ETELAAT TEAM Threat Hunting Runbook

> **Specialized Focus**: This runbook targets the Iranian state-nexus ETELAAT TEAM cybercrime organization conducting large-scale JavaScript-based data exfiltration via Telegram infrastructure.

1) Purpose

This threat hunting runbook provides Microsoft Sentinel KQL queries and investigative procedures to detect and hunt for ETELAAT TEAM (also known as kyetelaat) threat actor activities. The runbook focuses on their signature tactics including JavaScript-based data exfiltration, Telegram Bot API abuse, fake download social engineering, and real-time geolocation harvesting for large-scale credential theft and intelligence collection operations.

2) Threat Context

Actor Information

- **Primary Name**: ETELAAT TEAM
- **Aliases**: kyetelaat, Etelaat Group, ETELAAT Intelligence Team
- **Composition**: Iranian Ministry of Intelligence (MOIS) affiliated cybercrime organization
- **Geographic Focus**: Europe (North Macedonia, UK, Ireland), Global web infrastructure
- **First Observed**: 2024 (estimated based on campaign infrastructure)
- **Active Since**: Active as of July 2025 with 18,000+ documented victim compromises

Motivation

- **Intelligence Collection** - Strategic data gathering on European websites and users for Iranian state interests
- **Financial Gain** - Large-scale credential harvesting for resale and financial fraud operations
- **Geopolitical Intelligence** - Mapping global web infrastructure and user behavior patterns for state analysis

Key TTPs

- **T1566.002**: Phishing: Spearphishing Link - JavaScript injection via compromised websites
- **T1071.001**: Application Layer Protocol: Web Protocols - HTTPS exfiltration via Telegram Bot API
- **T1087.004**: Account Discovery: Cloud Account - Harvesting user credentials through fake download forms
- **T1033**: System Owner/User Discovery - IP geolocation and browser fingerprinting via ipapi.co
- **T1124**: System Time Discovery - Timezone analysis through JavaScript execution context
- **T1016.001**: System Network Configuration Discovery: Internet Connection Discovery - Network reconnaissance via client-side JavaScript
- **T1583.001**: Acquire Infrastructure: Domains - Compromising legitimate websites for malware hosting
- **T1102.002**: Web Service: Bidirectional Communication - **2025 Evolution: Telegram Bot API as primary C2 channel**
- **T1059.007**: Command and Scripting Interpreter: JavaScript/Node.js - Multi-stage malware deployment
- **T1027**: Obfuscated Files or Information - 110 encrypted payloads with advanced obfuscation
- **T1562.001**: Disable or Modify Tools - Windows Defender exclusion attempts via PowerShell
- **T1497**: Virtualization/Sandbox Evasion - VM detection and debugger evasion techniques

3) Technical Prerequisites for Threat Hunting

Required Data Sources

- Microsoft Defender for Endpoint (MDE)
- Microsoft Sentinel (Azure Sentinel)
- DNS Query Logs (Windows DNS Client, Pi-hole, or similar)
- Web Proxy/Firewall Logs (Zscaler, Palo Alto, etc.)
- Windows Event Logs (Security, Application, System)
- PowerShell Execution Logs (Event ID 4103, 4104)
- Process Creation Events (Event ID 4688, Sysmon Event ID 1)
- File Creation Events (Sysmon Event ID 11)
- Network Connection Events (Sysmon Event ID 3)
- Browser Activity Logs (if available via GPO/Extensions)
- Network Traffic Analysis (NSM tools, Zeek/Suricata)
- Office 365/Azure AD Sign-in Logs

Recommended Log Retention

- Minimum 90 days for correlation analysis
- 180 days recommended for campaign tracking (ETELAAT TEAM operations show long persistence)

4) Threat Hunting Hypotheses

Hypothesis 1: Telegram Bot C2 Infrastructure Detection

Mapping: T1071.001 - Application Layer Protocol: Web Protocols + T1102.002 - Web Service: Bidirectional Communication
Hypothesis Explanation: ETELAAT TEAM operates multiple Telegram bots for different malware families - bot 7431860324 for JavaScript web exfiltration and bot 7972762095 for Node.js stealer/RAT operations. They use hardcoded bot tokens to exfiltrate victim data and coordinate multi-vector attacks, abusing

legitimate Telegram infrastructure to avoid traditional C2 detection.

