LOGZILLA DOCUMENTATION

SOAR Security Orchestration

Orchestrate LogZilla security responses using SEC correlation for impossible-travel, credential abuse, and multi-stage attack patterns with GeoIP and PAM data

Event Correlation · Generated April 27, 2026 · logzilla.ai/docs/event-correlation/soar-security-orchestration

SOAR Security Orchestration

Security Orchestration, Automation, and Response (SOAR) capabilities detect sophisticated attack patterns that span multiple events, systems, and time periods. LogZilla's pre-processing extracts security fields, while SEC handles complex correlation logic. SEC generates synthetic events back to LogZilla, where triggers perform intelligent automated responses.

Prerequisites: Ensure Event Correlation is enabled and forwarder reloading is available as shown in the Event Correlation Overview (https://www.logzilla.ai/docs/event-correlation/event-correlation-overview).

Geographic Anomaly Detection

Impossible Travel Detection

Detect when the same user logs in from geographically distant locations within an impossible timeframe.

Required Apps and Configuration

Required Apps:

- `linux__pam` app (for `PAM User Tracking` and `PAM Remote Host` user tags)
- `geoip` app (for `SrcIP Country`, `SrcIP City`, `SrcIP State` user tags)

IP Address Mapping Rule

The GeoIP app requires `SrcIP` user tags, but PAM creates `PAM Remote Host`. Create a mapping rule:

File: `/etc/logzilla/rules/enabled/601-pam-mapping.lua`

```
-- Map PAM Remote Host to SrcIP for GeoIP processing
function process(event)
    if event.user_tags["PAM Remote Host"] then
        event.user_tags["SrcIP"] = event.user_tags["PAM Remote Host"]
    end
end
```

LogZilla Forwarder Configuration

```
# /etc/logzilla/forwarder.d/linux-security.yaml
window_size: 0
```

```
type: sec
sec_name: linux-security
match:
  - field: extra:_is_security
    value:
      - "true"
rules:
- rewrite:
    message: >-
      [LINUX_SECURITY]
      program="${:program}"
      host="${:host}"
      srcip="${:SrcIP}"
      srcport="${:SrcPort}"
      dstip="${:DstIP}"
      dstport="${:DstPort}"
      username="${:Username}"
      src_country="${:SrcIP Country}"
      src_city="${:SrcIP City}"
      src_state="${:SrcIP State}"
      pam_user_tracking="${:PAM User Tracking}"
      pam_action="${:PAM Action}"
      pam_remote_user="${:PAM Remote User}"
      pam_remote_host="${:PAM Remote Host}"
      bytes="${:bytes}"
      bytes_mb="${:BytesMB}"
      $MESSAGE
```

SEC Rule: Impossible Travel Detection

```
# /etc/logzilla/sec/linux-security/detect-impossible-travel.sec
# ============================================
# Detect impossible travel (Linux security)
# Same user authenticating from two different countries within 1 hour
# Uses [LINUX_SECURITY] rewritten message from linux-security.yaml
# ============================================
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="([^"]+)".*srcip="([^"]+)".*src_country="
([^"]+)".*pam_user_tracking="([^"]+)"
desc=Impossible travel check for user $4 on host $1
action=eval %host ( "$1" ); \
       eval %srcip ( "$2" ); \
       eval %src_country ( "$3" ); \
       eval %pam_user ( "$4" ); \
       shellcmd (mkdir -p /var/log/sec /var/log/logzilla/sec 2>/dev/null; \
                 statefile="/var/log/sec/impossible-travel-%pam_user"; \
                 old_country=""; old_ip=""; \
                 if [ -f "$statefile" ]; then \
                   old_country=$(sed -n '1p' "$statefile"); \
                   old_ip=$(sed -n '2p' "$statefile"); \
```

```
                fi; \
                if [ -n "$old_country" ] && [ "%src_country" != "$old_country" ]; then \
                  echo "IMPOSSIBLE_TRAVEL %t host=%host user=%pam_user old_ip=$old_ip
old_country=$old_country new_ip=%srcip new_country=%src_country" >>
/var/log/logzilla/sec/security.log; \
                  logger -t SEC-SECURITY -p local0.alert "IMPOSSIBLE_TRAVEL user=\"%pam_user\"
old_ip=\"$old_ip\" old_country=\"$old_country\" new_ip=\"%srcip\" new_country=\"%src_country\""; \
                fi; \
                echo "%src_country" > "$statefile"; \
                echo "%srcip" >> "$statefile")
```

Processing Chain

The complete geographic correlation works through this processing chain:

PAM Rule (`600-linux-pam.lua`) extracts authentication data:

- Creates `PAM User Tracking` (username)
- Creates `PAM Remote Host` (source IP)

Mapping Rule (`601-pam-mapping.lua`) prepares for GeoIP:

- Copies `PAM Remote Host` → `SrcIP`

GeoIP Rule (`999-add-geodata.lua`) adds geographic data:

- Creates `SrcIP Country`, `SrcIP City`, `SrcIP State`

Forwarder sends enriched event to SEC with all user tags appended

SEC Rule parses user tags from message and detects impossible travel

LogZilla Trigger: Impossible Travel Response

```yaml
# Trigger responds to SEC-generated impossible travel events
name: "Impossible Travel Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "IMPOSSIBLE_TRAVEL"
actions:
  send_email: true
  send_email_template: |
```

```
    Subject: SECURITY ALERT: Impossible Travel Detected

    User: {{event:ut:Username}}
    Impossible Country: {{event:ut:SrcIP Country}}
    Impossible State: {{event:ut:SrcIP State}}
    Impossible City: {{event:ut:SrcIP City}}

    This indicates potential credential compromise.
  exec_script: true
  script_path: "/usr/local/bin/impossible-travel-response.sh"
  issue_notification: true
```

Intelligent Response Script

```
#!/bin/bash
# /usr/local/bin/impossible-travel-response.sh
# Called by SEC shellcmd - receives data via command-line arguments

USERNAME="$1"
DETAILS="$2"

# Query HR system for user travel status
TRAVEL_STATUS=$(curl -s "https://hr-api.company.com/travel-status/$USERNAME")

# Query threat intelligence for IP reputation
# Note: IP would need to be extracted from DETAILS or passed as separate argument
IP_REPUTATION=$(curl -s "https://threat-intel.company.com/ip-reputation/$(echo $DETAILS | grep -oE
'[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+')")

# Make intelligent decision based on context
if [[ "$TRAVEL_STATUS" == "approved_travel" ]]; then
    # User has approved travel - lower priority
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - approved travel detected"
    # Create low-priority ticket
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=low&subject=Travel Login Verification&user=$USERNAME"
elif [[ "$IP_REPUTATION" == "malicious" ]]; then
    # High-risk scenario - immediate response
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - malicious IP detected"
    # Disable account immediately
    curl -X POST "https://identity.company.com/api/disable-account" -d "username=$USERNAME"
    # Create high-priority incident
    curl -X POST "https://servicedesk.company.com/api/incidents" \
         -d "priority=critical&subject=Account Compromise&user=$USERNAME&details=$DETAILS"
    # Alert SOC team
    curl -X POST "https://slack.company.com/api/webhooks/soc-alerts" \
         -d "text=CRITICAL: Account compromise detected for $USERNAME"
else
    # Standard response - investigate
    logger -t SECURITY-RESPONSE "Impossible travel for $USERNAME - investigation required"
    # Temporarily restrict account
```

```
    curl -X POST "https://identity.company.com/api/restrict-account" -d "username=$USERNAME"
    # Create medium-priority ticket
    curl -X POST "https://servicedesk.company.com/api/tickets" \
         -d "priority=medium&subject=Impossible Travel Investigation&user=$USERNAME"
fi
```

Lateral Movement Detection

Cross-System Access Correlation

Detect lateral movement by correlating edge system logins with internal network access.

SEC Rule: Lateral Movement Correlation

```
# /etc/logzilla/sec/linux-security/detect-lateral-movement.sec
# ============================================
# Lateral Movement Detection (Linux Security)
# Detects when users log into edge systems and then access internal networks
# Uses [LINUX_SECURITY] rewritten message from linux-security.yaml
# ============================================

# Create context for edge system logins (matches forwarder field order)
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="((?:web|dmz|edge|vpn)[^"]*)".*pam_user_tracking="
([^"]+)".*pam_action="session opened"
desc=Edge system access by $2
action=eval %host ( "$1" ); \
       eval %pam_user ( "$2" ); \
       create EDGE_ACCESS_%pam_user_%host 3600; \
       write /var/log/logzilla/sec/security.log "EDGE_LOGIN %t username=%pam_user host=%host"; \
       shellcmd (logger -t SEC-AUDIT -p local0.info "EDGE_LOGIN user=\"%pam_user\" host=\"%host\"")

# Detect internal network access from edge systems
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*host="((?:web|dmz|edge|vpn)[^"]*)".*dstip="([^"]+)".*dstport="
([^"]+)".*username="([^"]+)"
context=EDGE_ACCESS_$4_$1
desc=Potential lateral movement by $4 from $1 to $2:$3
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
       eval %username ( "$4" ); \
       shellcmd (echo "%dstip" | grep -Eq '^(10\.|172\.(1[6-9]|2[0-9]|3[01])\.|192\.168\.)' && { \
                  echo "LATERAL_MOVEMENT %t username=%username src_host=%host dst_ip=%dstip
dst_port=%dstport" >> /var/log/logzilla/sec/security.log; \
                  logger -t SEC-SECURITY -p local0.crit "LATERAL_MOVEMENT user=\"%username\"
```

```
src_host=\"%host\" dst_ip=\"%dstip\" dst_port=\"%dstport\""; \
                } || true)
```

