INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-07-04 14:50 UTC

FatFs Memory-Corruption Flaws Put Millions of Embedded Devices at Risk With No Upstream Fix in Sight

CVE VULNERABILITY | HIGH | CVSS 7.5

SCC Item ID	SCC-CVE-2026-0386
Type	CVE Vulnerability
CVE ID	CVE-2026-6682, CVE-2026-6683, CVE-2026-6684, CVE-2026-6685, CVE-2026-6686, CVE-2026-6687, CVE-2026-6688
Severity	HIGH
CVSS Base Score	7.5
EPSS Score	0.0021 (11th percentile)
Affected Products	FatFs embedded FAT/exFAT filesystem library (all versions carrying unpatched flaws); integrators: Espressif ESP-IDF, STMicroelectronics STM32Cube, Zephyr RTOS, MicroPython, ArduPilot, RT-Thread, Mbed, Samsung TizenRT, SWUpdate; downstream device classes: security cameras, drones, industrial controllers, hardware crypto wallets, ATMs, public kiosks, voting machines
Published	2026-07-03T16:19:31
Discovery Source	Rss

Executive Summary

Seven memory-corruption vulnerabilities in FatFs, a widely embedded FAT/exFAT filesystem library, affect firmware across industrial controllers, security cameras, drones, ATMs, hardware crypto wallets, voting machines, and public kiosks. Six of the seven flaws remain unpatched upstream, with the sole maintainer unresponsive to coordinated disclosure; public proof-of-concept exploit code is now available. Organizations that have not independently patched FatFs within their firmware supply chain face elevated risk of device compromise via malicious USB drives, SD cards, or crafted firmware update files.

Technical Analysis

runZero disclosed seven vulnerabilities in FatFs (all versions carrying unpatched flaws), an open-source FAT/exFAT filesystem library widely integrated into RTOS and IoT firmware stacks. The flaw set spans CWE-190 (integer overflow), CWE-120 (classic buffer overflow), CWE-121 (stack-based buffer overflow), CWE-369 (divide-by-zero), CWE-400 (uncontrolled resource consumption), and CWE-200 (information

exposure). CVE IDs assigned: CVE-2026-6682, CVE-2026-6683, CVE-2026-6684, CVE-2026-6685, CVE-2026-6686, CVE-2026-6687, CVE-2026-6688. Aggregate CVSS base score of 7.5 (High) is reported in security news sources; individual per-CVE CVSS scores from NVD were not confirmed in provided sources. Confidence in per-CVE granularity is LOW. EPSS: 0.0021 (11th percentile). CISA KEV: not listed. MITRE ATT&CK techniques mapped: T1200 (Hardware Additions), T1203 (Exploitation for Client Execution), T1400 (Modify System Image), T1091 (Replication Through Removable Media), T1195.002 (Compromise Software Supply Chain), T1542 (Pre-OS Boot / Firmware). Attack vectors include physical-layer delivery via malicious USB drives or SD cards and supply-chain firmware injection. Public proof-of-concept exploit code has been released by runZero. Six of seven CVEs remain unpatched upstream. Affected integrators include Espressif ESP-IDF, STMicroelectronics STM32Cube, Zephyr RTOS, MicroPython, ArduPilot, RT-Thread, Mbed, Samsung TizenRT, and SWUpdate. No coordinated upstream fix pipeline exists; remediation responsibility falls entirely to downstream integrators. Note: the NVD URL provided in source data references CVE-2026-47291, a distinct CVE, and has been disregarded for this report.

