

# Security in the age of cloud services

Trung Nguyen – CyStack Security

**CyStack**



**VIETNAM  
WEB  
SUMMIT**

# Whoami

- › Security Researcher with over 7 years experience in Security Industry
- › Co-Founder & CTO at CyStack Security
- › Discovered critical vulnerabilities and acknowledged by Microsoft, HP, Delloite...

# Agenda

- › AWS
- › Docker
- › Services exposed



# AWS Attack Vectors

- › IAM
- › Services
- › Network
- › Instances (Virtual Machines, EC2)
- › Custom applications & 3rd party software

# AWS Access Keys

Don't generate access keys for root users.

- › root user credentials allow full access to all resources in the account.
- › Losing keys means losing the whole data

# AWS Access Keys

Use Temporary Security Credentials (IAM Roles)

Instead of Long-Term Access Keys

- › When you don't control the client (mobile, desktop app, etc).
- › When you need to grant cross-account access.

# AWS Access Keys

## Manage IAM User Access Keys Properly

- › Don't embed access keys directly into code, use credentials file or environment variables instead
- › Use different access keys for different applications
- › Rotate access keys periodically
- › Remove unused access keys
- › Configure multi-factor authentication for your most sensitive operations



# IAM policy misuse

- › IAM is the core service behind access management within the AWS environment.
- › Misconfigurations of the service is the main source of vulnerabilities: privilege escalation or data exfiltration.
- › AWS allows users to apply two kinds of policies: **AWS managed policies** and **self-managed policies**

# IAM policy misuse

AWS managed policies can be even broken

## AWS Security Flaw which can grant admin access!



Sharath AV [Follow](#)  
May 22, 2018 · 5 min read



I recently discovered an AWS Managed Policy that potentially allowed granting admin access to self or any other IAM role. This blog-post describes my findings and my interactions with AWS Security team on the same.

<https://medium.com/ymedialabs-innovation/an-aws-managed-policy-that-allowed-granting-root-admin-access-to-any-role-51b409ea7ff0>

Filter: Policy type

Policy name	Type	Attachments	Description
AmazonElasticTranscoder_FullAccess	AWS managed	0	Grants users full access to Elastic Transcoder and
AmazonElasticTranscoderFullAccess	AWS managed	1	

AmazonElasticTranscoderFullAccess

Policy summary [{}JSON](#)

```
1- {
2-   "Version": "2012-10-17",
3-   "Statement": [
4-     {
5-       "Action": [
6-         "elastictranscoder:*",
7-         "cloudfront:*",
8-         "s3:List*",
9-         "s3:Put*",
10-        "s3:Get*",
11-        "s3:*MultipartUpload*",
12-        "iam:CreateRole",
13-        "iam:GetRolePolicy",
14-        "iam:PassRole",
15-        "iam:PutRolePolicy",
```

This access allowed any user or role who has been granted this policy to put a new inline role policy to any other roles. Potentially allowing to add an inline policy that grants root access to the account.

# IAM policy misuse

When a feature becomes a bug..

```
> curl http://169.254.169.254/latest/meta-data
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hostname
identity-credentials/
instance-action
instance-id
instance-type
local-hostname
local-ipv4
...
```

