PENTERA | Pentera Labs™ Research Series

Blurring Boundaries: Deciphering the Risks of AWS SSM in Hybrid Landscapes

Discover the risks of AWS SSM in hybrid environments, how attackers exploit it, real-world scenarios, and strategies to mitigate vulnerabilities effectively.

Ron Warshavsky

Blurring Boundaries: Deciphering the Risks of AWS SSM in Hybrid Landscapes

Discover the risks of AWS SSM in hybrid environments, how attackers exploit it, real-world scenarios, and strategies to mitigate vulnerabilities effectively.

Ron Warshavsky

Table of contents

03 Introduction

03 Brief Reminder: The Hybrid Network

04 The Hybrid Environment from the Attacker Perspective

04 AWS SSM Overview

05 Why AWS SSM? An Admin's Perspective

07 What Do SSM Attacks Look Like?

11 Limiting Exposure

12 About Pentera

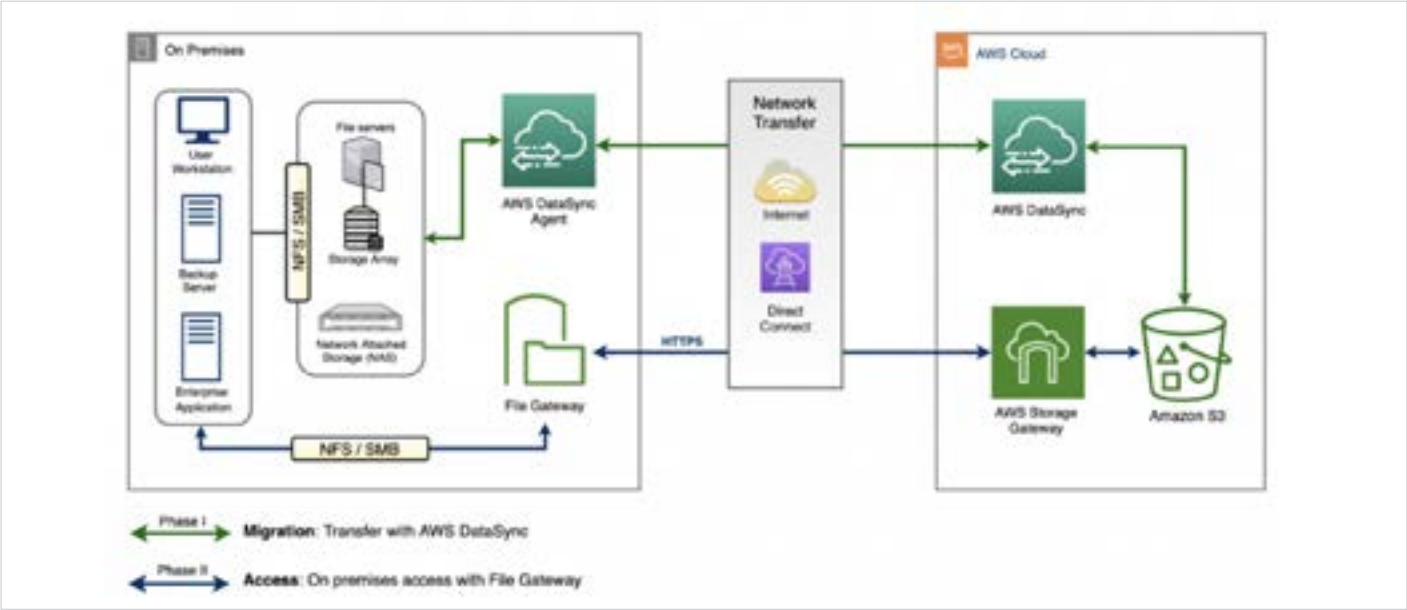
Introduction

Hybrid architectures are ubiquitous among enterprises. While they offer flexibility and scalability, they also expand the attack surface available to adversaries. Attackers can leverage tools like AWS SSM, which were designed to enhance hosts administration, as vectors for lateral movement, privilege escalation, and persistent footholds.