Action Checklist

1. Step 1: Containment. Inventory all firmware builds that incorporate FatFs (check ESP-IDF, STM32Cube, Zephyr, MicroPython, ArduPilot, RT-Thread, Mbed, TizenRT, SWUpdate integrations). Physically restrict or disable USB and SD card ports on affected device classes - security cameras, ATMs, kiosks, industrial controllers, voting machines, and hardware crypto wallets - where operationally feasible, per PE-3 (Physical Access Control) and SI-7 (Software, Firmware, and Information Integrity) principles. Reference CIS 1.1 to confirm your asset inventory captures all embedded device categories.
2. Step 2: Detection. Query firmware build manifests, software bills of materials (SBOMs), and vendor advisories for FatFs inclusion. Check whether any of the nine named integrator SDKs are in your supply chain. Monitor device logs for unexpected resets, filesystem parse errors, or anomalous removable media mount events that may indicate exploitation attempts. Reference AU-2 (Event Logging) and AU-6 (Audit Record Review) to ensure relevant device-level events are captured and reviewed. No confirmed IOC hashes or network indicators are available from provided sources at this time.
3. Step 3: Eradication. Contact each affected integrator (Espressif, STMicroelectronics, Zephyr, MicroPython, ArduPilot, RT-Thread, Mbed, Samsung TizenRT, SWUpdate) directly for patched SDK versions or backported fixes. Apply vendor-supplied firmware updates as they become available; do not wait for upstream FatFs resolution, as the maintainer has been unresponsive. Where integrator patches are not yet available, implement compensating controls: enforce allowlist-only removable media policies and restrict firmware update paths to verified, cryptographically signed images. Reference CIS 4.1 (Establish and Maintain a Secure Configuration Management Process) for patch management cadence.
4. Step 4: Recovery. After applying integrator-supplied patches, re-test affected devices for correct filesystem operation. Validate that removable media controls remain enforced post-update. Monitor device health logs and audit records for anomalies for a minimum of 30 days post-remediation, per AU-11 (Audit Record Retention) guidance. Confirm updated firmware versions are reflected in your asset and software inventories per CIS 1.1 and CIS 2.1.
5. Step 5: Post-Incident. Document firmware supply chain gaps this cluster exposed, specifically the absence of SBOM tracking for third-party embedded libraries. Establish a process for monitoring upstream open-source library maintainer responsiveness as part of vendor risk assessment. Implement or verify a formal vulnerability management process per CIS 7.1 and CIS 7.2 that includes embedded/IoT firmware components. Review supply-chain software composition controls mapped to SA-12 (Supply Chain

Protection) and AC-20 (Use of External Systems) to reduce future exposure from single-maintainer open-source dependencies.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately if forensic evidence (RAM dump heap analysis, serial console crash traces, or malformed FAT/exFAT volume found on a device) indicates active exploitation has occurred on any ATM, voting machine, industrial controller, or hardware crypto wallet, as these device classes trigger regulatory notification obligations (PCI DSS for ATMs, HAVA/state election law for voting machines, ICS-CERT reporting for industrial controllers) and represent high-consequence blast radius beyond confidentiality — including physical safety and financial integrity.
Recovery Notes	Re-flash all affected devices with integrator-supplied patched firmware that either backports fixes for CVE-2026-6682 through CVE-2026-6688 or disables the vulnerable FatFs code paths, and read back each flashed image to verify hash integrity before returning devices to service. Post-patch, enforce removable media allowlisting permanently — do not restore unrestricted USB/SD access even after patching, given public PoC availability and the six unpatched CVEs still pending upstream resolution. Monitor device health logs, UART serial output, and any available syslog feeds for filesystem parse errors, unexpected resets, or heap-corruption signatures for a minimum of 30 days, consistent with NIST AU-11 (Audit Record Retention) guidance, and retain all incident artifacts under your defined records retention schedule.
Forensic Artifacts	RAM dump from JTAG/SWD debug interface captured before isolation or firmware update — heap memory will contain corrupted FAT/exFAT parsing structures and potential shellcode or ROP chain fragments if CVE-2026-6682 through CVE-2026-6688 were weaponized for code execution on the target MCU | Serial/UART console output showing panic, hard fault, or watchdog reset traces attributable to memory corruption during SD card or USB mass-storage mount events — on STM32 targets this appears as HardFault_Handler stack traces; on ESP32 as Guru Meditation Errors with heap corruption backtrace | Forensic image of the removable media (SD card or USB drive) presented to the affected device, which may contain a deliberately malformed FAT32 or exFAT filesystem crafted to trigger the specific BPB parsing, cluster chain, or LFN entry heap overflow described in the CVE cluster | Firmware binary read-back from affected devices via esptool.py (ESP32), STM32CubeProgrammer (STM32), or equivalent — hash comparison against known-good signed release identifies whether unauthorized firmware was installed via a compromised SWUpdate or Mbed OTA update path following initial exploitation | Build manifest or SBOM artifacts (CMakeLists.txt, west.yml, idf_component.yml, requirements.txt) from each affected firmware project pinpointing the exact FatFs source commit hash included — establishes which specific CVEs from CVE-2026-6682 through CVE-2026-6688 apply to each device class and supports regulatory disclosure scoping

Per-Action IR Details

Step 1: Containment — Inventory all firmware builds that incorporate FatFs (check ESP-IDF, STM32Cube, Zephyr, MicroPython, ArduPilot, RT-Thread, Mbed, TizenRT, SWUpdate integrations). Physically restrict or disable USB and SD card ports on affected device classes — security cameras, ATMs, kiosks, industrial controllers, voting machines, and hardware crypto wallets — where operationally feasible, per AC-17 (Remote Access) and AC-19 (Access Control for Mobile Devices) principles. Reference CIS 1.1 to confirm your asset

inventory captures all embedded device categories.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: Choose a containment strategy based on the need to keep the system operational and the potential for further damage if the system remains operational.

