DEFENDING THE ATT&CK ENGINEERING LABS: DEFENSIVE COUNTERMEASURE DESIGN

Overview

This 4-month intensive lab series is designed to transform analysts into ATT&CK Engineers. The curriculum moves from foundational knowledge to advanced adversary emulation and defensive design using the MITRE ATT&CK and D3FEND frameworks.

Total Labs: 28 **Duration:** 4 Months Regime **Prerequisites:** Basic understanding of SOC operations and Windows/Linux command line.

Month 1: Foundations of ATT&CK & Data Sources

Focus: Understanding the matrix, mapping techniques, and validating data visibility.

Lab 1.1: Navigating the Matrix

- **Objective:** Familiarize with the ATT&CK Navigator and layers.
- **Activity:**
    1. Open ATT&CK Navigator.
    2. Create a layer highlighting "APT29" techniques.
    3. Create a second layer highlighting "APT28" techniques.
    4. Merge layers to find overlaps (common TTPs).
- **Code Example (Navigator JSON Layer Snippet):**

```
{
  "name": "APT29 vs APT28 Overlay",
  "versions": {
    "attack": "14",
    "navigator": "4.9.1"
  },
  "domain": "enterprise-attack",
  "techniques": [
    {
      "techniqueID": "T1059.001",
      "color": "#ff6666",
      "comment": "Common TTP: PowerShell"
    }
  ]
}
```

- **Deliverable:** JSON layer file showing the intersection of APT29 and APT28.

Lab 1.2: TTP Mapping Challenge - "The Phish"

- **Objective:** Map a textual threat report to ATT&CK Techniques.
- **Activity:**
    1. Read a provided CTI report on a phishing campaign.
    2. Identify at least 5 techniques used (e.g., T1566 Phishing, T1204 User Execution).
    3. Map observables (IPs, Hashes) to the techniques.
- **Code Example (Mapping Format):**

```
Report ID: INTEL-2024-001
Technique: T1566.001 (Spearphishing Attachment)
Evidence: "User received email with invoice.pdf"
Technique: T1059.001 (PowerShell)
Evidence: "PDF spawned a hidden powershell window"
```

- **Deliverable:** A marked-up PDF or text file with mapped Technique IDs.

Lab 1.3: Data Source Audit - Windows Event Logs

- **Objective:** Understand what is logged by default vs. what is needed.
- **Activity:**
  1. Execute a simple command: `whoami /priv`.
  2. Check Event Viewer (Security and Sysmon).
  3. Identify which Event ID captured this activity (e.g., 4688, Sysmon 1).
- **Code Example (Command & Log Check):**

```
# Execute command
whoami /priv

# Check for Event ID 4688 (Process Creation) in Security Log
Get-WinEvent -LogName Security | Where-Object {$_.Id -eq 4688} | Select-Object -First 1 | Format-List
```

- **Deliverable:** Screenshot of the log entry corresponding to the command execution.

Lab 1.4: Sysmon Configuration Tuning

- **Objective:** Tune Sysmon to capture Process Creation with full command lines.
- **Activity:**
  1. Install Sysmon with a default config (SwiftOnSecurity).
  2. Run `powershell.exe -enc <base64>`.
  3. Verify the log contains the decoded command line or adequate specific details.
  4. Modify the config to exclude a noisy process (e.g., SplunkUniversalForwarder).
- **Code Example (Sysmon Config Exlusion):**

```
<RuleGroup name="" groupRelation="or">
  <ProcessCreate onmatch="exclude">
    <!-- Exclude Splunk Universal Forwarder -->
    <Image condition="end with">splunk-optimize.exe</Image>
  </ProcessCreate>
</RuleGroup>
```

- **Deliverable:** The modified `sysmon.xml` configuration snippet.

Lab 1.5: Mapping Data Sources to Techniques

- **Objective:** Link abstract techniques to concrete logs.
- **Activity:**
  1. Select T1003 (OS Credential Dumping).
  2. Identify the Data Sources list in ATT&CK (e.g., Process Monitoring, File Monitoring).
  3. Determine specifically *which* Windows Event IDs provide this visibility.
- **Code Example (Mapping Table):** | Technique | Data Source | Component | specific Event ID | | :--- | :--- | :--- | :--- | | T1003.001 (LSASS Dump) | Process Monitoring | Windows Security | 4688 (Process Creation) | | T1003.001 (LSASS Dump) | Process Access | Sysmon | 10 (ProcessAccess) |
- **Deliverable:** A mapping table: Technique -> Data Source -> Specific Event ID.

