Cybersecurity Threat Hunting Runbook: Detecting APT38/Lazarus Group Activities

1. Purpose

This runbook outlines a proactive threat hunting plan within Microsoft Sentinel to detect sophisticated Advanced Persistent Threat (APT) activities, specifically focusing on tactics, techniques, and procedures (TTPs) associated with North Korean state-sponsored groups like **APT38 (also known as Contagious Interview or Famous Chollima) and Lazarus Group**. These groups are known for targeting the financial sector, especially cryptocurrency and blockchain technology, with increasing sophistication and speed of execution, often by compromising developers with access to critical systems and code repositories.

2. Threat Context

- **Threat Actors:** APT38 (Contagious Interview, Famous Chollima), Lazarus Group, TradeTraitor.
- **Motivation:** Financial gain, having successfully stolen billions in cryptocurrency and financial assets.
- **Key TTPs:**
  - **Social Engineering:** Threat actors leverage fake job interviews, recruitment offers, and elaborate fake identities/front companies (e.g., BlockNovas LLC, Angeloper Agency, and SoftGlide LLC). They often use **AI-generated images for "employees"** to appear legitimate.
  - **GitHub Abuse:** Malicious code is disguised as "skill assessments" or hidden within external dependencies in GitHub repositories. This includes techniques like inserting numerous spaces before malicious code to render it off-screen in the GitHub interface, or hiding backdoors within seemingly legitimate JavaScript files.
  - **Custom Malware:** APT actors build custom code tailor-made for the victim's infrastructure, tools, and DevOps environment, rather than relying on generic toolkits. Attacks are often layered and modular, starting within a specific environment (e.g., Java) and then breaking out to the operating system for privilege escalation. Examples include BeaverTail, InvisibleFerret, OtterCookie, and ForexProvider.
  - **"Compiling on Target":** This is an advanced tradecraft where attackers compile malicious artefacts using legitimate compilers like `csc.exe` (C#/.NET compiler) directly on the target

machine, which helps them avoid traditional Indicator of Compromise (IoC)-based detection.
    - **Persistence:** Techniques observed include exploiting Java serialization vulnerabilities, deploying as Maven plugins, and establishing persistence via Windows Registry Run Keys, Linux Desktop entries, or MacOS LaunchAgents.
    - **Evasion:** Attackers use common-looking executable names (e.g., `Edgebrowser.exe`, `explorer.exe`, or files with `.db` extensions). These files are often dropped into writable locations such as `\Users\Public`. They may also use proxyloaders (e.g., `rundll32` to execute `.db` files as Win32 DLLs) to bypass Endpoint Detection and Response (EDR).

3. Prerequisites for Threat Hunting

- **Microsoft Sentinel Workspace:** Configured and operational.
- **Data Connectors:** Ensure relevant logs are ingested, primarily **`DeviceProcessEvents` from Microsoft Defender for Endpoint (MDE)** for process execution analysis. For comprehensive hunting, also integrate logs from mail delivery systems (e.g., Office 365, Exchange Online) and GitHub audit logs, if applicable to your environment.
- **Understanding of "Normal" Behaviour:** Familiarity with typical developer activities and DevOps environment usage within your organisation is crucial for effective anomaly detection.

4. Threat Hunting Hypotheses & KQL Queries

The following sections detail specific threat hunting hypotheses derived from APT38/Lazarus Group TTPs, along with corresponding KQL queries for Microsoft Sentinel.

4.1. Hunt for Suspicious GitHub Activity

MITRE ATT&CK Techniques:

- T1566: Phishing
- T1204.003: User Execution: Malicious Image
- T1195.001: Supply Chain Compromise: Compromise Software Dependencies and Development Tools

Hypothesis: Threat actors use fake GitHub repositories, disguised as skill assessments or legitimate projects, to entice developers to clone and execute malicious code.

Hunting Focus: Identify `git clone` commands on corporate devices that are *not* targeting known, legitimate corporate GitHub repositories.

KQL Query:

```
DeviceProcessEvents
| where Timestamp > ago(7d) // Adjust timeframe as needed, e.g.,
`ago(30d)` for longer historical searches
| where ProcessCommandLine contains "git" and ProcessCommandLine contains
"clone"
| where ProcessCommandLine !contains "<YourCorporateGitHubName>" //
**Replace with your organization's known GitHub names (e.g.,
`github.com/mycompany`)**
| where ProcessCommandLine !contains "com.microsoft" // Exclude
Microsoft's own repos if not relevant to your internal dev
```

```
| project TimeGenerated, AccountName, DeviceName,
InitiatingProcessCommandLine, ProcessCommandLine, FolderPath, FileName,
SHA256
| summarize count() by AccountName, DeviceName, ProcessCommandLine
| sort by count_ desc
```