Controls: NIST AC-19 (Access Control For Mobile Devices), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices)

Compensating: For devices where physical port disablement is not possible, use epoxy or tamper-evident port blockers for USB/SD slots on ATMs and kiosks. On Linux-based embedded devices (e.g., Yocto-built industrial controllers), add a udev rule to block mass-storage class devices: `echo 'SUBSYSTEM=="usb", ATTRS{bInterfaceClass}=="08", ATTR{authorized}="0"' > /etc/udev/rules.d/99-block-msc.rules && udevadm control --reload`. On ESP-IDF or STM32Cube targets, rebuild firmware with SD/SDIO peripheral disabled in the hardware abstraction layer (HAL) configuration header (e.g., `#define HAL_SD_MODULE_DISABLED` in stm32xxx_hal_conf.h) until a patch is available.

Evidence: Before physically restricting ports or modifying device configuration, capture: (1) current firmware version string and build hash from device bootloader output or `/proc/version` on Linux targets; (2) a list of currently mounted filesystems (`/proc/mounts` or `cat /etc/mtab`) to confirm whether a malicious FAT/exFAT volume is already mounted; (3) dmesg output (`dmesg | grep -E 'sd|mmc|fat|exfat|usb'`) for evidence of recent removable media mount events that could indicate prior exploitation; (4) full asset inventory snapshot (device model, firmware version, integrator SDK version, FatFs version string if extractable via strings on the firmware binary). These are volatile or configuration-state artifacts that a firmware update or port disablement will overwrite or obscure.

Step 2: Detection — Query firmware build manifests, software bills of materials (SBOMs), and vendor advisories for FatFs inclusion. Check whether any of the nine named integrator SDKs are in your supply chain. Monitor device logs for unexpected resets, filesystem parse errors, or anomalous removable media mount events that may indicate exploitation attempts. Reference AU-2 (Event Logging) and AU-6 (Audit Record Review) to ensure relevant device-level events are captured and reviewed. No confirmed IOC hashes or network indicators are available from provided sources at this time.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Analyze all available precursors and indicators; look for correlating information across multiple sources to confirm whether an incident has occurred.

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, And Reporting), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 8.2 (Collect Audit Logs)

Compensating: Use `strings` and `grep` against extracted firmware binaries to detect FatFs inclusion: `strings firmware.bin | grep -E 'ff_[a-z]+|FatFs|f_mount|f_open|f_read|diskio'` — the FatFs public API and internal function names are distinctive. For SBOM generation on built firmware, use `syft` (free, Anchore) or `cyclonedx-cli` against firmware filesystem extracts unpacked with `binwalk -e firmware.bin`. Monitor syslog on Linux-based embedded devices for kernel messages matching `EXT_ERROR`, `FAT-fs`, or `exFAT-fs` error strings that indicate malformed filesystem parsing. For devices with serial debug consoles (ATMs, industrial PLCs), capture UART output during SD/USB mount events using a USB-to-serial adapter and `minicom` or `screen /dev/ttyUSB0 115200`.

Evidence: This is a detection/analysis step that does not alter live state, so order-of-volatility sequencing is not a blocking concern here. However, preserve the following before any remediation proceeds: (1) SBOM or build manifest artifacts showing FatFs version linked into each firmware image (check `CMakeLists.txt`, `Kconfig`, `idf_component.yml` for ESP-IDF; `Makefile` or `west.yml` for Zephyr; `requirements.txt` or `setup.py` for MicroPython); (2) kernel or RTOS log files showing filesystem parse errors — on Linux targets at `/var/log/kern.log` or via `journalctl -k`; (3) serial console capture (UART) showing panic/reset traces attributable to heap corruption, stack smashing, or null-pointer dereferences during FAT/exFAT volume parsing; (4) any removable media (SD card, USB drive) presented to affected devices, which should be forensically imaged with `dd if=/dev/sdX of=media_image.dd bs=4M` and preserved as potential exploit delivery vehicles.

Step 3: Eradication — Contact each affected integrator (Espressif, STMicroelectronics, Zephyr, MicroPython, ArduPilot, RT-Thread, Mbed, Samsung TizenRT, SWUpdate) directly for patched SDK versions or backported fixes. Apply vendor-supplied firmware updates as they become available; do not wait for upstream FatFs resolution, as the maintainer has been unresponsive. Where integrator patches are not yet available, implement compensating controls: enforce allowlist-only removable media policies and restrict firmware update paths to verified, cryptographically signed images. Reference CIS 7.3 and CIS 7.4 for OS and application patch management cadence.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: After containing the incident, eradicate the cause of the incident; delete malicious code, identify and mitigate all vulnerabilities that were exploited.

