70b90c87c12f...	244
/assets/touch-icon-ipad-a6eec6aeb9da138e507593b464fdac...	202



It took attackers..

# 3 mins

to **access** our HTTP honeypot

From the URIs we can see that attackers were searching for standard HTTP paths that could help with typical HTTP exploitation attacks and for credentials that could be relevant. However they were not searching for an exposed secret, which is why they did not discover or use the key.



# 5

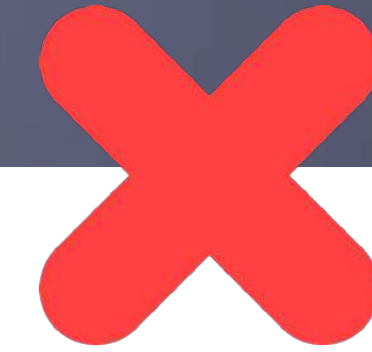
## DockerHub

For our Docker honeypot, we created a Docker image that builds a container with an AWS config file that contains keys, then published it in a [public Docker repository](#) and waited..

However, the Docker image was never downloaded.

We think the reason for this is that scanning Docker is a **far harder task** than for instance scanning GitHub. In GitHub, all an attacker needs to do is scan the content that is visible on the website. However on DockerHub, an attacker would need to actually download the Docker image before they are able to access any information.

This is probably why there does not seem to be automated reconnaissance on DockerHub.



**No attackers  
accessed our  
Docker image**

The cost/benefit ratio is far less attractive on DockerHub and this is why attackers are less likely to target it.





# 6

## ECR Honeypot

We created a [public registry](#) in Amazon's Elastic Container Registry (ECR) with names we believed would attract the interest of attackers:

- 1 [Production-app-new](#)
- 2 [Evelyn-image](#)
- 3 [images-uploads](#)

The images contained an embedded AWS key. For two months, we waited for someone to download the image and use the keys. After no actors took the bait, we decided to post a question in [Stack Overflow](#) about the image stored in ECR with the details and authentication of the image.

The question was immediately reviewed 20-30 times, then received more views over time. After 4 months, we observed two actors downloading the images and initiating calls with our keys. One of the keys was used by an IP registered in the Asia Pacific region, another one by an IP registered to Microsoft.



amazon  
ECR



It took..

# 4 months

before keys were  
used

It appears that there is not much automated reconnaissance on ECR itself, but if digital breadcrumbs are found, like on Stack Overflow, actors come and scope out the ECR target.



# 7

## Elasticsearch Honeypot

Elasticsearch is a popular data analytics and visualization program. In the default configuration, the API endpoint for Elasticsearch is on tcp/9200 and is unauthenticated. This combination makes it easy for an actor to access data if the API endpoint is open to the Internet.

For our honeypot, we created a machine that was publicly accessible on the well-known API port, enabling easy access to query the indices and contents of our server. This Elasticsearch instance had one index called keys, with one entry that was an AWS secret key. While the key stored in our Elasticsearch instance wasn't accessed or used, the instance itself was scanned repeatedly.

```
[devenv2] ~/r/orca >>> curl -XGET "http://[redacted]/keys/_search?pretty"
{
  "took": 5,
  "timed_out": false,
  "_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": {
      "value": 1,
      "relation": "eq"
    },
    "max_score": 1.0,
    "hits": [
      {
        "_index": "keys",
        "_id": "c2jQiIMBX7Bgm9GdSmJ1",
        "_score": 1.0,
        "_source": {
          "key_type": "index_test",
          "service": "aws access key",
          "key_value": [redacted],
          "secret": [redacted]
        }
      }
    ]
  }
}
```



It took..

# 2 hours

before our Elasticsearch honeypot was **accessed**

Most of the queries looked like scans and we didn't see any reference to our 'keys' index. Because there was very little indexed data, attackers may have decided that our instance wasn't very interesting.

### Top queries on our honeypot

Query	Count
/	1,666
/_cluster/stats	490
/_nodes	465
/favicon.ico	385
/_stats/indexing,store,docs	358
/_all/_mapping	95
/_cat/indices?format=json	95
/_search	66
/.env	56
/_aliases	31



8

## Amazon EBS (AMI) Honeygot



Amazon Elastic Block Store (EBS) is a block storage service that allows you to use EBS Snapshots with automated lifecycle policies to back up your volumes.

Our Amazon EBS honeypot was a snapshot of an AWS Elastic Cloud Compute (EC2) virtual machine with access keys embedded in the snapshot. This Amazon EBS backup of the VM was configured to be publicly accessible without authentication. We placed the keys in the regular `.aws/credentials` directory of a VM.

The snapshot was not downloaded, and we saw no usage of the key configured in the image.



**No attackers accessed  
our Amazon EBS  
honeypot**

Since there were no download requests, we can assume that there is **not much automated reconnaissance** on Amazon EBS.



EBS Snapshot



Amazon Machine Image (AMI)



9

## Redis Honeypot



Redis is a popular in-memory data structure store that enables very fast access to stored data. In a default configuration, Redis exposes tcp/6379 on localhost for access to the store with an assumption that access is trusted.