Investigation Steps if Results Found:

- **Examine `ProcessCommandLine`:** What specific repository was cloned? Is it publicly accessible? Does it contain elements like "skill assessment" or "interview"?.
- **Investigate `AccountName` and `DeviceName`:** Is this user typically involved in cloning external repositories? Is the device a development workstation?
- **Contextualise:** Was this activity part of a legitimate recruitment process or an unexpected interaction? Check for associated email lures or LinkedIn messages.
- **Check Repository Content:** If accessible, examine the cloned repository for obfuscated code, unusual dependencies, or "second grade" vulnerable code (e.g., numerous spaces before malicious code, hidden backdoors in JavaScript, or deserialization vulnerabilities).

4.2. Hunt for Suspicious Mail Delivery Logs (Recruitment Lures)

MITRE ATT&CK Techniques:

- T1566.001: Phishing: Spearphishing Attachment
- T1566.002: Phishing: Spearphishing Link
- T1534: Internal Spearphishing

Hypothesis: Threat actors use recruitment lures, often via email (after initial contact on platforms like LinkedIn), to deliver malicious content or direct victims to fake GitHub repositories.

Hunting Focus: Identify suspicious recruitment emails or messages, particularly from platforms like LinkedIn, that appear inappropriate or non-compliant, or those directing users to external code repositories.

KQL Query:

```
EmailEvents
| where Timestamp > ago(7d) // Adjust timeframe as needed, e.g.,
`ago(30d)` for longer historical searches
| where SenderFromAddress has "linkedin"
      or SenderDisplayName has "linkedin"
      or SenderDomain has "linkedin"
| project Timestamp, SenderFromAddress, SenderDisplayName, SenderDomain,
RecipientEmailAddress, Subject, DeliveryAction, DeliveryLocation
| order by Timestamp desc
```

the query above can be very verbouse when applied to long timeframes, the following query is more specific and selects typical first touch from recruiters

```
```

```
EmailEvents
| where Timestamp > ago(7d) // Adjust timeframe as needed, e.g.,
`ago(30d)` for longer historical searches
| where SenderDisplayName has "LinkedIn"
      and Subject has_any
("job","offer","position","interview","opportunity", "technical
assessment", "skill assessment", "cryptocurrency job")
| project Timestamp, SenderFromAddress, SenderDisplayName,
SenderFromDomain, RecipientEmailAddress, Subject, DeliveryAction,
DeliveryLocation
| order by Timestamp desc
```

Note: The provided sources mention observing "mail delivery logs" but do not offer specific KQL queries for this data type within Microsoft Sentinel's `DeviceProcessEvents` schema. This activity would typically require data from email platforms (e.g., `EmailEvents`, `EmailUrlInfo` if using Microsoft 365 Defender, or similar tables from other email security solutions).

Conceptual Approach (no direct KQL from provided sources):

- **Review Email Logs:** Search for emails with keywords like "job offer," "technical assessment," "interview," "skill assessment," "cryptocurrency job".
- **Filter by Source:** Pay attention to emails originating from unfamiliar or newly registered domains, especially those purporting to be from cryptocurrency companies.
- **Examine Links/Attachments:** Identify emails containing links to external, untrusted GitHub repositories or suspicious attachments.
- **Cross-reference:** Check if recipients of such emails subsequently exhibit suspicious `git clone` activity or other unusual process executions.
- **Investigate Sender Identity:** Scrutinise the sender's email address and any associated company names for signs of fake identities or front companies (e.g., BlockNovas LLC, Angeloper Agency, SoftGlide LLC). Look for signs of AI-generated employee images on associated websites.

4.3. Hunt for Abnormal DevOps Environment Usage & "Compiling on Target"

MITRE ATTACK T1027.004: Obfuscated Files or Information: Compile After Delivery

Hypothesis: APT actors build custom code tailored for victims' infrastructure and DevOps environments, which might involve executing commands that appear legitimate but are atypical for the specific developer or group. This includes the advanced tradecraft of "compiling on target".

Hunting Focus: Establish a baseline of "normal" developer behaviour within your DevOps environment and look for deviations or commands that are rare or unusual for a specific user or machine. Specifically, monitor for instances of `csc.exe` (the C#/.NET compiler) executing unusual compile commands, especially when launched as a child process of non-console applications (e.g., a Java process or Maven invoking `csc`).

Note: The sources state that identifying "abnormal use of the DevOps environment" is a "challenging task" that can be accomplished using "sophisticated threat hunting using notebooks (e.g., Jupyter notebooks)". No direct KQL query is provided for this broad category, as it often requires behavioural baselining and advanced analytics beyond a simple query.