Controls: NIST SI-2 (Flaw Remediation), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Where integrator patches are unavailable, apply source-level compensating mitigations directly in the FatFs library: add bounds checks before `ff_memcpy`/`ff_memset` calls in `ff.c` targeting the heap allocation paths associated with CVE-2026-6682 through CVE-2026-6688, and recompile with stack-canary and heap-overflow protections enabled (`-fstack-protector-strong -D_FORTIFY_SOURCE=2` for GCC-based toolchains such as xtensa-esp32-elf-gcc or arm-none-eabi-gcc). For firmware update path integrity, verify cryptographic signatures before flashing: on ESP-IDF use `espsecure.py verify_signature --keyfile pub.pem firmware.bin`; on STM32 use STM32CubeProgrammer with secure boot enabled. Track integrator patch releases via GitHub release feeds: e.g., `https://github.com/espressif/esp-idf/releases` and `https://github.com/zephyrproject-rtos/zephyr/releases` — subscribe via RSS to detect patch availability without polling.

Evidence: Before applying any firmware update or recompiled image to a device that may have been actively exploited, capture the following volatile state: (1) full RAM dump if the device supports JTAG or SWD debug interface (`openocd -f interface/cmsis-dap.cfg -f target/stm32f4x.cfg -c 'dump_image ram.bin 0x20000000 0x20000'`), as heap corruption artifacts from CVE-2026-6682 through CVE-2026-6688 exploitation will exist only in live memory; (2) current running firmware image hash (`sha256sum` of the flashed binary read back via `esptool.py read_flash` or STM32CubeProgrammer) to confirm whether unauthorized firmware was loaded via a compromised SWUpdate or Mbed update path; (3) filesystem state of the FAT/exFAT volume that triggered the vulnerability — forensic image of the SD card or USB device used as the attack vector, preserved before the device is wiped or reformatted.

Step 4: Recovery — After applying integrator-supplied patches, re-test affected devices for correct filesystem operation. Validate that removable media controls remain enforced post-update. Monitor device health logs and audit records for anomalies for a minimum of 30 days post-remediation, per AU-11 (Audit Record Retention) guidance. Confirm updated firmware versions are reflected in your asset and software inventories per CIS 1.1 and CIS 2.1.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: Restore systems to normal operation, confirm that the systems are functioning normally, and remediate vulnerabilities to prevent similar incidents.

Controls: NIST AU-11 (Audit Record Retention), NIST AU-6 (Audit Record Review, Analysis, And Reporting), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Validate patched firmware behavior by mounting a deliberately malformed FAT32/exFAT volume — craft one using `mformat` with oversized cluster values or corrupted BPB (BIOS Parameter Block) fields — and confirm the patched device rejects or safely handles it without reset or hang. Use a logic analyzer or UART serial console to observe device response. For 30-day post-patch monitoring without SIEM, configure a cron job on a central syslog collector to alert on FatFs-specific error strings: `grep -E 'FAT-fs|exFAT|ff_mount|diskio|heap|panic|reset' /var/log/remote/device_*.log | mail -s 'FatFs anomaly alert' soc@yourorg.com`. Update SBOM records in your asset inventory immediately after each firmware flash, recording integrator SDK version, FatFs patch commit hash, and flash date.

Evidence: Recovery does not alter forensic evidence already collected in prior phases, but confirm the following before declaring recovery complete: (1) read back and hash the newly flashed firmware from each device (`esptool.py

read_flash` for ESP32, STM32CubeProgrammer read-back for STM32) and compare against the known-good signed image hash to confirm successful flash; (2) verify no persistence mechanism was installed by a prior exploit — on Linux-embedded targets, check for unexpected files in `/etc/init.d/`, `/etc/rc.local`, or unusual cron entries added during a potential compromise window; (3) confirm removable media port block rules (udev, HAL config) survived the firmware update, as OTA update processes on some integrators (SWUpdate, Mbed) may reset peripheral configuration to defaults.

Step 5: Post-Incident — Document firmware supply chain gaps this cluster exposed, specifically the absence of SBOM tracking for third-party embedded libraries. Establish a process for monitoring upstream open-source library maintainer responsiveness as part of vendor risk assessment. Implement or verify a formal vulnerability management process per CIS 7.1 and CIS 7.2 that includes embedded/IoT firmware components. Review supply-chain software composition controls mapped to NIST AC-20 (Use of External Systems) and T1195.002 to reduce future exposure from single-maintainer open-source dependencies.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: Review lessons learned, update IR plan, improve detection, share intelligence, and update policies to prevent recurrence.

