# Step-by-Step lambda function Code Explanation

### 1. Importing Dependencies

```python
import json
import boto3
import os
import requests
from datetime import datetime, timedelta
```

This block imports necessary libraries:

- **json**: Used for handling JSON data.
- **boto3**: AWS SDK for Python, used for interacting with AWS services (e.g., SecurityHub).
- **os**: For accessing environment variables.
- **requests**: For making HTTP requests (used to create Jira issues).
- **datetime**: For handling date and time, although not fully utilized in the provided code.

### 2. Fetch Environment Variables

```python
JIRA_URL = os.getenv("JIRA_URL", "https://abc-example.atlassian.net")
JIRA_EMAIL = os.getenv("JIRA_EMAIL", "abc@exmple.com")
JIRA_API_TOKEN = os.getenv("JIRA_API_TOKEN")
JIRA_PROJECT_KEY = os.getenv("JIRA_PROJECT_KEY", "SCRUM")
FILTER_ACCOUNTS = os.getenv("FILTER_ACCOUNTS", "").split(",")  # Comma-separated account IDs
FILTER_REGIONS = os.getenv("FILTER_REGIONS", "us-east-1").split(",")  # Default to us-east-1
FILTER_SEVERITY = os.getenv("FILTER_SEVERITY", "").split(",")  # Comma-separated severities
```

- Fetches environment variables needed for interacting with Jira and AWS.
- If not set, default values are used for some parameters like `JIRA_URL` and `JIRA_PROJECT_KEY`.

### 3. Define Services to Monitor

```python
MONITORED_SERVICES = ["EC2", "RDS", "VPC", "S3", "IAM", "Config"]
```

- Specifies the AWS services that you want to monitor for security findings. These services are used to filter relevant findings.

### 4. Get Enabled Regions

```python
def get_enabled_regions():
    """Get list of regions where SecurityHub is enabled."""
    enabled_regions = []
    try:
        # Start with specified regions or default region
        regions_to_check = FILTER_REGIONS if FILTER_REGIONS and FILTER_REGIONS[0] else ["us-east-1"]
        for region in regions_to_check:
            try:
                securityhub = boto3.client("securityhub", region_name=region)
                # Try to make a simple API call to check if SecurityHub is enabled
                securityhub.get_findings(Filters={}, MaxResults=1)
                enabled_regions.append(region)
                print(f":white_check_mark: SecurityHub is enabled in region: {region}")
            except Exception as e:
                if "not subscribed to AWS Security Hub" in str(e):
                    print(f":warning: SecurityHub is not enabled in region: {region}")
```

```
                elif "The security token included in the request is invalid" in str(e):
                    print(f":warning: Region {region} is not available or accessible")
                else:
                    print(f":warning: Error checking region {region}: {str(e)}")
                continue
        return enabled_regions if enabled_regions else ["us-east-1"]  # Fallback to us-east-1 if no regions are enabled
    except Exception as e:
        print(f":x: Error getting enabled regions: {str(e)}")
        return ["us-east-1"]  # Fallback to us-east-1 on error
```

- This function checks if AWS Security Hub is enabled in each region specified in `FILTER_REGIONS` or default regions. It attempts to call the `get_findings` API to verify.
- If an error occurs, it falls back to the default region `us-east-1`.

### 5. Check Relevance of Findings

```
def is_relevant_finding(title, resource_type=None):
    """Check if the finding is related to monitored services."""
    if any(service.lower() in title.lower() for service in MONITORED_SERVICES):
        return True
    if resource_type and any(service.lower() in resource_type.lower() for service in MONITORED_SERVICES):
        return True
    return False
```

- This function checks if a finding is related to the services defined in `MONITORED_SERVICES` (e.g., EC2, RDS, IAM). It evaluates the finding title and resource type.

### 6. Fetch Remediation Recommendation

```
def get_remediation_recommendation(finding):
    """Extract remediation recommendations from the finding."""
    recommendation = ""
    if finding.get("Remediation", {}).get("Recommendation", {}).get("Text"):
        recommendation = finding["Remediation"]["Recommendation"]["Text"]
    elif finding.get("Description"):
        desc = finding["Description"].lower()
        if "recommend" in desc:
            sentences = desc.split(". ")
            for sentence in sentences:
                if "recommend" in sentence:
                    recommendation = sentence.strip().capitalize()
                    break
    if not recommendation:
        title = finding.get("Title", "").lower()
        resource_type = finding.get("ResourceType", "").lower()
        if "port" in title or "ingress" in title:
            recommendation = "Review and restrict network access to only required IP ranges and ports. Consider using Security Groups and NACLs to implement the principle of least privilege."
        elif "s3" in resource_type:
            recommendation = "Review S3 bucket policies, enable bucket encryption, and ensure proper access controls are in place."
        elif "ec2" in resource_type:
            recommendation = "Review EC2 security groups, network access controls, and implement proper instance hardening measures."
        elif "iam" in resource_type:
            recommendation = "Review IAM policies and ensure principle of least privilege is followed. Remove any unused permissions or roles."
```

```
        else:
            recommendation = "Review security controls and compliance requirements for this resource.
Implement security best practices as per AWS guidelines."
    return recommendation
```

- This function extracts remediation recommendations from the finding data. If not found in the response, it checks the description for potential recommendations or provides default suggestions based on the resource type (e.g., EC2, S3).