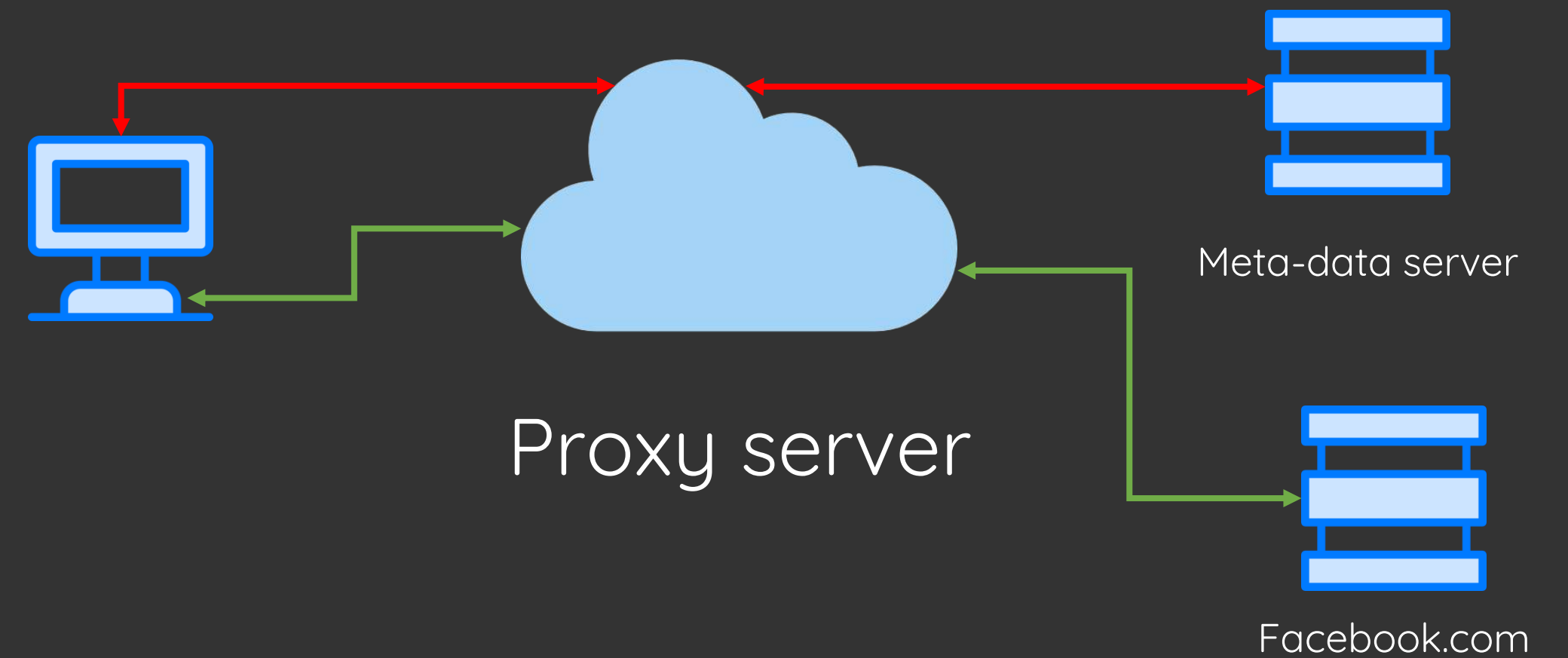
# IAM policy misuse

Let's assume that we have a role that look goods and is attached to an EC2 instance

```
{  
  "Effect": "Allow",  
  "Action": [  
    "iam:Create*",  
    "iam:Add*"  
  ],  
  "Resource": [  
    "arn:aws:iam::12345678:user/*"  
  ]  
}
```

# IAM policy misuse

```
> curl http://169.254.169.254/la  
test/meta-data/iam/security-  
credentials/<role-name>
```



```
{  
  "Code": "Success",  
  "LastUpdated": "2012-04-26T16:39:16Z",  
  "Type": "AWS-HMAC",  
  "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",  
  "SecretAccessKey": "xxxxxxx",  
  "Token": "xxxxxxxx",  
  "Expiration": "2017-05-17T15:09:54Z"  
}
```

# S3

- › One of the most awesome services of AWS
- › However, presumably, the most common cause of security breaches related to Amazon services, are misconfigurations of S3 buckets
- › 7% of all S3 buckets have unrestricted public access



<https://www.bleepingcomputer.com/news/security/7-percent-of-all-amazon-s3-servers-are-exposed-explaining-recent-surge-of-data-leaks/>

# S3 bucket breaches

## > Misconfiguration

The screenshot displays the AWS IAM console interface for an S3 bucket's permissions. The 'Permissions' tab is active, showing a 'Public' status. The 'Access for bucket owner' section is visible, along with 'Access for other AWS accounts' and 'Public access' sections. A red box highlights the 'Everyone' group in the 'Public access' section, which has 'List objects', 'Write objects', and 'Read bucket permissions' enabled. On the right, a modal window titled 'Everyone' shows a warning: 'This bucket has public access' and lists the permissions granted to everyone: 'List objects', 'Write objects', 'Read bucket permissions', and 'Write bucket permissions'.