Lab 1.6: Introduction to OSQuery

- **Objective:** Query endpoint state as a database.
- **Activity:**
  1. Install OSQuery.
  2. Run a query to list all running processes: `SELECT * FROM processes;`.
  3. Run a query to find processes running from temporary folders.
- **Code Example (OSQuery SQL):**

```
-- Find processes running from Temp directories
SELECT name, pid, path, cmdline
FROM processes
WHERE path LIKE '%AppData\\Local\\Temp%';
```

- **Deliverable:** SQL query used to find suspicious process paths.

Lab 1.7: Month 1 Capstone - Visibility Gap Analysis

- **Objective:** Assess coverage for a specific threat group.
- **Activity:**
  1. Choose a Threat Group (e.g., FIN7).
  2. List their top 10 techniques.
  3. Determine if your current lab environment (Sysmon + Default Logs) detects them.
- **Deliverable:** A "Coverage Map" (Green/Yellow/Red) for the top 10 techniques.

Month 2: Detection Engineering & Sigma

Focus: Writing robust detection logic and reducing false positives.

Lab 2.1: First Sigma Rule - Process Creation

- **Objective:** Write a standard Sigma rule.
- **Activity:**
  1. Target: `certutil.exe` downloading a file.
  2. Write a Sigma rule strictly detecting `certutil` with `-urlcache` and `-split` flags.
  3. Convert the rule to Splunk SPL or Elastic Query.
- **Code Example (Sigma YAML):**

```
title: Certutil Download
id: 1234
status: experimental
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith: '\certutil.exe'
        CommandLine|contains: 'urlcache'
    condition: selection
```

- **Deliverable:** Valid `certutil.yml` Sigma rule.

Lab 2.2: Regex for Defenders

- **Objective:** Master Regular Expressions for log parsing.
- **Activity:**
  1. Given a raw log file.
  2. Write a Regex to extract every IPv4 address.
  3. Write a Regex to extract Base64 strings longer than 20 characters.
- **Code Example (Regex):**

```
# IPv4 Extraction
\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b

# Base64 String (>20 chars)
[a-zA-Z0-9+/]{20,}={0,2}
```

- **Deliverable:** The regex patterns.

Lab 2.3: Detecting Parent-Child Relationships

- **Objective:** Detect suspicious process trees.
- **Activity:**
  1. Simulate Word spawning PowerShell (Macro execution).
  2. Identify the Parent Process ID (PPID) and Image.
  3. Write a rule: `ParentImage="*WINWORD.EXE"` AND `Image="*powershell.exe"`.
- **Code Example (Sigma Detection Logic):**

```
detection:
    selection:
        ParentImage|endswith: '\WINWORD.EXE'
        Image|endswith: '\powershell.exe'
    condition: selection
```

- **Deliverable:** Sigma rule for suspicious office child processes.

Lab 2.4: PowerShell Script Block Logging (ETW)

- **Objective:** Detect obfuscated PowerShell.
- **Activity:**
  1. Enable PowerShell Script Block Logging (EID 4104).
  2. Run an obfuscated script (invoke-obfuscation).
  3. Locate the de-obfuscated code in Event Viewer.
- **Code Example (Enable Logging):**

```
# Enable Script Block Logging via Registry
New-ItemProperty HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ScriptBlockLogging -Name EnableScriptBlockLogging -Value
```

- **Deliverable:** Screenshot of EID 4104 showing cleartext code.

Lab 2.5: False Positive Tuning

- **Objective:** Refine a noisy rule.
- **Activity:**
  1. Deploy a rule detecting "Net.exe user /add".
  2. Simulate a legitimate admin activity that triggers it.
  3. Modify the rule to whitelist the admin's specific workstation or user account.
- **Code Example (Sigma Exclusion):**

```
detection:
    selection:
        Image|endswith: '\net.exe'
        CommandLine|contains: 'user /add'
    filter_admin_workstation:
        ComputerName: 'ADMIN-PC-01'
    condition: selection and not filter_admin_workstation
```

- **Deliverable:** Tuned rule with `condition: selection and not filter`.

Lab 2.6: YARA Rules for Static Analysis

- **Objective:** Detect malicious files at rest.
- **Activity:**
  1. Download a sample Mimikatz binary (or simulation).
  2. Write a YARA rule looking for specific strings (e.g., "gentilkiwi").
  3. Scan a directory to verify it hits.
- **Code Example (YARA):**

```
rule Detect_Mimikatz_Strings {
    meta:
        description = "Detects Mimikatz by Common Strings"
    strings:
        $s1 = "gentilkiwi" ascii wide
        $s2 = "sekurlsa" ascii wide
    condition:
        any of them
}
```

- **Deliverable:** `mimikatz.yar` file.

Lab 2.7: Month 2 Capstone - The Detection Pipeline

- **Objective:** End-to-end detection creation.

