INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-13 18:55 UTC

GemStuffer: RubyGems Packages Weaponized as Dead-Drop Channels Targeting UK Government Infrastructure

THREAT CAMPAIGN | **HIGH** | CVSS 5.0

SCC Item ID	SCC-CAM-2026-0310
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	5.0
Affected Products	RubyGems package ecosystem (rubygems.org registry); UK government public-facing servers; Ruby application dependency chains
Published	2026-05-13T17:09:20
Discovery Source	Rss

Executive Summary

Threat actors published hundreds of malicious Ruby software packages designed to scrape publicly accessible UK government servers, using the RubyGems public registry as a covert data-staging channel rather than attacker-owned infrastructure. Any organization whose development teams use Ruby and pull packages from rubygems.org without strict integrity controls may have introduced malicious code into their own software supply chain. The business risk spans unauthorized data collection from government-adjacent systems, potential backdoor implantation in internal applications, and exposure of sensitive information processed by affected Ruby applications.

Technical Analysis

GemStuffer is an active supply chain campaign in which adversaries registered accounts on rubygems.org and published hundreds of malicious packages embedding scrapers targeting publicly accessible UK government servers. The primary tradecraft element is use of the RubyGems registry as a dead-drop resolver (MITRE T1102, Web Service), staging collected data or relaying commands through a trusted public platform rather than attacker-owned C2 infrastructure. This bypasses egress controls that block unknown external destinations. Attack chain maps to: T1195.001 (Compromise Software Dependencies and Development Tools) for initial supply chain entry; T1059/T1059.004 (Command and Scripting Interpreter: Unix Shell) for execution; T1119 (Automated Collection) for scraping; T1071.001 (Application Layer Protocol: Web Protocols) and T1567 (Exfiltration Over Web Service) for data movement. Relevant CWEs: CWE-829 (Inclusion of Functionality from

Untrusted Control Sphere), packages pulled without namespace or publisher verification; CWE-494 (Download of Code Without Integrity Check), absence of checksum or signature enforcement on gems; CWE-200 (Exposure of Sensitive Information to Unauthorized Actor), scraped government server data exfiltrated covertly. No CVE assigned. No vendor patch available; RubyGems has suspended new account registrations as an emergency containment measure. Attribution remains unconfirmed. The technique is assessed as deliberate and operationally sophisticated; extension to npm, PyPI, or other registries cannot be ruled out.

Action Checklist

1. Containment, Immediately audit Gemfile.lock and Gemfile across all Ruby projects for packages published by unknown or recently created rubygems.org accounts. Cross-reference installed gem names and versions against the rubygems.org registry for anomalous publisher activity. Place any build or CI/CD environments that pulled gems from rubygems.org in the past 90 days under enhanced monitoring pending review; evaluate isolation as a containment option based on operational risk tolerance. Block outbound connections from build systems to rubygems.org until audit completes if operational risk permits.
2. Detection, Search application and network logs for outbound HTTP/S connections from Ruby application servers or CI/CD pipelines to rubygems.org endpoints that do not correspond to standard gem installation events. Look for automated, high-frequency GET requests from application runtime processes (not package managers) to registry URLs, these indicate dead-drop polling or data staging. Review endpoint process logs for unexpected shell spawning from Ruby interpreter processes (ruby, bundle, rake). Correlate against MITRE T1102 and T1567 behavioral patterns: web service used as communication channel, exfiltration disguised as normal web traffic.
3. Eradication, Remove any gems identified as malicious from all environments: uninstall via 'gem uninstall [package_name]', remove from Gemfile and Gemfile.lock, run 'bundle install' from a clean, pinned dependency state. Rotate any credentials, API keys, or tokens accessible to Ruby application processes on affected systems. If scraper activity is confirmed, treat the affected host as compromised and follow full IR eradication procedures including image reimaging where feasible.
4. Recovery, After eradication, verify clean state by re-running dependency audit against a known-good baseline. Confirm no residual scheduled tasks, cron jobs, or background processes are executing gem-related activity outside normal package management. Monitor outbound traffic from Ruby application hosts for 30 days post-remediation for recurrence of dead-drop communication patterns. Validate integrity of data processed by affected applications, particularly any data accessible to UK government-facing services.
5. Post-Incident, This campaign exposes a control gap in dependency integrity enforcement. Implement gem signature verification and enforce checksum validation in all Ruby build pipelines. Adopt a private gem mirror or proxy (e.g., Gemstash, Nexus, Artifactory) so that only vetted packages enter the environment. Apply namespace and publisher verification controls. Map this gap to NIST SP 800-161 (Cyber Supply Chain Risk Management) and update your software supply chain risk assessment. Evaluate whether equivalent gaps exist in other package ecosystems (npm, PyPI, Maven) used in your environment.