**Permissions Overview**

- Block public access
- Access Control List (Public)
- Bucket Policy (Public)
- CORS configuration

**Access for bucket owner**

Canonical ID	List objects	Write objects	Read bucket permissions
[Redacted]	Yes	Yes	Yes

**Access for other AWS accounts**

+ Add account | Delete

Canonical ID	List objects	Write objects	Read bucket permissions
--------------	--------------	---------------	-------------------------

**Public access**

Group	List objects	Write objects	Read bucket permissions
Everyone	Yes	Yes	Yes

**Everyone**

**Warning: This bucket has public access**  
Everyone has access to one or all of the following: list objects, write objects, read and write permissions.

**Access to the objects**

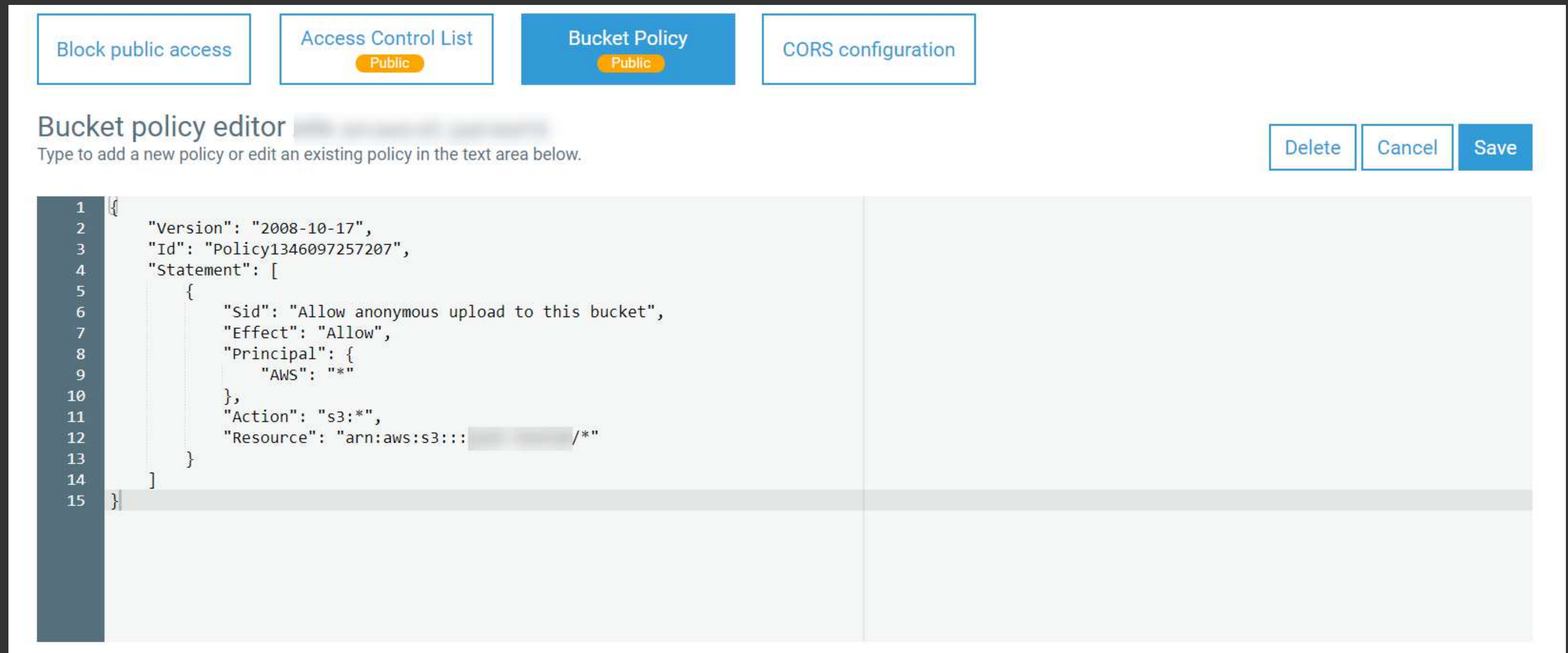
- List objects
- Write objects

**Access to this bucket's ACL**

- Read bucket permissions
- Write bucket permissions

# S3 bucket breaches

## > Misconfiguration



The screenshot shows the AWS IAM console interface for editing a bucket policy. At the top, there are four tabs: "Block public access", "Access Control List" (with a "Public" indicator), "Bucket Policy" (with a "Public" indicator), and "CORS configuration". The "Bucket Policy" tab is active. Below the tabs, the title "Bucket policy editor" is followed by a blurred bucket name. To the right of the title are "Delete", "Cancel", and "Save" buttons. Below the title, a text prompt reads: "Type to add a new policy or edit an existing policy in the text area below." The main area is a code editor with a line number column on the left (1-15) and a text area containing the following JSON policy:

```
1 {
2   "Version": "2008-10-17",
3   "Id": "Policy1346097257207",
4   "Statement": [
5     {
6       "Sid": "Allow anonymous upload to this bucket",
7       "Effect": "Allow",
8       "Principal": {
9         "AWS": "*"
10      },
11      "Action": "s3:*",
12      "Resource": "arn:aws:s3:::[redacted]/*"
13    }
14  ]
15 }
```

