INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-29 18:43 UTC

Phishing Infrastructure Evolves: React-Built Pages and EmailJS Abuse Bypass Traditional Detection

THREAT CAMPAIGN | **MEDIUM** | CVSS 5.0

SCC Item ID	SCC-CAM-2026-0018
Type	Threat Campaign
Severity	MEDIUM
CVSS Base Score	5.0
Affected Products	EmailJS (legitimate transactional email service, abused for exfiltration); React (JavaScript framework used to construct phishing page); end users of any service mimicked by the lure
Published	2026-03-13

Executive Summary

A phishing campaign detected in mid-March 2026 uses React-built credential harvesting pages and routes stolen credentials through EmailJS, a legitimate email delivery service, rather than attacker-controlled infrastructure. This design bypasses two common defensive layers: static content inspection and outbound SMTP/domain blocklists. Any organization whose users receive email lures mimicking trusted services is at risk of credential theft that leaves no trace in standard email security or proxy logs.

Technical Analysis

The campaign deploys a credential harvesting page constructed with React, a mainstream JavaScript front-end framework. Dynamic rendering means the malicious login form is assembled client-side at runtime; static HTML scanners and many email link-inspection sandboxes will not resolve the full DOM and may return a benign result. Captured credentials are exfiltrated via EmailJS (emailjs.com), a legitimate transactional email API used by developers. Outbound requests go to EmailJS service endpoints (api.emailjs.com), which appear in network telemetry as normal SaaS traffic. No attacker-controlled SMTP infrastructure is required, so reputation-based domain blocklists and known-bad IP feeds do not fire. Applicable weaknesses: CWE-319 (Cleartext Transmission of Sensitive Information), credentials traverse a third-party service outside victim visibility; CWE-693 (Protection Mechanism Failure), standard controls are structurally bypassed by design. MITRE ATT&CK coverage: T1566/T1566.002 (Phishing/Spearphishing Link), T1056.003 (Web Portal Capture), T1567/T1102 (Exfiltration Over Web Service/Web Service abuse), T1071.001 (Application Layer Protocol: Web), T1027 (Obfuscated Files or Information), T1598 (Phishing for Information). Lure quality was reported as low; attribution is not established. No CVE is assigned. No patch is applicable, this is an

abuse-of-legitimate-service technique, not a software vulnerability.

Action Checklist

1. Step 1, Immediate: Assess legitimate EmailJS usage (Step 3). If none exists, add api.emailjs.com and *.emailjs.com to your proxy/firewall blocklist. If EmailJS is in use, add to watchlist and flag all outbound POST requests to api.emailjs.com for anomalous volume or timing.
2. Step 2, Detection: Query proxy and DNS logs for outbound requests to emailjs.com and api.emailjs.com originating from endpoints outside your development or product teams. Cross-reference with user-reported phishing submissions from the same timeframe (mid-March 2026 onward). Review email gateway logs for links pointing to React-based single-page app URLs (indicators: URLs loading JS bundles, no server-rendered HTML in link preview scan results).
3. Step 3, Assessment: Determine whether EmailJS is used legitimately anywhere in your environment. Inventory SaaS developer tools (transactional email APIs, form-submission services, webhook relay services) that could be abused for credential exfiltration in the same pattern. Review phishing simulation and awareness metrics to identify user populations with lower detection rates, these are likely higher-risk targets for low-quality lures.
4. Step 4, Communication: Brief the SOC on the detection gap: dynamic React rendering and legitimate-service exfiltration will not surface through standard static analysis or domain reputation feeds. Update phishing reporting guidance to instruct users to submit suspicious links even when they appear to load a blank or broken page (indicative of a React SPA that failed to render). Notify IT and development teams to account for any EmailJS usage in asset inventories.
5. Step 5, Long-term: Extend email security sandbox configuration to execute JavaScript and resolve single-page app content before rendering verdicts, static HTML inspection is insufficient for React-based lures. Develop or tune a detection rule for outbound HTTP POST to transactional email API endpoints (EmailJS, Formspree, Web3Forms, and similar) from non-application hosts. Review acceptable-use policy for third-party developer API services and establish an approval process to distinguish legitimate internal use from shadow integrations that create detection blind spots.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and external IR firm if evidence of credential compromise is discovered (confirmed phishing click-through or password reset anomalies), or if EmailJS abuse has propagated to production systems.
Recovery Notes	Post-containment: (1) Reset credentials for all users in lower-awareness quartile who received phishing lures (per Step 3 inventory). (2) Conduct lessons-learned session with SOC to document detection gaps, update runbooks with JavaScript execution requirements, and integrate React-SPA detection into standard phishing triage checklist. (3) Re-baseline email security sandbox configuration; measure time-to-verdict improvements for React-based content.