- **Activity:**
    1. Pick a technique (e.g., T1053 Scheduled Task).
    2. Simulate it.
    3. Collect Logs.
    4. Write Sigma Rule.
    5. Validate the alert triggers.
- **Deliverable:** A report documenting the 5 steps.

Month 3: Countermeasures (D3FEND) & Deception

Focus: Active defense, hardening, and interfering with attacks.

Lab 3.1: Introduction to D3FEND

- **Objective:** Map ATT&CK Techniques to D3FEND Countermeasures.
- **Activity:**
    1. Take T1003 (Credential Dumping).
    2. Find the corresponding D3FEND artifact (e.g., Credential API Call Analysis).
    3. Explain how this countermeasure works theoretically.
- **Code Example (SPARQL Query for D3FEND):**

```
PREFIX d3f: <http://d3fend.mitre.org/ontologies/d3fend.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

# Find defensive techniques associated with Credential Dumping
SELECT ?defense ?label WHERE {
  ?attack d3f:attack-id "T1003" .
  ?defense d3f:mitigates ?attack .
  ?defense rdfs:label ?label .
}
```

- **Deliverable:** Short summary of the D3FEND technique.

Lab 3.2: Hardening - Attack Surface Reduction (ASR)

- **Objective:** Implement Microsoft ASR rules.
- **Activity:**
    1. Enable ASR rule: "Block all Office applications from creating child processes".
    2. Attempt to run the macro from Lab 2.3.
    3. Verify the block in Windows Defender logs.
- **Code Example (PowerShell ASR):**

```
# Rule GUID for "Block Office apps from creating child processes": D4F940AB-401B-4EFC-AADC-AD5F3C50688A
Add-MpPreference -AttackSurfaceReductionRules_Ids D4F940AB-401B-4EFC-AADC-AD5F3C50688A -AttackSurfaceReductionRules_Actions Ena
```

- **Deliverable:** Screenshot of the ASR block event.

Lab 3.3: Canary Tokens (Deception)

- **Objective:** Deploy a honeytoken.
- **Activity:**
    1. Generate a Canary Token (Word doc or URL).
    2. Place it on a file share or desktop.
    3. "Trigger" it by opening the file.
    4. Receive the alert.
- **Code Example (Deployment Concept):**

```
# Concept: Creating a hidden link in a PDF/Doc (not actual code, but logic)
# <link href="http://canarytoken.com/uniqid/image.png">

# Placing it:
cp malicious_looking_invoice.pdf /var/www/html/secure_docs/
```

- **Deliverable:** Email or dashboard screenshot of the canary trigger.

Lab 3.4: Honey Accounts

- **Objective:** Detect enumeration.
- **Activity:**
    1. Create a fake domain admin account (e.g., `Start_Admin`).
    2. Monitor for *any* authentication attempt or query regarding this account (4624, 4769).
    3. Simulate an attacker trying to brute force it.
- **Code Example (Splunk Detection Logic):**

```
index=windows EventCode=4624 TargetUserName="Start_Admin"
| stats count by SourceNetworkAddress
| table _time, TargetUserName, SourceNetworkAddress, count
```

- **Deliverable:** Alert specifically for the Honey Account.

Lab 3.5: Network Interfering - DNS Sinkholing

- **Objective:** Prevent C2 communication.
- **Activity:**
    1. Configure a `hosts` file or local DNS to redirect `malicious-site.com` to `127.0.0.1`.
    2. Attempt `ping malicious-site.com`.
    3. Observe the traffic failure.
- **Code Example (Hosts File):**

```
# C:\Windows\System32\drivers\etc\hosts
127.0.0.1       malicious-c2.com
127.0.0.1       bad-site.org
```

- **Deliverable:** Screenshot of the ping resolving to localhost.

Lab 3.6: Decoy Processes

- **Objective:** Confuse automated malware.
- **Activity:**
    1. Run a script that renames a innocuous executable to `wireshark.exe` or `fiddler.exe`.
    2. Run a piece of malware (in a safe VM) that checks for analysis tools.
    3. Observe if the malware terminates itself.
- **Code Example (PowerShell Decoy):**

```
# Launch Notepad but rename the window/process to look like a tool
Start-Process notepad.exe
# (Advanced: Typically requires renaming the binary on disk to 'wireshark.exe' then running it)
Copy-Item C:\Windows\System32\notepad.exe C:\Tools\wireshark.exe
Start-Process C:\Tools\wireshark.exe
```

- **Deliverable:** Observation notes.