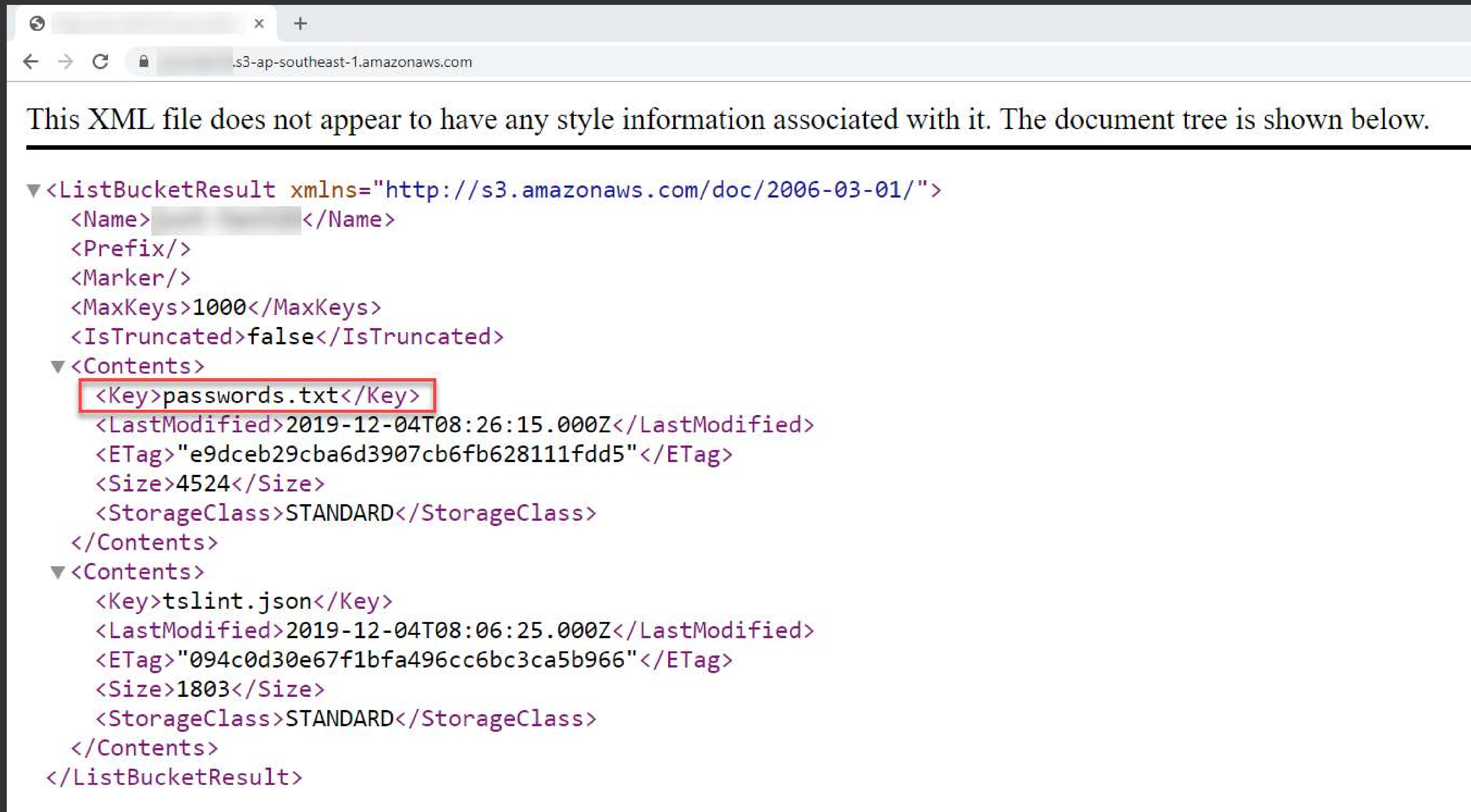


# S3 bucket breaches

Now, attackers can:

- › get access to list and read files in S3 bucket
- › write/upload files to S3 bucket
- › change access rights to all objects and control the content of the files