In this paper, we explore the risks associated with AWS SSM in a hybrid environment. We show the capabilities of SSM, its appeal to attackers and demonstrate a number of attack scenarios that highlight its misuse. Finally, we provide strategies to limit exposure and mitigate risks, ensuring that the advantages of hybrid architecture are not overshadowed by vulnerabilities. After reading this paper, you will have a better understanding of how to identify and mitigate risks related to hybrid environments, their management tools and specifically to AWS SSM.

Brief Reminder: The Hybrid Network Architecture

A hybrid network architecture combines on-premises infrastructure, like user workstations and servers (including application and file servers), with cloud resources that extend and enhance on-premises capabilities. For example, AWS provides storage gateways and S3 buckets. Connections between the on-premises and cloud environments can be established via dedicated network tunnels or the public internet.



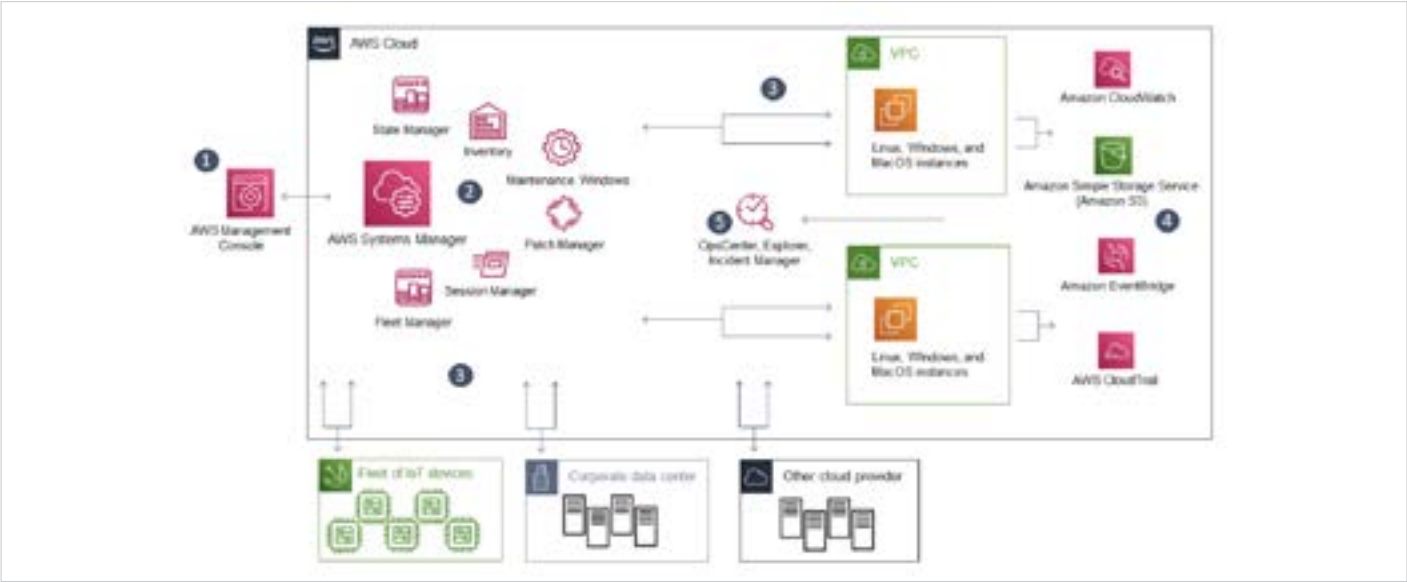
Hybrid cloud adoption offers flexibility and scalability, making it easy to adapt network design and expand resources as needed. This architecture also supports data sovereignty and regulatory requirements in certain industries, which require companies to keep some sensitive data on-premises.

The Hybrid Environment from the Attacker Perspective

While a hybrid network environment provides multiple advantages, it also significantly expands the attack surface. With both local and cloud components in play, attackers gain multiple entry points and have more opportunities to establish persistence and move laterally within the environment.