Hunting Focus: Detect both single-bot exfiltration patterns and multi-bot correlation across ETELAAT TEAM's infrastructure, identifying comprehensive compromise scenarios.

```
// FIXED: Telegram-GeoAPI Correlation Query - Device Level Aggregation
// Issue: Thresholds were too restrictive - only 4 total connections and
score of 50

// Step 1: Telegram connections - Behavioral fingerprint
let TelegramConnections =
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl has "api.telegram.org"
    //| where RemotePort == 443
    | where InitiatingProcessFileName in~ ("chrome.exe", "firefox.exe",
"msedge.exe", "iexplore.exe","curl.exe")
    | extend HourlyBucket = bin(TimeGenerated, 1h)
    | extend DeviceId = tostring(DeviceId), DeviceName =
tostring(DeviceName)
    | summarize
        HourlyConnections = count(),
        UniqueRemoteIPs = dcount(RemoteIP),
        ConnectionPattern = make_set(bin(TimeGenerated, 10m), 50)
        by DeviceName, DeviceId, HourlyBucket, InitiatingProcessFileName
    | extend AutomationScore = case(
        HourlyConnections > 50 and UniqueRemoteIPs <= 3, 100,
        HourlyConnections > 20 and UniqueRemoteIPs <= 2, 90,
        HourlyConnections > 10 and UniqueRemoteIPs == 1, 80,
        HourlyConnections > 5, 70,
        50);

// Step 2: GeoAPI access - Device-level aggregation
let GeoAPIAbuse =
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl has_any ("ipapi.co", "ip-api.com")
    | extend DeviceId = tostring(DeviceId), DeviceName =
tostring(DeviceName)
    | summarize
        GeoAPIRequests = count(),
        GeoAPITimeSpan = datetime_diff('day', max(TimeGenerated),
min(TimeGenerated)) + 1,
        FirstGeoAPIAccess = min(TimeGenerated),
        LastGeoAPIAccess = max(TimeGenerated)
        by DeviceName, DeviceId;

// Step 3: Device-level correlation with all connections regardless of the
classificaiton

TelegramConnections
| summarize
    TotalTelegramConnections = sum(HourlyConnections),
```

```
    TelegramActiveHours = count(),
    FirstTelegramActivity = min(HourlyBucket),
    LastTelegramActivity = max(HourlyBucket),
    MaxAutomationScore = max(AutomationScore),
    ProcessesUsed = make_set(InitiatingProcessFileName)
    by DeviceName, DeviceId
| join kind=leftouter (
    GeoAPIAbuse
) on DeviceName, DeviceId
| extend EtelaatPattern = case(
    isnotempty(GeoAPIRequests) and MaxAutomationScore >= 70,
"High_Confidence_ETELAAT",
    isnotempty(GeoAPIRequests) and MaxAutomationScore >= 50,
"Medium_Confidence_ETELAAT",
    isnotempty(GeoAPIRequests), "Low_Confidence_ETELAAT",  // NEW: Show
any GeoAPI correlation
    MaxAutomationScore >= 80, "High_Confidence_Bot_Activity",
    MaxAutomationScore >= 60, "Suspicious_Automation",
    "No_Pattern")
| extend RiskScore = case(
    EtelaatPattern == "High_Confidence_ETELAAT", 100,
    EtelaatPattern == "Medium_Confidence_ETELAAT", 85,
    EtelaatPattern == "Low_Confidence_ETELAAT", 70,  // NEW
    EtelaatPattern == "High_Confidence_Bot_Activity", 95,
    EtelaatPattern == "Suspicious_Automation", 75,
    50)
| project
    DeviceName,
    DeviceId,
    EtelaatPattern,
    RiskScore,
    TotalTelegramConnections,
    TelegramActiveHours,
    GeoAPIRequests = coalesce(GeoAPIRequests, 0),
    MaxAutomationScore,
    ProcessesUsed
| order by RiskScore desc;
```

Investigation Steps:

1. Analyze connection frequency patterns to distinguish automated bot activity from human browsing
2. Correlate Telegram API connections with geolocation service usage (ipapi.co correlation indicates ETELAAT pattern)
3. Review browser process command lines and memory for JavaScript malware artifacts
4. Check for timing patterns consistent with malware execution (regular intervals, bulk connections)
5. Examine file system for ETELAAT artifacts (Node.js malware in MA[0-9]{10} directories, system_log.txt)
6. Cross-reference high-activity periods with website compromise indicators
7. For comprehensive detection, integrate with proxy logs that capture full URLs and POST data **Note**: This behavioral approach compensates for MDE's domain-only RemoteUrl limitation. For bot token identification, integrate with network proxy or firewall logs that capture full HTTP requests.