Controls: NIST AC-20 (Use Of External Systems), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Implement SBOM generation as a mandatory CI/CD pipeline gate using `syft` or `cyclonedx-cli` on every firmware build, outputting CycloneDX or SPDX format SBOMs that explicitly enumerate third-party C libraries including FatFs, version pinned by git commit hash. For maintainer-responsiveness monitoring, use a lightweight script to poll the upstream FatFs GitHub repository (`github.com/abbrev/fatfs` or the ChaN mirror) for commit activity and open CVE references, alerting if no upstream activity occurs within a defined SLA window. Document the single-maintainer risk of FatFs (ChaN/elm-chan.org) explicitly in your vendor risk register, categorized under AC-20 (Use of External Systems), and require integrator-level patching SLAs as a procurement condition for new embedded device acquisitions. Note: the T1195.002 reference in the original step is a MITRE ATT&CK technique ID (Supply Chain Compromise: Compromise Software Supply Chain) — it describes attacker behavior and is not a defensive control; it has been excluded from the controls field per control discipline rules and is noted here for analyst situational awareness only.

Evidence: Post-incident documentation should preserve: (1) the complete timeline of integrator advisory publication dates versus your internal detection date, to calculate dwell-time exposure for each affected device class; (2) SBOM snapshots from pre-patch firmware builds confirming which FatFs version (and which of CVE-2026-6682 through CVE-2026-6688) was present in each device category — these become evidentiary records if regulatory notification is required for ATM or voting machine compromise; (3) any serial console crash dumps or heap-corruption traces collected during the detection phase, retained per your AU-11 (Audit Record Retention) policy as incident evidence; (4) vendor response logs (email threads, advisory URLs, ticket IDs) from all nine integrators documenting patch availability timelines, supporting future vendor risk scoring.

Detection Guidance

No network-layer IOCs (IPs, domains, hashes) are available from provided sources. Detection focus is supply-chain and host-based. (1) SBOM audit: search firmware build manifests for FatFs as a dependency; flag any build incorporating ESP-IDF, STM32Cube, Zephyr, MicroPython, ArduPilot, RT-Thread, Mbed, TizenRT, or SWUpdate until integrator patch status is confirmed. (2) Device event monitoring: look for filesystem parse errors, unexpected device reboots, or anomalous removable media mount/unmount events in device-level syslog or serial console logs; these may indicate malformed FAT/exFAT image processing. (3) Physical access audit: review physical access logs for USB or SD card insertion events on sensitive device classes (ATMs, kiosks, industrial controllers, voting machines). (4) Firmware integrity: where devices support it, validate

firmware image signatures and compare against known-good hashes before and after any removable media contact. Reference AU-2 and AU-6 for logging requirements; reference SI-7 (Software, Firmware, and Information Integrity) monitoring for file system artifact analysis. Confidence in behavioral indicators is MODERATE; no confirmed in-the-wild exploitation has been reported in provided sources.

Framework Mappings

MITRE-ATTACK

- **T1200** — Hardware Additions
- **T1203** — Exploitation for Client Execution
- **T1400**
- **T1091** — Replication Through Removable Media
- **T1195.002** — Compromise Software Supply Chain
- **T1542** — Pre-OS Boot

NIST-800-53R5

- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-16** — Memory Protection
- **AC-3** — Access Enforcement
- **SC-28** — Protection of Information at Rest
- **SC-5** — Denial-of-Service Protection

OWASP-TOP10-2021

- **A03:2021** — Injection
- **A01:2021** — Broken Access Control

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **13.8** — Deploy a Network Intrusion Prevention Solution
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1200	Hardware Additions	Initial-Access
T1203	Exploitation for Client Execution	Execution
T1400		
T1091	Replication Through Removable Media	Lateral-Movement
T1195.002	Compromise Software Supply Chain	Initial-Access
T1542	Pre-OS Boot	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/07/unpatched-flaws-disclosed-in-file...	T2
CVE Record: CVE-2026-6682	https://www.cve.org/CVERecord?id=CVE-2026-6682	T1
CVE-2026-47291 detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-47291	T1
Post -Patch Tuesday Roundup: June 2026	https://www.softcat.com/blog/post-patch-tuesday-roundup-june-2026	T3
May 2026 CVE Landscape	https://www.recordedfuture.com/blog/may-2026-cve-landscape	T1

Generated 2026-07-04 14:50 UTC by TJS Security Command Center