Lab 3.7: Month 3 Capstone - Designing a Defensible Network

- **Objective:** Architect a network segment with embedded countermeasures.
- **Activity:**
    1. Draw a diagram of a subnet.
    2. Place 3 distinct countermeasures (e.g., Deception User, ASR policies, Egress Filtering).
    3. Explain which ATT&CK techniques each stops.
- **Deliverable:** Network Diagram with annotations.

Month 4: Purple Teaming & Advanced Emulation

Focus: Testing defenses against realistic adversarial behaviors.

Lab 4.1: Atomic Red Team - Setup & Execution

- **Objective:** Run a prescribed atomic test.
- **Activity:**

1. Install Atomic Red Team (Invoke-Atomic).
2. Run a test for T1059.001 (PowerShell).
3. Verify execution without errors.

- **Code Example (Invoke-Atomic):**

```
# Install
IEX (IWR 'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredteam/master/install-atomicredteam.ps1' -UseBasicParsing

# Execute T1059.001
Invoke-AtomicTest T1059.001 -TestNumbers 1
```

- **Deliverable:** Console output of the atomic test.

Lab 4.2: Validating Detections with Atomic

- **Objective:** Closed-loop testing.
- **Activity:**
  1. Ensure your Sigma rule from Lab 2.4 (Script Block) is active.
  2. Run the Atomic test for PowerShell obfuscation.
  3. Confirm the alert fired. If not, fix the rule.
- **Code Example (Validation Loop):**

```
# 1. Clear Logs
wevtutil cl "Microsoft-Windows-PowerShell/Operational"

# 2. Run Attack
Invoke-AtomicTest T1027 -TestNumbers 1

# 3. Check Logs (Automated)
Get-WinEvent -LogName "Microsoft-Windows-PowerShell/Operational" | Where-Object {$_.Id -eq 4104}
```

- **Deliverable:** "Pass" validation status.

Lab 4.3: Caldera - The Basics

- **Objective:** Automated adversary emulation.
- **Activity:**
  1. Install Caldera server.
  2. Deploy a Sandcat agent to a victim VM.
  3. Verify the agent check-in.
- **Code Example (Agent Install):**

```
$server="http://localhost:8888";
$url="$server/file/download";
$wc=New-Object System.Net.WebClient;
$wc.Headers.add("platform","windows");
$wc.Headers.add("file","sandcat.go");
$data=$wc.DownloadData($url);
get-process | ? {$_.modules.FileName -like "C:\Users\Public\sandcat.exe"} | stop-process -f;
[io.file]::WriteAllBytes("C:\Users\Public\sandcat.exe",$data) | start-process -argumentlist "-server $server -group red"
```

- **Deliverable:** Screenshot of the Agent in the Caldera dashboard.

Lab 4.4: Caldera - Running an Operation

- **Objective:** Execute a campaign.
- **Activity:**
  1. Create an Operation in Caldera using the "Discovery" profile.
  2. Run the operation against the victim agent.
  3. Review the steps taken (whoami, net user, etc.).
- **Code Example (Operation Config YAML - Conceptual):**

```
name: Discovery Operation
adversary: 247d519d-b8d9-482f-9818-47754d9c7e09
group: red
planner: atomic
auto_close: true
```

- **Deliverable:** Operation Report export (PDF/JSON).

Lab 4.5: Emulating Ransomware Behavior

- **Objective:** Simulate the impact without the damage.
- **Activity:**
  1. Use a simulator (like RTA or a custom script) to bulk-rename files in a dummy folder (mock encryption).
  2. Delete Shadow Copies (vssadmin delete shadows).
  3. Detect the high-volume file modification or the vssadmin command.
- **Code Example (Ransomware Simulator Script):**

```
# Mock Encryption (Rename files to .enc)
Get-ChildItem -Path "C:\TestDocs" -Filter "*.txt" | ForEach-Object {
    Rename-Item $_.FullName -NewName ($_.Name + ".enc")
}

# Delete Shadow Copies
vssadmin delete shadows /all /quiet
```

- **Deliverable:** Alert logic for "Mass file modification".

Lab 4.6: Purple Team Reporting

- **Objective:** Communicating value.
- **Activity:**
  1. Compile results from Lab 4.4.
  2. Create a "Purple Team Scorecard".
  3. Columns: Technique | Status (Blocked/Detected/Missed) | Improvement Plan.
- **Deliverable:** A filled-out matrix/scorecard.

Lab 4.7: Month 4 Capstone - Full Chain Emulation

- **Objective:** The Final Exam.
- **Scenario:** Intruder moves from Phishing -> Execution -> Discovery -> Lat Movement -> Impact.
- **Activity:**
  1. Script or manually execute a 5-step kill chain.
  2. Record every detection and every miss.
  3. Draft a "Post-Mortem" report.
- **Deliverable:** Final Incident Report & Defense Improvement Plan.

End of Regime.