KQL Query for `csc.exe` Anomalies (Compiling on Target):

```
DeviceProcessEvents
| where Timestamp > ago(7d) // Adjust timeframe
| where FileName =~ "csc.exe"
| where InitiatingProcessFileName !in ("cmd.exe", "powershell.exe",
"msbuild.exe", "devenv.exe", "Developer Command Prompt for VS") // Exclude
common, legitimate parent processes for csc.exe
| project TimeGenerated, AccountName, DeviceName,
InitiatingProcessCommandLine, ProcessCommandLine, FolderPath, FileName,
SHA256
| where AccountName !in ("local service","network service", "system")
//excluding cases where the possible attacker already escalated
| extend ParentProcess = InitiatingProcessCommandLine // Helpful for
contextual analysis
| summarize count() by AccountName, DeviceName, ParentProcess,
ProcessCommandLine, FolderPath, FileName
| sort by count_ desc
```

- the query might require additional tuning on filtering by InitiatingProcessCommandline when something "usual" is identified
- the query is excluding cases when the attacker already escalated, filtering out also usual system activities of processes like citrix or other vendors doing at run compiling

Investigation Steps if Results Found:

- **Investigate Context:** Why is `csc.exe` being invoked by a non-console application (e.g., a Java process or Maven) or an unusual parent process? Is this a new, legitimate build process or a rare, authorised development activity?.
- **Examine `ProcessCommandLine`:** What source files are being compiled? Are they expected? Is the compilation output going to an unusual location?
- **Review `AccountName` and `DeviceName`:** Is this a specific developer's machine? Is this activity unusual for them?

4.4. Hunt for Unusual Executable Locations and Processes and exec via rundll32

MITRE Attack T1218.001: System Binary Proxy Execution: Signed Binary Proxy Execution: Rundll32 **MITRE Attack** T1218.010: System Binary Proxy Execution: Regsvr32

Hypothesis: Threat actors drop malicious files with common-looking names into writable, non-standard locations and execute them to blend in with legitimate activity and bypass detection. Malicious `.db` files can function as Win32 DLLs executed via `rundll32`.

Hunting Focus: Identify common executable names (e.g., `Edgebrowser.exe`, `explorer.exe`, `MusicUtil.exe`, `desktop-build.exe`) running from unexpected or user-writable locations (e.g., `\Users\Public`, `\AppData\Local\Temp`, `\TEMP`). Look for `.db` files being executed as Win32 DLLs via `rundll32`.

KQL Query:

```
// Query for suspicious common-looking executables in unusual locations
DeviceProcessEvents
| where Timestamp > ago(7d) // Adjust timeframe
| where FileName in ("edgebrowser.exe", "EdgeWebView.exe", "explorer.exe",
"musicutil.exe", "desktop-build.exe") // Include other common-looking
malicious executable names from intelligence
| where FolderPath contains @"\Users\Public\" or FolderPath contains
@"\AppData\Local\Temp\" or FolderPath contains @"\TEMP\" // Common user-
writable, non-standard locations
| project TimeGenerated, AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName, SHA256, InitiatingProcessFileName,
InitiatingProcessCommandLine
| summarize count() by AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName
| sort by count_ desc

// Query for suspicious .db file execution via rundll32
DeviceProcessEvents
| where Timestamp > ago(7d)
| where FileName =~ "rundll32.exe"
| where (ProcessCommandLine contains ".db" or ProcessCommandLine contains
".dll") and ProcessCommandLine contains "#" // Typical rundll32 for DLLs
(e.g., rcmdun.db,#1)
| where ProcessCommandLine contains @"\Users\Public\" // Look for
execution from public writable locations, as seen in attacks
| project TimeGenerated, AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName, SHA256, InitiatingProcessFileName,
InitiatingProcessCommandLine
| summarize count() by AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName
| sort by count_ desc
```

alternative and more effectively you can run a union query between the two info obtained above (use the query below)

```
union
(
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName in ("edgebrowser.exe", "EdgeWebView.exe",
"explorer.exe", "musicutil.exe", "desktop-build.exe")
    | where FolderPath contains @"\Users\Public\"
        or FolderPath contains @"\AppData\Local\Temp\"
        or FolderPath contains @"\TEMP\"
    | project TimeGenerated, AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName, SHA256, InitiatingProcessFileName,
InitiatingProcessCommandLine
),
(
    DeviceProcessEvents
```

```
    | where Timestamp > ago(7d)
    | where FileName =~ "rundll32.exe"
    | where (ProcessCommandLine contains ".db" or ProcessCommandLine
contains ".dll")
        //and ProcessCommandLine contains "#"
        and ProcessCommandLine contains @"\Users\Public\"
    | project TimeGenerated, AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName, SHA256, InitiatingProcessFileName,
InitiatingProcessCommandLine
)
| summarize count() by AccountName, DeviceName, ProcessCommandLine,
FolderPath, FileName
| sort by count_ desc
```