### 7. Format the Finding for Jira

```
def format_finding(finding):
    """Format Security Hub finding for Jira creation."""
    resource = finding.get("Resources", [{}])[0] if finding.get("Resources") else {}
    return {
        "Title": finding.get("Title", ""),
        "Description": finding.get("Description", ""),
        "Severity": finding.get("Severity", {}).get("Label", "Unknown"),
        "Region": finding.get("Region", "Unknown"),
        "AccountId": finding.get("AwsAccountId", "Unknown"),
        "ProductName": finding.get("ProductFields", {}).get("aws/securityhub/ProductName", "Unknown"),
        "ResourceType": resource.get("Type", "Unknown"),
        "ResourceId": resource.get("Id", "Unknown"),
        "ResourceUrl": resource.get("Details", {}).get("AwsEc2Instance", {}).get("WebLink", "N/A"),
        "Remediation": finding.get("Remediation", {}),
        "Compliance": finding.get("Compliance", {}).get("Status", "Unknown"),
        "WorkflowState": finding.get("Workflow", {}).get("Status", "Unknown"),
        "CreatedAt": finding.get("CreatedAt", "Unknown"),
        "UpdatedAt": finding.get("UpdatedAt", "Unknown")
    }
```

- This function formats the Security Hub finding details into a structured dictionary that is compatible with the Jira API. It retrieves various details like severity, description, recommendation, and resource information.

### 8. Create Jira Issue

```
def create_jira_issue(finding_data):
    """Send Security Hub finding to Jira as an issue."""
    try:
        title = finding_data["Title"]
        if title in processed_titles:
            print(f"⏭️ Skipping duplicate finding: {title}")
            return
        jira_api_url = f"{JIRA_URL}/rest/api/2/issue"
        headers = {"Content-Type": "application/json"}
        auth = (JIRA_EMAIL, JIRA_API_TOKEN)
        recommendation = get_remediation_recommendation(finding_data)
        service_type = "Other"
        resource_type = finding_data.get("ResourceType", "Unknown")
        for service in MONITORED_SERVICES:
            if service.lower() in title.lower() or service.lower() in resource_type.lower():
                service_type = service
                break
        severity = finding_data.get('Severity', 'Unknown')
        severity_label = {
            "CRITICAL": "🔴 *CRITICAL*",
            "HIGH": "🟠 *HIGH*",
            "MEDIUM": "🟡 *MEDIUM*",
            "LOW": "🟢 *LOW*"
```

```
        }.get(severity, f"⚪ *{severity}*")
        issue_description = f"""
h2. 🚨 Security Finding Details
*Description:* {finding_data.get('Description', 'No description provided.')}
*Created:* {finding_data.get('CreatedAt', 'Unknown')}
*Last Updated:* {finding_data.get('UpdatedAt', 'Unknown')}
*Workflow State:* {finding_data.get('WorkflowState', 'Unknown')}
h2. 🎯 Resource Information
* *Service:* {service_type}
* *Resource Type:* {resource_type}
* *Resource ID:* {finding_data.get('ResourceId', 'Unknown')}
* *Resource URL:* {finding_data.get('ResourceUrl', 'N/A')}
* *Severity:* {severity_label}
* *Region:* {finding_data.get('Region', 'Unknown')}
* *Account ID:* {finding_data.get('AccountId', 'Unknown')}
* *Product:* {finding_data.get('ProductName', 'Unknown')}
* *Compliance Status:* {finding_data.get('Compliance', 'Unknown')}
h2. 🛠️ Recommended Actions
{recommendation}
h2. ⚠️ Impact
This security finding indicates potential vulnerabilities that could compromise system security.
* Severity Level: {severity}
* Affected Resource: {resource_type}
* Potential Risk: Data exposure, unauthorized access, or system compromise
* Business Impact: Security posture degradation, compliance violations, potential data breaches
h2. ⏰ Required Actions
* Review and implement the recommended actions within:
** CRITICAL: 72 hours
** HIGH: 14 days
** MEDIUM: 90 days
** LOW: 180 days
* Document all remediation steps taken in this ticket
* Verify the fix by re-running security checks
* Update compliance documentation if needed
* Close this ticket only after confirming the issue is resolved
"""
        labels = [
            service_type,
            severity,
            f"AWS-{finding_data.get('Region', 'Unknown')}",
            f"Account-{finding_data.get('AccountId', 'Unknown')}",
            finding_data.get('Compliance', 'Unknown').replace(" ", "-"),
            "SecurityHub"
        ]
        issue_data = {
            "fields": {
                "project": {"key": JIRA_PROJECT_KEY},
                "summary": f"[{service_type}] {title} [{severity}]",
                "description": issue_description,
                "issuetype": {"name": "Task"},
                "labels": labels
            }
        }
        response = requests.post(jira_api_url, headers=headers, auth=auth, json=issue_data, timeout=10)
        if response.status_code == 201:
            print(f"✅ Jira issue created successfully for: {title}")
            processed_titles.add(title)
        else:
```

```
            print(f"❌ Failed to create Jira issue for {title}: {response.text}")
    except Exception as e:
        print(f"❌ Error creating Jira issue: {str(e)}")
```

- This function sends the formatted finding as a new Jira issue. It constructs the issue description with relevant information, including severity, impacted resources, and remediation steps.