Forensic Artifacts	Email gateway logs (.eml files, metadata: sender, subject, recipient, timestamp, URL preview scans, sandbox verdict) | Proxy/firewall egress logs (source IP, destination IP/hostname, port, timestamp, protocol, request/response size) | DNS query logs (source IP, query name, query type A/AAAA, response RR, timestamp) | Browser cache and history (C:\Users*\AppData\Local\Microsoft\Windows\INetCache on Windows; ~/.cache/google-chrome on Linux; timestamps, cached content, referrer URLs) | Windows Event Logs (Event ID 4624 for user logons; Event ID 4720 for new account creation; Event ID 1102 for log clearing) | Application logs (.env files, application.log, debug logs from transactional email API integrations, API call timestamps and parameters) | Cloud IAM audit logs (API key creation, last-used timestamp, associated user/application) | Network packet captures (TLS handshakes to identify SNI, HTTP POST payloads if unencrypted, DNS query-response pairs)

Per-Action IR Details

Step 1, Immediate: Assess legitimate EmailJS usage (Step 3). If none exists, add api.emailjs.com and *.emailjs.com to your proxy/firewall blocklist. If EmailJS is in use, add to watchlist and flag all outbound POST requests to api.emailjs.com for anomalous volume or timing.

NIST Phase: Preparation

Reference: NIST 800-61r3 §2 (Preparation phase: tools, policies, and safeguards)

Controls: NIST 800-53 SC-7 (Boundary Protection), NIST 800-53 CA-7 (Continuous Monitoring), CIS 6.1 (Establish a logging and monitoring program)

Compensating: Without enterprise proxy: (1) Export DNS query logs from your recursive resolver or Windows DNS server (Get-DnsClientCache | Export-CSV on Windows; tcpdump -i any -w capture.pcap 'udp port 53' on Linux). (2) Query netstat -ano on endpoints weekly, filter for established connections to api.emailjs.com (resolve IPs via nslookup api.emailjs.com to establish baseline). (3) Configure Windows Firewall inbound/outbound rules to log blocks: netsh advfirewall set allprofiles logging droppedconnections enable, then parse %windir%\System32\LogFiles\Firewall\pfirewall.log for EmailJS IPs.

Evidence: Before blocking or monitoring: (1) Capture baseline outbound DNS queries for 7 days (dns.log from your recursive resolver or Sysmon Event ID 22 on endpoints). (2) Extract current firewall rules (netsh advfirewall show rule name=all for Windows; iptables-save for Linux). (3) Enumerate all web.config (IIS), appsettings.json (.NET), or .env files on development endpoints for EmailJS API keys (grep -r 'emailjs' /app/ --include='*.json' --include='*.config'). (4) Capture network baseline: netstat -ano -p TCP | findstr 'ESTABLISHED' (Windows) or ss -tanop (Linux) filtered for port 443/80 egress.

Step 2, Detection: Query proxy and DNS logs for outbound requests to emailjs.com and api.emailjs.com originating from endpoints outside your development or product teams. Cross-reference with user-reported phishing submissions from the same timeframe (mid-March 2026 onward). Review email gateway logs for links pointing to React-based single-page app URLs (indicators: URLs loading JS bundles, no server-rendered HTML in link preview scan results).

