How to Get Your AWS Account Hacked

(And How to Avoid It)

The phrase “innovate or die”, which years ago seemed a little radical, has now taken instant roots in many executives’ minds – mainly due to the aftermath of the Covid-related lock-downs that sparked a massive increase in the need for digital solutions

A McKinsey survey of global executives showed that digitization of their companies in terms of their customer and supply chain interactions as well as internal operations have accelerated by three to four years.

But in the process of refreshing your systems, upgrading your software and moving to become more digitally apt – all at a faster rate than what the status quo typically saw pre-pandemic – you might not always give every aspect that comes with that change the proper attention.

One often overlooked aspect is the security of your IT solution. According to the 2021 Global Threat Intelligence Report, 30.000 websites are hacked every day. Over 90% of these security breaches stem from misconfigurations and can be prevented just by having the right setup in place.

In this e-book, I’ll show you how I was able to easily access real AWS accounts through a number of access points, loopholes and weak points in 3 applications typically used to work with AWS solutions: Atlassian (company), Jenkins (application) and Kubernetes (framework). I’ll show you the process by which this can happen and what you can do to prevent yourself from falling victim to a hacker attack.

„The ultimate security is your understanding of reality.”

H. Stanley Judd

Contents

How to Get Your AWS Account Hacked

01

A Case for White Hat Hackers

The Trick to Covering all Bases

The need for solid security is especially true for Cloud solutions, because your data is constantly flowing from one point to another. If not secured properly, that data can be vulnerable to exploits on their way from endpoint to endpoint.

There are people who may want to cause not only disruption to your website, communications and applications for financial gain, but also to damage your reputation. They operate in the dark, away from the mainstream, and specialize in exploiting loopholes and weak points.

So, the million-dollar question is: where are the weak points in your system?

The CISO of Société Générale, Stephane Nappo, once said that "one of the main cyber-risks is to think they don't exist." It can indeed be tricky to identify security threats using conventional testing methods, such as unit tests, integration tests, smoke tests and regression tests. More often than not, these tests – typically conducted from inside your own organization – will not always manage to realistically create the conditions that really exist when someone is looking to get in from outside of your organization.

It takes a thief to catch a thief goes the old saying. This is also true in the digital realm. Coupled with a sound security strategy, this principle forms the basis of ethical hacking, also called white hat hacking.

What is White Hat Hacking?

Contrary to common belief, hackers aren't always malicious!

We generally differentiate between black hat and white hat hackers – all of whom usually have deep knowledge about breaking into computer systems and bypassing security protocols. The difference lies in the intention.

A white hat hacker is a security expert who uncovers security risks in a software, which they then report to the owner in order for them to put in place the relevant security measures; a white hat hacker always operates with good intentions.

What exactly are those intentions? To raise the overall security of your systems

And then there's the hackers we all know from the movies: black hat hackers. They're security experts who intend on exploiting uncovered security breaches to profit from it themselves – naturally, without prior consent from the system owner. Their primary goal is to extort, destroy reputations and other nefarious things – I'm sure you've seen the headlines.

An Important Piece of the QA Process

It's a cat-and-mouse game: white hats try to uncover systems exploits and have them fixed before black hats can take advantage of them.

External white hat hacking should be an essential part of an organization's QA process, because it closes a gap that internal penetration testers can't bridge – even if realistic circumstances are simulated. That's because they'll typically work within a framework, given certain assignments and, most importantly, they often won't be personally motivated to dig really deep.

In my role as security researcher, I often put on the old white hat and search the internet for vulnerable software. During my search I'll find just that – misconfigured machines. The trick lies in finding the owner of a given machine. That's the key challenge that I – similarly to my counterparts, the black hat hackers – try to solve.

The CTO of one of the companies I helped plug certain security gaps put it like this: "It's like you stumble upon a house with an open door, walk right in, find a picture of the owner and their number written on the back. Then you call them and say, hey I found your stuff, might want to lock your doors."

The difference between internal and external pen tests is that when you do penetration testing from inside an organization, you'll usually just report vulnerabilities without checking for credentials, because you know who the house belongs to. Yet checking credentials is exactly what black hat hackers do – their primary target will often be unknown to them at first. They'll try everything to find those valuable credentials. And when they do, they can begin making a profit.

Like I said, it takes one to know one!

02

Atlassian

On my quest to learn more about effectively securing Cloud solutions, I sometimes do some penetration testing on some unsuspecting companies to see how secure a given software is. Usually, I'll find interesting topics in the Cyber Security Subreddit and exploit db and then decide to explore something more in depth.

On a recent "recon raid", I went through some old vulnerabilities in Atlassian Software that were already uncovered in 2018 to see if some companies were still vulnerable. Back in 2018, this bug was quite widespread, and many companies scrambled to fix that security gap – but not all, as it turned out.

The Method

I started by searching the internet using search engines like Shodan, Censys or BinaryEdge, which index responses from various ports on all IPs and constantly monitors the internet to index vulnerabilities. During my search, I found multiple machines that were running outdated Jira software.

The thing with Jira versions below v7.3.5 is that they contain a vulnerable open proxy that can be used anonymously to return any data from internal networks.