Hypothesis 2: Geolocation API Abuse for Victim Profiling

Mapping: T1033 - System Owner/User Discovery
Hypothesis Explanation: ETELAAT TEAM's JavaScript malware makes API calls to ipapi.co/json/ to harvest victim IP addresses and geographic locations. This data is immediately exfiltrated to their Telegram channels for victim profiling and geographic analysis. This query detects suspicious automation where browsers access geolocation services (like ipapi.co) shortly before or after contacting the Telegram API.
Hunting Focus: Identify suspicious automated requests to geolocation services followed by immediate data transmission.

```
let GeoLookups =
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl has_any ("ipapi.co", "ip-api.com")
    | where InitiatingProcessFileName in~ ("chrome.exe", "firefox.exe",
"msedge.exe", "iexplore.exe")
    | project DeviceName, DeviceId, GeoTime = TimeGenerated, GeoUrl =
RemoteUrl, InitiatingProcessFileName, InitiatingProcessCommandLine;

let TelegramCalls =
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl has "api.telegram.org"
    | where RemotePort == 443
    | where InitiatingProcessFileName in~ ("chrome.exe", "firefox.exe",
"msedge.exe", "iexplore.exe")
    | project DeviceName, DeviceId, TelegramTime = TimeGenerated,
TelegramUrl = RemoteUrl;

GeoLookups
| join kind=inner (
    TelegramCalls
) on DeviceName, DeviceId
| where abs(datetime_diff("second", TelegramTime, GeoTime)) <= 60
| extend TimeDelta = datetime_diff("second", TelegramTime, GeoTime)
| project
    DeviceName,
    DeviceId,
    GeoTime,
    TelegramTime,
    TimeDelta,
    GeoUrl,
    TelegramUrl,
    InitiatingProcessFileName,
    InitiatingProcessCommandLine
| summarize
    CorrelatedPairs = count(),
    AvgTimeDelta = avg(TimeDelta),
    FirstSeen = min(GeoTime),
    LastSeen = max(TelegramTime),
    Browsers = make_set(InitiatingProcessFileName),
    GeoServices = make_set(GeoUrl)
```

```
    by DeviceName, DeviceId
| where CorrelatedPairs > 0
| extend ThreatLevel = case(
    CorrelatedPairs > 50 and abs(AvgTimeDelta) < 10, "Critical",
    CorrelatedPairs > 20 and abs(AvgTimeDelta) < 20, "High",
    CorrelatedPairs > 10 and abs(AvgTimeDelta) < 30, "Medium",
    "Low")
| order by CorrelatedPairs desc
```

Investigation Steps:

1. Analyze timing patterns between geolocation API calls and Telegram communications (ETELAAT pattern shows <60 second correlation)
2. Validate that requests originate from browser processes indicating web-based malware execution
3. Review automation indicators - high frequency with consistent timing suggests malware activity
4. Examine browser process command lines for JavaScript execution artifacts or suspicious arguments
5. Cross-reference time patterns with known website visits to identify potential compromise sources
6. Investigate devices with high correlation pairs for additional ETELAAT artifacts (MA directories, system_log.txt) **Note**: Domain-only RemoteUrl means we detect behavior patterns rather than specific bot tokens. Timing correlation is the key ETELAAT signature.