It's possible to misconfigure the service and expose the service port to the Internet, as we did, and this enabled actors to access our honeypot without requiring any authentication.

The Redis server was accessed **within 2.5 hours** of creation, but the keys in our Redis honeypot were never used.

More than half of the attacks on our Redis port originated from China (52%), a third of the attacks originated from the United States (32%), and 7% from Russia. The rest of the regions accounted for 9% in total.

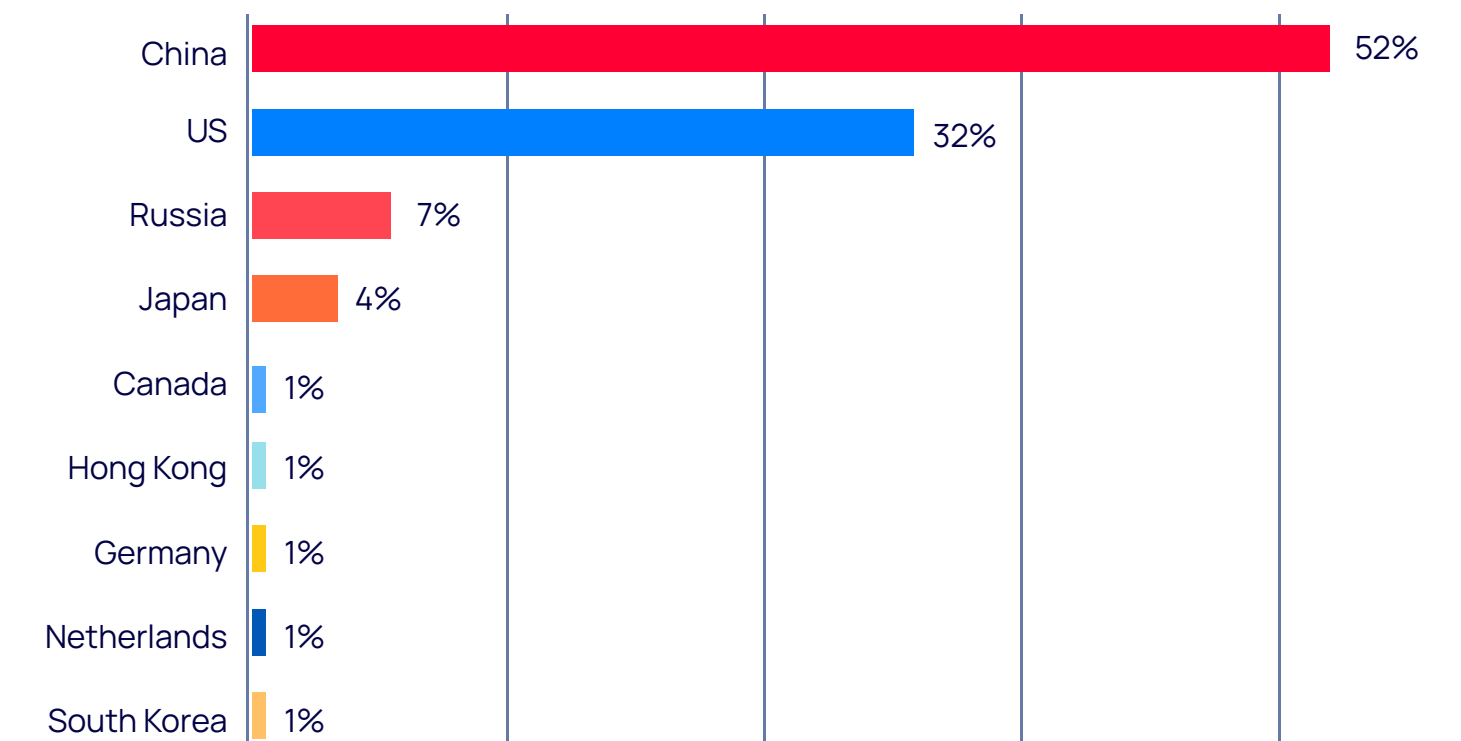


It took attackers..

**2.5 hours**

to discover and access our honeypot

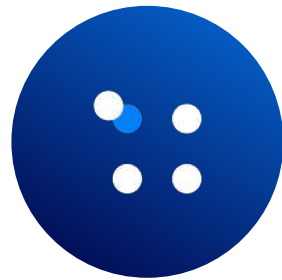
Attacks originated from:







# Redis Honeypot - Tactics



The two calls that the attackers used the most are 'NewConnect' and 'Closed', which are used for straightforward session management. Additional calls, such as 'INFO' and 'CLIENT LIST', appear to be reconnaissance to understand more about the asset.

Some of the other calls indicate more nefarious attempts. For example, 'MODULE LOAD ./red2.so' is [associated with a campaign](#) to exploit exposed Redis servers in order to run a cryptominer.

This confirms that attackers are actively looking for misconfigured and exposed Redis servers in order to run cryptocurrency mining operations.