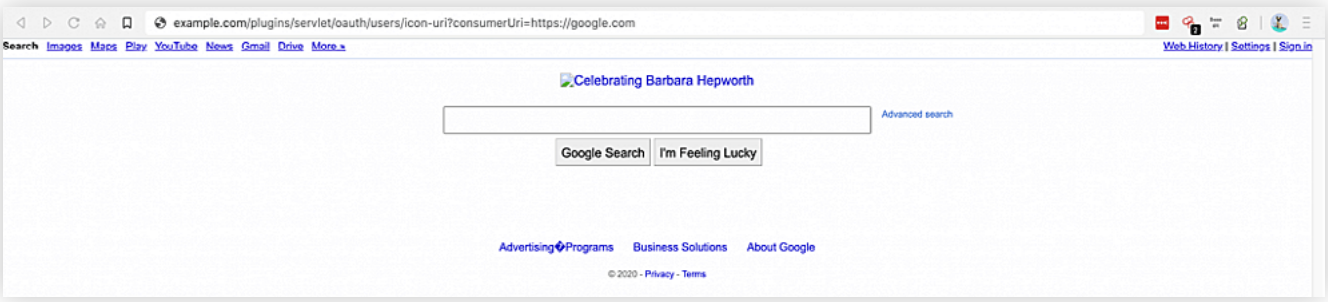
Let's assume the Jira server works under the following address:

https://example.com/secure/Dashboard.jspa

So, I knew that Jira in this version contained that bug, but just to make sure, I checked the vulnerable plugin to confirm that the vulnerability still existed in this machine. And it did!

Proof of concept – I tried to open google.com under example.com domain:

https://example.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://google.com



Then I made sure the machine ran on AWS by checking the DNS name. It said that this IP belongs to EC2 instance on AWS, so I was on the right track – and delving deeper into familiar territory.

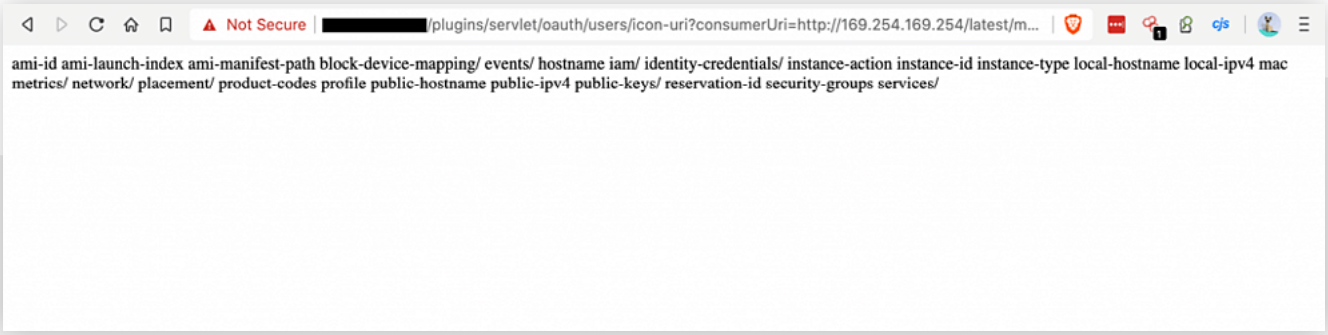
Now, I know that an EC2 instance can query the local address for meta data.

Meta data contains details of the EC2 instance like AWS Region, Availability Zone, SSH key name – and when there is any IAM role attached, you can even extract temporary credentials!

The EC2 instance generates these temporary credentials in order to communicate with other AWS services. You can extract them and use them from your local machine.

So that' what I did! I simply asked for the EC2 metadata.

https://example.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/



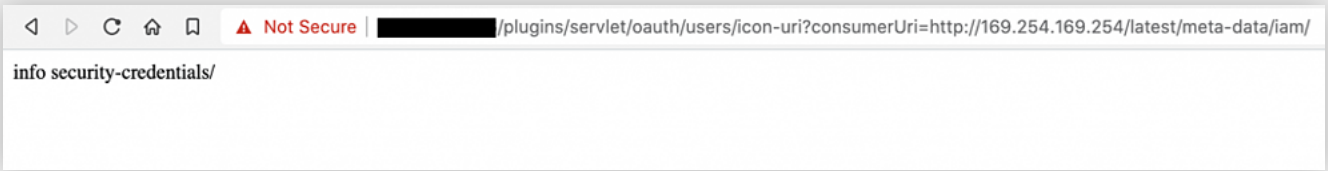
Policies and how they can be exploited

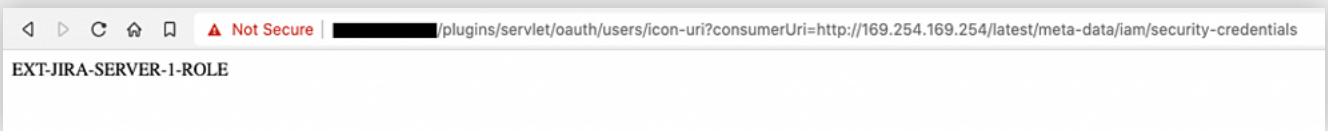
Depending on the policies used by the role attached to the EC2 instance, you can do different things with it. If a given role only lets you send logs to CloudWatch, you could for example flood CloudWatch with weird data to confuse it and force an instance restart or initiate autoscaling.