# S3 bucket breaches

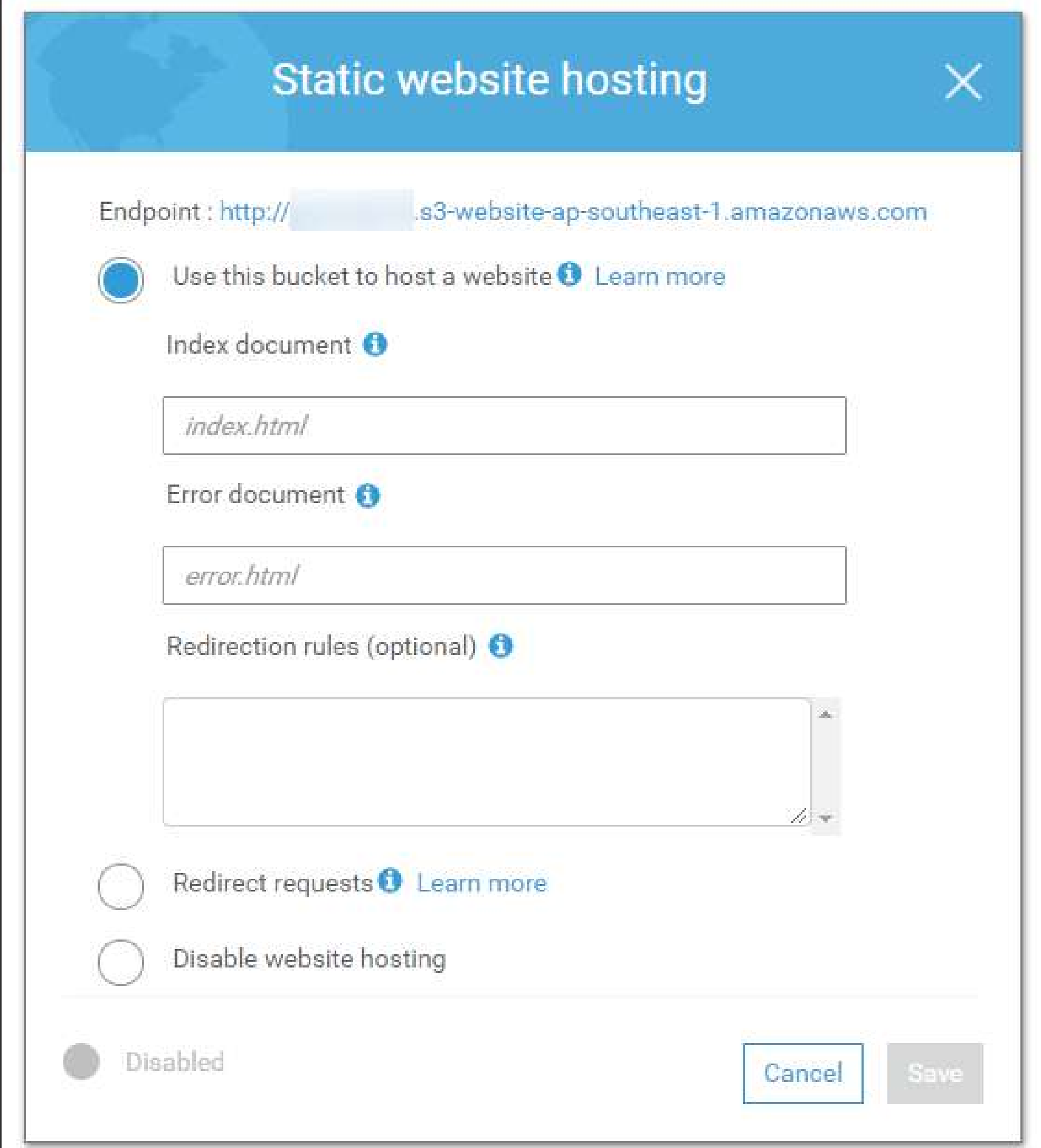


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼ <ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>[REDACTED]</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>>false</IsTruncated>
  ▼ <Contents>
    <Key>passwords.txt</Key>
    <LastModified>2019-12-04T08:26:15.000Z</LastModified>
    <ETag>"e9dceb29cba6d3907cb6fb628111fdd5"</ETag>
    <Size>4524</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  ▼ <Contents>
    <Key>tslint.json</Key>
    <LastModified>2019-12-04T08:06:25.000Z</LastModified>
    <ETag>"094c0d30e67f1bfa496cc6bc3ca5b966"</ETag>
    <Size>1803</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
</ListBucketResult>
```

# S3 sub-domain take over

- › Create a bucket named **sub.company.com**
- › Enable the feature static web hosting then put static files to this bucket



The screenshot shows the 'Static website hosting' configuration page in the AWS console. The page has a blue header with the title 'Static website hosting' and a close button (X). Below the header, the 'Endpoint' is displayed as 'http://[redacted].s3-website-ap-southeast-1.amazonaws.com'. There are three radio button options: 'Use this bucket to host a website' (which is selected), 'Redirect requests', and 'Disable website hosting'. Under the selected option, there are three input fields: 'Index document' with the value 'index.html', 'Error document' with the value 'error.html', and 'Redirection rules (optional)' which is currently empty. At the bottom left, there is a 'Disabled' radio button. At the bottom right, there are 'Cancel' and 'Save' buttons.