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 (Detection and analysis)

Controls: NIST 800-53 SI-4 (Information System Monitoring), NIST 800-53 AU-6 (Audit review, analysis, and reporting), CIS 3.10 (Ensure that proper logging is enabled on cloud storage)

Compensating: Without SIEM: (1) Export proxy access logs (access.log from nginx/Apache or PAN-OS if available): grep -i 'emailjs\|formspree\|web3forms' /var/log/proxy.log | awk '{print $1, $4, $7, $9}' (source IP, timestamp, URL, response code). (2) Cross-reference source IPs against Active Directory user login logs (Event ID 4624 on domain controller: Get-EventLog -LogName Security -InstanceId 4624 | Where-Object {$_.TimeGenerated -ge '2026-03-15'} | Select-Object TimeGenerated, Message). (3) Query email gateway for phishing reports: extract sender, subject, recipient, submission timestamp from email archive (exiftool or grep on .eml files). (4) Retrieve URL preview data from

email gateway (Proofpoint, Mimecast, etc. retain this in audit logs). Manually inspect for React bundle indicators: URLs containing '.js?bundle', 'chunk.js', or serving 200 with Content-Type: text/html but loading external .js.

Evidence: Before querying: (1) Preserve proxy logs (typically rotate daily; archive 2026-03-15 to present). (2) Export email gateway phishing submission export (CSV or JSON with full headers, URLs, sender, timestamp). (3) Capture domain controller Security event logs for all 4624 (successful login) events 2026-03-15 onward. (4) Document email gateway URL scanning configuration to determine if JavaScript is being executed during preview (most don't by default). (5) Extract browser cache from at-risk endpoints before users clear it: C:\Users*\AppData\Local\Microsoft\Windows\INetCache (Windows) or ~/.cache/google-chrome (Linux) — preserve directory structure and timestamps.

Step 3, Assessment: Determine whether EmailJS is used legitimately anywhere in your environment. Inventory SaaS developer tools (transactional email APIs, form-submission services, webhook relay services) that could be abused for credential exfiltration in the same pattern. Review phishing simulation and awareness metrics to identify user populations with lower detection rates, these are likely higher-risk targets for low-quality lures.

NIST Phase: Preparation

Reference: NIST 800-61r3 §2.3.1 (Risk assessment and mitigation recommendations)

Controls: NIST 800-53 RA-3 (Risk assessment), NIST 800-53 SA-12 (Supply chain protection), CIS 4.1 (Establish and maintain a secure configuration management process)

Compensating: Without SaaS discovery tools (Netskope, Zscaler): (1) Code audit: search application source repositories for API key usage: git log -p --all -S 'emailjs_public_key' | grep -i commit, then diff against main branch. Use grep recursively across source trees: grep -r 'api\.emailjs\|formspree\|web3forms' . --include='*.js' --include='*.jsx' --include='*.ts' --include='*.tsx' --include='*.json'. (2) Network baseline: packet capture from development endpoints for 48 hours (tcpdump -i any -w dev.pcap 'port 443' on Linux; Wireshark on Windows), then filter for TLS SNI names (tshark -r dev.pcap -Y 'tls.handshake.extensions_server_name' -T fields -e tls.handshake.extensions_server_name | sort -u). (3) Cloud API audit: log in to AWS/Azure/GCP and export all API keys with last-used timestamp (AWS: aws iam list-access-keys, then get-access-key-last-used). (4) Phishing simulation baseline: manually survey training completion records and compare low-score users (bottom quartile) to breach notification list to identify overlap.

Evidence: Before assessment: (1) Export entire application codebase as-of mid-March 2026 (git archive HEAD or full source snapshot). (2) Capture all .env files, secrets manager exports (if accessible), and application configuration from development, staging, and production tiers. (3) Obtain email gateway full message logs (raw .eml or equivalent) for 2026-02-01 to 2026-03-31. (4) Extract phishing simulation metadata: user ID, simulation completion date, click/submission result, training completion status. (5) Preserve IAM audit logs from cloud providers showing API key creation and last-used timestamps.

