SAMPLE INCIDENT REPORT · CLICKFIX

Sample Incident Report: ClickFix Malware Across 30+ WordPress Sites

This plain-English sample explains why cleaning a serious WordPress infection takes far more than running a security scan. More than 30 WordPress and WooCommerce sites were affected across files, the database, the active theme, and PHP settings.

Prepared by G. Schad, WordPress and Linux security specialist

Published 22 August 2026 · Last reviewed 22 August 2026

ENVIRONMENT	SEVERITY
30+ WordPress and WooCommerce sites	Critical — coordinated multi-site persistence
HIDING PLACES	**STATUS**
Files, database, active theme, and PHP settings	Fleet-wide eradication and retesting required

---

START HERE · PLAIN ENGLISH

The malware was hiding in four places at the same time

More than 30 WordPress and WooCommerce sites contained the same ClickFix-style infection pattern. A visitor might see only one fake verification screen, but the code behind it was spread across several parts of each website.

Some malicious code was hidden in PHP files. More was stored in the WordPress database. Another piece was called from the active theme, while a PHP setting could load a malicious file before WordPress even started.

That is why deleting one suspicious file or clicking “remove” in a scanner is not enough. If any one of the other hiding places survives, it can bring the malicious behavior back.

THE KEY POINT

A security scan can find important clues. A complete cleanup must also remove every persistence path, close the attacker's access, replace untrusted code, rotate credentials, and prove through retesting that the infection is gone.

30+ affected sites

The same infection pattern appeared across WordPress and WooCommerce installations in one operating estate.

Four hiding places

Malicious behavior was connected to PHP files, database values, the active theme, and PHP startup settings.

Critical risk

Affected sites could show visitors a fake verification flow designed to make them run a malicious command.

Entry point not yet proven

The evidence did not justify blaming one plugin, password, hosting account, or administrator.

SCAN VERSUS CLEANUP

A Wordfence scan finds warning signs. A complete cleanup goes much further.

Wordfence is useful for detecting known malicious patterns, changed files, suspicious code, and dangerous content in WordPress options. Those alerts can provide an excellent starting point.

But a scan result is a snapshot, not proof that the whole environment is trustworthy. Running the scanner alone does not preserve evidence, determine how access was obtained, rebuild every affected component, rotate hosting credentials, inspect shared infrastructure, or retest more than 30 sites after cleanup.

This comparison refers specifically to running a scanner by itself. It does not describe a managed incident-response service in which security professionals investigate and clean the environment.

SECURITY SCAN

Finds clues and raises alerts

- Matches files against known malware signatures
- Flags suspicious or modified code
- Can identify dangerous content in WordPress options
- Shows what the scanner can observe at that moment
- Helps an analyst decide where to investigate next

≠

COMPLETE INCIDENT RESPONSE

Restores trust across the environment

- ✓ Confirms which alerts are genuinely malicious
- ✓ Stops delivery while preserving useful evidence
- ✓ Rebuilds files and cleans the database and PHP settings
- ✓ Rotates credentials and closes the attacker's access path
- ✓ Retests every site and monitors for reinfection

A clean scan is useful evidence. It is not, by itself, a guarantee that a compromised WordPress or WooCommerce site is safe again.

VISUAL INFECTION MAP

Four hiding places could feed the same malicious result

The infection was resilient because it did not depend on one file. Each layer offered another way to store, call, or reload the malicious behavior.

01

Obfuscated PHP files

Hidden loaders and encoded server-side logic on disk.

02

WordPress database

Base64-encoded script content stored in options records.

03

Theme functions.php

An unauthorized call ran during normal WordPress page handling.

04

PHP .user.ini

A startup instruction could load a malicious file before WordPress.

↓

ANY SURVIVING LAYER COULD LOAD THE CHAIN AGAIN

RESULT

The fake verification page returns—or another malicious action runs.

WHY THE INFECTION APPEARS TO COME BACK

01

A scan finds one bad file

The visible symptom is identified.

02

Only that file is deleted

The page may look normal for a while.

