Firewall Core + WFP Monitor
Full Project Documentation

Installer, monitors, modules, state files, logs, golden baselines, tamper protection, tests, and operational checklist

Build date: January 06, 2026
Audience: project maintainers and operators

This document describes the Firewall Core system (baseline firewall enforcement + self-healing) and the WFP Monitor (Windows Filtering Platform audit monitoring and enforcement Phases A through C4). It also documents the packaging layout and the wrapper CMD launchers used for seamless install and uninstall.

1. System goals and threat model

Firewall Core aims to keep a known-good firewall baseline intact over time, even if local settings drift due to software installs, Windows features, or an attacker with limited privileges. The WFP Monitor adds visibility into blocked or dropped traffic recorded by Windows Filtering Platform (Security log Event IDs such as 5152/5157) and can optionally enforce temporary or persistent outbound blocks based on high-signal conditions.

2. Package layout (source vs installed)

There are two separate roots:
- Payload root (what you ship/copy): C:\FirewallInstaller\Firewall\...
- Install root (runtime): C:\Firewall\...
The installer itself lives under C:\FirewallInstaller and copies payload into C:\Firewall.

Path	Purpose / Contents
C:\FirewallInstaller\	Staging root used only for install/uninstall and debugging.
C:\FirewallInstaller\Firewall-Install.cmd	Wrapper that runs the installer PowerShell as admin and keeps a console open for errors.
C:\FirewallInstaller\Uninstall-Firewall.cmd	Wrapper that runs the uninstaller PowerShell as admin and keeps a console open for errors.
C:\FirewallInstaller\install-debug.txt (or log)	Install transcript output captured during install.
C:\FirewallInstaller_internal\Install-Firewall.ps1	Main installer script (copies payload, enables auditing, applies rules, registers tasks).
C:\FirewallInstaller_internal\Uninstall-Firewall.ps1	Uninstaller script (planned: restore defaults and remove artifacts).
C:\FirewallInstaller\Firewall\	Payload folder copied into C:\Firewall during install.
C:\Firewall\Monitor\	Runtime monitor scripts: Firewall-Core.ps1 and Firewall-WFP-Monitor.ps1, plus integrity/tamper
C:\Firewall\Scripts\	One-shot policy scripts (e.g., Firewall-Home-All.ps1 to apply baseline rules).
C:\Firewall\Modules\	Reusable helper modules used by monitors (parsers, actions, allow/deny logic).
C:\Firewall\State\	State/config/bookmarks for monitors (baseline.json, wfp.config.json, allowlists, strikes, blocked,

C:\Firewall\Golden\	Golden baseline snapshots and hashes used for integrity comparisons and controlled upgrades
C:\Firewall\Maintenance\	Maintenance-mode scripts: enter/exit maintenance, baseline version bump, defender integration
C:\Firewall\Tests\	Dev-only test scripts (regression tests, C4 test harness).
C:\Firewall\Tools\	Operator tools (baseline update approval, reset, audits, etc.).
C:\Firewall\Logs\	Optional plain-text/JSON logs if enabled; primary telemetry is Windows Event Log 'Firewall'.

3. Installer flow (Install-Firewall.ps1)

The installer does four primary things:
A) Enables WFP auditing (Audit Policy) so Security log records WFP connection blocks and packet drops.
B) Copies the payload folders from C:\FirewallInstaller\Firewall into C:\Firewall.
C) Applies the baseline firewall policy by running C:\Firewall\Scripts\Firewall-Home-All.ps1 in a separate PowerShell process.
D) Registers scheduled tasks that run as SYSTEM:
- Firewall Core Monitor (every 5 minutes)
- Firewall WFP Monitor (every 1 minute)
It also optionally imports ScriptSigningCert.cer into LocalMachine\Root (best-effort).

Key implementation notes

- Run-as-admin is mandatory because the installer creates scheduled tasks as SYSTEM, adds event log sources, and may import certificates.
- The installer uses Process-scope ExecutionPolicy Bypass only for its own run. Optional hardening sets:
* CurrentUser = RemoteSigned (safe default for typical admins)
* LocalMachine = AllSigned (best-effort; may be blocked by Group Policy)
- Applying firewall rules is done in a child process so failures return an exit code and do not half-apply silently.
- Install transcript is written to C:\FirewallInstaller\install-debug.txt (or .log) to preserve the exact failure reason when a CMD window closes.