LogZilla Trigger: Lateral Movement Response

```
name: "Lateral Movement Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-SECURITY
  - field: message
    op: "=~"
    value: "LATERAL_MOVEMENT"
actions:
  exec_script: true
  script_path: "/usr/local/bin/lateral-movement-response.sh"
  issue_notification: true
  send_webhook: true
  send_webhook_template: |
    {
      "alert_type": "lateral_movement",
      "user": "{{event:ut:Username}}",
      "source_host": "{{event:host}}",
      "destination": "{{event:ut:DstIP}}:{{event:ut:DstPort}}",
      "severity": "critical"
    }
```

Advanced Persistent Threat (APT) Detection

Multi-Stage APT Correlation

Detect sophisticated attack campaigns across multiple stages and systems. This demonstrative example shows how SEC can correlate events to identify multi-stage attacks.

Prerequisites:

- Deploy `bandwidth-monitor.sh` script (see below)
- Add to crontab: `*/5 * * * * /path/to/bandwidth-monitor.sh`

Bandwidth Monitor Script

```
#!/bin/bash
# Simple bandwidth monitor for APT detection
# Logs large outbound transfers to syslog
```

```
THRESHOLD_MB=100
LOG_TAG="bandwidth-monitor"

ss -tn state established | awk 'NR>1 {
    split($4, src, ":")
    split($5, dst, ":")
    src_ip = src[1]
    dst_ip = dst[1]
    dst_port = dst[length(dst)]

    # Check for external connections (demo)
    if (dst_ip !~ /^(10\.|172\.(1[6-9]|2[0-9]|3[01])\.|192\.168\.)/) {
        bytes = int(rand() * 400000000) + 100000000
        printf "%s %s %s %s\n", src_ip, dst_ip, dst_port, bytes
    }
}' | while read src_ip dst_ip dst_port bytes; do
    bytes_mb=$((bytes / 1048576))
    if [ "$bytes_mb" -ge "$THRESHOLD_MB" ]; then
        logger -t "$LOG_TAG" \
            "LARGE_TRANSFER srcip=${src_ip} dstip=${dst_ip} dstport=${dst_port} bytes=${bytes}
bytes_mb=${bytes_mb}"
    fi
done
```

SEC Rule: APT Campaign Detection

```
# /etc/logzilla/sec/linux-security/detect-apt-campaign.sec
# ============================================
# APT Campaign Detection - Demonstrative Example
# Three-stage attack detection showing SEC correlation capabilities
# ============================================

# Stage 1: Suspicious outbound connection to external IP
# Detects connections to known-bad or suspicious IPs from kernel/firewall logs
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="kernel".*host="([^"]+)".*dstip="
((?:185\.220\.|45\.153\.|23\.129\.)[^"]+)".*dstport="([^"]+)"
desc=Suspicious outbound connection from $1 to $2:$3
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
       create APT_INDICATOR_%host 86400; \
       write /var/log/logzilla/sec/security.log "APT_STAGE1 %t host=%host
destination=%dstip:%dstport"; \
       shellcmd logger -t SEC-APT -p local0.warning "APT_STAGE1 host=\"%host\"
dst=\"%dstip:%dstport\" - suspicious outbound detected"

# Stage 2: Suspicious process execution on flagged host
# Looks for credential dumping or hacking tools in program field
type=Single
```

```
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="(mimikatz|pwdump|procdump|hashcat|hydra|metasploit)".*host="
([^"]+)"
context=APT_INDICATOR_$2
desc=Suspicious process $1 on compromised host $2
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       write /var/log/logzilla/sec/security.log "APT_STAGE2 %t host=%host process=%program"; \
       shellcmd logger -t SEC-APT -p local0.alert "APT_STAGE2 host=\"%host\" process=\"%program\" -
credential tool detected"

# Stage 3 Use Perl for numeric comparison
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="bandwidth-monitor".*host="([^"]+)".*dstip="([^"]+)".*dstport="
([^"]+)".*bytes="([0-9]+)"
context=APT_INDICATOR_$1
desc=Large transfer from compromised host $1 to $2:$3 ($4 bytes)
action=eval %host ( "$1" ); \
       eval %dstip ( "$2" ); \
       eval %dstport ( "$3" ); \
       eval %bytes ( "$4" ); \
       eval %large ( %bytes > 104857600 ); \
       if %large ( write /var/log/logzilla/sec/security.log "APT_CAMPAIGN %t host=%host
dst=%dstip:%dstport bytes=%bytes - FULL ATTACK CHAIN DETECTED"; \
                   shellcmd logger -t SEC-APT -p local0.crit "APT_CAMPAIGN host=\"%host\"
dst=\"%dstip:%dstport\" bytes=\"%bytes\" - FULL ATTACK CHAIN DETECTED" )
```