## Top Calls on our Redis honeypot:








Action	Count
NewConnect	15,491
Closed	15,387
info	13,458
slaveof NO ONE	754
CLIENT LIST	466
SCAN 9000	442
config set dbfilename dump.rdb	387
MODULE LOAD ./red2.so	384
MODULE UNLOAD system	384





# Summary

A comparison of access times, key usage, top actions, and attack vector popularity for each of our honeypots.

 Resource	 Type	 Time to Access	 Top Access Action	 Time to Key Use	 Top Key Action	 Attack Vector Popularity
GitHub	Service	2 minutes	N/A	2 mins	GetCallerIdentity	High
HTTP	TCP Port	3 minutes	Reconnaissance	No use	N/A	High
SSH	TCP Port	4 minutes	shell	No use	N/A	High
AWS S3 Bucket	Service	1 hour	HeadBucket	8 hours	GetCallerIdentity	High/Medium
Elasticsearch	TCP Port	2 hours	Reconnaissance	No use	N/A	High/Medium
Redis	TCP Port	2.5 hours	NewConnect	No use	N/A	High/Medium
Elastic Container Registry	Service	4 months	Reconnaissance	4 months	GetCallerIdentity	Medium
Amazon EBS (AMI)	Service	No access	N/A	N/A	N/A	Low
DockerHub	Service	No access	N/A	N/A	N/A	Low

POPULARITY



## Summary

# Why are some resources targeted more than others?

Attackers run their operations like a business. It basically comes down to: “Where can I get the best bang for my buck?”



### Cost/benefit ratio:

The easier the discoverability of the resource, the more attractive the resource will be for attackers:

- For instance, on GitHub it is easy to discover public repos and new commits in those repos.
- If the asset is exposed to the Internet via a TCP port, such as HTTP, Elasticsearch, Redis, and SSH these systems can be found efficiently via a resource like Shodan.
- However, for S3 buckets, there is no way to query all S3 buckets in existence and there is no unauthenticated way to query all of a particular account's S3 buckets; instead, a dictionary attack approach is required, cycling through a space of potential bucket names to find publicly accessible ones, something which takes more effort, even though it can also be done in an automated way.



### How much the resource is used:

The more users of a resource, the more chance of finding potentially useful data.



### How prone it is to contain secrets:

GitHub, for instance, is very prone to contain secrets since it contains all the source code of a project, sometimes even of an entire organization. Other resources are less prone to this.