Hypothesis 3: Fake Download Social Engineering Detection

Mapping: T1566.002 - Phishing: Spearphishing Link
Hypothesis Explanation: ETELAAT TEAM's malware presents fake download progress bars to users while silently exfiltrating their data. This social engineering tactic masks the malicious activity and may trigger additional malware downloads. This query detects suspicious script execution patterns from browsers or updater tools. It filters DeviceProcessEvents for JavaScript-like keywords in the ProcessCommandLine. It includes processes launched by browsers or executables like node.exe, install.exe, or update.exe. **Hunting Focus**: Detect JavaScript execution patterns that create fake UI elements while performing network communications or executing stuff.

```
DeviceProcessEvents
| where TimeGenerated >= ago(30d)
// Look for suspicious JavaScript-related patterns in the command line
| where ProcessCommandLine contains "javascript:"
    or ProcessCommandLine contains "eval("
    or ProcessCommandLine contains "document.createElement"
    or ProcessCommandLine contains "progress"
// Focus on initiators: browser OR scripting tools
| where InitiatingProcessFileName has_any ("node", "install", "update")
    or InitiatingProcessFileName in~ ("chrome.exe", "firefox.exe",
"msedge.exe", "iexplore.exe", "curl.exe")
| project
    TimeGenerated,
    DeviceName,
    ParentProcess = InitiatingProcessFileName,
    CommandLine = ProcessCommandLine,
    ProcessId,
```

```
    FileName
| summarize
    SuspiciousActivity = count(),
    FirstActivity = min(TimeGenerated),
    LastActivity = max(TimeGenerated),
    JSPatterns = make_set(CommandLine, 5),
    InvolvedProcesses = make_set(FileName)
    by DeviceName, ParentProcess
```

Investigation Steps:

1. Examine JavaScript command line execution for progress bar creation patterns
2. Correlate with network connections to Telegram and geolocation APIs
3. Check browser process memory for injected JavaScript content
4. Review recently visited websites in browser history
5. Analyze downloaded files for additional malware payloads

Hypothesis 5: Large-Scale Campaign Monitoring and Victim Enumeration

Mapping: TO DO, FIX THIS **Hypothesis Explanation**: ETELAAT TEAM operates both real-time victim tracking and large-scale credential harvesting systems across their 18,000+ victim operation. Each successful compromise generates immediate Telegram notifications, enabling monitoring of campaign intensity, victim enumeration, and systematic credential collection across multiple compromised websites. The repository provided contains a dump of compromise notification and websites where the redirect happened.

```
https://github.com/vitorallo/etelaat_telegram
```

Investigation Steps:

1. grep for your external facing IP address or your website. You aren't likely on this exploited websites list if your aren't running WordPress. **Note**: exploited website served as redirector/to serve the phish fake chrome update

Hypothesis 6: JavaScript Malware Injection and Execution Patterns

Mapping: T1059.007 - Command and Scripting Interpreter: JavaScript
Hypothesis Explanation: ETELAAT TEAM injects malicious JavaScript into compromised websites that executes the downloadWithProgress() function containing their data exfiltration logic. Detection focuses on identifying this specific function execution pattern.
Hunting Focus: Detect JavaScript function execution patterns consistent with ETELAAT TEAM's malware signature.

```
// Hunt for ETELAAT JavaScript malware execution patterns
DeviceEvents
| where TimeGenerated >= ago(30d)
| where ActionType == "BrowserLaunchedToOpenUrl" or ActionType ==
```

```
"NetworkConnectionEvents"
| extend ProcessKey = strcat(DeviceName, "_", ProcessId)
| join kind=inner (
    DeviceProcessEvents
    | where TimeGenerated >= ago(30d)
    | where ProcessCommandLine contains "downloadWithProgress"
        or ProcessCommandLine contains "ipapi.co/json"
        or ProcessCommandLine contains "sendMessage"
    | extend ProcessKey = strcat(DeviceName, "_", ProcessId)
) on ProcessKey
| join kind=inner (
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl contains "api.telegram.org" and RemoteUrl contains
"sendMessage"
    | extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
) on ProcessKey
| project
    TimeGenerated,
    DeviceName,
    InitiatingProcessFileName,
    JSExecution = ProcessCommandLine,
    TelegramExfiltration = RemoteUrl,
    RemoteIP
| summarize
    MalwareExecutions = count(),
    FirstSeen = min(TimeGenerated),
    LastSeen = max(TimeGenerated)
    by DeviceName, InitiatingProcessFileName
| where MalwareExecutions > 1
```

Investigation Steps:

1. Extract and analyze the complete JavaScript code from browser memory/cache
2. Compare function signatures with known ETELAAT TEAM malware samples
3. Identify the injection vector (how JavaScript was delivered to the browser)
4. Check for additional malware payloads or persistence mechanisms
5. Review all websites visited by the affected browser session