But sometimes people make the mistake of not following the least privilege principle, i.e. not allowing a role only as much access to information for them to be able to do their job.

So, it could be that a given role has AdministratorAccess policy attached, and then the JIRA instance can be used to do pretty much anything: create resources, modify resources, terminating resources, encrypting, decrypting, accessing billing details, etc.

The meta data returned an "iam/" endpoint, which is available only when a role is attached to the instance.





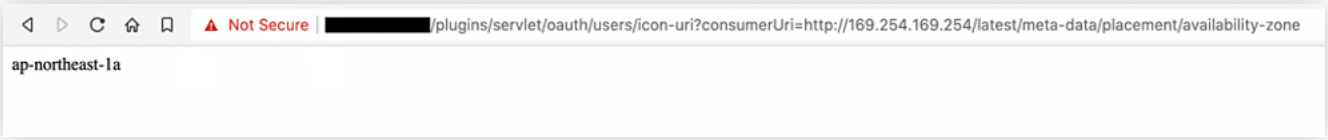
Asking for the temporary role credentials gave me fresh AWS access, secret keys and a session token:

https://example.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/iam/security-credentials/EXT-JIRA-SERVER-1-ROLE


```
{
  "Code": "Success",
  "LastUpdated": "2020-06-08T05:26:05Z",
  "Type": "AWS-HMAC",
  "AccessKeyId": "ASIA6CENSOREDAOO7JQB",
  "SecretAccessKey": "lJJxM9K5ExXR-CENSORED-8367E9vQ",
  "Token": "IQo-CENSORED-sDaRGP0g==",
  "Expiration": "2020-06-08T11:35:07Z"
}
```


It's also good to know the region in which resources are created, because it's required when you want to use AWS CLI. This can also be extracted from metadata:

https://example.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/placement/availability-zone



With all this information, you can use AWS CLI to check what actions are possible using that role.

My script initially simply asked for the S3 Buckets list, the IAM users list and billing information from last month.

All requests were successful. That usually means the role has an AdministratorAccess policy attached, because usually only administrators can access billing.

To identify the owner, I additionally asked for AWS Organization details and Route53 domains.

As mentioned before, using the Jira machine, you can access the internal network it's located in. All private hosted zones (example.local) are also open to you.

The different Atlassian applications that may run in the internal network only – like Confluence or BitBucket – are now open to me. Sometimes there was even no login needed because it was a local resource, and since I'm accessing it from Jira, the system thinks I'm on the internal network.

With this kind of access, a black hat hacker could:

- create an IAM user for themselves,
- create additional access keys for existing users,
- attach admin roles to other vulnerable services,
- copy objects from S3 buckets,
- create AMIs from running instances and start new ones with their keys to log in,
- share AMIs with other AWS accounts,
- remove everything from the account,
- leave the AWS Organization,
- remove CloudTrail logs,
- and more...

Money well Spent, but not well Secured.

The above list illustrates the many options a black hat hacker has to mess with your AWS account.

On the long list of leaked access keys I found, some gave me access to accounts spending nearly $100,000 a month on AWS.

Here's a breakdown of the costs incurred, taken from the billing information:

Monthly Spending on AWS in $:

- AWS CloudTrail: 0.00
- AWS Config: 66.92
- AWS Direct Connect: 25.25
- AWS Glue: 0.00
- AWS IoT: 0.00
- AWS Key Management Service: 45.80
- AWS Lambda: 298.20
- AWS Secrets Manager: 0.00
- Amazon DynamoDB: 24.94
- Amazon EC2 Container Registry (ECR): 0.06
- Amazon EC2 Container Service: 0.00
- Amazon ElastiCache: 1,041.44
- EC2 – Other: 6976.19
- Amazon Elastic Compute Cloud – Compute: 61,008.41
- Amazon Elastic File System: 3.61
- Amazon Elastic Load Balancing: 2,020.69
- Amazon Elasticsearch Service: 1,686.58
- Amazon Kinesis: 145.08
- Amazon Relational Database Service: 2,270.92
- Amazon Route 53: 1,148.44
- Amazon Simple Email Service: 2.67
- Amazon Simple Notification Service: 0.01
- Amazon Simple Queue Service: 0.00
- Amazon Simple Storage Service: 1,770.76
- Amazon SimpleDB: 0.00
- Amazon Translate: 0.08
- Amazon Virtual Private Cloud: 559.91
- Amazon CloudWatch: 345.10
- Tax: 7,673.75

Total monthy spending on AWS: $87,114.84

As you can see, this company is using a lot of AWS resources.
However, their security measures could definitely be improved!

Finding the Owners

To find the responsible organization, there are a few ways you can go about it. Here's are some methods I frequently use to find identities.

After finding the owners of the Jira machine, I contacted them to inform them of the gap in their security. They said that my discovery truly shocked them and the vulnerability was immediately fixed.

They also said they would introduce some changes in their cloud security policies and reassess their access management.