IR / Forensic Enrichment

Triage Priority	URGENT

Escalation Criteria	Escalate immediately to CISO and legal counsel if forensic analysis confirms that malicious gems executed during application runtime (not only during build/install) and had read access to data processed by UK government-facing services, as this may trigger breach notification obligations under UK GDPR Article 33 (72-hour supervisory authority notification) and potential Cabinet Office supply chain incident reporting requirements.
Recovery Notes	After eradication, rebuild all affected CI/CD environments from clean base images rather than remediating in place, as GemStuffer gems may have introduced persistence mechanisms via post-install hooks that survive gem uninstall. Monitor all outbound connections from Ruby application runtime processes to any rubygems.org-affiliated infrastructure (including Fastly CDN ranges and `s3.amazonaws.com/production.s3.rubygems.org`) for a minimum of 30 days, treating any such connection outside of explicitly approved `bundle install` windows as an active indicator of compromise. Validate with application owners that no data accessible to UK government-facing service endpoints was exfiltrated by comparing application audit logs against expected data access patterns for the compromise window.
Forensic Artifacts	Gemfile.lock and Gemfile from all affected Ruby projects — primary artifacts establishing which malicious gem versions and publisher accounts were introduced, when they were pinned, and which application components depended on them during the GemStuffer campaign window | CI/CD build logs (Jenkins build console output, GitHub Actions workflow run logs, GitLab CI job traces) covering the past 90 days — establish precise timestamps of `bundle install` executions that pulled malicious rubygems.org packages and identify which pipeline stages and environments are in scope | Malicious gem source files preserved from `$(gem environment gemdir)/gems/-/lib/` — contain the actual scraper logic targeting UK government public endpoints and the dead-drop staging code using rubygems.org as a covert channel, directly evidencing MITRE T1102 and T1567 TTPs | NetFlow or proxy logs filtered for outbound connections to rubygems.org API paths (`/api/v1/`, `/gems/`) originating from Ruby application runtime process PIDs during non-deployment hours — distinguishes legitimate package manager activity from dead-drop polling by the malicious gem during application execution | Application-level audit logs for UK government-facing service endpoints covering the compromise window — required to assess what public-facing data was accessible to the scraper and to scope any downstream breach notification or supply chain incident reporting obligations

Per-Action IR Details

Containment — Immediately audit Gemfile.lock and Gemfile across all Ruby projects for packages published by unknown or recently created rubygems.org accounts. Cross-reference installed gem names and versions against the rubygems.org registry for anomalous publisher activity. Isolate any build or CI/CD environments that pulled gems from rubygems.org in the past 90 days pending review. Block outbound connections from build systems to rubygems.org until audit completes if operational risk permits.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST CM-3 (Configuration Change Control), NIST SA-12 (Supply Chain Protection), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run `bundle exec gem list --local > installed_gems_$(hostname).txt` on every Ruby host to snapshot installed gems. Cross-reference publisher accounts using the rubygems.org API: `curl https://rubygems.org/api/v1/gems/.json | python3 -m json.tool | grep -E 'owners|created_at|version_created_at'` for each installed gem. Flag any gem where the publisher account was created within 180 days of the gem's first publication or where the account owns only 1-3 gems. Use `iptables -A OUTPUT -p tcp --dport 443 -d rubygems.org -j DROP` to block outbound gem registry access on CI/CD hosts pending review. A 2-person team can script the API

checks across all Gemfile.lock entries using `grep -E '^ [a-z]' Gemfile.lock | awk '{print $1}' | xargs -I{} curl -s https://rubygems.org/api/v1/gems/{}.json`.