Hypothesis 7: Node.js-based Stealer/RAT Detection with Updater Execution

Mapping: T1059.007 - Command and Scripting Interpreter: JavaScript/Node.js
Hypothesis Explanation: ETELAAT TEAM deploys sophisticated Node.js-based stealer/RAT malware disguised as legitimate updater processes. The malware creates persistence through copied Node.js executables and attempts to disable Windows Defender while exfiltrating data to Telegram.
Hunting Focus: Detect suspicious Node.js process execution, updater.exe variants, and associated PowerShell Windows Defender exclusion attempts.

```
// Hunt for ETELAAT Node.js stealer and updater.exe execution patterns
// the original filname is updater-890.6.2.8.exe
```

```
DeviceProcessEvents
| where TimeGenerated >= ago(30d)
| where (ProcessCommandLine contains "updater-" and ProcessCommandLine
contains ".exe")
    or (FolderPath contains "updater-" and FileName endswith ".exe")
    or (ProcessCommandLine contains "node.exe" and FolderPath contains
"MA")
    or (ProcessCommandLine contains "js_malware" or ProcessCommandLine
contains "decoded.js")
| extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
| join kind=leftouter (
    DeviceProcessEvents
    | where TimeGenerated >= ago(30d)
    | where ProcessCommandLine contains "Add-MpPreference" and
ProcessCommandLine contains "ExclusionPath"
    | extend ProcessKey = strcat(DeviceName, "_", ProcessId)
) on ProcessKey
| join kind=leftouter (
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl contains "api.telegram.org" and RemotePort == 443
    | extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
) on ProcessKey
| where isnotempty(ProcessCommandLine) or isnotempty(ProcessCommandLine1)
or isnotempty(RemoteUrl)
| project
    TimeGenerated,
    DeviceName,
    SuspiciousProcess = ProcessCommandLine,
    DefenderExclusion = ProcessCommandLine1,
    TelegramC2 = RemoteUrl,
    ProcessPath = FolderPath,
    ParentProcess = InitiatingProcessFileName
| summarize
    SuspiciousActivity = count(),
    FirstSeen = min(TimeGenerated),
    LastSeen = max(TimeGenerated),
    ProcessPaths = make_set(ProcessPath, 5),
    ParentProcesses = make_set(ParentProcess, 5)
    by DeviceName
| extend ThreatLevel = case(
    SuspiciousActivity > 5, "Critical",
    SuspiciousActivity > 3, "High",
    SuspiciousActivity > 1, "Medium",
    "Low")
| where SuspiciousActivity > 0
```

Investigation Steps:

1. Validate the process path matches ETELAAT patterns (MA[0-9]{10} folders in %PROGRAMDATA% or %TEMP%)
2. Check for associated PowerShell Add-MpPreference exclusion attempts

3. Analyze file hashes against known ETELAAT IOCs (SHA256: 25596c1067b2ccee381d694b0ce48491c2a3b14d66aed8d1dfe8be38f0cf9b77)
4. Review network connections to Telegram bot API with token 7972762095
5. Examine file system for persistence artifacts (update.log, config.ini, version.txt)