AWS SSM Overview

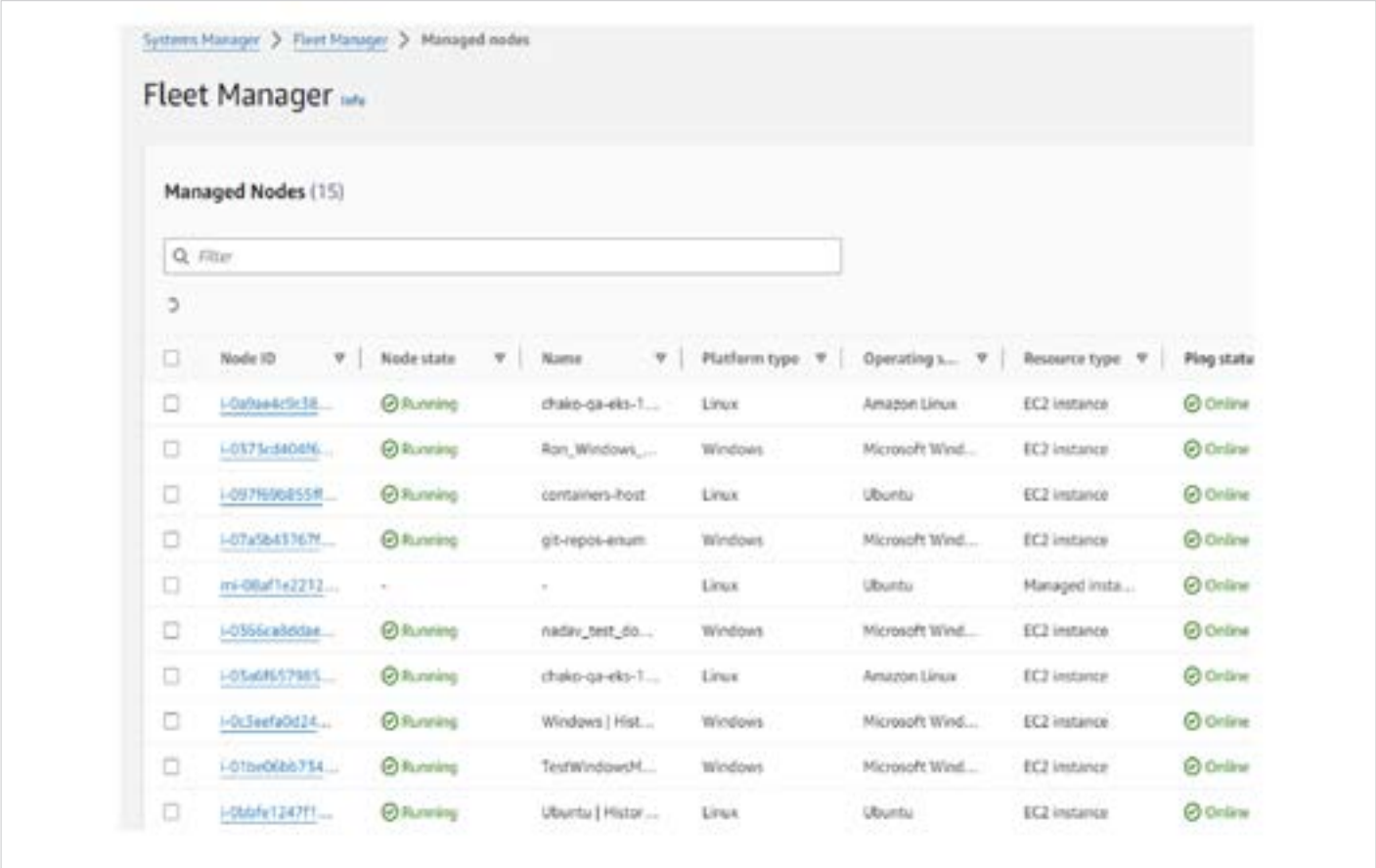
AWS Systems Manager (SSM) is a management tool designed to help organizations manage infrastructure across AWS, on-premises, and hybrid cloud environments.



Key Components of AWS Systems Manager:

- **Fleet Manager** - AWS Fleet Manager is a tool for managing and monitoring servers remotely. It uses the SSM Agent installed on instances to enable secure communication and execution of commands.

SSM Agent typical configuration



SSM Agent managed hosts centralized view

- **Session Manager –** Provides a secure and auditable way to manage instances through interactive shell or command execution, leveraging the SSM Agent to establish and manage the sessions without needing SSH or RDP.
- **Patch Manager –** Provides automated patching for operating systems and applications across your instances, leveraging the SSM Agent to scan, detect, and apply patches according to defined compliance baselines.