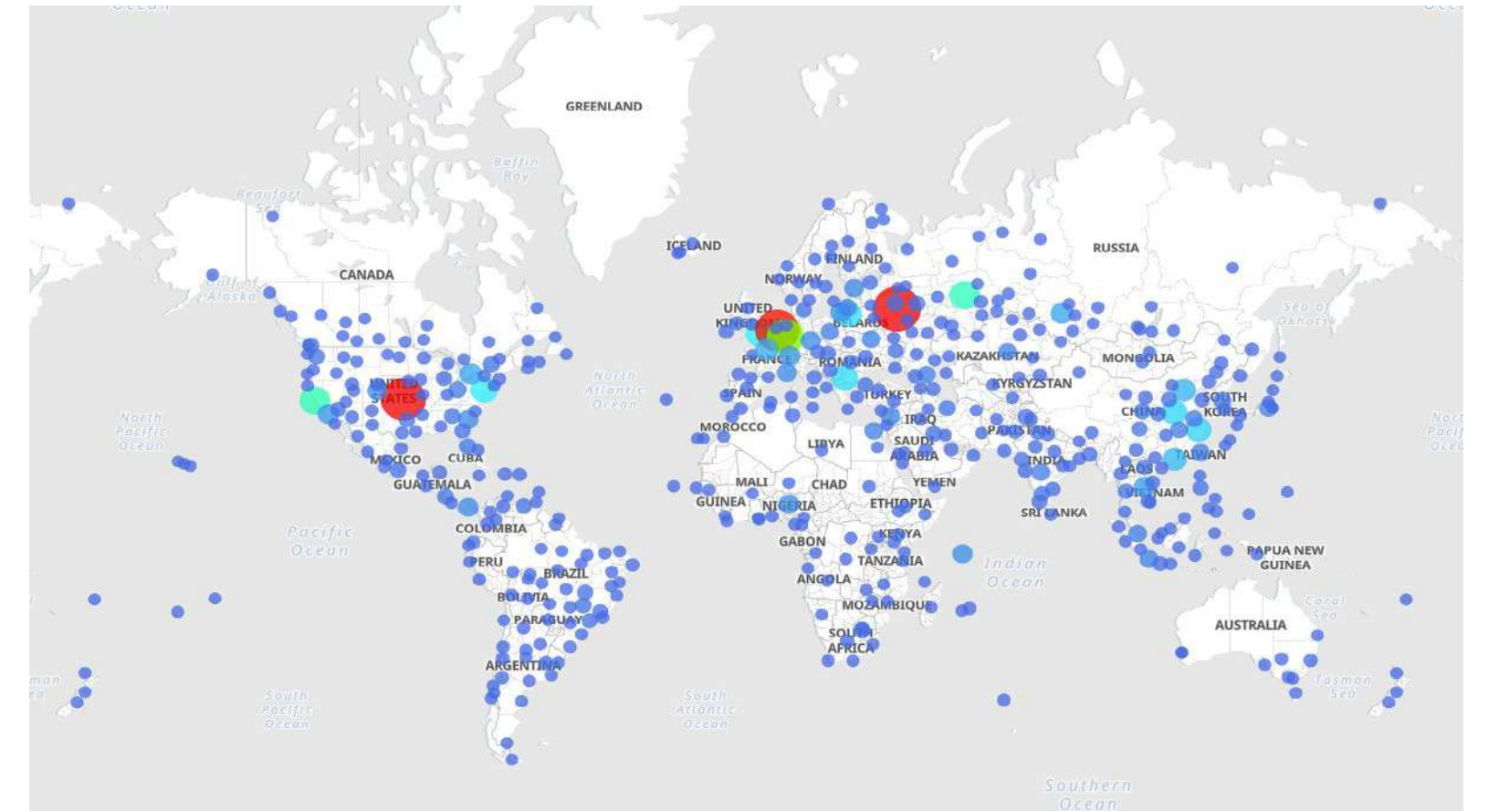


## Summary

# Attacker Heatmap

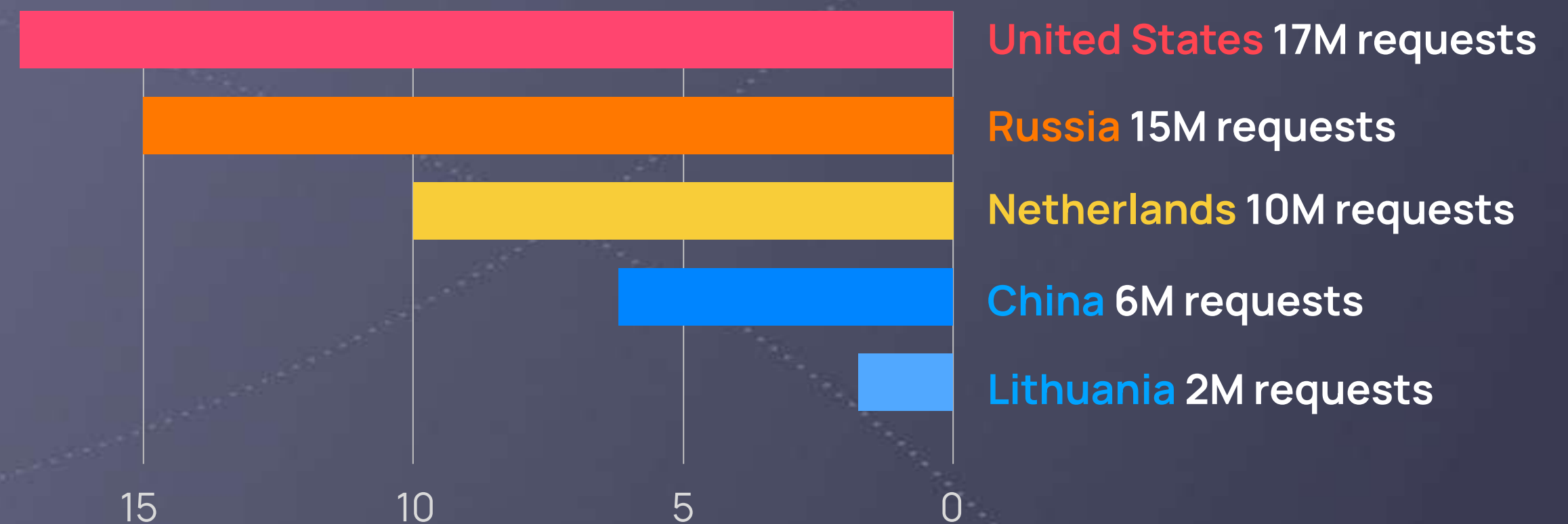


IP origins  
of our honeypot  
attackers



Most attacks originated from IP addresses in the US, which makes sense because most IPs are from the US. However, in second place is Russia, indicating that a fair number of attacks are originating from there. Surprisingly, third in the list is the Netherlands.

## Top 5 countries





## Summary

# Targeted Ports

Port scanning is a method employed by penetration testers and malicious hackers to examine publicly accessible devices on the Internet. Its purpose is to identify which applications or services are actively operating on the network, often with the intention of launching targeted attacks.

The top targeted ports in attack and scanning attempts on our SSH, HTTP, Redis, and Elasticsearch honeypots were port 5900 Virtual Network Computing (VNC) and port 23, as both are easily exploitable.

