```
import os, json, boto3
globalVars  = {}
globalVars['Owner']                = "bangari"
globalVars['REGION_NAME']          = "us-east-1"

globalVars['security_group_id']     = os.environ['security_group_id']

def lambda_handler(event, context):
    print(event)

    print(globalVars['security_group_id'])
    # Ensure that we have an event name to evaluate.
    if 'detail' not in event or ('detail' in event and 'eventName' not in event['detail']):
        return {"Result": "Failure", "Message": "Lambda not triggered by an event"}

    # Remove the rule only if the event was to authorize the ingress rule for the given
    # security group id is one provided in the Environment Variables.
    if (event['detail']['eventName'] == 'AuthorizeSecurityGroupIngress' and
            event['detail']['requestParameters']['groupId'] == globalVars['security_group_id']):
        result = revoke_security_group_ingress(event['detail'])

        message = "AUTO-MITIGATED: Ingress rule removed from security group: {} that was added
by {}: {}".format(
            result['group_id'],
            result['user_name'],
            json.dumps(result['ip_permissions'])
        )


def revoke_security_group_ingress(event_detail):
    request_parameters = event_detail['requestParameters']

    # Build the normalized IP permission JSON struture.
    ip_permissions = normalize_paramter_names(request_parameters['ipPermissions']['items'])

    response = boto3.client('ec2').revoke_security_group_ingress(
        GroupId=request_parameters['groupId'],
        IpPermissions=ip_permissions
    )

    # Build the result
    result = {}
    result['group_id'] = request_parameters['groupId']
    result['user_name'] = event_detail['userIdentity']['arn']
```

```
    result['ip_permissions'] = ip_permissions

    return result

def normalize_paramter_names(ip_items):
    # Start building the permissions items list.
    new_ip_items = []

    # First, build the basic parameter list.
    for ip_item in ip_items:

        new_ip_item = {
            "IpProtocol": ip_item['ipProtocol'],
            "FromPort": ip_item['fromPort'],
            "ToPort": ip_item['toPort']
        }

        # CidrIp or CidrIpv6 (IPv4 or IPv6)?
        if 'ipv6Ranges' in ip_item and ip_item['ipv6Ranges']:
            # This is an IPv6 permission range, so change the key names.
            ipv_range_list_name = 'ipv6Ranges'
            ipv_address_value = 'cidrIpv6'
            ipv_range_list_name_capitalized = 'Ipv6Ranges'
            ipv_address_value_capitalized = 'CidrIpv6'
        else:
            ipv_range_list_name = 'ipRanges'
            ipv_address_value = 'cidrIp'
            ipv_range_list_name_capitalized = 'IpRanges'
            ipv_address_value_capitalized = 'CidrIp'

        ip_ranges = []

        # Next, build the IP permission list.
        for item in ip_item[ipv_range_list_name]['items']:
            ip_ranges.append(
                {ipv_address_value_capitalized: item[ipv_address_value]}
            )

        new_ip_item[ipv_range_list_name_capitalized] = ip_ranges

        new_ip_items.append(new_ip_item)

    return new_ip_items
```