Why AWS SSM? An Admin's Perspective

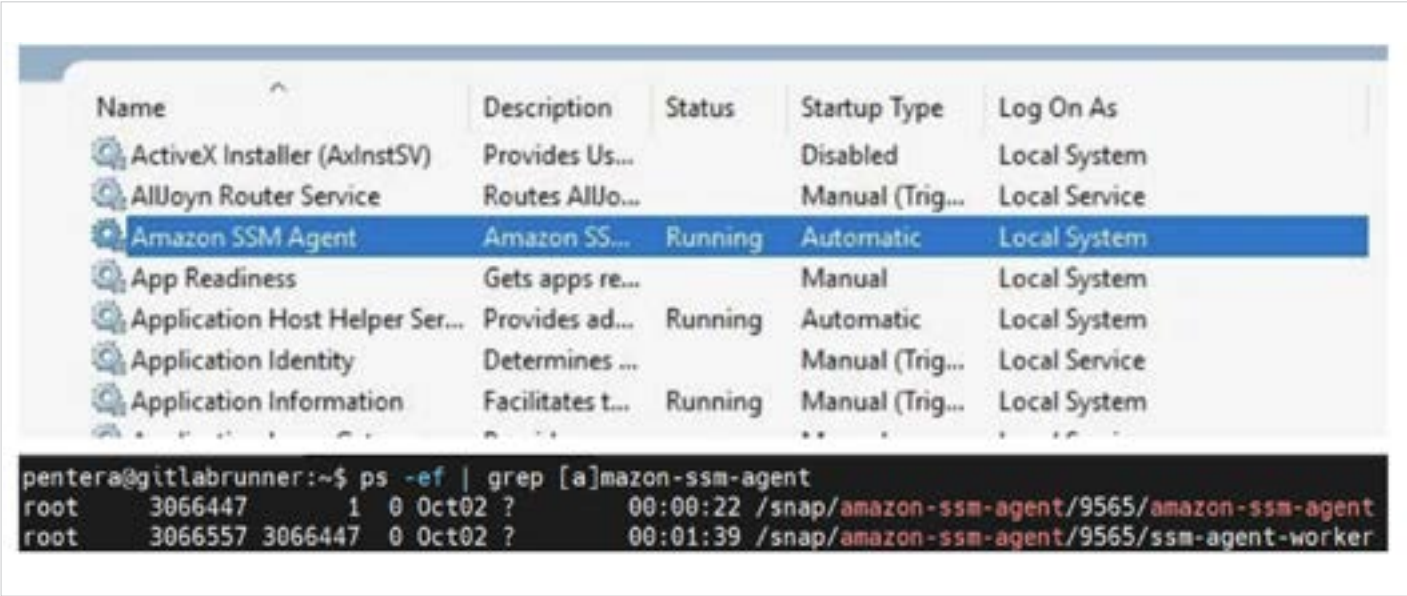
Many organizational admins choose AWS SSM because it offers:

- Unified and simplified management across on-premises and cloud environments, regardless of where resources are located.
- Automation of essential maintenance activities, like patching, configuration changes, and software updates.
- Improved security, since there is no need to remotely connect to endpoints using SSH or RDP.

AWS SSM as an Attraction to Attackers

AWS SSM's powerful management capabilities make it attractive for admins and attackers alike. This is due to the following characteristics:

- **Centralized Management with High Privileges** - AWS Systems Manager (SSM) grants administrators elevated (root/SYSTEM) privileges, allowing them to perform critical tasks such as software installations, system configurations, and patch management across both cloud and on-premises environments.These high-level permissions also make SSM attractive to attackers, enabling them to move laterally, escalate privileges, and maintain persistent access within the infrastructure.