It's a good practice to thank for the person for the responsible disclosure with a Bug Bounty – a reward given to a security researcher for highlighting security issues. It motivates the researcher to continue his work of helping companies create tighter security processes.

Unfortunately, there was no bug bounty this time!

- Domain name from SSL Certificate
- Domains from Route53 hosted zones
- Domain names from S3 buckets list
- Email addresses on IAM users list
- First name, Last name from IAM users list
- -> LinkedIn
- URLs/Logos on Jira login page
- AWS Organization master payer account root email
- AWS account root user email
- Responsible disclosure email address from contact page
- Data protection email address from privacy page

Atlassian Security Tips

So, what went wrong in this example?

1. Broken principle of least privilege – Any user, program, or process should have only the bare minimum privileges to be able to perform their function. In this case, the administrator role attached to the EC2 gave it unlimited privileges.

2. Outdated software – Software updates can include new or improved features and better compatibility with other devices and applications. But more importantly, they also provide security fixes. In this case, it was a well-known exploit from 2018 that allowed the sensitive information to be extracted this easily. Atlassian fixed it a long time ago, but users who haven't patched their applications are still vulnerable.

Solutions to these are:

1. Apply the principle of least privilege – This would reduce the risk of unauthorized access to critical systems or sensitive data through low-level user accounts, devices or applications.

2. Update your software regularly – Besides new features and greater compatibility, tighter security is a main component of software updates.

Vulnerable apps & versions:

- Bamboo < 6.0.0
- Confluence < 6.1.3
- Jira < 7.3.5
- Bitbucket < 4.14.4
- Crowd < 2.11.2
- Crucible & Fisheye < 4.3.2

I end this chapter with a quote from the great American writer Henry David Thoreau – straight out of his opus magnum, Walden.

„If you have built castles in the air, your work need not be lost; that is where they should be. Now put the foundations under them."

03

Jenkins

Jenkins is an open-source automation server that helps automate those parts of software development that are related to building, testing and deploying. It plays a major role in ensuring continuous integration and delivery.

It is also used as a management tool to help develop pipelines for building, testing and deploying Cloud infrastructure; it's a popular choice for many development teams who work within the AWS cloud.

In this chapter, we'll review potential security issues and establish what you can do to secure your AWS accounts against any unwanted access through Jenkins.

Privileges and the Damage They do

Jenkins requires a lot of privileges to deploy and destroy infrastructure and resources in the Cloud, making it a great entry point for hackers!

That's why most companies usually restrict access by hiding it behind a Virtual Private Network (VPN), making it available only for their employees.

It does happen, however, that a firm's Jenkins server is publicly accessible and that anonymous visitors use it through different access levels – the easiest being read-only permissions.

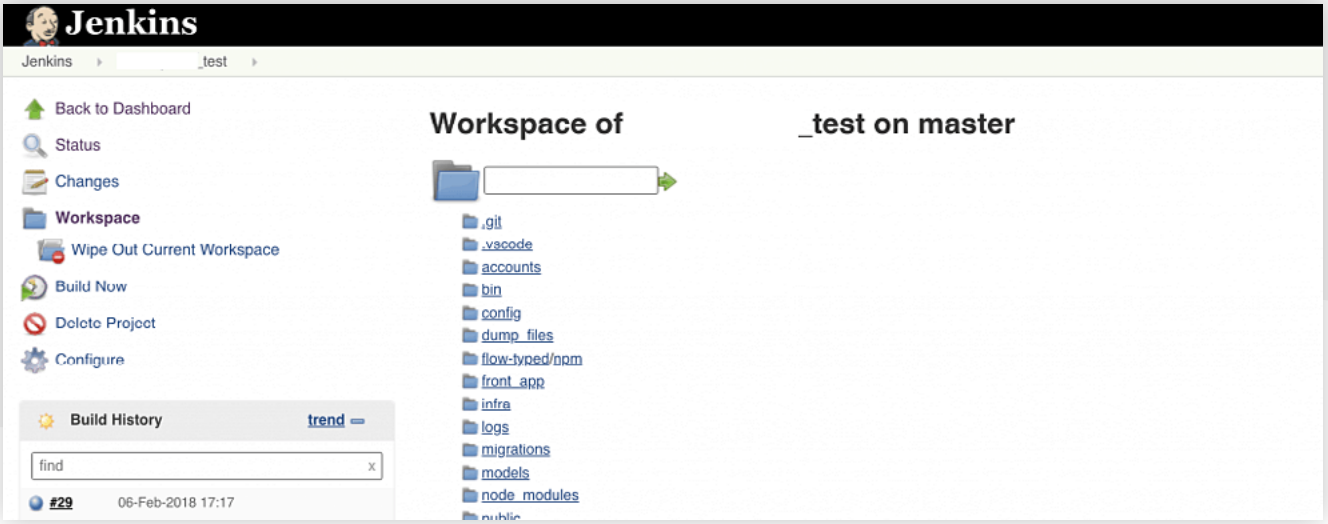
Once again donning my white hat, let me first shed some light on this particular access type.