Evidence: Before isolating CI/CD environments, preserve: (1) full `Gemfile.lock` and `Gemfile` from every affected Ruby project — these are the authoritative record of which gem versions and publishers were introduced; (2) rubygems.org API responses for every installed gem capturing publisher account age, ownership, and version timestamps; (3) CI/CD build logs from the past 90 days showing which `bundle install` or `gem install` commands executed and when, located at build system artifact storage (Jenkins: `$JENKINS_HOME/jobs//builds/*/log`, GitHub Actions: workflow run logs, GitLab CI: job trace artifacts); (4) filesystem snapshot of the Ruby gem installation directory (`gem environment gemdir` to locate, typically `/usr/local/bundle` or `~/.gem/ruby//gems/`) including file hashes of `.gemspec` and `.rb` files within suspicious packages.

Detection — Search application and network logs for outbound HTTP/S connections from Ruby application servers or CI/CD pipelines to rubygems.org endpoints that do not correspond to standard gem installation events. Look for automated, high-frequency GET requests from application runtime processes (not package managers) to registry URLs — these indicate dead-drop polling or data staging. Review endpoint process logs for unexpected shell spawning from Ruby interpreter processes (ruby, bundle, rake). Correlate against MITRE T1102 and T1567 behavioral patterns: web service used as communication channel, exfiltration disguised as normal web traffic.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST IR-5 (Incident Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon on Windows build/app hosts using a config that captures process creation (Event ID 1) and network connections (Event ID 3); filter for `ruby.exe`, `bundle.exe`, or `rake.exe` as parent or initiating processes making outbound port 443 connections to `rubygems.org`, `fastly.net` (RubyGems CDN), or `s3.amazonaws.com/production.s3.rubygems.org`. On Linux, use `auditd` with rules: `auditctl -a always,exit -F arch=b64 -S execve -F exe=/usr/bin/ruby -k rubygems_exec` and `auditctl -a always,exit -F arch=b64 -S connect -F exe=/usr/bin/ruby -k rubygems_net`. For network detection without SIEM, run `tcpdump -i eth0 -w rubygems_capture.pcap 'host rubygems.org or host s3.amazonaws.com' ` during application runtime (not during build/install windows) and analyze with Wireshark filtering for non-`bundle`/`gem` User-Agent strings making GET requests to `/api/v1/` or `/gems/` endpoints. Apply the public Sigma rule for T1102 (Web Service as C2) adapted for Ruby process ancestry.

Evidence: Before concluding detection scope: (1) Web server access logs and application server logs (e.g., Nginx `/var/log/nginx/access.log`, Apache `/var/log/apache2/access.log`) showing inbound requests that could have been scraped by the malicious gem — these reveal what public-facing data was targeted; (2) NetFlow or proxy logs covering the past 90 days filtered for outbound connections from Ruby application runtime processes to `rubygems.org` API endpoints during non-deployment hours (dead-drop polling would occur during application runtime, not `bundle install` windows); (3) Sysmon Event ID 1 (Process Creation) or Linux `auditd` `execve` records showing `ruby` spawning child processes such as `curl`, `wget`, `sh`, or `bash` — GemStuffer-style scrapers may shell out to collect and stage data; (4) DNS query logs for `rubygems.org` subdomains originating from application process PIDs rather than package manager processes, which would indicate runtime registry polling consistent with MITRE T1102 dead-drop behavior.

Eradication — Remove any gems identified as malicious from all environments: uninstall via 'gem uninstall [package_name]', remove from Gemfile and Gemfile.lock, run 'bundle install' from a clean, pinned dependency state. Rotate any credentials, API keys, or tokens accessible to Ruby application processes on affected systems. If scraper activity is confirmed, treat the affected host as compromised and follow full IR eradication procedures including image reimaging where feasible.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST IA-5 (Authenticator Management), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 5.2 (Use Unique

Passwords)

Compensating: For gem removal: `gem uninstall --all --executables --ignore-dependencies` followed by `rm -rf $(gem environment gemdir)/gems/-*` to ensure no residual files remain in the gem install directory. Verify removal with `gem list | grep ` (should return empty). Regenerate a clean `Gemfile.lock` by deleting the existing lockfile and running `bundle install --deployment --frozen` against a private mirror or with `--local` against a pre-vetted gem cache. For credential rotation, identify all environment variables and config files accessible to the Ruby process: inspect `/proc//environ` (Linux) or the application's `.env`, `config/credentials.yml.enc`, and `database.yml` files for secrets that were in-scope during the compromise window. Rotate identified secrets in the upstream secret store (e.g., HashiCorp Vault, AWS Secrets Manager, or manually via the issuing service's API) before bringing applications back online.