Investigation Steps if Results Found:

- **Verify Legitimacy:** Is there a legitimate reason for this executable to be in this location? For instance, `Edgebrowser.exe` should typically reside in `C:\Program Files\Microsoft Edge\Application`.
- **Process Tree Analysis:** What parent process launched the suspicious executable? Look for unusual chains (e.g., Java process launching `Edgewebview.exe` which then downloads `MusicUtil.exe`).
- **Network Connections:** Check `DeviceNetworkEvents` for outbound connections from these processes to suspicious IPs or domains (e.g., `<redacted>`).
- **File Analysis:** Collect the suspicious file for static and dynamic analysis. Pay particular attention to `.db` files that are actually Win32 DLLs functioning as stagers.

4.5. Investigate for java process dropping files

MITRE ATTACK T1036.005: Masquerading: Match Legitimate Name or Location **MITRE ATTACK** T1027.002: Obfuscated Files or Information: Software Packing **MITRE ATTACK** T1105: Ingress Tool Transfer

Hypothesis: APT actors exploited java environment and can drop, download or use jave to write files on public local folder

Hunting Focus: look into c:\users\public or any other writable folder for Windows OS in the DeviceFileEvents table to identiy any create, modify or delete of files, any file, where the initiate process is java*

Note: we are looking for an event like this: FileCreated {"FileType":"PortableExecutable"} rcmdun.db 130560 C:\Users\Public\rcmdun.db user@organisation EdgeWebView.exe edgewebview.exe c:\users\public\edgewebview.exe 25020 Low java.exe 27900 ... DeviceFileEvents

KQL Query for java process dropping:

```
DeviceFileEvents
| where Timestamp > ago(60d) // Adjust timeframe as needed
| where FolderPath contains @"\Users\Public\" or FolderPath contains
@"\AppData\Local\" or FolderPath contains @"\Temp\" // Common writable
locations
| where InitiatingProcessFileName matches regex @"java.*\.exe" // Any Java
```

```
process
| where ActionType in ("FileCreated", "FileModified")
| where FolderPath !contains "Android"
| where FileName !contains ".jar"
| where FileName has_any (".exe", ".dll", ".db",".js",".py",".ts")
| project TimeGenerated, DeviceName, ActionType, FileName, FolderPath,
FileSize,
         InitiatingProcessAccountName, InitiatingProcessFileName,
InitiatingProcessCommandLine,
         InitiatingProcessFolderPath, InitiatingProcessId, SHA1, SHA256
| summarize count() by DeviceName, InitiatingProcessFileName, FileName,
FolderPath
| sort by count_ desc
```

we have a problem with this, it might be very very verbouse if the developers are running on frameworks like jetbrain (based) or other complex j2ee. it might be convenient to reduce the target folder excluding /appdata/local

Investigation Steps if Results Found:

1. **Verify File Type**: Examine the file extension and content type. Executable files (`.exe`, `.dll`, `.db`) or script files (`.js`, `.py`) dropped by Java processes are particularly suspicious.
2. **Process Context**: Review the full Java process command line to understand what legitimate application might be running. Compare against known development activities in your environment.
3. **File Analysis**: Submit suspicious files for malware analysis. Look for signs of obfuscation or encryption.
4. **User Context**: Determine if the user regularly works with Java applications and if this behavior is normal for their role.
5. **Network Activity**: Check if the Java process or the dropped file made any suspicious network connections after execution.
6. **Persistence Mechanisms**: Check if the dropped files are being used to establish persistence (e.g., through registry modifications or scheduled tasks).

4.6. Investigate execution of possible malicious app

MITRE ATTACK T1059.006: Command and Scripting Interpreter: Python **MITRE ATTACK** T1059.007: Command and Scripting Interpreter: JavaScript **MITRE ATTACK** T1059: Command and Scripting Interpreter **MITRE ATTACK** T1204.002: User Execution: Malicious File **MITRE ATTACK** T1064: Scripting

Hypothesis: APT actors where able to convince user to download a malicious development app, or skill-test, in languages like java, node, python. the app is on the disk and the victim user is testing it running as usual

Hunting Focus: we want to identify all the execution of atypical or unusal apps, in general list all the execution of:

1. node a_javascript_file.js
2. npm start and npm run
3. python a_python_file.py

Note: the result of this query might contain obviously false positives and normal files that a developer user can usually run. we must identify if this is typical or not and AI could help in the process of identify execution of files that are not known. Use AI to do a first check. this check might require human valuation.