You'd be surprised what damage can be done with "just" a read-only permission. Access to AWS credentials, data and secret information... these and more can be revealed to you – if you know where to find them.

Let's look at three specific entry points that can grant potential access to some juicy details.

Workspace

Within Jenkins, Workspace gives you an overview and easy access to your project structure and project files.



Here, an unauthorized user can easily find the source code of a given application, which they can then use in conjunction with the read-only permission to preview all project files.

Besides the source code, you can also find configuration files, data dumps, CSV files containing PII or even database dumps.

That's all fine and dandy, but here's the real kicker: you can find plain text AWS credentials in config files or even hardcoded in the application.

Once a hacker finds this, they can do as much as IAM Policies attached this user allow him to.

By the way: another entry point is the Step Workspace. If you work with pipelines, this is where the project structure and files in your pipelines are accessible.

Environment Variables

These are defined for each job individually and are available only during the job run. With the read-only authorization, you can find AWS access keys, logins, passwords and other sensitive information here. The Environment Variables page is available for most already completed jobs.

Console Log

The console log contains the results and output of every command run for each job. You can find a lot of interesting information here, like logs of credentials, endpoints and e-mail addresses.

Example jobs which I found containing sensitive data:

- Creating new AWS IAM Users (and printing access keys);
- Generating temporary AWS access keys using STS (and printing them);
- Testing the application on different access levels (and printing login and password used).

Username/Password allows you, for example, to log into the stage environment where the test was conducted. Now, the stage environment can be connected to a production database. And when that's the case, then we're in it for the big time!

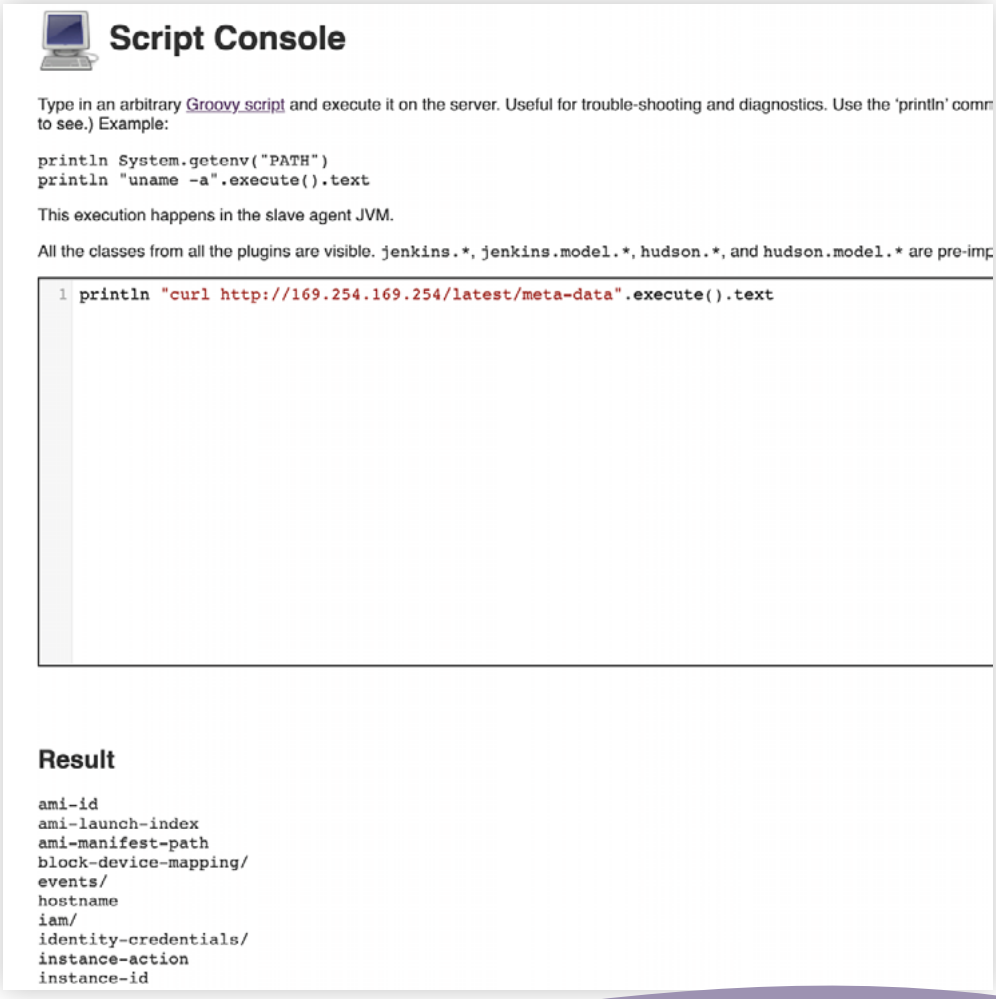
Jobs

Another access level commonly found on publicly exposed Jenkins machines allows any anonymous user to create a Jenkins Job.

A Job could be a script that is then run by the Jenkins machine. This means you can create any script you like, and Jenkins will run it for you.