03

Another layer survives

The database, theme, .user.ini, or attacker access remains.

04

The infection returns

A later request, restart, cache refresh, or deployment loads it again.

TERMINOLOGY AND CLASSIFICATION

ClickFix describes the delivery technique, not one malware family

ClickFix is a social-engineering technique that presents a fake error, CAPTCHA, or verification step and attempts to persuade a person to copy and run a command. In this incident, compromised WordPress sites acted as part of the delivery surface for that visitor-facing behavior.

The report therefore uses “ClickFix-style infection” for the observed website delivery pattern and describes the PHP and database artifacts separately as malicious loaders, injections, and persistence mechanisms. This avoids treating ClickFix as the name of a single server-side payload.

Read Microsoft Threat Intelligence’s ClickFix analysis↗

Review Proofpoint’s ClickFix threat research↗

SCOPE AND LIMITATIONS

What this sanitized report does and does not establish

> EVIDENCE STANDARD
> Confirmed means directly supported by retained artifacts or reproducible behavior. Likely and possible describe reasoned assessment, not proven fact.

AREA	EVIDENCE CONSIDERED	BOUNDARY
Affected website estate	WordPress files, database records, active theme code, PHP configuration, and rendered responses.	Client names, domains, IP addresses, credentials, hashes, and attacker infrastructure are redacted.
Malware execution chain	References between loaders, encoded option values, theme invocation, and PHP startup behavior.	Live payload code and operational command strings are intentionally omitted.
Initial access	Shared components, identities, hosting boundaries, and available logs.	The original entry vector is not claimed without sufficient historical evidence.
Visitor impact	Website-side delivery behavior and potential exposure.	Visitor command execution, endpoint infection, or data theft requires endpoint evidence and was not established by the website review alone.

OBSERVED EXECUTION CHAIN

What happened when an affected website handled a request

01 ### Distributed PHP loaders

Several installations contained unexpected or modified PHP files whose meaningful behavior was concealed through layered encoding, generated strings, and obfuscated control flow.

02 ### Database-hosted script content

Base64-encoded script fragments and related loader data were present in affected WordPress options records, allowing malicious content to persist outside the visible theme and plugin files.

03 **Theme-level invocation**

The active theme’s functions.php contained an automated call connected to the malicious chain, giving it an execution opportunity during normal WordPress request handling.

04 **PHP startup persistence**

A .user.ini directive caused an injected PHP component to load automatically before normal application handling, creating a persistence path outside the ordinary WordPress lifecycle.

05 **Visitor-facing ClickFix behavior**

When the configured conditions matched, the compromised site could present a fake verification or recovery flow intended to persuade a visitor to execute a command on their own device.

06 **Redundant recovery paths**

Because multiple layers could reintroduce or reload the behavior, a single-file cleanup could leave the environment capable of reinfection.

REDACTED VISUAL EVIDENCE

The visitor-facing lure imitated a security verification step

The compromised page presented a familiar-looking verification flow rather than an obvious malware warning. Its purpose was to transfer execution to the visitor by persuading that person to open a system interface and paste a command.

The reconstruction below preserves the observable layout and social-engineering pattern while removing the client domain, command content, attacker infrastructure, and other operational details.

FIGURE 1 · VISITOR-FACING LURE

Sanitized reconstruction—not original client evidence. The domain and command are deliberately redacted; no executable content is included.

> SAFETY RULE
>
> A legitimate website should not ask a visitor to open a Run dialog or terminal and paste an unknown command. Close the page, preserve the URL for your security team, and do not follow the instructions.

CRITICAL FINDING IR-01

Coordinated ClickFix delivery with redundant multi-site persistence

The affected estate should be treated as a coordinated compromise rather than 30 isolated website infections. Repeated artifact structure across sites, multiple execution layers, and the use of configuration outside WordPress all raise the likelihood that the attacker had access to a shared control point, reusable credential, common deployment path, or shared vulnerable component.

That pattern increases both cleanup complexity and reinfection risk. A site cannot be considered restored merely because its public pages stop displaying the malicious behavior at one moment in time.