KQL Query for execution of possible malicious app:

```
// Look for execution of JavaScript, Python, and other scripting languages
in unusual locations
DeviceProcessEvents
| where Timestamp > ago(7d) // Adjust timeframe as needed
| where
    // Node.js execution patterns
    (FileName =~ "node.exe" and ProcessCommandLine matches regex
@"\bnode\s+\S+\.js\b") or
    (FileName =~ "npm" and (ProcessCommandLine contains "start" or
ProcessCommandLine contains "run")) or
    // Python execution patterns
    (FileName =~ "python.exe" and ProcessCommandLine matches regex
@".*\.py") or
    // Add other interpreters as needed
    (FileName =~ "java.exe" and ProcessCommandLine contains "-jar")
//| where
    // Focus on suspicious locations
    //ProcessCommandLine contains @"\Users\Public\" or
    //ProcessCommandLine contains @"\Downloads\" or
    //ProcessCommandLine contains @"\Temp\" or
    //ProcessCommandLine contains @"\Desktop\"
|where
    ProcessCommandLine !contains "surefire" and
    ProcessCommandLine !contains "_in_process.py" and
    InitiatingProcessFileName in ("cmd.exe","java.exe")
| project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
         FolderPath, InitiatingProcessFileName,
InitiatingProcessCommandLine
| extend ScriptPath = extract(@"((?:\w:\\|\\\\)(?:[^\\]+\\)*[^\\]+\.
[jspyjar]{2,3})", 1, ProcessCommandLine)
| summarize FirstSeen=min(TimeGenerated), LastSeen=max(TimeGenerated),
          ExecutionCount=count() by DeviceName, AccountName, FileName,
ScriptPath, ProcessCommandLine
| order by ExecutionCount asc // Focus on rare executions first
```

if the queries is returning too many results, focus only on suspicious location and remove the comments tags

Investigation Steps if Results Found:

1. **Baseline Comparison**: Determine if this is normal activity for the user. Developers may regularly run scripts, but the location and frequency matter.
2. **Script Analysis**: Retrieve and analyze the script content for suspicious code, obfuscation, or encoded commands.

3. **Execution Context**: Review what triggered the script execution (email attachment, download, etc.).
4. **Network Activity**: Check for any unusual network connections established during or after script execution.
5. **User Interview**: If appropriate, ask the user about the script execution - was it part of a legitimate task or test?
6. **Reputation Check**: Check if the script hash or name matches known IOCs from threat intelligence sources.
7. **Additional Artifacts**: Look for additional files created or modified by the script execution.

4.7. Investigate possible UAC Bypass attempts

MITRE ATTACK T1548.002: Abuse Elevation Control Mechanism: Bypass User Account Control **MITRE ATTACK** T1134: Access Token Manipulation **MITRE ATTACK** T1574: Hijack Execution Flow **MITRE ATTACK** T1546: Event Triggered Execution

Hypothesis: APT actors where able to exploit and take remote control of a system under normal non-privileged user and the process is running with Low integrity. The process attempts one of the known techniques to perform UAC bypass.

Hunting Focus: in this hunt we would love to look into all the possible techniques, but there are more than 40 techniques; the entire coverage might require a runbook in his own. however we focus on covering at least the 3 main techniques explained in this article: https://falconforce.nl/falconfriday-detecting-uac-bypasses-0xff16/

Note: UAC bypass techniques often involve manipulating registry keys and using trusted Windows binaries to execute malicious code with elevated privileges. We're focusing on several key techniques:

1. Trusted binaries (fodhelper.exe, eventvwr.exe, sdclt.exe) with registry hijacking
2. Elevated COM interface techniques (41, 43, and 65) from the FalconForce article, which involve COM interface abuse