## Port 5900

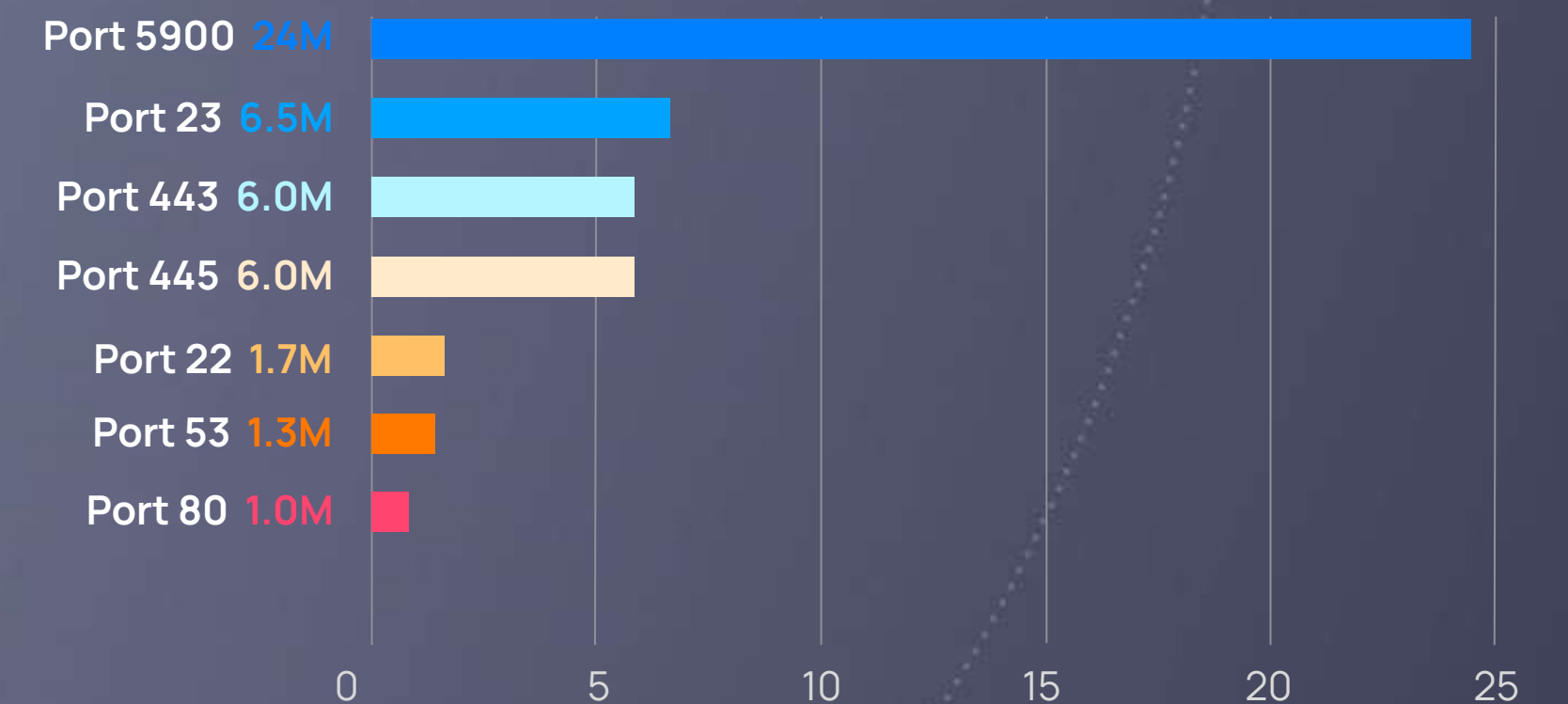
is associated with the VNC service, and is often targeted by attackers since many users fail to change the default configuration of port 5900, leaving their VNC servers vulnerable to asset compromise and data leakage.

## Port 23

is associated with the Telnet protocol, which connects users with remote computers. While Telnet has largely been replaced by SSH, some websites still use it. However, due to its outdated and insecure nature, Telnet is susceptible to various attacks such as credential brute-forcing, spoofing, and credential sniffing.



## Top targeted ports



### #1

## Port 5900

24M scans

### #2

## Port 23

6.5M scans



## Summary

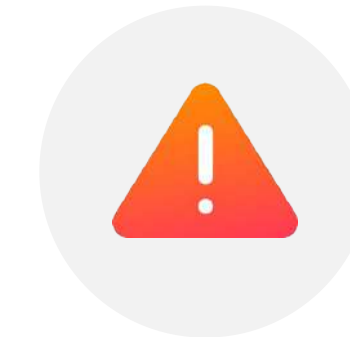
# Breached Key Reporting is Limited



With its push protection, GitHub scans for 1,000 types of exposed keys and if it finds one (even if it's in the Git History), the commit is blocked. We can confirm that the permissions of the leaked keys were locked down as soon as the git push occurred - using the [AWS Compromised Key](#) policy, which denies access to destructive actions.

However, except for the breached keys from AWS in GitHub, none of our exposed keys were reported as breached, despite the fact that some of them were used by unauthorized users.

This means that defenders need to be **extra careful** not to include secrets on S3 Buckets and Elastic Container Registry (though to a lesser extent), since we also saw relatively fast key usage from these resources.