Step 4, Communication: Brief the SOC on the detection gap: dynamic React rendering and legitimate-service exfiltration will not surface through standard static analysis or domain reputation feeds. Update phishing reporting guidance to instruct users to submit suspicious links even when they appear to load a blank or broken page (indicative of a React SPA that failed to render). Notify IT and development teams to account for any EmailJS usage in asset inventories.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2.7 (Documenting the incident)

Controls: NIST 800-53 IR-4 (Incident handling), NIST 800-53 IR-6 (Incident reporting), CIS 17.1 (Maintain a security awareness program)

Compensating: Without formal IR communication platform: (1) Issue email advisory directly to SOC, IT, and development DLs with subject 'URGENT: Phishing Detection Blind Spot — React SPAs + EmailJS.' Include: samples of phishing URLs (redacted but showing .js bundle indicators), screenshot of broken/blank page rendering, and step-by-step reporting instructions. (2) Update phishing reporting process: provide clickable link or QR code for users to submit suspected phishing directly to security@, with one-line instruction: 'Submit even if page appears blank or broken.' (3) Post critical SOC alert in Slack/Teams pinned to #security-alerts: 'Do not rely on static content inspection or domain reputation for EmailJS phishing; JavaScript execution and human verification required.' (4) Schedule

15-minute all-hands briefing for IT/development covering EmailJS misuse patterns and inventory requirements.

Evidence: Before communication: (1) Confirm no ongoing breach investigation that would require confidentiality (check IR ticket status). (2) Prepare 3-5 sanitized phishing URL examples showing React bundle loading and EmailJS endpoints. (3) Document current phishing reporting submission rate and channels to establish baseline for tracking improved reporting post-advisory.

Step 5, Long-term: Extend email security sandbox configuration to execute JavaScript and resolve single-page app content before rendering verdicts, static HTML inspection is insufficient for React-based lures. Develop or tune a detection rule for outbound HTTP POST to transactional email API endpoints (EmailJS, Formspree, Web3Forms, and similar) from non-application hosts. Review acceptable-use policy for third-party developer API services and establish an approval process to distinguish legitimate internal use from shadow integrations that create detection blind spots.

NIST Phase: Recovery

Reference: NIST 800-61r3 §4.4.1 (Follow-up); NIST 800-61r3 §2.2 (Tools and resources)

Controls: NIST 800-53 SI-3 (Malware protection), NIST 800-53 SI-4 (Information system monitoring), NIST 800-53 AC-2 (Account management), CIS 2.1 (Establish and maintain an inventory of all software), CIS 6.2 (Establish and maintain a baseline configuration for network devices)

Compensating: Without advanced email sandbox (Proofpoint, Mimecast): (1) Manual JavaScript execution: configure cron job (Linux) or scheduled task (Windows) to download URLs flagged by SPF/DKIM/DMARC failures, execute via Node.js or browser headless mode (puppeteer-extra or Playwright), and capture rendered DOM: node -e "const puppeteer = require('puppeteer-extra'); const StealthPlugin = require('puppeteer-extra-plugin-stealth'); puppeteer.use(StealthPlugin()); (async () => { const browser = await puppeteer.launch(); const page = await browser.newPage(); await page.goto(process.argv[1]); console.log(await page.content()); })()" URL 2>&1 | grep -i 'emailjs\|formspree'. (2) Detection rule (Suricata/Snort): http.client_ip !$HOME_NET and http.method == POST and (http.uri contains "emailjs" or http.uri contains "formspree" or http.uri contains "web3forms") and http.response_body contains "200"; alert and log request body. (3) Policy audit: export current SaaS acceptable-use policy from compliance system, add clause: 'All third-party API integrations (transactional email, form handlers, webhooks) require written approval from AppSec; shadow integrations will result in network access revocation.' (4) Inventory process: add quarterly audit task to asset management system requiring development team sign-off on all external API endpoints in use.