Severity: Critical

Public delivery of a command-execution lure, arbitrary server-side PHP execution, and fleet-wide persistence create material risk to visitors and site owners.

Affected assets

More than 30 WordPress and WooCommerce sites plus any shared hosting, deployment, identity, or management layer within scope.

Likelihood

High while any loader, encoded option, modified theme file, startup directive, or shared access path remains available.

Potential impact

Visitor malware delivery, site manipulation, credential theft, reinfection, service disruption, reputation damage, and search or browser warnings.

Confidence

High for compromise and observed persistence; moderate for a shared access path; undetermined for the exact initial vector.

Retest status: Required

The finding remains open until every site and the relevant shared control plane pass the defined validation checks.

EVIDENCE MATRIX

Correlated observations that supported the finding

LOCATION	SANITIZED OBSERVATION	SECURITY SIGNIFICANCE	ASSESSMENT
Multiple PHP files	Unexpected or modified files contained high-entropy strings, decoder patterns, and indirect execution logic.	Concealed server-side loaders could execute or reconstruct malicious behavior.	Confirmed
WordPress options table	Affected option values contained base64-encoded script content and related loader material.	The database provided persistence independent of a visible theme or plugin file.	Confirmed
Active theme functions.php	An unauthorized automated call referenced the malicious chain during normal theme execution.	Ordinary front-end requests could activate the loader.	Confirmed
.user.ini	A PHP startup directive referenced an injected component before normal application processing.	The malicious file could load even when WordPress-level hooks were removed.	Confirmed
Rendered website response	Conditional visitor-facing behavior matched a fake verification or recovery flow.	The compromised site could serve as a ClickFix delivery surface.	Confirmed
Initial-access evidence	Available artifacts did not uniquely identify the first successful access path.	Attribution to one plugin, account, or host would be speculative.	Undetermined

---

MITRE ATT&CK MAPPING

Standard technique names make the observed chain easier to compare

The mapping below uses MITRE ATT&CK vocabulary to describe behavior, not to identify a threat actor or claim evidence that was not retained. Technique confidence is stated separately from the underlying incident finding.

INITIAL ACCESS REMAINS UNMAPPED

No initial-access technique is assigned because the available evidence did not prove exploitation of a public-facing application, credential theft, a compromised administrator endpoint, or another specific entry path.

T1204.004 EXECUTION

User Execution: Malicious Copy and Paste

The fake verification flow attempted to persuade a visitor to paste and execute a command. The website-side lure was observed; successful execution on a visitor endpoint was not established.

High-confidence behavioral match

MITRE ATT&CK ↗

T1505.003 PERSISTENCE

Server Software Component: Web Shell

Unexpected PHP components provided server-side execution and persistence. This sub-technique applies only where retained evidence supports a web-accessible backdoor or command interface; an obfuscated loader alone is not automatically a web shell.

Qualified analytical mapping

MITRE ATT&CK ↗

T1546 PERSISTENCE · PRIVILEGE ESCALATION

Event Triggered Execution

Theme request handling and PHP startup configuration created automatic execution opportunities. ATT&CK does not define a PHP auto_prepend_file sub-technique, so the parent technique is used with that limitation stated.

Closest parent-technique mapping

MITRE ATT&CK ↗

T1027 DEFENSE EVASION

Obfuscated Files or Information

Layered encoding, generated strings, and concealed control flow made the PHP and database-hosted content harder to understand and detect.

High-confidence behavioral match

MITRE ATT&CK ↗

TECHNICAL INTERPRETATION

Why the database and .user.ini layers changed the cleanup strategy

Base64 is encoding rather than encryption. Its presence alone is not proof of malware, because legitimate software may store encoded values. Here it became relevant because the decoded structure correlated with unauthorized PHP loaders, theme changes, and visitor-facing behavior across multiple sites.

WordPress stores settings and extension data in its options table, so malicious content placed there can survive replacement of a theme file and can be retrieved during later requests. The database therefore required targeted review and cleanup rather than a file-only malware scan.