Static website hosting

Endpoint : [http://\[redacted\].s3-website-ap-southeast-1.amazonaws.com](http://[redacted].s3-website-ap-southeast-1.amazonaws.com)

Use this bucket to host a website [Learn more](#)

Index document [i](#)

Error document [i](#)

Redirection rules (optional) [i](#)

Redirect requests [Learn more](#)

Disable website hosting

Disabled

[Cancel](#) [Save](#)

# S3 sub-domain take over



- › One day, you removed the bucket but didn't update the DNS records
- › And attackers can create a new bucket with the same name → Take control your sub-domain
- › Not only you, Microsoft, Google and other big corp faced the same issue.

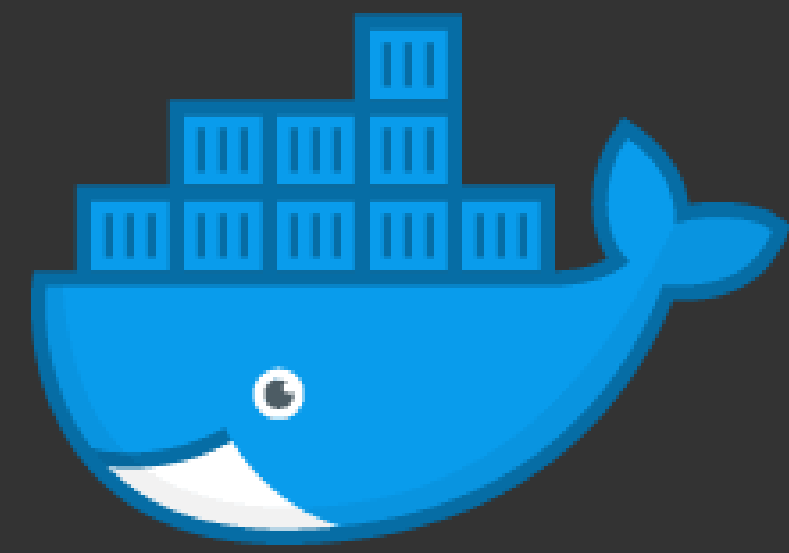
# Serverless

NOT YOUR SERVERS, NOT YOUR PROBLEMS?

NOT QUITE...

# Serverless

- › Event injection
- › Broken authentication
- › Insecure deployment settings
- › Misuse of permissions and roles
- › Insufficient logging
- › Insecure storing of app secrets
- › DoS attacks and financial exhaustion
- › Improper exception handling



docker

# Image Authenticity

- › Use Private or Trusted Repositories
- › Prefer [Docker Certified](#) images
- › Prefer minimal base images



DON'T

# Privileges

› `docker run app --privileged`

gives all capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller. In other words, the container can then do almost everything that the host can do. This flag exists to allow special use-cases, like running Docker within Docker.

# Privileges

- › By default, the application in container runs as root privileges