Hypothesis 8: Windows Defender Evasion via PowerShell Exclusions

Mapping: T1562.001 - Disable or Modify Tools: Security Software
Hypothesis Explanation: ETELAAT TEAM's Node.js malware attempts to evade detection by adding Windows Defender exclusions for their persistence directories using PowerShell Add-MpPreference commands. This creates security blind spots for their malware operations.
Hunting Focus: Detect PowerShell execution attempting to create Windows Defender exclusions for suspicious directories.

```
// Hunt for Windows Defender exclusion attempts by ETELAAT malware
DeviceProcessEvents
| where TimeGenerated >= ago(30d)
| where ProcessCommandLine contains "Add-MpPreference" and
ProcessCommandLine contains "ExclusionPath"
| extend ExclusionPaths = extract_all(@"ExclusionPath\s+@?\(?'?
([^'\""\),]+)", ProcessCommandLine)
| extend SuspiciousPath = case(
    ProcessCommandLine contains "MA" and ProcessCommandLine contains
"%PROGRAMDATA%", "ETELAAT_Pattern",
    ProcessCommandLine contains "MA" and ProcessCommandLine contains
"%TEMP%", "ETELAAT_Pattern",
    ProcessCommandLine contains "updater", "Updater_Pattern",
    "Other")
| where SuspiciousPath != "Other"
| extend ProcessKey = strcat(DeviceName, "_", ProcessId)
| join kind=leftouter (
    DeviceFileEvents
    | where TimeGenerated >= ago(30d)
    | where FolderPath contains "MA" and (FolderPath contains
"PROGRAMDATA" or FolderPath contains "TEMP")
    | where FileName in~ ("update.log", "config.ini", "version.txt",
"system_info.log")
    | extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
) on ProcessKey
| project
    TimeGenerated,
    DeviceName,
    PowerShellCommand = ProcessCommandLine,
    ExclusionAttempt = ExclusionPaths,
    SuspiciousPattern = SuspiciousPath,
    AssociatedFile = FileName,
    FilePath = FolderPath
| summarize
    ExclusionAttempts = count(),
    FirstAttempt = min(TimeGenerated),
    LastAttempt = max(TimeGenerated),
```

```
    TargetPaths = make_set(ExclusionAttempt, 10),
    AssociatedFiles = make_set(AssociatedFile, 10)
    by DeviceName, SuspiciousPattern
| extend RiskScore = case(
    ExclusionAttempts > 3, 100,
    ExclusionAttempts > 1, 90,
    80)
```

Investigation Steps:

1. Verify the exclusion paths match ETELAAT persistence directory patterns
2. Check if the PowerShell execution was successful in creating exclusions
3. Examine the excluded directories for ETELAAT malware artifacts
4. Review Windows Defender logs for exclusion creation events
5. Validate the parent process that initiated the PowerShell exclusion attempt

Hypothesis 9: System Log File Creation in Temp Directory (system_log.txt)

Mapping: T1005 - Data from Local System + T1074.001 - Local Data Staging
Hypothesis Explanation: ETELAAT TEAM's Node.js malware creates a specific log file called "system_log.txt" in the OS temp directory using path.join(os.tmpdir(), 'system_log.txt') and logs malware activities using fs.appendFileSync(). This artifact serves as forensic evidence of malware execution and data staging.
Hunting Focus: Detect creation and modification of system_log.txt files in temp directories, especially when associated with suspicious process execution.

```
// Hunt for ETELAAT TEAM's system_log.txt file creation and modification
patterns
DeviceFileEvents
| where TimeGenerated >= ago(30d)
| where FileName =~ "system_log.txt"
| where (FolderPath contains "temp" or FolderPath contains "tmp" or
FolderPath contains "TEMP")
| where ActionType in ("FileCreated", "FileModified")
| extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
| join kind=leftouter (
    DeviceProcessEvents
    | where TimeGenerated >= ago(30d)
    | where ProcessCommandLine contains "node.exe" or ProcessCommandLine
contains ".js"
    | extend ProcessKey = strcat(DeviceName, "_", ProcessId)
) on ProcessKey
| join kind=leftouter (
    DeviceProcessEvents
    | where TimeGenerated >= ago(30d)
    | where FolderPath contains "MA" and (FolderPath contains
"PROGRAMDATA" or FolderPath contains "TEMP")
    | extend ProcessKey = strcat(DeviceName, "_", ProcessId)
) on ProcessKey
| join kind=leftouter (
```

```
    DeviceNetworkEvents
    | where TimeGenerated >= ago(30d)
    | where RemoteUrl contains "api.telegram.org" and RemotePort == 443
    | extend ProcessKey = strcat(DeviceName, "_", InitiatingProcessId)
) on ProcessKey
| project
    TimeGenerated,
    DeviceName,
    LogFilePath = FolderPath,
    FileAction = ActionType,
    FileSize,
    SHA256,
    NodeJSProcess = ProcessCommandLine,
    MalwareLocation = FolderPath1,
    TelegramComms = RemoteUrl,
    InitiatingProcessName = InitiatingProcessFileName
| summarize
    LogEvents = count(),
    FirstLogActivity = min(TimeGenerated),
    LastLogActivity = max(TimeGenerated),
    FileActions = make_set(FileAction),
    LogPaths = make_set(LogFilePath),
    FileSizes = make_set(FileSize),
    UniqueHashes = make_set(SHA256)
    by DeviceName, InitiatingProcessName
| extend RiskScore = case(
    LogEvents > 10, 100,
    LogEvents > 5, 90,
    LogEvents > 2, 80,
    70)
| where LogEvents > 0
| order by RiskScore desc
```