KQL Query for UAC Bypass detection:

```
// UAC Bypass Detection - Based on FalconForce article
// Part 1: Detecting trusted binary UAC bypasses

// Method 1: Detecting fodhelper.exe UAC bypass
union
(
    // Method 1: Detecting fodhelper.exe UAC bypass
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName =~ "fodhelper.exe"
    | join kind=inner (
        DeviceRegistryEvents
        | where RegistryKey contains @"\Software\Classes\ms-
settings\shell\open\command"
        | where RegistryValueName == "(Default)" or RegistryValueName ==
"DelegateExecute"
    ) on DeviceId, DeviceName
```

```
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
            RegistryKey, RegistryValueName, RegistryValueData
),
(
    // Method 2: Detecting eventvwr.exe UAC bypass
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName =~ "eventvwr.exe"
    | join kind=inner (
        DeviceRegistryEvents
        | where RegistryKey contains
@"\Software\Classes\mscfile\shell\open\command"
        | where RegistryValueName == "(Default)"
    ) on DeviceId, DeviceName
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
            RegistryKey, RegistryValueName, RegistryValueData
),
(
    // Method 3: Detecting sdclt.exe UAC bypass
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName =~ "sdclt.exe"
    | join kind=inner (
        DeviceRegistryEvents
        | where RegistryKey contains
@"\Software\Classes\exefile\shell\runas\command"
           or RegistryKey contains
@"\Software\Classes\AppX82a6gwre4fdg3bt635tn5ctqjf8msdd2\Shell\open\comman
d"
        | where RegistryValueName == "(Default)" or RegistryValueName ==
"IsolatedCommand"
    ) on DeviceId, DeviceName
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
            RegistryKey, RegistryValueName, RegistryValueData
)
| summarize count() by DeviceName, AccountName, FileName,
ProcessCommandLine, RegistryKey, RegistryValueName, RegistryValueData
| order by count_ desc
```

```
// Part 2: Detecting Elevated COM Interface UAC bypasses (Techniques 41,
43, and 65) from the article: https://falconforce.nl/falconfriday-
detecting-uac-bypasses-0xff16/

// Technique 41: CMSTPLUA COM interface abuse
union
DeviceProcessEvents
| where Timestamp > ago(7d)
| where InitiatingProcessFileName in~ ("dllhost.exe", "rundll32.exe")
```

```
| where InitiatingProcessCommandLine contains "CMSTPLUA" or
InitiatingProcessCommandLine contains "{3E5FC7F9-9A51-4367-9063-
A120244FBEC7}"
| project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
         InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessIntegrityLevel, ProcessIntegrityLevel
| where (InitiatingProcessIntegrityLevel == "Low" or
InitiatingProcessIntegrityLevel == "Medium") and ProcessIntegrityLevel ==
"High"
| summarize count() by DeviceName, AccountName, FileName,
ProcessCommandLine, InitiatingProcessFileName,
InitiatingProcessCommandLine

// Technique 43: WUAUCLT COM interface abuse
union
DeviceProcessEvents
| where Timestamp > ago(7d)
| where FileName =~ "wuauclt.exe"
| where ProcessCommandLine contains "/RunHandlerComServer" or
ProcessCommandLine contains "/UpdateDeploymentProvider"
| project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
         InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessIntegrityLevel, ProcessIntegrityLevel
| where (InitiatingProcessIntegrityLevel == "Low" or
InitiatingProcessIntegrityLevel == "Medium")
| join kind=inner (
    DeviceProcessEvents
    | where ProcessIntegrityLevel == "High"
    | where InitiatingProcessFileName =~ "wuauclt.exe"
) on DeviceId, DeviceName
| summarize count() by DeviceName, AccountName, FileName,
ProcessCommandLine, InitiatingProcessFileName,
InitiatingProcessCommandLine

// Technique 65: DiskCleanup COM interface abuse
union
(
    // Technique 41: CMSTPLUA COM interface abuse
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where InitiatingProcessFileName in~ ("dllhost.exe", "rundll32.exe")
    | where InitiatingProcessCommandLine contains "CMSTPLUA" or
InitiatingProcessCommandLine contains "{3E5FC7F9-9A51-4367-9063-
A120244FBEC7}"
    | where (InitiatingProcessIntegrityLevel == "Low" or
InitiatingProcessIntegrityLevel == "Medium") and ProcessIntegrityLevel ==
"High"
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
             InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessIntegrityLevel, ProcessIntegrityLevel
),
```

```
(
    // Technique 43: WUAUCLT COM interface abuse
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName =~ "wuauclt.exe"
    | where ProcessCommandLine contains "/RunHandlerComServer" or
ProcessCommandLine contains "/UpdateDeploymentProvider"
    | where (InitiatingProcessIntegrityLevel == "Low" or
InitiatingProcessIntegrityLevel == "Medium")
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
            InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessIntegrityLevel, ProcessIntegrityLevel, DeviceId
    | join kind=inner (
        DeviceProcessEvents
        | where ProcessIntegrityLevel == "High"
        | where InitiatingProcessFileName =~ "wuauclt.exe"
        | project DeviceId, DeviceName
    ) on DeviceId, DeviceName
),
(
    // Technique 65: DiskCleanup COM interface abuse
    DeviceProcessEvents
    | where Timestamp > ago(7d)
    | where FileName =~ "cleanmgr.exe" or FileName =~ "dllhost.exe"
    | where ProcessCommandLine contains "DiskCleanup"
        or ProcessCommandLine contains "{77137B0F-F36B-466C-B00E-
B6A8AFEFC392}"
        or ProcessCommandLine contains "{B41DB860-8EE4-11D2-9906-
E49FADC173CA}"
    | where (InitiatingProcessIntegrityLevel == "Low" or
InitiatingProcessIntegrityLevel == "Medium") and ProcessIntegrityLevel ==
"High"
    | project TimeGenerated, DeviceName, AccountName, FileName,
ProcessCommandLine,
            InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessIntegrityLevel, ProcessIntegrityLevel
)
| summarize count() by DeviceName, AccountName, FileName,
ProcessCommandLine,
            InitiatingProcessFileName, InitiatingProcessCommandLine
| order by count_ desc
```