AWS SSM's powerful management capabilities make it attractive for admins and attackers alike. This is due to the following characteristics:

- **Legitimate High-Traffic Channels Between AWS and On-Premises Environments** - The integration between on-premises systems and AWS through SSM relies on continuous communication. Since these channels are often high-traffic and critical to operations, they are typically trusted and may bypass stricter scrutiny from security tools. For attackers, this creates a "low noise" environment where malicious activity may blend in with normal operations, reducing the chances of discovery and allowing them to move through the network more stealthily.
- **Signed Binary Trusted by Most EDR Systems** - SSM uses AWS-signed binaries, which are widely trusted by endpoint security solutions like EDR systems. This means that actions conducted through SSM are often deemed legitimate by security tools, which might flag custom executables but ignore the SSM agent. It's a type of LOTL (Living Off the Land) attack. Attackers leveraging SSM can bypass these security controls, enabling them to operate within the network without immediately triggering alarms.

What Do SSM Attacks Look Like?

Let's look at two examples of SSM attacks, and how attackers compromise the on-premises environment and use it to harvest credentials from AWS and target AWS SSM.

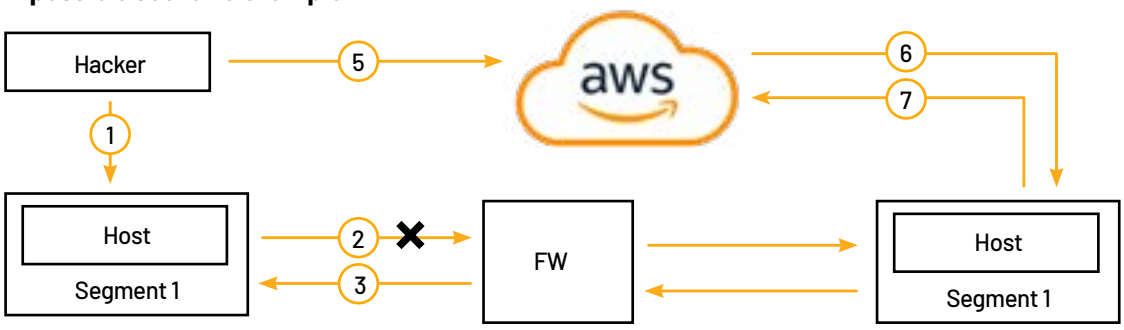
Example #1: The Basic Attack

1. Hacker has obtained privileged AWS credentials via a phishing attack, posing as an AWS legitimate request for periodical password change.
2. The hacker connects to the AWS API and starts mapping out the infrastructure using Boto3 AWS Python SDK.
3. The hacker focuses on SSM Managed devices, specifically the on-prem ones.
4. It so happens that one of the IT administrators deployed an SSM agent on one of the company's servers for functionality testing
4. The hacker manages to connect to the on-premises server, and using the SSM agent's high privileges, harvests sensitive data and credentials.
5. In addition, the attacker creates himself another user through the IAM interface of the AWS API, thus achieving persistence.

Example #2: The Advanced Attack

In this example, an attacker uses AWS SSM to circumvent the effective firewall restrictions that were applied on the on premise network.

A possible scenario example



1. A hacker successfully compromises a host located in Segment 1 of an on-prem environment. The attacker would like to propagate further and gain access to more of the victim's valuable assets: DBs, file servers, finances, know-how, etc.
2. The attacker starts to perform reconnaissance, gathering IP configuration information from the host, reviewing local firewall rules, active connections, etc.
3. The attacker tries to access hosts on segment 2 on commonly used services: SSH, RDP, SMB, LDAP but to no avail. This is due to correct and strict firewall configuration. It looks as if the attacker is stuck...
4. The attacker turns his attention back to the host for additional reconnaissance.
5. The attacker successfully manages to harvest stored AWS credentials (code in previous example).
6. The attacker connects to the AWS infrastructure using the AWS API.
7. He targets the SSM manager component as a whole, focusing on the on-premise managed nodes.