Even if key permissions are locked down (as they were on GitHub), the key is **not entirely blocked**. Although the policy denies most permissions, an attacker can potentially still perform malicious actions on some services, such as RDS, EKS, and Elasticsearch.





# Key Recommendations

So how can defenders up their game and stay one step ahead of the attackers? We have listed our key recommendations below:



01

## Identify & Manage Your Secrets

If you don't properly store your secrets, it's just a matter of time before attackers will find them. Combining a strong secret management approach using vaults with automatic identification of where secrets are stored, enables you to reduce the chance a secret is compromised and identify where sensitive data & secrets are inadvertently exposed. With Orca's [data classification](#) and [attack path analysis](#), you can identify that exposure even if, unlike our honeypots, it would require an attacker to exploit multiple weaknesses in the environment to get to the data.



02

## Check for Secrets Before Deploying Code

It's essential to scan for secrets prior to committing code. For developers working in a platform like GitHub, in particular, it may make sense to combine [Orca's Shift-Left secret scanning](#) with [pre-commit hooks](#), allowing you to scan code on a developer's workstation before it's pushed to a platform like GitHub. When it only takes attackers minutes to find your mistake, it makes sense to go to great lengths to prevent this sort of exposure. Even if GitHub does include push protection, it's better to be safe than sorry and not rely on GitHub to find your secrets, since the stakes can be high. In addition, even though the permissions of our honeypot key were greatly reduced, this does not mean that an attacker could not still do damage using the remaining permissions.



03

## Check Your Git History

Don't forget that attackers are not only scanning new GitHub commits, but are also looking for secrets in your Git History. So make sure that when you do remove a secret from a commit, you also remove it from your history. The Orca platform detects when keys and other [secrets are in your Git History](#) and need to be removed.