Do you want to extract temporary access keys from the Jenkins EC2 metadata? No problem:

curl http://169.254.169.254/latest/meta-data/iam/security-credentials/



Do you want access keys stored in Jenkins Credentials? No problem, print them on the screen or send in an email. How about Jenkins using IAM User access keys instead of a role? Great:

cat ~/.aws/credentials

If that's not enough, there's another powerful permission level – Manage Jenkins.

Manage Jenkins

This access type allows you to open “Manage Jenkins” page with all configuration details.

With this permission level, you can preview (and modify!) the current configuration of all extensions. It can give you email addresses, private keys, credentials, git repository addresses and more.

Manage Jenkins also gives you access to a script console. The console may be used to run any scripts without running a job. Attack scenarios are similar to the previous access level with job creation. But the console is also useful to decrypt Jenkins Credentials. Just grab the hash of the interesting credential (inspect the html form field) and run the following script:

https://example.com/plugins/servlet/oauth/users/icon-uri?consumerUri=http://169.254.169.254/latest/meta-data/placement/availability-zone

It will print the value hidden behind selected credential hash.

If you get access to AWS access keys that belong to Jenkins, you can do everything the Jenkins user/role is allowed to do.

The above examples are all true and possible. To illustrate that this does indeed happen, let me tell you a little story of one case I was involved in recently.

How Easy is it to get Access to Your Account?

Xebia was doing a security audit for one of our clients a few months ago. As part of this audit, I was given AWS credentials to the client’s account with read – only access.

Using these credentials, I checked every part of the project infrastructure – and, not after long, I found a Jenkins machine running on one of the AWS regions.

The machine was located in a public subnet and with a public IP. It also looked like it was created using terraform templates. Security group allowed everyone to access it (0.0.0.0/0).

Unfortunately, there was a login page with no anonymous access.

I also found a full log from the provisioning of the Jenkins instance on CloudWatch. Astonishingly, the log ended with the credentials of a newly created Jenkins admin user.

So I opened the Jekins login page once again – but this time I was able to login to an admin account. By checking IAM Role and Policies attached to the EC2, It already knew that Jenkins had an AWS AdministratorAccess policy attached to this role, and so I now have full access to the AWS account.

The Client assumed that their account was secured because users had to log in. They didn’t count on the login data being easily accessible on the CloudWatch log with a read-only permission.

Jenkins' Potential Weak Points

The above access points do not have to be vulnerable. As I mentioned before, cyber security is a strategy that needs to be implemented on all fronts.

> **„If you reveal your secrets to the wind, you should not blame the wind for revealing them to the trees."**
>
> H. Kahlil Gibran

Any security breaches become possible when security measures aren't applied consequentially and effectively. In the case of Jenkins, we can identify four main issues:

Jenkins is Publicly Available

Not having a VPN in place can have serious consequences. The internet is a vast jungle of connectivity that has its shares of predators, and they will not hesitate to jump on easy prey. If not VPN, lock it at least at a Security Group level, allowing only approved IPs to connect. Home IPs of your employees are not considered safe, as a single IP could cover half of a big city.

Jenkins Security is Disabled

Jenkins has some security options. For example, by default it requires you to log in. Yet, setting these up can be cumbersome and time consuming, so some companies choose to disable the security module and... forget about it later. "How does someone know the IP of my newly created Jenkins machine? Only my employees know that, and they don't need to login to do things"

Bad Virtual Host Configuration

Domain access is properly restricted, but IP access is using default vhost with no restrictions. This means that when I try to open https://jenkins.example.com the connection will NOT be permitted. But then if I try with the IP behind this domain, it will use a different configuration file on the server and will let me in.

Broken Principle of Least Privilege

"Give Admin rights to everyone, they may need it at some point". This is a common one – admin rights are given to people because they think that, at some point, they might need it. This can happen when you don't want your technical lead to be bothered every 5 minutes with a request.

How to Avoid These Leaks

The above points can definitely be avoided with the right security measures and strategy. Here are some tips on what you can do so secure your AWS accounts running Jenkins.

1. Make sure that Jenkins is only accessible from a local network or use a VPN
2. Ensure that the security settings in Jenkins are properly set up – check the domain and direct IP access.
3. Only give Jenkins access to your AWS environment as required. The Principle of Least Privilege is your friend.

Xebia

04

Kubernetes

I'm excited to dive right into this one because it's a hot topic, as so many businesses are starting to work with Kubernetes in connection with their AWS accounts!

So in this chapter, I'll give you an overview of the potential weak points of Kubernetes and show you how I uncovered real-life AWS credentials during my last research run.

First thing's first: a short overview of what Kubernetes is and how you typically use it in connection with AWS accounts. If you know this part, skip ahead to the next page.

In the Beginning, There was Configuration

Managed Kubernetes like EKS usually have APIs secured with mutual TLS, IAM roles and API-level RBAC (Role-based Access Control). These generally constitute a pretty secure setup.

But when developers configure Kubernetes, it can happen that these security settings are overruled in favor of a less complicated setup. As a result, the API gets exposed.

Working with public Cloud providers doesn't mean you should have public APIs. And yet, this is one of the most common issues when it comes to Kubernetes.

These k8s APIs give anyone with access the opportunity to look into different parts of the environment configuration.