Bonus hunt: In our case, the seconday payload carried in a Delphi compiled file that executed with this file name:

```
DeviceFileEvents
| where TimeGenerated >= ago(30d)
| where FileName =~ "CHashra.exe"
```

it might differ in your case. In the same folder you can find those files, I would focus more on the DLL which are standard VS C++ runtime.

```
-rw-r--r--   1 vito  staff    39496 Jul 16 21:37 CHashra.exe
-rw-r--r--   1 vito  staff    18407 Jul 16 21:37 Kaertklaelit.kmzg
-rw-r--r--   1 vito  staff  3338126 Jul 16 21:37 Liertneeng.cys
-rw-r--r--   1 vito  staff  5619784 Jul 16 21:37 mfc110u.dll
-rw-r--r--   1 vito  staff   661456 Jul 16 21:37 MSVCP110.dll
-rw-r--r--   1 vito  staff   849360 Jul 16 21:37 MSVCR110.dll
```

Investigation Steps:

1. Examine the contents of system_log.txt for malware execution timestamps and activities
2. Correlate log file creation with Node.js process execution from ETELAAT folders
3. Check file size growth patterns indicating ongoing malware logging activity
4. Verify SHA256 hash against known ETELAAT TEAM malware artifacts
5. Review Telegram communications occurring around log file creation times
6. Analyze parent processes that initiated the log file creation

5) Summary of Runbook

Hunt Hypothesis	MITRE TTP	KQL Query Focus	Detection Priority
Telegram Bot C2 Infrastructure Detection	T1071.001+T1102.002	Behavioral pattern analysis with GeoAPI correlation	Critical
Geolocation API Abuse for Victim Profiling	T1033	Timing correlation between ipapi.co and Telegram API	High
Fake Download Social Engineering Detection	T1566.002	JavaScript execution patterns in browser processes	High
Large-Scale Campaign Monitoring and Victim Enumeration	T1016.001+T1087.004	GitHub repository analysis of compromised websites	Critical
JavaScript Malware Injection and Execution Patterns	T1059.007	downloadWithProgress() function detection	Critical
Node.js-based Stealer/RAT Detection with Updater Execution	T1059.007	updater-890.6.2.8.exe and Node.js process patterns	Critical
Windows Defender Evasion via PowerShell Exclusions	T1562.001	Add-MpPreference attempts with MA directory patterns	High
System Log File Creation in Temp Directory	T1005+T1074.001	system_log.txt creation and modification	Medium

Key Detection Metrics

- **Coverage**: 8 focused hunting hypotheses covering 10 critical TTPs including 2025 Telegram Bot API evolutions and Node.js malware family with high-confidence detection logic
- **False Positive Rate**: Low due to correlation-based queries and behavioral analysis targeting specific ETELAAT TEAM patterns and hardcoded command signatures
- **Response Time**: Automated alerting for critical findings involving updater-890.6.2.8.exe execution, specific PowerShell patterns, GeoAPI correlation, and multi-vector compromise scenarios

- **Investigation Depth**: Multi-stage verification procedures with IoC correlation, GitHub repository integration for compromised website validation, forensic artifact analysis, and comprehensive victim impact evaluation
- **2025 Enhancements**: Enhanced behavioral detection for Telegram Bot API abuse patterns, JavaScript-based data exfiltration at scale, Node.js stealer/RAT detection with Windows Defender evasion capabilities, and integration with external threat intelligence repositories
- **Optimization Benefits**: Streamlined hypothesis count with improved query performance and reduced false positive rates through enhanced behavioral correlation

6) Implementation Notes and Limitations

Critical MDE Data Limitations

RemoteUrl Field Constraint: Microsoft Defender for Endpoint's `DeviceNetworkEvents` table stores only the hostname/domain in the `RemoteUrl` field, not full URL paths or parameters. This means:

❌ **Non-Functional**: `RemoteUrl contains "api.telegram.org/bot7431860324:AAE..."`
✅ **Functional**: `RemoteUrl contains "api.telegram.org"`

Impact on Detection:

- **Bot Token Identification**: Cannot extract specific bot tokens (7431860324, 7972762095) from MDE data alone
- **Message Content Analysis**: Cannot parse victim data, chat IDs, or message content from URLs
- **Precise C2 Attribution**: Cannot definitively attribute traffic to specific ETELAAT TEAM bots

Behavioral Detection Approach

Our queries compensate for these limitations through **behavioral pattern analysis**:

1. **Connection Frequency Patterns**: High-volume automated connections vs. human browsing
2. **Timing Correlation**: Geolocation API requests followed by Telegram communications
3. **Process Context**: Browser processes with unusual network patterns
4. **Geographic Distribution**: Multiple devices showing similar connection behaviors

Enhanced Detection Requirements

For **complete ETELAAT TEAM detection**, integrate with additional data sources:

Network Security Tools

- **Proxy Logs**: Zscaler, BlueCoat, Palo Alto (full URL capture)
- **Firewall Logs**: Next-gen firewalls with application inspection
- **Web Security Gateways**: Content inspection and URL filtering logs

SIEM Integration

```
// Example: Enhanced detection with proxy log integration
DeviceNetworkEvents
```

```
| where RemoteUrl contains "api.telegram.org"
| join kind=inner (
    // Proxy logs table with full URL content
    ProxyLogs
    | where Url contains "/bot7431860324:" or Url contains
"/bot7972762095:"
    | extend BotToken = extract(@"bot(\d+)", 1, Url)
) on $left.RemoteIP == $right.ClientIP
```

Network Monitoring Tools

- **Zeek/Suricata**: HTTP content inspection
- **Wireshark/tcpdump**: Packet-level analysis
- **DNS Monitoring**: Recursive query analysis

Alternative Detection Strategies

1. DNS-Based Detection

```
// Detect frequent DNS queries to api.telegram.org
DeviceEvents
| where ActionType == "DnsRequest"
| where AdditionalFields contains "api.telegram.org"
| summarize QueryCount = count() by DeviceName, bin(TimeGenerated, 1h)
| where QueryCount > 50  // Threshold for automation
```

2. Certificate Pinning Analysis

```
// Detect SSL/TLS connections to Telegram infrastructure
DeviceNetworkEvents
| where RemoteUrl contains "telegram.org"
| where RemotePort in (443, 8443)
| summarize ConnectionCount = count() by DeviceName, RemoteIP
```

3. Process Memory Analysis

- Use memory forensics tools to extract JavaScript from browser processes
- Search for hardcoded bot tokens in process memory dumps
- Analyze browser cache and temporary files for malware artifacts

Deployment Recommendations

Phase 1: Immediate (MDE Only)

1. Deploy behavioral detection queries (Hypotheses 1-10)

2. Monitor for connection pattern anomalies
3. Alert on high-frequency Telegram API connections

Phase 2: Enhanced (MDE + Network Logs)

1. Integrate proxy logs for full URL inspection
2. Deploy bot token extraction queries
3. Enable content-based message analysis

Phase 3: Comprehensive (Full Stack)

1. Deploy network monitoring for packet inspection
2. Implement browser memory analysis capabilities
3. Create automated IOC extraction from HTTP content

Query Performance Considerations

- **Time Windows**: Limit queries to necessary time ranges (30d max recommended)
- **Device Filtering**: Focus on high-risk device populations first
- **Correlation Efficiency**: Use indexed fields (DeviceName, TimeGenerated) for joins
- **Threshold Tuning**: Adjust connection count thresholds based on environment size

False Positive Mitigation

1. **Legitimate Telegram Usage**: Filter known legitimate applications and users
2. **Browser Extension Traffic**: Account for Telegram web clients and extensions
3. **Development/Testing**: Exclude development environments and security testing
4. **Baseline Establishment**: Create environmental baselines for normal Telegram usage

---

7) References

- Sucuri Blog: Telegram's Role in Website Malware
- Mandiant: Telegram Malware in Iranian Cyber Espionage
- MITRE ATT&CK Framework

This document is prepared by Crimson7 - 2025 v1.0

Runbook Deployment Notes:

- All KQL queries tested for Microsoft Sentinel and MDE compatibility
- Queries include risk scoring and correlation logic to minimize false positives
- Investigation procedures account for ETELAAT TEAM's specific operational patterns
- Includes both real-time detection and historical campaign analysis capabilities