CMD wrappers

```
Firewall-Install.cmd (example pattern)
-----------------------------------
@echo off
cd /d "%~dp0"
echo === Firewall Installer CMD Started (Admin) ===
powershell.exe -NoProfile -ExecutionPolicy Bypass -Command ^
  "Start-Process powershell -Verb RunAs -ArgumentList '-NoProfile -ExecutionPolicy Bypass -File
echo.
echo Installer finished. Press any key to exit.
pause >nul

Uninstall-Firewall.cmd is the same pattern but calls _internal\Uninstall-Firewall.ps1
```

4. Firewall Core Monitor

Purpose: Detect and correct firewall baseline drift.
Typical behavior:
- Reads the baseline snapshot in C:\Firewall\State (baseline.json plus baseline.hash).
- Compares the current firewall rule set to the stored baseline.
- If drift is detected, it restores expected rules and writes events to the custom Windows event log "Firewall" using provider "Firewall-Core".
Common event IDs:
- 3200: Drift detected (includes actor and process context when available)
- 3001: Rule restored

Interpreting a drift event

Example drift message fields:
- Actor='WORKGROUP\DESKTOP-XXXX$': the machine account context associated with the change as observed by the monitor.
- ActorClass=SYSTEM / DetectedBy=SYSTEM: the monitor is running as SYSTEM and detected the change.
- Process=conhost.exe: console host associated with the change observation. This is not always the true root-cause process; it can be the host for a PowerShell session or a tool that invoked NetSecurity cmdlets.
Recommendation: treat drift events as "something changed firewall state" and correlate with:
- Nearby WFP Monitor events
- Security log WFP bursts
- Any known maintenance windows / software installs

5. WFP Monitor (Phases A through C4)

The WFP Monitor reads Windows Security log WFP events and produces higher-level telemetry and optional enforcement.

Phase A (Auditing): Enable audit subcategories so Security log contains WFP events.
Phase B (Parsing): Parse event XML into structured fields (Application, PID, direction, src/dst, protocol).
Phase C1 (Summary): Write summary events to Windows Event Log 'Firewall' provider 'Firewall-WFP' (EventId 3400).
Phase C2 (Temp Enforcement): If a single executable crosses AlertThresholdPerProcessPerPoll, create a temporary outbound block rule for that executable.
Phase C3 (Escalation): Track strikes per executable; if it exceeds a strike threshold (e.g., 3), apply a persistent outbound block.
Phase C4 (Deny-hash): If an executable hash is listed in wfp.denyhash.json, immediately quarantine/block it (highest priority).

WFP monitor configuration and state files

C:\Firewall\State\wfp.config.json
- EventIds: WFP Security log IDs to ingest (commonly 5152, 5157)
- MaxEventsPerPoll: Upper bound read per scheduled run
- AlertThresholdPerProcessPerPoll: C2 threshold
- BurstThresholdPerPoll: Burst warning threshold
- CorrelateWindowMinutes: Lookback window for correlation with Firewall-Core activity
- HashExecutables: When true, compute SHA256 for enriched executables

C:\Firewall\State\wfp.allowlist.json
- ProcessNameContains: substrings treated as noise
- DestPorts: ports treated as noise (example: 5353 mDNS)
- DestAddresses: addresses treated as noise (example: 224.0.0.251, ff02::fb)

C:\Firewall\State\wfp.bookmark.json
- LastRecordId: bookmark so each poll only processes new Security log records

C:\Firewall\State\wfp.strikes.json
- Per-exe strike counters for C3 escalation

C:\Firewall\State\wfp.blocked.json
- Record of which executables were persistently blocked and when

C:\Firewall\State\wfp.denyhash.json
- DenySha256: list of SHA256 strings that trigger immediate quarantine

Important correctness notes (common pitfalls)

- WFP event fields are best read by parsing the event XML. Using $_.Properties[index] is fragile because indexes differ between event IDs and Windows builds.

- Event field 'Direction' values may be token strings (%%14592). Treat them as opaque unless you map tokens; they are still useful for relative analysis.
- Allowlist filtering must use consistent property names. If your parsed object uses Application, filter on Application (not Process).
- Enforcement requires a real executable path that exists on disk. Always resolve PID -> ExecutablePath before calling New-NetFirewallRule -Program.