A Kubernetes API is the looking glass through which we can discover a plethora of information, some of which may be of interest to a hacker. What follows are three ways to get to that juicy meat.

Did you know?

Kubernetes (/koo-ber-nay'-tace/) comes from Greek and means "sailing master".

So now that the plain-English definition is out of the way and we're all on the same page, let me put on my white hat and see where the common weak points of this thing are.

1. /api/v1/secrets

Sensitive info, such as passwords and keys, is stored in the aptly named Secrets.

These are encoded with base64, which doesn't really have anything to do with security but rather serves to avoid characters escaping in requests/yaml files.

$ echo ZXhhbXBsZQo= | base64 – d

If set up properly, they are encrypted with AES. But the encryption is being handled transparently by the API, so if you're given the appropriate permissions in RBAC, you're able to decrypt and read the secrets.

For example, you could find GIT Repository or database credentials, .dockerconfig files and AWS access keys stored here. You shouldn't be able to, but I've seen it happen in my research.

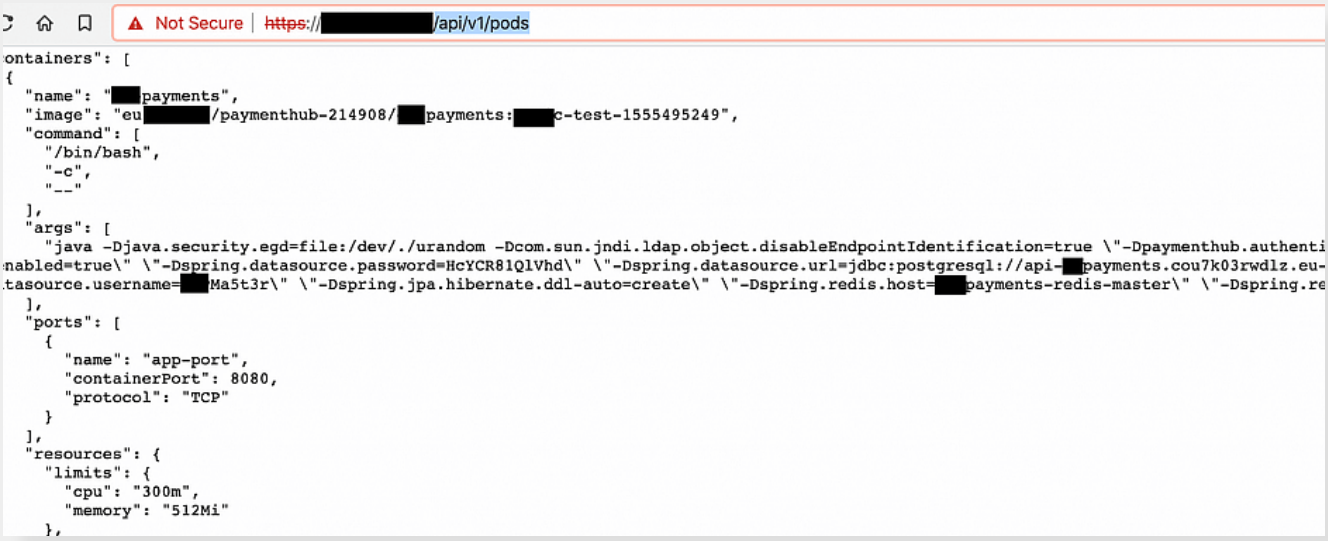
2. /api/v1/configmaps

ConfigMaps is basically the content of various configuration files which are used in your application – instead of keeping them in your repository.

Within it, the “annotations” parts can give you access to email addresses and project names, while the “config file contents” could even harbor plaintext secrets!

3. /api/v1/pods

You can get access to plaintext secrets through “arguments” and entry point “command”.



Reading about the theory can be fun and good, but it doesn’t hold up to a solid, real-life example.

How I Accessed an AWS Account of a Huge Multinational

I’ll walk you through my Kubernetes security research step by step – from the way I found a “victim”, uncovered their AWS credentials through /api/v1/ secrets to finding the contact e-mail of the person responsible for the AWS account.

All actual references to the account owner are anonymized for their own protection.

1. Finding the Victim

In chapter 1, I mentioned one of the tools that helps me find exposed data: **Binaryedge.io**. It scans the entire public internet to uncover exposures to help you secure them.

So as a first step, I switch on the VPN and simply search for an unsecured machine.

My eyes fall on some AWS EC2s running in the US, so I go into that list and just pick one at random.

The credentials are right there, just waiting to be base64-decoded. And lo and behold! I am now in possession of the aws_access_key_id and the aws_secret_access_key.

I can see some tokens, certificates... **and suddenly I hit the jackpot!**

2. Uncovering Secrets

The most straightforward path can be the most effective, so that's the one I start with. I simply go to:

/api/v1/secrets

Now I can see the entire configuration details of the Kubernetes – and this is just after a few clicks. There ain't no magic tricks!

3. Striking Gold

And now the lucrative part begins. I open my terminal and configure a new AWS profile with the access keys.

Then I run my script to check the access level of the account by asking for IAM users, S3 buckets, last month's costs, etc...