PHP can be configured to parse a named file automatically before the requested script. A malicious .user.ini instruction at the site or parent-directory level can therefore run before WordPress, bypass theme-focused cleanup, and affect multiple applications under the same directory hierarchy.

Review the official WordPress Options API documentation↗

Review PHP's auto_prepend_file documentation↗

See Wordfence guidance on encoded files and suspicious WordPress options↗

IMPACT ASSESSMENT

Potential consequences were broader than website defacement

> NOT ESTABLISHED BY THIS EVIDENCE
>
> The report does not claim that payment data was accessed, that visitor devices executed the lure, or that data was exfiltrated. Those outcomes require their own evidence.

Visitor endpoint risk

A successful ClickFix lure can persuade a visitor to run a command that downloads or launches malware on the visitor's workstation.

WooCommerce trust

Fake verification behavior can interrupt shopping journeys, increase abandonment, and undermine confidence in checkout and account pages.

Website integrity

Unauthorized PHP execution can alter content, users, configuration, redirects, scheduled tasks, and other application behavior.

Fleet-wide reinfection

A surviving shared credential, parent configuration, deployment artifact, or common component can reintroduce the infection after local cleanup.

Search and reputation

Malicious behavior may lead to browser warnings, security-vendor blocklisting, search visibility loss, complaints, and brand damage.

Evidence loss

Uncoordinated edits or restores can destroy timestamps and logs needed to determine how access was obtained.

ROOT-CAUSE ASSESSMENT

A common control point was plausible, but the exact entry path remained unproven

A similar infection across more than 30 sites makes a shared cause more plausible than unrelated compromises happening at the same time. That is a prioritization signal, not proof of one specific vector.

The investigation should compare timestamps and access records across the full estate before naming the root cause. Restoring clean files without closing the common access path would leave every site at risk of reinfection.

- ✔ Reused or compromised hosting-panel, SSH, SFTP, deployment, or administrator credentials

- ✔ A vulnerable plugin, theme, management agent, or other component deployed across the fleet
- ✔ Compromise of a shared hosting account, parent directory, automation system, or backup source
- ✔ Unsafe cross-site filesystem permissions or insufficient account isolation
- ✔ A compromised administrator workstation, browser session, API token, or password manager entry
- ✔ An infected deployment package, staging source, or restoration image reused across sites

---

CHECK YOUR OWN SITE · IOC TRIAGE

Use these indicator classes to decide whether you need incident response

This public report does not publish client-specific hashes, filenames, domains, or payloads. Instead, the checks below describe indicator classes that site owners and administrators can compare with their own environment without assuming that every encoded value or changed file is malicious.

Preserve a backup, timestamps, logs, and representative artifacts before changing anything. If a visitor-facing lure is active, restrict public delivery first and investigate from a trusted administrative device. Never paste a command supplied by the suspicious page.

> ESCALATE IMMEDIATELY WHEN
> A lure is still being served, PHP executes outside approved paths, startup configuration loads an unknown component, administrative access cannot be trusted, or the same indicators appear across multiple sites.

SIGNAL	WHERE TO CHECK	WHAT RAISES CONCERN	FIRST SAFE ACTION
Fake verification or error flow	Public pages in a clean browser profile, representative referrers, mobile and desktop views, and response/network captures.	The page asks a visitor to open Run, Terminal, PowerShell, or another system interface and paste a command.	Do not interact with the lure. Capture the URL and screen, restrict delivery, and begin incident handling.
Unexpected PHP	WordPress root, uploads, cache, mu-plugins, theme directories, drop-ins, and recently modified files.	PHP where executable code is not expected, high-entropy strings, indirect function calls, or unexplained changes from trusted packages.	Record paths, timestamps, ownership, and hashes; compare with trusted sources before removal.
Suspicious option values	WordPress options records, especially autoloaded values and records referenced by unfamiliar code.	Long encoded blobs, injected script markup, unknown external resources, or values that correlate with the observed lure.	Export the affected rows and database before decoding or editing; validate context to avoid false positives.
Theme invocation	Active theme functions.php and custom include paths.	Unapproved remote calls, dynamic includes, decoding followed by execution, or references to unfamiliar option names and files.	Diff against the known-good theme and preserve legitimate customizations separately.
PHP startup persistence	Site and parent .user.ini, php.ini, PHP-FPM pool configuration, and effective runtime settings.	Unexpected auto_prepend_file or auto_append_file directives, particularly when they reference hidden or recently created PHP files.	Document the effective configuration and parent inheritance before removing the unauthorized directive.