Investigation Steps if Results Found:

1. **Process Timeline**: Establish a timeline of events leading up to and following the potential UAC bypass, paying special attention to integrity level changes from Medium/Low to High.
2. **Registry Modifications**: Review all registry modifications made by the process or its parent processes.
3. **Privilege Escalation**: Check if the process or child processes subsequently ran with elevated privileges.
4. **Command Execution**: Analyze what commands were executed after the potential bypass.

5. **Parent Process Analysis**: Determine what process initiated the UAC bypass attempt and whether it's legitimate. For COM interface abuse, check if dllhost.exe or other COM-related processes are being used in an unusual context.
6. **Lateral Movement**: Check if the compromised account was used to access other systems after privilege escalation.
7. **Persistence Mechanisms**: Look for any persistence mechanisms established with elevated privileges.
8. **Memory Analysis**: If possible, capture and analyze memory from the affected system to identify injected code or other artifacts.
9. **COM Interface Analysis**: For COM interface abuse techniques (41, 43, 65), examine COM registration and interface usage. Look for unusual COM object instantiation or interface method calls.

4.8. Hunt for Known APT38/Lazarus Group IoCs

MITRE ATT&CK Techniques:

- T1588.001: Obtain Capabilities: Malware
- T1071: Application Layer Protocol
- T1102: Web Service
- T1583.001: Acquire Infrastructure: Domains
- T1584: Compromise Infrastructure

Hypothesis: APT38/Lazarus Group has been observed using specific infrastructure, malware, and GitHub repositories in their campaigns. Direct matches to known IoCs can provide high-confidence indicators of compromise.

Hunting Focus: Search for known IoCs across network connections, file hashes, and GitHub repository access. This approach complements the behavioral detection methods in previous sections with a more direct IoC-based detection strategy.

KQL Query for Domain/IP IoCs:

```
// Search for known malicious domains and IPs in network connections
let KnownDomains = dynamic([
    <redacted>
]);
let KnownIPs = dynamic([
    <redacted>
]);
DeviceNetworkEvents
| where Timestamp > ago(30d) // Longer timeframe for IoC hunting
| where RemoteUrl in~ (KnownDomains) or RemoteIP in (KnownIPs) or
       RemoteUrl has_any (KnownDomains) or RemoteIP has_any (KnownIPs)
| project TimeGenerated, DeviceName, LocalIP, RemoteIP, RemoteUrl,
RemotePort,
         InitiatingProcessFileName, InitiatingProcessCommandLine,
InitiatingProcessAccountName
| summarize FirstSeen=min(TimeGenerated), LastSeen=max(TimeGenerated),
          ConnectionCount=count() by DeviceName, RemoteIP, RemoteUrl,
```

```
InitiatingProcessFileName
| sort by LastSeen desc
```

KQL Query for File Hash IoCs:

```
// Search for known malicious file hashes
let KnownHashes = dynamic([
   <redacted>
]);
let KnownFileNames = dynamic([
    <redacted>
]);
union DeviceFileEvents, DeviceProcessEvents
| where Timestamp > ago(7d)
| where SHA1 in~ (KnownHashes) or MD5 in~ (KnownHashes) or
       SHA256 in~ (KnownHashes) or FileName in~ (KnownFileNames)
| project TimeGenerated, DeviceName, ActionType, FileName, FolderPath,
         SHA1, MD5, SHA256, InitiatingProcessFileName,
InitiatingProcessCommandLine
| summarize FirstSeen=min(TimeGenerated), LastSeen=max(TimeGenerated),
          Count=count() by DeviceName, FileName, FolderPath, SHA1, MD5,
SHA256
| sort by LastSeen desc
```

KQL Query for GitHub Repository IoCs:

```
// Search for access to known malicious GitHub repositories
let KnownRepos = dynamic([
 <redacted>
]);
DeviceProcessEvents
| where Timestamp > ago(7d)
| where ProcessCommandLine has_any (KnownRepos) or
InitiatingProcessCommandLine has_any (KnownRepos)
| project TimeGenerated, DeviceName, AccountName, ProcessCommandLine,
         InitiatingProcessCommandLine, FileName, FolderPath
| summarize FirstSeen=min(TimeGenerated), LastSeen=max(TimeGenerated),
          Count=count() by DeviceName, AccountName, ProcessCommandLine
| sort by LastSeen desc
```

Investigation Steps if Results Found:

1. **Immediate Containment**: Any direct IoC matches should be treated as high-confidence indicators of compromise. Consider immediate containment of affected systems.
2. **Correlation Analysis**: Check if the system exhibits other behaviors described in previous hunting sections (e.g., suspicious Java process activity, UAC bypass attempts).
3. **Lateral Movement Check**: Determine if the compromised system has been used as a jumping point to access other systems in the environment.