# Key Recommendations

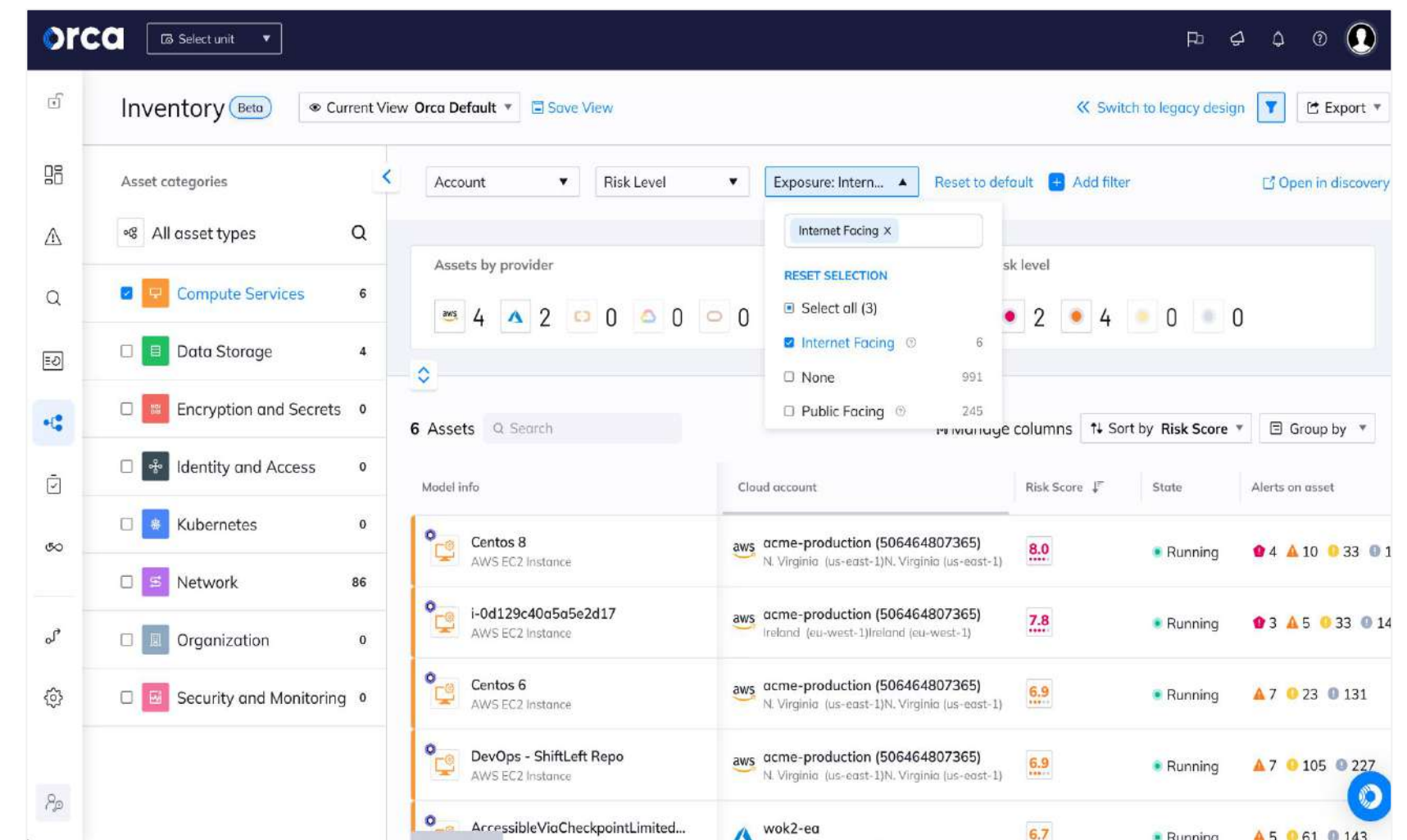


04

## Set Public Access Only When Strictly Needed

Although it may seem obvious that you should limit public access to cloud assets whenever possible, unfortunately, in real-world deployments, accidental public exposure is common enough that attackers are scanning for many of these exposed resources *all the time*. Organizations should continually assess which resources in their cloud estate are publicly exposed and ensure that there is an important business reason for doing so.

To further reduce the chances of human error opening something to the outside world, it also makes sense to lock down cloud environments so that changes are made as code (“Infrastructure as Code”, or IaC). This can be combined with [security policy scanning of IaC artifacts](#) to ensure that mistakes are caught & addressed before they’re deployed in the cloud.



*Having a cloud security solution in place that can alert when assets are publicly exposed is essential.*



# Key Recommendations



05

## Follow “Security by Obscurity”

We saw that attackers often take an opportunistic approach and try to guess usernames, passwords, and asset names by simply going down a list of the most commonly used names. For instance, with our SSH honeypots, attackers were simply trying the ones that were the most commonly used.

As defenders, it is therefore recommended to use obscure usernames and asset names that cannot easily be guessed (admin as a username should be off the table, and customer-database is not a great name for a bucket either). Although this should of course *not be your only security recourse*, the more difficult you can make it for the attacker, the sooner they will get tired and move on to the next target.



06

## Bolster Authentication and Limit Authorization

One of the commonalities across our honeypots was that they had weak authentication – while this is what we wanted in a honeypot, it is a common contributing factor in security incidents. For those assets that must be exposed to the Internet, ensure that strong authentication is enabled. These options differ from service to service but may include certificate-based authentication and, if the service is accessed directly by humans, multi-factor authentication (MFA).

Additionally, where possible, ensure that the accounts that connect to the service have the minimum possible access. Limiting the scope of authorization can help to reduce the impact of an attacker managing to authenticate to the service.





# Key Recommendations



## Monitor for Malicious Process Execution and Malware

For those services where process execution is a possibility, monitor the systems for abnormal execution and potentially malicious files & processes. SSH is a particular risk here as it is, explicitly, a shell for executing things. However, other services may also explicitly or, via a vulnerability, allow an attacker to execute code. In our Redis honeypot, we saw evidence of an attempt to execute a cryptominer.



## Assess and Patch Vulnerabilities

Vulnerability assessment and patching should be a priority for assets directly exposed to the Internet. Common Vulnerabilities and Exposures (CVEs) that allow for remote code execution or privilege escalation may be used by attackers to bypass many of our other recommendations.



## Implement Port Hygiene

Determine which ports are necessary for your specific applications and services to function properly and limit access to only those applications. If you are using cloud providers like AWS, Azure, or Google Cloud, configure security groups or firewall rules to limit access to only required ports and restrict access from any unauthorized sources.

The screenshot shows the Orca Security dashboard with a list of alerts. The top section shows 396 alerts, with 12 high severity alerts. Below that, there are 96 medium severity alerts, including 56 alerts for 'API access from malicious source IP was detected'. A table lists several of these alerts:

Alert	Last seen	Discovered	Asset	Account	Action
6.3 API access from malicious source IP was detected	3 days ago 2023 Jun 06, 13:15	2 days ago 2023 Jun 07, 11:43	project-r-312719 CloudAccount	project-r-312719	TAKE ACTION
6.3 API access from malicious source IP was detected	4 days ago 2023 Jun 05, 19:46	2 days ago 2023 Jun 07, 11:43	project-r-312719 CloudAccount	project-r-312719	TAKE ACTION
6.3 API access from malicious source IP was detected	3 days ago 2023 Jun 06, 13:15	2 days ago 2023 Jun 07, 11:43	project-r-312719 CloudAccount	project-r-312719	TAKE ACTION
6.3 API access from malicious source IP was detected	4 days ago 2023 Jun 05, 19:46	2 days ago 2023 Jun 07, 11:43	project-r-312719 CloudAccount	project-r-312719	TAKE ACTION
API access from malicious source IP was detected	2 days ago	8 days ago	orca-demo-01	demo-01	TAKE ACTION

*[Cloud Detection & Response](#), which automatically detects anomalies and suspicious events in cloud environments, should be deployed to alert security teams to possible attacks in progress.*