Evidence: Before implementing: (1) Baseline current email gateway verdict accuracy: extract 30-day sample of messages flagged as phishing and verify false-positive rate. (2) Document current email security sandbox configuration (what headers/content are scanned, JavaScript execution capability, timeout settings). (3) Capture current acceptable-use policy version and effective date. (4) Obtain list of all approved third-party SaaS integrations from IT/procurement to avoid blocking legitimate services.

Detection Guidance

Primary detection surface is network egress. Query proxy or firewall logs for outbound POST requests to api.emailjs.com, particularly from workstations or servers with no known development function. A single POST to api.emailjs.com/api/v1.0/email/send from an endpoint user machine is anomalous. DNS query logs for emailjs.com from non-developer hosts are a secondary signal. For email gateway telemetry: flag links where the URL resolves to a page that returns JavaScript bundle files (e.g., main.[hash].js, chunk.[hash].js) rather than rendered HTML, this pattern is consistent with React SPAs used as phishing lures and inconsistent with legitimate transactional or marketing pages. Behavioral indicator: users reporting a login page that 'looks off' or loads slowly, then redirects or shows an error, consistent with a credential harvesting SPA completing its POST and then failing gracefully. No confirmed IOCs (domains, IPs, hashes) were published in the primary source material at the time of this report. Monitor the SANS ISC diary entry (isc.sans.edu/diary/32794) for updates if IOCs are released.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`api.emailjs.com`	Legitimate EmailJS API endpoint abused in this campaign for credential exfiltration via HTTP POST. Flag outbound connections from non-developer endpoints.	MEDIUM
DOMAIN	`emailjs.com`	Parent domain of abused transactional email service. DNS queries from workstations warrant review.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1598** — Phishing for Information
- **T1071.001** — Web Protocols
- **T1567** — Exfiltration Over Web Service
- **T1566.002** — Spearphishing Link
- **T1102** — Web Service
- **T1027** — Obfuscated Files or Information
- **T1056.003** — Web Portal Capture
- **T1566** — Phishing

NIST-800-53R5

- **AT-2** — Literacy Training and Awareness
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-8** — Spam Protection
- **CA-7** — Continuous Monitoring
- **SC-8** — Transmission Confidentiality and Integrity
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A02:2021** — Cryptographic Failures

CIS-V8

- **3.10**
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks

- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.312(e)(1)** — Transmission Security
- **164.312(d)** — Person or Entity Authentication
- **164.308(a)(5)(i)** — Security Awareness and Training

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1598	Phishing for Information	Reconnaissance
T1071.001	Web Protocols	Command-And-Control
T1567	Exfiltration Over Web Service	Exfiltration
T1566.002	Spearphishing Link	Initial-Access
T1102	Web Service	Command-And-Control
T1027	Obfuscated Files or Information	Defense-Evasion
T1056.003	Web Portal Capture	Collection
T1566	Phishing	Initial-Access

Sources

Source	URL	Tier
Security News	https://isc.sans.edu/diaryimages/images/26-03-17-phish.png	T1

Source	URL	Tier
A React-based phishing page with credential exfiltration via EmailJS	https://isc.sans.edu/diary/A+Reactbased+phishing+page+with+credenti...	T1
Phishing Alert: React-Based Page Uses EmailJS for Credential Theft	https://cyberpings.com/article/phishing-alert-react-based-page-uses...	T3
A React-based phishing page with credential exfiltration via EmailJS ...	https://www.reddit.com/r/SecOpsDaily/comments/1rshb46/a_reactbased_...	T3
Cybersecurity researchers revealed a phishing campaign abusing ...	https://www.facebook.com/thescorpsec/posts/-cybersecurity-researche...	T3

Generated 2026-03-29 18:43 UTC by TJS Security Command Center