4. **Timeline Analysis**: Establish a complete timeline of events before and after the IoC detection to understand the full scope of the compromise.
5. **Malware Analysis**: If file hash IoCs are found, collect samples for in-depth malware analysis to understand capabilities and potential impact.
6. **Network Traffic Analysis**: For domain/IP IoCs, analyze the full network traffic to and from these destinations to identify data exfiltration or additional command and control activity.
7. **GitHub Repository Analysis**: If suspicious GitHub repositories were accessed, examine what code was downloaded and if it was executed in the environment.
8. **User Interview**: Carefully interview the affected user to understand the context of the activity, particularly if it relates to recruitment offers or skill assessments.

5. Summary of Threat Hunting Approaches

The following table provides an overview of the threat hunting hypotheses outlined in this runbook, along with their associated MITRE ATT&CK techniques and tactics. This summary can be used as a quick reference guide when planning and executing threat hunting activities.

Hunt ID	Hunt Name	Description	MITRE ATT&CK Techniques	MITRE ATT&CK Tactics
4.1	Suspicious GitHub Activity	Identify `git clone` commands targeting non-corporate repositories	T1566: Phishing T1204.003: User Execution: Malicious Image T1195.001: Supply Chain Compromise	Initial Access Execution Defense Evasion
4.2	Suspicious Mail Delivery Logs	Detect recruitment lures via email directing to malicious repositories	T1566.001: Spearphishing Attachment T1566.002: Spearphishing Link T1534: Internal Spearphishing	Initial Access Execution
4.3	Abnormal DevOps Environment Usage	Detect "compiling on target" and unusual compiler usage	T1027.004: Obfuscated Files or Information: Compile After Delivery	Defense Evasion
4.4	Binaries in unusual locations and exec via rundll32	Identify common-looking executables in non-standard locations and possible attempts to run them via rundll	T1218.001: Signed Binary Proxy Execution: Rundll32 T1218.010: System Binary Proxy Execution: Regsvr32	Defense Evasion Execution

Hunt ID	Hunt Name	Description	MITRE ATT&CK Techniques	MITRE ATT&CK Tactics
4.5	Java Process Dropping Files	Detect Java processes writing files to suspicious locations	T1036.005: Masquerading: Match Legitimate Name or Location T1027.002: Obfuscated Files or Information: Software Packing T1105: Ingress Tool Transfer	Defense Evasion Command and Control
4.6	Malicious App Execution	Identify execution of suspicious scripts and applications	T1059.006: Command and Scripting Interpreter: Python T1059.007: Command and Scripting Interpreter: JavaScript T1204.002: User Execution: Malicious File	Execution Defense Evasion
4.7	UAC Bypass Attempts	Detect attempts to bypass User Account Control	T1548.002: Abuse Elevation Control Mechanism: Bypass UAC T1134: Access Token Manipulation T1574: Hijack Execution Flow T1546: Event Triggered Execution	Privilege Escalation Defense Evasion Persistence
4.8	Known APT38/Lazarus Group IoCs	Search for known malicious domains, IPs, file hashes, and repositories	T1588.001: Obtain Capabilities: Malware T1071: Application Layer Protocol T1102: Web Service T1583.001: Acquire Infrastructure: Domains T1584: Compromise Infrastructure	Resource Development Command and Control

Mapping to MITRE ATT&CK Tactics

The threat hunting hypotheses in this runbook cover the following MITRE ATT&CK tactics:

1. **Initial Access**: Detecting phishing attempts and malicious repositories used to gain initial foothold
2. **Execution**: Identifying execution of malicious code, scripts, and applications
3. **Persistence**: Detecting mechanisms used to maintain access to systems
4. **Privilege Escalation**: Identifying attempts to gain higher-level permissions
5. **Defense Evasion**: Detecting techniques used to avoid detection
6. **Command and Control**: Identifying communication with command and control infrastructure
7. **Resource Development**: Detecting acquisition of resources used for the attack

This comprehensive coverage allows for detection across multiple stages of the attack lifecycle, increasing the likelihood of identifying APT38/Lazarus Group activities before they achieve their objectives.

This document is prepared by Crimson7 BV 2025