After a few seconds waiting time, I'm in – and can see all details pertaining to the AWS account, complete with number of users, associated costs, S3 buckets and the AWS Organization master account e-mail!

In the account summary, I can see that it's got 11 Users, 57 policies and 91 roles attached to it, so it's not huge but not small either. And there's the total cost of this infrastructure for the last month.

```
$ costs
AWS CloudTrail: 140.07
AWS Config: 27.91
AWS Glue: 0.00
AWS Key Management Service: 5.36
AWS Lambda: 0.01
AWS Secrets Manager: 0.00
AWS Service Catalog: 0.71
AWS Step Functions: 0.00
AWS Systems Manager: 0.02
Amazon DynamoDB: 0.00
Amazon EC2 Container Registry (ECR): 0.14
EC2 - Other: 4914.00
Amazon Elastic Compute Cloud - Compute: 359.96
Amazon Elastic Container Service for Kubernetes: 1043.17
Amazon Elastic Load Balancing: 473.06
Amazon Glacier: 0.00
Amazon GuardDuty: 41.98
Amazon Route 53: 3.54
Amazon Simple Notification Service: 0.00
Amazon Simple Queue Service: 0.00
Amazon Simple Storage Service: 2.55
Amazon Virtual Private Cloud: 377.22
AmazonCloudWatch: 381.55
Tax: 0.00
----------
TOTAL: 7771.26
```

4. Finding the owner

Now that I have all this information, I want to find the owner in order to inform them of the exposure.

I already have the AWS Organization master e-mail account (which has some revealing info in the domain name) but I want to get one more just to be on the safe side.

Using the same AWS profile, I can extract a root e-mail from a recent AWS support case – if there is any. The account owner e-mail domain was the same as the previously found one, so I know that I have the owner.

Securing Your Kubernetes

As you can see by the example above, it can be very easy to get into an AWS account through Kubernetes. But it should never be that easy.

The exposed account belonged to a large multinational company. After I contacted them with the info, they managed to fix everything within a few hours.

This could have been avoided from the start. Let's see how in our conclusion.

As is often the case with these types of vulnerabilities, they can be avoided by implementing sound security measures.

Here are three recommendations on how to secure your Kubernetes.

1. Keep your k8s API secured

Don't disable the default settings, which have RBAC and mTLS. If you need to do this for any reason, at least limit inbound connections to your office IPs only.

2. Use IAM Roles instead of access keys

There is no need to generate an access key for your application running on EKS. Use roles for your service accounts used by your pods.

3. Limit the privileges of your application

If your application needs to talk to S3, you can limit it to a single bucket. It really doesn't need an admin-like policy to work! This also applies to k8s RBAC. It's a common mistake to assign cluster-admin roles to users and service accounts.

Bonus tip:

Another common problem is that rights to exec into containers and/or debug containers are too open. Combined with a missing seccomp/apparmor setup (which aren't on by default) – this makes it possible to escape container and access host and other container resources.

Summary

How to Get Your AWS Account Hacked

Cyber security is not an isolated act – it's a process involving policies, training, compliance, risk assessment and infrastructure. Not having this process in place can lead to important data losses and reputational damage. It's basically like working out – you gotta keep at it to stay strong!

This is particularly true for solutions that exist in the Cloud. These are often front and center of modernization efforts, and in setting up or running software in the Cloud, often times simple mistakes can have big consequences.

As exhausting as it sounds, cyber security is a never-ending story. As you use more and more digital tools, you have to make sure that you stay on top of your platforms and accounts. Educating yourself and conducting internal and external security audits will help you in understanding and owning the tools to secure your systems and AWS environments.

Cyber security will always play a major role on the stage of business in the digital age. Don't take the topic lightly and don't make mistakes that can be avoided.

My advice to you is make the security analysis of each component of your digital solution a standard practice. That way, you stay in control and will be able to fend off any unwanted intruders.

„Without the opinion of an expert, there's no such thing as certainty."

Joanna Ruocco

Michał Brygidyn

Cloud Solutions Architect & Security Expert
michal.brygidyn@xebia.com
https://www.linkedin.com/in/michalbrygidyn/

Michał is an experienced Security Researcher (White Hat) and Cloud Solutions Architect with DevOps skillset. He is passionate about finding sources of leaking data and has already helped companies from various industries to improve their infrastructure's security. His experience also covers building a successful AWS partnership including preparation for competencies and programs. Outside of work, he's a big fan of ice hockey.

About Xebia:

Xebia is an IT Consultancy and Software Development Company that has been creating digital leaders across the globe since 2001.
With offices on every continent, we help the top250 companies worldwide embrace innovation, adopt the latest technologies, and implement the most successful business models. To meet every digital demand, Xebia is organized into chapters. These are teams with tremendous knowledge and experience in Agile, DevOps, Data & AI, Cloud, Software Development, Security, Quality Assurance, Low Code, and Microsoft Solutions. In addition to high-quality consulting and state-of-the-art software, Xebia Academy offers the training that modern companies need to work better, smarter, and faster. Today, Xebia continues to expand through a buy and build strategy.
We partner with leading IT companies to gain a greater foothold in the digital space.

Find more information on how Xebia is driving innovation at **www.xebia.com**.