Evidence: Before uninstalling malicious gems, forensically preserve: (1) Full contents of the malicious gem directory at `$(gem environment gemdir)/gems/-/` — extract and hash all `.rb` source files to document the scraper logic, data targets, and staging mechanism used against UK government endpoints; (2) The gem's `.gemspec` file which contains publisher metadata, declared dependencies, and any `post_install_message` or executable hooks that may have run on install; (3) Contents of `~/.bundle/config` and the project `.bundle/config` to determine if `BUNDLE_MIRROR` or `BUNDLE_DISABLE_CHECKSUM_VALIDATION` settings were in place that facilitated the compromise; (4) Memory dump or `/proc//maps` snapshot if the Ruby application process is still running with the malicious gem loaded — the scraper may hold credentials or staged data in process memory that will be lost on process termination.

Recovery — After eradication, verify clean state by re-running dependency audit against a known-good baseline. Confirm no residual scheduled tasks, cron jobs, or background processes are executing gem-related activity outside normal package management. Monitor outbound traffic from Ruby application hosts for 30 days post-remediation for recurrence of dead-drop communication patterns. Validate integrity of data processed by affected applications, particularly any data accessible to UK government-facing services.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-10 (System Recovery and Reconstitution), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Verify clean state by running `bundle exec bundle-audit check --update` (requires the `bundler-audit` gem from a trusted source) against the rebuilt `Gemfile.lock` to confirm no known malicious or vulnerable gems are present. Enumerate cron jobs and scheduled tasks for all users on affected hosts: `crontab -l && ls -la /etc/cron* /var/spool/cron/` (Linux) or `schtasks /query /fo LIST /v` (Windows) — filter for any entries invoking `ruby`, `bundle`, `rake`, or `gem` outside of approved CI/CD maintenance windows. For 30-day traffic monitoring without SIEM, configure a cron-driven `tcpdump` capture with daily rotation: `0 0 * * * tcpdump -i eth0 -G 86400 -w /var/log/rubygems_monitor_%Y%m%d.pcap 'host rubygems.org'` and review weekly using Wireshark's `tshark -r -Y 'http.request.method == GET && http.host contains rubygems'`.

Evidence: Before declaring recovery complete: (1) Diff of pre- and post-eradication `gem list --local` output to confirm only approved gems remain; (2) Output of integrity check tool (e.g., `sha256sum` across all files in `$(gem environment gemdir)/gems/`) compared against a baseline generated from a fresh install of the approved `Gemfile.lock` on a clean system — delta files indicate persistence or reinfection; (3) Application log review for any continuation of anomalous outbound requests to `rubygems.org` API endpoints (`/api/v1/gems/`, `/gems/*.gem`) from runtime processes post-remediation, which would indicate a persistence mechanism not yet identified; (4) Review of data logs or audit trails for UK government-facing application endpoints to assess scope of data that may have been scraped and staged during the compromise window — relevant for downstream breach notification assessment.

Post-Incident — This campaign exposes a control gap in dependency integrity enforcement. Implement gem signature verification and enforce checksum validation in all Ruby build pipelines. Adopt a private gem mirror or proxy (e.g., Gemstash, Nexus, Artifactory) so that only vetted packages enter the environment. Apply namespace and publisher verification controls. Map this gap to NIST SP 800-161 (Cyber Supply Chain Risk Management) and update your software supply chain risk assessment. Evaluate whether equivalent gaps exist in other package ecosystems (npm, PyPI, Maven) used in your environment.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SA-12 (Supply Chain Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST RA-3 (Risk Assessment), 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: Deploy Gemstash (open source, RubyGems-maintained) as an internal gem proxy: `gem install gemstash && gemstash setup && gemstash start` — configure all project `Gemfile` sources to point to the internal Gemstash URL and disable direct rubygems.org access from build hosts via firewall rule. Enforce checksum validation in CI/CD by adding `bundle config set --global frozen true` and `bundle config set --global deployment true` to prevent any `Gemfile.lock` modifications during pipeline execution. For npm and PyPI equivalents, deploy Verdaccio (npm proxy) and devpi (PyPI proxy) on the same internal host. Implement a YARA rule scanning new gems introduced to the private mirror for patterns consistent with GemStuffer behavior: HTTP client instantiation targeting government domain patterns (`gov.uk`, `mod.uk`, `nhs.uk`) combined with rubygems.org API write calls within the same gem source file. Add `bundle exec bundle-audit` as a required CI/CD gate that fails the build on any unrecognized gem checksum.