### 9. Fetch Findings from Security Hub

```
def fetch_findings():
    """Fetch Security Hub findings using pagination."""
    findings = []
    enabled_regions = get_enabled_regions()
    for region in enabled_regions:
        try:
            securityhub = boto3.client("securityhub", region_name=region)
            next_token = None
            while True:
                # Updated filters to correctly identify Prowler findings
                filters = {
                    "RecordState": [{"Value": "ACTIVE", "Comparison": "EQUALS"}],
                    "ProductFields": [
                        {
                            "Key": "ProviderName",  # Changed from CompanyName to ProviderName
                            "Value": "Prowler",
                            "Comparison": "EQUALS"
                        }
                    ],
                    "WorkflowStatus": [{"Value": "NEW", "Comparison": "EQUALS"}]
                }
                # Add account filter if specific accounts are provided
                if FILTER_ACCOUNTS and FILTER_ACCOUNTS[0]:
                    filters["AwsAccountId"] = [
                        {"Value": account.strip(), "Comparison": "EQUALS"}
                        for account in FILTER_ACCOUNTS
                    ]
                # Add severity filter if specific severities are provided
                if FILTER_SEVERITY and FILTER_SEVERITY[0]:
                    filters["SeverityLabel"] = [
                        {"Value": severity.strip().upper(), "Comparison": "EQUALS"}
                        for severity in FILTER_SEVERITY
                    ]
                try:
                    params = {
                        "Filters": filters,
                        "MaxResults": 100
                    }
                    if next_token:
                        params["NextToken"] = next_token
                    # Debug: Print the exact filters being used
                    print(f":information_source: Using filters: {json.dumps(filters, indent=2)}")
                    response = securityhub.get_findings(**params)
                    current_findings = response.get("Findings", [])
                    # Debug: If we found findings, print some details of the first one
                    if current_findings:
                        sample = current_findings[0]
                        print(f":information_source: Sample finding details:")
                        print(f"Title: {sample.get('Title')}")
```

```
                        print(f"ProductFields: {json.dumps(sample.get('ProductFields', {}), indent=2)}")
                    print(f":information_source: Found {len(current_findings)} findings in {region}")
                    for finding in current_findings:
                        # Additional check to verify it's a Prowler finding
                        product_fields = finding.get('ProductFields', {})
                        provider = product_fields.get('ProviderName', '')
                        generator_id = finding.get('GeneratorId', '')
                        if 'prowler' in provider.lower() or 'prowler' in generator_id.lower():
                            title = finding.get("Title", "")
                            resource_type = finding.get("Resources", [{}])[0].get("Type", "") if
finding.get("Resources") else ""
                            if is_relevant_finding(title, resource_type) and title not in processed_titles:
                                findings.append(format_finding(finding))
                    next_token = response.get("NextToken")
                    if not next_token:
                        break
                except Exception as e:
                    print(f":x: Error fetching findings batch in region {region}: {str(e)}")
                    print(f"Error details: {str(e)}")
                    break
            print(f":white_check_mark: Successfully processed findings from region {region}")
        except Exception as e:
            print(f":x: Error processing region {region}: {str(e)}")
            continue
    print(f":white_check_mark: Total unique Prowler findings fetched: {len(findings)}")
    return findings
```

- This function fetches Security Hub findings from enabled regions and applies filters for account IDs, severity, and other criteria. It ensures only findings from Prowler are retrieved, if Prowler is listed as the provider.

### 10. Lambda Handler

- The Lambda handler is the entry point for the Lambda function when triggered by an event (e.g., from CloudWatch or manually). It fetches findings from Security Hub, processes them, and creates Jira issues.

```
def lambda_handler(event, context):
    """Process Security Hub findings with title-based deduplication."""
    try:
        if event.get("detail", {}).get("findings"):
            for finding in event["detail"]["findings"]:
                title = finding.get("Title", "")
                resource_type = finding.get("Resources", [{}])[0].get("Type", "") if finding.get("Resources")
else ""
                if is_relevant_finding(title, resource_type):
                    formatted_finding = format_finding(finding)
                    create_jira_issue(formatted_finding)
        print(":information_source: Starting to fetch existing findings...")
        existing_findings = fetch_findings()
        if not existing_findings:
            print(":information_source: No new Prowler findings to process")
        for finding in existing_findings:
            create_jira_issue(finding)
        return {
            "statusCode": 200,
            "body": json.dumps({
                "message": "Processed Prowler findings from enabled regions",
                "findingsCount": len(existing_findings)
            })
        }
```

```
    except Exception as e:
        print(f":x: Error in lambda_handler: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps({"message": f"Error processing findings: {str(e)}"})
        }
```