SIGNAL	WHERE TO CHECK	WHAT RAISES CONCERN	FIRST SAFE ACTION
Identity or scheduled persistence	Administrators, application passwords, sessions, SSH/SFTP keys, control-panel users, WP-Cron, system cron, and deployment automation.	Unknown accounts or keys, unexplained privilege changes, unfamiliar scheduled tasks, or access continuing after password rotation.	Revoke unknown access, rotate affected secrets from a trusted device, and retain the audit trail.
Cross-site repetition	Other sites under the same host, user account, deployment workflow, management agent, or backup source.	Matching artifacts, timestamps, option names, loader structure, or configuration changes across more than one site.	Treat the shared control plane as part of the incident; do not declare sites clean independently.

Use the complete WordPress incident-response scope →

Request a confidential review of the indicators you found →

CONTAINMENT AND ERADICATION

A proper cleanup follows five stages—not one scan-and-delete action

01

Detect

Confirm what is malicious and how widely it appears.

02

Contain

Stop visitors receiving the lure and protect evidence.

03

Clean

Remove malicious files, database values, theme calls, and PHP settings.

04

Close access

Patch the entry path, rotate credentials, and isolate sites.

05

Retest

Prove every site stays clean after normal activity resumes.

01 **Preserve evidence**

Capture timestamps, hashes, logs, database exports, configuration files, and representative malicious artifacts before destructive cleanup.

02 **Contain delivery**

Temporarily restrict affected sites, block confirmed attacker infrastructure where appropriate, and prevent visitors from receiving the malicious flow.

03 **Secure shared access**

Rotate control-panel, SSH, SFTP, database, WordPress, deployment, API, and CDN credentials; revoke sessions and remove unknown keys.

04 **Rebuild trusted application files**

Replace WordPress core, plugins, and themes from verified sources. Do not rely on manually deleting only the visible encoded lines.

05 **Remove database persistence**

Back up the database, identify the unauthorized options records, remove the malicious values in a controlled transaction, and verify application behavior.

06 **Remove startup and theme triggers**

Restore functions.php from a known-good source and remove unauthorized .user.ini directives, including inherited configuration in parent directories.

07 **Close the enabling condition**

Patch or remove vulnerable shared components, correct isolation and permissions, and address the access path supported by the available evidence.

08

Retest the full fleet

Validate each site independently, then monitor the common environment long enough to cover cache expiry, scheduled execution, deployments, and normal traffic.

PER-SITE REMEDIATION CHECKLIST

Each WordPress and WooCommerce installation needs an independent trust decision

- ✔ Preserve a pre-cleanup backup and record timestamps and hashes for representative artifacts.
- ✔ Inventory WordPress core, active and inactive plugins, themes, must-use plugins, drop-ins, uploads, cache directories, and unexpected PHP files.
- ✔ Replace application packages from verified vendor or WordPress.org sources instead of editing suspicious files in place.
- ✔ Review the options table for unauthorized encoded values, suspicious autoloaded records, injected external resources, and deleted-plugin remnants.
- ✔ Restore the active theme’s functions.php and other modified theme files from a trusted source, then reconcile legitimate customizations separately.
- ✔ Review .user.ini, php.ini, .htaccess, web-server configuration, parent directories, scheduled tasks, and PHP process configuration for hidden loaders.
- ✔ Inspect wp-config.php, index files, mu-plugins, uploads, cron events, administrator accounts, application passwords, sessions, and API tokens.
- ✔ Rotate WordPress salts and all credentials whose trust cannot be preserved; avoid reusing one replacement secret across the fleet.
- ✔ Clear WordPress, object, opcode, reverse-proxy, CDN, and browser caches so removed content cannot continue to execute.