```
RUN groupadd -r gooduser && useradd -m -r -g gooduser -s /sbin/nologin -c "create a good user" gooduser
```

```
USER gooduser
```

```
CMD ["python", "app.py"]
```

# Read-only mode

```
> docker run --read-only --tmpfs /tmp app
```

# DDoS preventing

```
> docker run --cpus=0.5 --memory=512m app
```

Services exposed



kubernetes

> kubectl proxy --address 0.0.0.0 --accept-hosts '.\*'

The screenshot shows the Kubernetes Dashboard interface. The left sidebar contains a navigation menu with categories: Cluster (Cluster Roles, Namespaces, Nodes, Persistent Volumes, Storage Classes), Namespace (default), Overview (selected), Workloads (Cron Jobs, Daemon Sets, Deployments, Jobs, Pods, Replica Sets, Replication Controllers, Stateful Sets), Discovery and Load Balancing (Ingresses, Services), and Config and Storage (Config Maps, Persistent Volume Claims). The main content area displays two tables: 'Daemon Sets' and 'Deployments'. Both tables have columns for Name, Namespace, Labels, Pods, Age, and Images. The 'Daemon Sets' table shows 1 item, and the 'Deployments' table shows 10 items. The bottom right corner of the dashboard indicates '1 - 10 of 19' items.



kubernetes

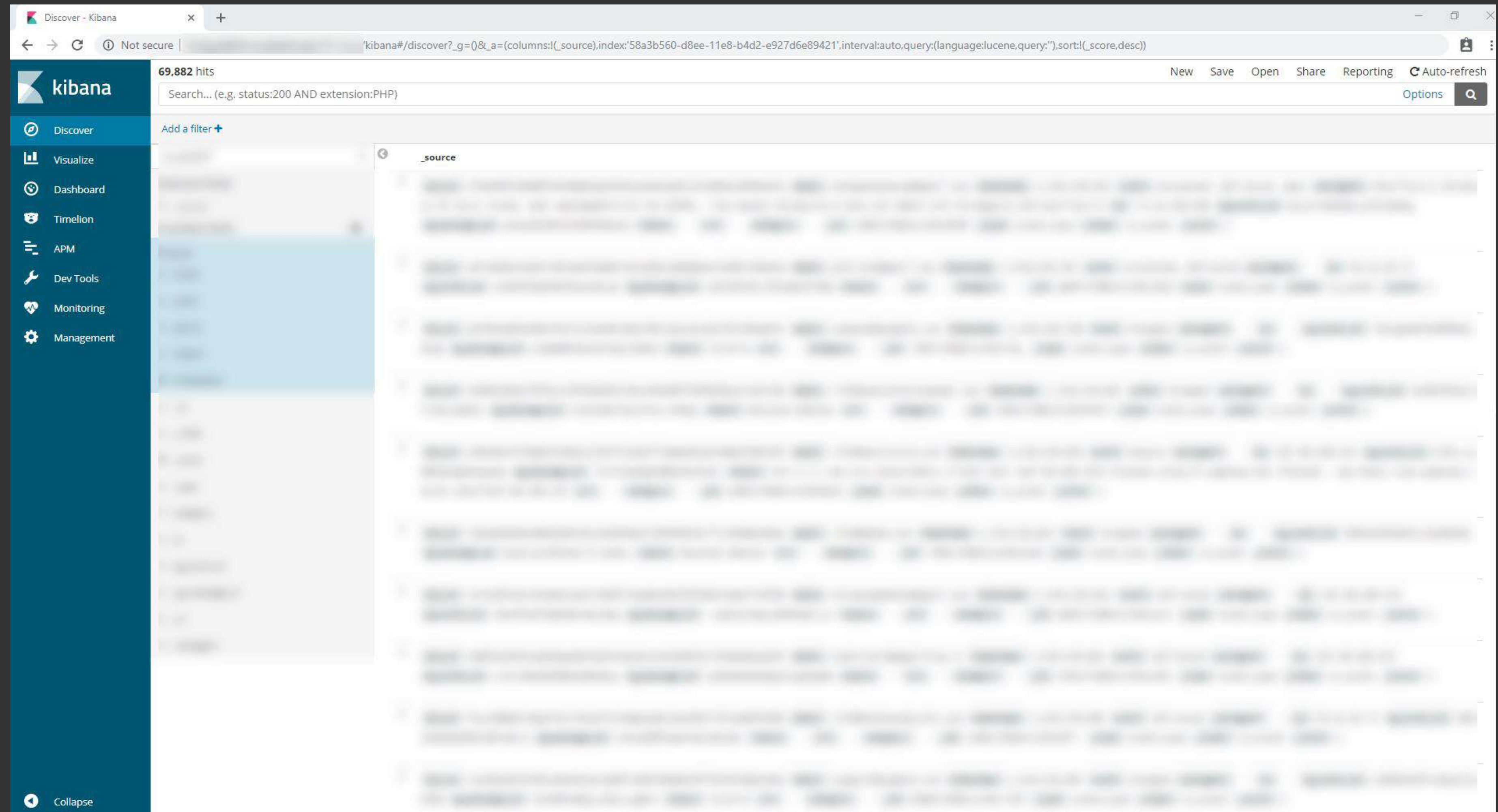
- › Tesla cloud resources are hacked to run cryptocurrency-mining malware



kibana.yml

server.port: 5601

server.host : 0.0.0.0







elasticsearch.yml

http.port: 9200

network.host: 0.0.0.0

```

{
  "name" : "elasticsearch",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "98123456789012345678901234567890",
  "version" : {
    "number" : "5.6.0",
    "build_hash" : "781a835",
    "build_date" : "2017-09-07T03:09:58.087Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.0"
  },
  "tagline" : "You Know, for Search"
}

```

# Thanks !

trungnh@cystack.net  
@everping