LogZilla Trigger: APT Campaign Response

```
name: "APT Campaign Incident Response"
filter:
  - field: program
    op: eq
    value: SEC-APT
  - field: message
    op: "=~"
    value: "APT_CAMPAIGN"
actions:
  send_email: true
  send_email_template: |
    Subject: CRITICAL: APT Campaign Detected

    A multi-stage attack has been detected and correlated by SEC.

    Full attack chain: Suspicious connection → Credential tool → Data exfiltration

    Immediate investigation and containment required.
  send_webhook: true
  send_webhook_template: |
    {
```

```
      "alert_type": "apt_campaign",
      "severity": "critical",
      "message": "{{event:message}}"
    }
```

Privilege Escalation Detection

Account Privilege Changes Correlation

Monitor for suspicious privilege escalations and administrative access patterns.

SEC Rule: Privilege Escalation Correlation

```
# ============================================
# Privilege Escalation Detection
# Two-stage detection: failed admin attempts → successful escalation
# Uses [LINUX_SECURITY] rewritten messages from linux-security.yaml
# ============================================

# Stage 1: Track multiple failed administrative access attempts (sudo/su)
# Looks for failed authentication in PAM logs
type=SingleWithThreshold
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="(sudo|su)".*host="([^"]+)".*pam_user_tracking="
([^"]+)".*authentication failure
desc=Multiple failed admin access attempts by $3 on $2
thresh=5
window=300
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %pam_user ( "$3" ); \
       create PRIV_ESC_ATTEMPT_%pam_user_%host 600; \
       write /var/log/logzilla/sec/security.log "PRIVILEGE_ESCALATION_ATTEMPT %t program=%program
user=%pam_user host=%host count=5"; \
       shellcmd (logger -t SEC-SECURITY -p local0.warning "PRIVILEGE_ESCALATION_ATTEMPT
program=\"%program\" user=\"%pam_user\" target=\"%host\" count=5")

# Stage 2: Detect successful admin access after failed attempts
# Looks for successful PAM session opened after failures
type=Single
ptype=RegExp
pattern=\[LINUX_SECURITY\].*program="(sudo|su)".*host="([^"]+)".*pam_user_tracking="
([^"]+)".*pam_action="(session opened|command executed)"
context=PRIV_ESC_ATTEMPT_$3_$2
desc=Successful privilege escalation by $3 on $2 after failed attempts
action=eval %program ( "$1" ); \
       eval %host ( "$2" ); \
       eval %pam_user ( "$3" ); \
```

```
        eval %pam_action ( "$4" ); \
        delete PRIV_ESC_ATTEMPT_%pam_user_%host; \
        write /var/log/logzilla/sec/security.log "PRIVILEGE_ESCALATION_SUCCESS %t program=%program
user=%pam_user host=%host action=%pam_action"; \
        shellcmd (logger -t SEC-SECURITY -p local0.alert "PRIVILEGE_ESCALATION_SUCCESS
program=\"%program\" user=\"%pam_user\" target=\"%host\" action=\"%pam_action\" - successful after
failures")
```

Related Topics

- Brute Force Attack Detection (https://www.logzilla.ai/docs/event-correlation/brute-force-attack-detection)
- BGP Network Monitoring Correlation (https://www.logzilla.ai/docs/event-correlation/bgp-network-monitoring-correlation)
- Event Correlation Rule Types (https://www.logzilla.ai/docs/event-correlation/event-correlation-rule-types)
- Creating Triggers (https://www.logzilla.ai/docs/creating-triggers/)