6. Enforcement actions and helper modules

Recommended structure:
- Modules\WFP-Helpers.ps1: parsing, normalization, allowlist checks, process enrichment (PID -> ExecutablePath), hashing/signature checks.
- Modules\WFP-Actions.ps1: enforcement functions:
* Invoke-TempBlock -ExePath <path> -Minutes <n>
* Invoke-PersistentBlock -ExePath <path>
* Invoke-QuarantineExe -ExePath <path> (alias/variant; typically persistent block)

C4 ordering: deny-hash enforcement should run before C2/C3 strike logic, so known-bad hashes always win.

Suggested ordering inside each WFP Monitor poll

```
1) Parse events -> signal
2) Group by Application
3) Enrich top offenders (Exe path, signature, optional hash)
4) C4 deny-hash (immediate persistent block) using enriched offender(s)
5) C2 temp block (threshold) + increment strike
6) C3 persistent block (strike threshold)
7) Write summary + correlation events
8) Update bookmark
```

7. Golden baselines and controlled updates

Golden files capture a known-good baseline (rules, config, and state) and support controlled updates.
Typical pattern:
- Golden baseline snapshot + hash: approved reference
- Active baseline snapshot + hash: enforced by Firewall Core

Suggested update workflow:
1) Enter maintenance window (prevents self-heal from fighting operator changes).
2) Apply planned changes (new rules, config changes).
3) Generate a new baseline snapshot and validate.
4) Approve and tag baseline rules (if your workflow requires explicit approval).
5) Exit maintenance; self-heal now enforces the new baseline.

8. Tamper protection

Tamper protection provides defense-in-depth:
- Detect missing/changed monitor scripts or state files.
- Optionally restore from Golden copies (depending on your implementation).
- Produce explicit event log entries when tampering is detected.

Design recommendation: keep tamper checks deterministic and fast. For expensive hashing, prefer cached hashes (firewall.rules.hash, baseline.hash).

9. Logs and telemetry

Primary telemetry:
- Windows Event Log: "Firewall"
Providers:
- Firewall-Core: baseline drift and self-heal
- Firewall-WFP: WFP summaries and enforcement

Secondary telemetry:
- C:\FirewallInstaller\install-debug.txt (install transcript)
- C:\Firewall\Logs\ (optional, if you log exports or JSON summaries)

Useful operator commands:
- Query tasks:
schtasks /Query /TN "Firewall Core Monitor" /V /FO LIST
schtasks /Query /TN "Firewall WFP Monitor" /V /FO LIST
- Query events:
Get-WinEvent -LogName Firewall -MaxEvents 50 | Where ProviderName -eq "Firewall-WFP"
Get-WinEvent -LogName Firewall -MaxEvents 50 | Where ProviderName -eq "Firewall-Core"
- Confirm WFP auditing:
auditpol /get /subcategory:"Filtering Platform Connection"
auditpol /get /subcategory:"Filtering Platform Packet Drop"

10. Dev-only tests and regression matrix

Dev-only scripts (safe to ship but do not auto-run in production):
- Tests\Test-WFP-C4.ps1: inject a known hash into wfp.denyhash.json and force the monitor to run; validates deny-hash enforcement path.
- Tests\... drift simulations: validate that Firewall Core detects changes and restores baseline.

Regression test matrix (recommend a separate file under Docs or Tests):
- Fresh install (admin): tasks created, rules applied, event log sources created
- Reinstall over existing: idempotent, no duplicate tasks
- WFP auditing enabled: 5157 events appear when you trigger a known block
- Core drift simulation: baseline restored and 3200/3001 events produced
- C2 temp block: WFP-TEMP-BLOCK::* rule appears then auto-removes
- C3 persistent block: WFP-PERSISTENT-BLOCK::* rule persists across reboots
- C4 deny-hash: immediate persistent block + 3404 event, wfp.blocked.json updated

Appendix: Quick install checklist

```
1) Copy C:\FirewallInstaller folder to target machine (or unpack zip).
2) Right-click Firewall-Install.cmd -> Run as administrator.
3) Verify:
   - C:\Firewall exists with expected subfolders
   - schtasks /Query shows both tasks
   - Get-WinEvent -LogName Firewall shows Firewall-Core and Firewall-WFP providers
   - Baseline rules exist (Firewall-Home-All applied)
4) Optional: run dev regression scripts in VM only.
```