- ✔ Apply security updates and correct the verified enabling condition before returning the site to normal service.

FLEET RECOVERY PRIORITIES

Sequence the work by containment risk and shared dependencies

WINDOW	PRIORITY ACTIONS	EXIT CONDITION
0–4 hours	Preserve evidence, restrict malicious delivery, identify shared infrastructure, revoke exposed sessions, and establish one incident owner.	Visitors no longer receive the malicious flow and evidence needed for investigation is retained.
4–24 hours	Rotate shared access, remove all confirmed persistence layers, restore trusted application files, and patch verified urgent exposure.	Every known execution path is removed and access used for recovery is trusted.
2–7 days	Complete per-site validation, review logs and identities, improve isolation, test backups, and monitor for reappearance.	All sites pass the retest matrix and no new indicators appear through normal traffic and scheduled activity.
Within 30 days	Address architectural causes, separate client accounts, harden deployment and secrets management, document response ownership, and conduct a lessons-learned review.	The common control plane has measurable preventive and detective controls.

RETEST PROTOCOL

A clean homepage is not sufficient evidence of recovery

> **SAMPLE STATUS**
>
> Retest remains pending until the checks above pass for every affected site and the shared hosting or management layer. One clean site does not close a fleet-wide finding.

CONTROL	PASS CONDITION	EVIDENCE RETAINED
Filesystem integrity	No unauthorized PHP loaders remain; core and packaged components match trusted sources; approved custom code is reconciled.	Hash comparison, exception list, and reviewed file inventory.
Database integrity	No malicious encoded options, external loader references, or unauthorized autoloaded records remain.	Sanitized before-and-after option inventory and reviewer notes.
PHP startup path	Site and parent .user.ini or related configuration contain no unauthorized prepend or append behavior.	Configuration inventory and effective-runtime verification.
Theme execution	functions.php and related theme entry points contain only approved code and no references to removed loaders.	Trusted-source comparison and custom-change review.
Rendered behavior	Representative public, account, cart, checkout, and administrative routes return expected content without the ClickFix flow.	Controlled browser captures, response review, and network-request summary.
Identity and persistence	No unknown administrators, application passwords, sessions, cron events, deployment keys, or control-panel accounts remain.	Account and scheduled-task review with revocation record.
Observation period	No indicators recur after caches expire, PHP workers restart, cron runs, deployments execute, and normal traffic resumes.	Monitoring log and dated final retest status.

DETECTION AND PREVENTION

Controls should detect both the malicious content and the mechanism that loads it

File integrity monitoring

Baseline application packages and alert on PHP changes in themes, uploads, mu-plugins, root files, and other executable paths.

Database change monitoring

Track suspicious changes to autoloaded options, injected scripts, unfamiliar external domains, and unusually encoded values.

Configuration monitoring

Alert on .user.ini, php.ini, .htaccess, web-server, cron, and PHP pool changes, including parent-directory inheritance.

Account isolation

Use separate system users and credentials per client or site group, least privilege, MFA, and controlled deployment identities.

Response telemetry

Retain control-panel, SSH, SFTP, web, WordPress, WAF, CDN, database, and deployment logs long enough to investigate delayed discovery.

User awareness

Teach staff and customers that a website should never instruct them to open a run dialog or terminal and paste an unknown command.

CONCLUSION

Recovery depends on restoring trust across the whole estate

This incident cannot be closed by deleting one obfuscated file or removing one visible browser script. The filesystem, WordPress database, active theme, PHP startup path, identities, shared infrastructure, and recovery sources all require a documented trust decision.

The most important unresolved question is the common access path. Until it is identified or the plausible shared paths are neutralized, the fleet remains exposed to reinfection even

when individual sites appear clean.

This sample intentionally withholds client identifiers, attacker infrastructure, live payloads, and executable commands. A client report would include controlled evidence references, ownership, timestamps, remediation status, and a signed-off retest record for each affected asset.