# Key Recommendations



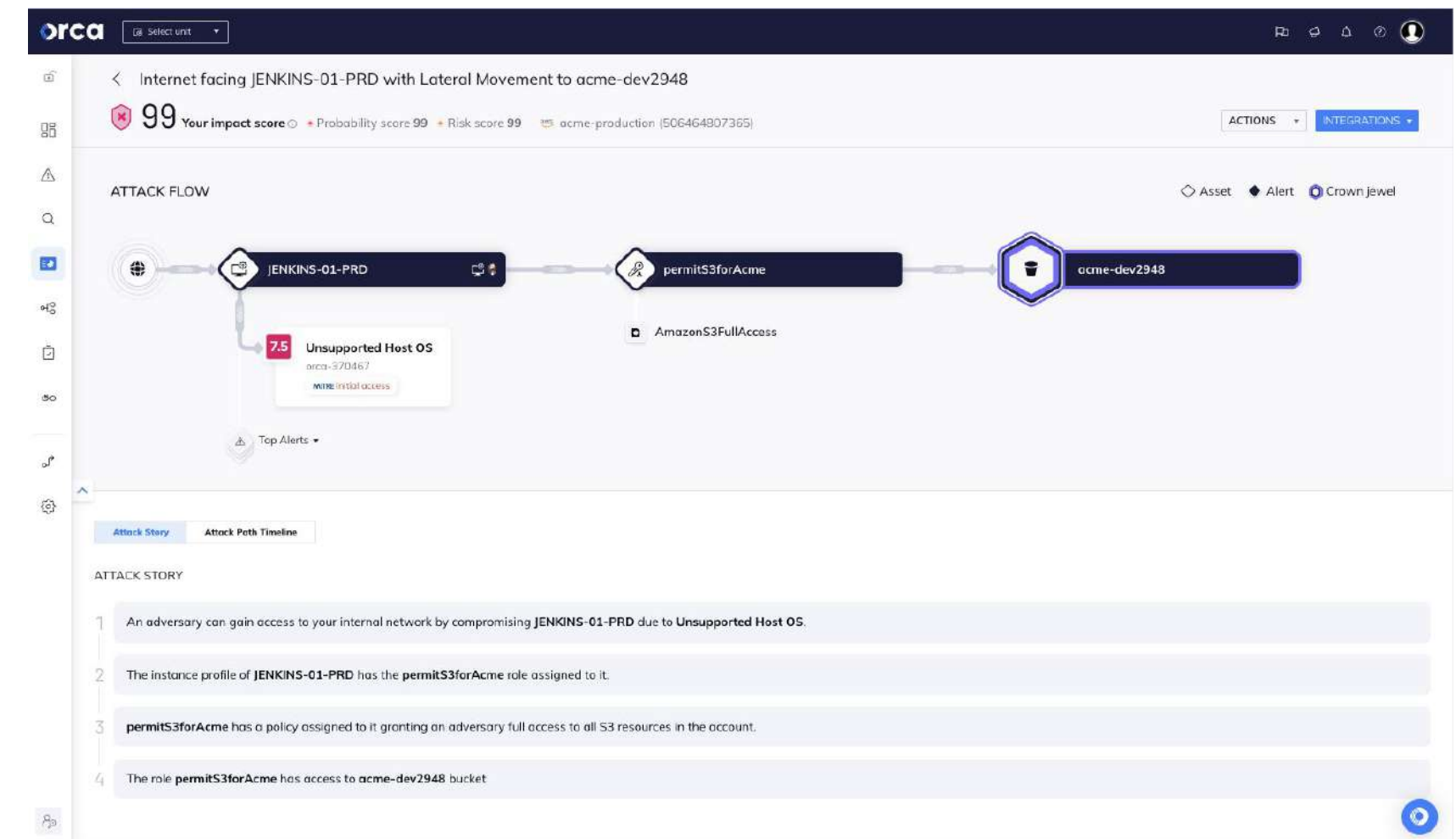
## Prioritize Protection of Your Crown Jewels

We know that attackers are continually scanning cloud environments looking for vulnerable resources. Unfortunately, it's just a matter of time before attackers are going to find a vulnerability in your defense.

Therefore, instead of trying to fix all vulnerabilities and remediate all risks, which is frankly a Sisyphean task, a better strategy is to ensure that your crown jewels, such as PII, intellectual property, financial information, and other sensitive data, are protected. Any risks that endanger your crown jewels should always be prioritized and fixed first.

However, this does not mean that you only focus on risks that are directly connected to your crown jewels. Attackers will take advantage of different weaknesses in your environment to move laterally and ultimately reach their target.

Therefore, it's essential that security teams have insight into the different [attack paths](#) (i.e. combinations of risks) that endanger the organization's crown jewels and then ensure that these attack paths are deactivated in the fastest and most effective way.



*Attack paths show the combinations of risks that are a direct path to your critical assets*



# About Orca Security

Orca's agentless cloud security platform connects to your environment in minutes and provides 100% visibility of all your assets on AWS, Azure, Google Cloud, Kubernetes, and more.

Orca detects, prioritizes, and helps remediate cloud risks across every layer of your cloud estate, including vulnerabilities, malware, misconfigurations, lateral movement risk, API risks, sensitive data at risk, weak and leaked passwords, and overly permissive identities.



Watch a [recorded demo](#) or take our [free cloud risk assessment](#).