Code example:

```
import boto3
import getpass
import logging
from botocore.exceptions import NoCredentialsError, PartialCredentialsError

def get_ssm_managed_instances(
    aws_access_key=None,
    aws_secret_key=None,
    aws_session_token=None,
    region_name='eu-west-1',
    filters=None
):
    """
    Retrieves a list of on-premises managed instances registered with AWS
Systems Manager.

    """
    try:
        # Create a session with provided AWS credentials or default
credentials
        session = boto3.Session(
            aws_access_key_id=aws_access_key,
            aws_secret_access_key=aws_secret_key,
            aws_session_token=aws_session_token,
            region_name=region_name
        )

        ssm_client = session.client('ssm')

        # Get list of managed instances
        paginator = ssm_client.get_paginator('describe_instance_information')
        page_iterator = paginator.paginate(
            Filters=filters or []
        )

        on_prem_instances = []

        for page in page_iterator:
            for instance in page['InstanceInformationList']:
                # Check if it's an on-premises instance (not EC2)
                if instance['ResourceType'] == 'ManagedInstance':
                    on_prem_instances.append(instance)

        return on_prem_instances

    except NoCredentialsError:
        logging.error("AWS credentials not provided or invalid.")
    except PartialCredentialsError:
        logging.error("Partial AWS credentials provided. Please provide both
access key and secret key.")
    except Exception as e:
        logging.error(f"An error occurred: {e}")
    return []  # Return empty list on error

def main():
    use_default_credentials = input("Use default AWS credentials? (y/n):
").lower() == 'y'

    if use_default_credentials:
        aws_access_key = None
        aws_secret_key = None
        aws_session_token = None
```

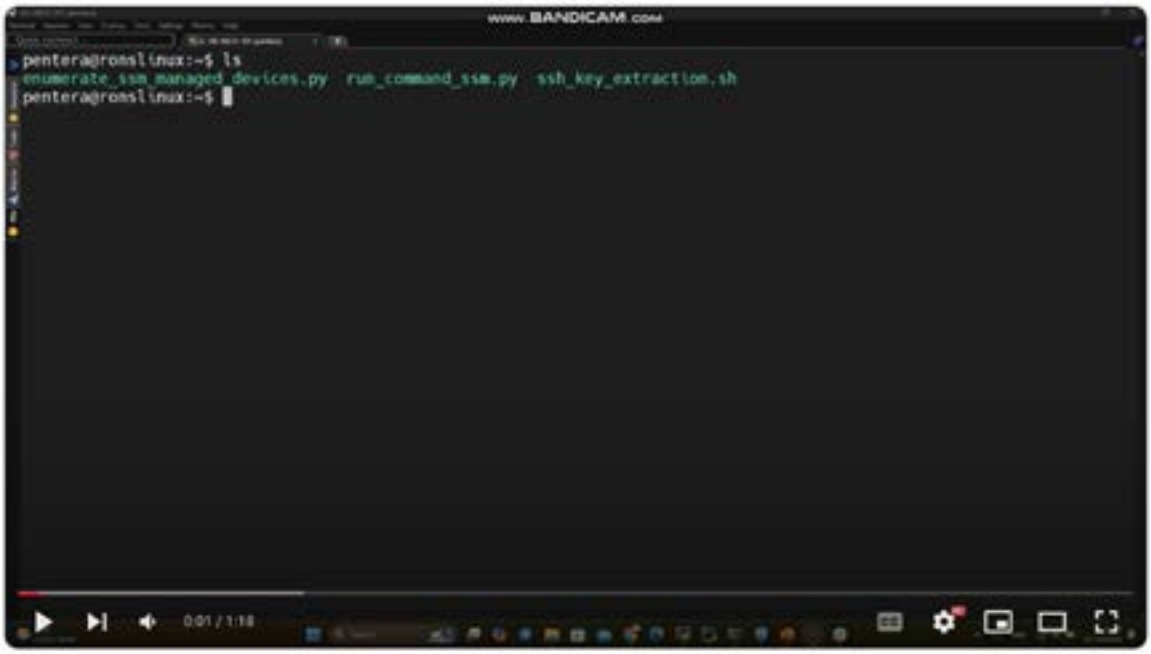
```
    else:
        aws_access_key = getpass.getpass("Enter AWS Access Key: ")
        aws_secret_key = getpass.getpass("Enter AWS Secret Key: ")
        aws_session_token = getpass.getpass("Enter AWS Session Token (if
applicable, otherwise leave blank): ")
        if not aws_session_token:
            aws_session_token = None  # Optional

    region_name = input("Enter AWS Region (default 'eu-west-1'): ") or 'eu-
west-1'

    # Optionally, you can prompt the user for filters here

    on_prem_instances = get_ssm_managed_instances(
        aws_access_key,
        aws_secret_key,
        aws_session_token,
        region_name
    )

    if on_prem_instances:
        print("On-Premises Managed Instances:")
        for instance in on_prem_instances:
            print(f"""
Instance ID       : {instance['InstanceId']}
Computer Name     : {instance.get('ComputerName', 'N/A')}
Platform Name     : {instance.get('PlatformName', 'N/A')}
Platform Version  : {instance.get('PlatformVersion', 'N/A')}
IPAddress         : {instance.get('IPAddress', 'N/A')}
Registration Date : {instance.get('RegistrationDate', 'N/A')}
            """)
    else:
        print("No on-premises managed instances found.")

if __name__ == "__main__":
    main()
```