Evidence: For the post-incident review and lessons-learned documentation: (1) Complete timeline reconstructed from CI/CD build logs showing first introduction of each malicious gem into the dependency chain — this establishes the breach window for supply chain risk assessment and potential regulatory reporting; (2) Aggregated rubygems.org API records (preserved during containment) documenting publisher account creation dates, gem publication timestamps, and download counts — quantifies how many of the hundreds of reported GemStuffer packages were actually ingested by the organization; (3) Comparative audit results across npm (`npm audit` + `package-lock.json`), PyPI (`pip-audit` + `requirements.txt`), and Maven (`mvn dependency:tree`) ecosystems to document whether the same publisher-verification gap exists in other package managers used in the environment; (4) Software supply chain risk assessment update artifact documenting the control gap mapped to NIST SP 800-161r1 practice C-SCRM Level 2 controls, with acceptance, remediation, or transfer decision recorded for governance purposes.

Detection Guidance

Primary behavioral indicator: outbound connections to rubygems.org originating from Ruby application runtime processes (not package managers during scheduled build events). In proxy/firewall logs, filter for rubygems.org as destination, exclude known CI/CD build windows, flag any connections from production application servers. In SIEM, correlate process lineage on Ruby hosts: ruby.exe or ruby binary spawning child shell processes (bash, sh, cmd) is anomalous and maps to T1059.004. For endpoint detection, alert on file writes to temp directories by Ruby interpreter processes followed by outbound network connections. If you have DNS logging, watch for high-frequency resolution of rubygems.org or gems.rubygems.org from non-build systems. IOC note: specific malicious package names are not confirmed in available T3 source reporting at the time of this writing. Monitor https://rubygems.org/security-advisories and the RubyGems blog (https://blog.rubygems.org) for the official list of malicious packages and update detection rules as IOCs are published. MITRE technique coverage to validate in your detection stack: T1195.001, T1102, T1119, T1567, T1071.001.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://rubygems.org`	RubyGems public registry used as dead-drop resolver for covert data staging and C2 relay — outbound connections from production Ruby processes (outside scheduled build events) are anomalous	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1059.004** — Unix Shell
- **T1119** — Automated Collection
- **T1102** — Web Service
- **T1567** — Exfiltration Over Web Service
- **T1071.001** — Web Protocols
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-6** — Least Privilege
- **SC-7** — Boundary Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **AC-3** — Access Enforcement
- **SC-28** — Protection of Information at Rest
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A01:2021** — Broken Access Control

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.004	Unix Shell	Execution
T1119	Automated Collection	Collection
T1102	Web Service	Command-And-Control
T1567	Exfiltration Over Web Service	Exfiltration
T1071.001	Web Protocols	Command-And-Control
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Security News	https://www.darkreading.com/application-security/attackers-weaponiz...	T3
RubyGems Suspends New Signups After Hundreds of Malicious ...	https://thehackernews.com/2026/05/rubygems-suspends-new-signups-aft...	T3
Hundreds of Malicious Packages Force RubyGems to Suspend ...	https://www.securityweek.com/hundreds-of-malicious-packages-force-r...	T3
RubyGems has suspended new signups after a major malicious ...	https://x.com/TheHackersNews/status/2054212503991898338	T3

Source	URL	Tier
GemStuffer: Attackers Weaponize RubyGems as a Covert Data Drop ...	https://securityonline.info/gemstuffer-rubygems-data-exfiltration-s...	T3

Generated 2026-05-13 18:55 UTC by TJS Security Command Center