8. The attacker cross checks the previously gathered information and discovers an SSM Managed host on Segment 2.
9. The attacker connects to the host using the available SSM agent and is able to perform various malicious activities they have in mind: data theft, ransomware, achieving persistence, etc.

Example #3: Bringing it all together: The full attack kill chain

In this example, an attacker, after obtaining AWS credentials, uses three scripts to exploit the AWS environment.

1. The attacker first enumerates SSM managed devices, specifically targeting an online managed instance.
2. Once the instance ID is identified, the attacker launches the Run Command SSM script to execute a payload(the third script) on the target host.
3. The payload in this command is designed to search for and extract stored SSH keys from the target host, effectively enabling further propagation through the compromised keys.
4. The attacker focuses on identifying on-premises instances that are online. Once the instance ID and IP address are obtained, the attacker uses the Run Command SSM to create a client that can execute a command on the remote target host. The payload in this command is designed to extract SSH keys from the target host.
5. By using AWS API calls, the attacker retrieves the managed instance ID, and after specifying it, successfully executes the payload.
6. The output confirms successful extraction of SSH keys for both the root and privileged users on the targeted endpoint, demonstrating a critical security vulnerability.



Limiting Exposure

To mitigate risks associated with AWS SSM, it's recommended to implement the following security strategies:

- **Assign IAM Roles with Least Privilege** - Specific permissions, such as `SSM SendCommand` and `SSM GetCommandInvocation`, grant users the ability to execute commands on SSM-managed hosts. Because of the elevated access these permissions provide, they should be granted only to essential roles, and access should be regularly reviewed and minimized where possible. Additionally, monitoring for anomalous activity related to these permissions can help detect unauthorized command execution early.

- **Rotate and Encrypt AWS Credentials** - Compromised credentials can be used to gain unauthorized access to resources or for elevating permissions. Regularly rotate and encrypt AWS credentials to limit the potential impact of any compromised credential.

- **Monitor Local SSM User Accounts** - When the SSM agent is installed on an instance, a local SSM user account is created. This account is added to the `sudoers` file on Linux systems or the administrators group on Windows, granting it elevated privileges. Monitor the activity associated with these accounts for login attempts, configuration changes and other misuse.

- **Enable Detailed Logging for SSM** - AWS CloudTrail can be configured to log SSM actions, capturing details about command executions, changes to configurations, and access attempts. This logging provides a comprehensive audit trail. Regularly reviewing these logs to detect unauthorized access, configuration drift, or other indicators of compromise.

By following these practices, organizations can reduce their exposure to potential misuse of AWS SSM, helping to protect both cloud and on-premises environments while enjoying the benefits of a hybrid cloud architecture.

About the author

Ron Warshavsky is a Security Researcher at Pentera, focusing on Windows, AWS and CI/CD-based attacks.

Reach out to us with any questions about the research at labs@pentera.io.

About Pentera

PENTERA

Pentera is the category leader for Automated Security Validation, allowing every organization to test with ease the integrity of all cybersecurity layers, unfolding true, current security exposures at any moment, at any scale. Thousands of security professionals and service providers around the world use Pentera to guide remediation and close security gaps before they are exploited.

For more info, visit: pentera.io

PENTERA | Pentera Labs™ Research Series

Blurring Boundaries: Deciphering the Risks of AWS SSM in Hybrid Landscapes