FREE CHAPTER PREVIEW

# WORDPRESS MALWARE RECOVERY

A practical, evidence-led field guide

CHAPTER 1

## Is the WordPress Site Actually Hacked?

**MD PABEL**

Founder, 3Zero Digital

FREE PREVIEW

# Before You Read

This is the complete first chapter of a book currently in development. It begins where real investigations begin: with a symptom that may have more than one explanation. The cases are real; the conclusions are limited to what their evidence supports.

> **The same symptom can lead to two opposite conclusions. Evidence is what separates them.**

## WHAT THIS CHAPTER WILL HELP YOU DO

- See why two visitors can receive different versions of one website.
- Watch one blank screen lead to malware and another to an ordinary coding fault.
- Follow browser evidence into the database, user records, logs, and DNS.
- Understand why finding a payload does not automatically reveal the entry point.
- Know when the evidence is strong enough to begin a full incident response.

Selected screenshots are drawn from retained investigations. Public-preview crops remove or exclude client identifiers, database details, and operational payload data. A screenshot proves only what is visible in that frame.

Educational material only. Work only on systems you own or are authorized to examine. Payment, privacy, legal, and regulatory incidents may require qualified specialists.

CHAPTER 1

# Is the WordPress Site Actually Hacked?

Two people can open the same website and see two different things.

The owner opens the homepage on a laptop. It looks normal. A customer opens the same page on a phone and is sent to a spam website. The owner checks again and still sees nothing wrong.

Both people may be telling the truth.

I worked on a case with exactly this symptom. Desktop requests returned a normal `200` response. A request using a mobile browser identity received a `302` redirect. The server was making a decision based on the visitor's device. A malicious rule at the top of `.htaccess` checked for words such as `Android` and `iPhone` in the user-agent string, then redirected matching visitors to an external domain.[1]

The WordPress dashboard did not need to look broken. The desktop homepage did not need to show an error. The redirect happened before the visitor reached the normal page.

This case explains the most important rule in this chapter:

> **A site looking normal to you does not prove that it is clean.**

SAME WEBSITE, DIFFERENT VISITOR CONDITIONS

| Owner on desktop | Customer on phone |
| --- | --- |
| 200 - normal WordPress page | 302 - external redirect |

It also explains why diagnosis cannot depend on one screen or one tool. A browser shows what one visitor received. Page Source shows what the server sent in the main document. The Network panel shows what the page contacted. The database shows stored content. Logs show requests and failures. DNS may decide which server the visitor reaches before WordPress runs at all.

Each view can be useful. None is the whole website.

This chapter will help you decide whether an unusual symptom is an ordinary failure, a credible security warning, or confirmed evidence of compromise. It will not teach the complete cleanup yet. Changing files too early can destroy evidence, break the site, or remove the visible symptom while leaving the attacker's access in place.

## Start with the symptom, not the word “malware”

When somebody says, “My site is hacked,” I first ask what they actually saw.

Did the browser change to another URL? Did the address stay the same while the page changed? Did a warning appear? Was the site blank? Did Google show pages nobody created? Did checkout suddenly ask for card details in an unfamiliar form? Did an administrator disappear?

“Hacked” is a conclusion. The symptom is the observation that lets you test it.

| The site redirects | |
|---|---|
| ORDINARY CAUSE | An intentional rule, wrong site URL, stale cache, or CDN setting |
| SECURITY CAUSE | Malicious `.htaccess`, PHP, JavaScript, database content, DNS, or CDN rule |
| NEXT CHECK | Record the destination and the response chain |

| The page is blank or returns `500` | |
|---|---|
| ORDINARY CAUSE | PHP error, plugin conflict, theme fault, memory limit, or failed update |
| SECURITY CAUSE | Replaced entry file, malicious loader, or attacker-written configuration |
| NEXT CHECK | Check error logs and compare the relevant file with a trusted copy |

| The site returns `403` or a security warning | |
|---|---|
| ORDINARY CAUSE | Firewall rule, permissions problem, or stale reputation record |
| SECURITY CAUSE | Malicious access rule, hostile resource, redirect, or download |
| NEXT CHECK | Record the exact URL and identify which layer blocked it |

| A user, post, or search result is unfamiliar | |
|---|---|
| ORDINARY CAUSE | Forgotten content, migration record, agency account, or indexing error |
| SECURITY CAUSE | Hidden administrator, spam content, or unauthorized publishing |
| NEXT CHECK | Compare the visible screen with an independent source |

The table does not diagnose the site. It stops the first symptom from deciding the entire investigation.

## One blank screen, two opposite conclusions

In one case, the whole WordPress site was blank. The root `index.php` was about 10 KB, far larger than the small, predictable WordPress entry file. Size alone was not proof, but it gave me a reason to open it. Inside was obfuscated PHP, remote-request logic, visitor checks, and special handling for sitemap and robots requests. The normal front door of WordPress had been replaced.[2]

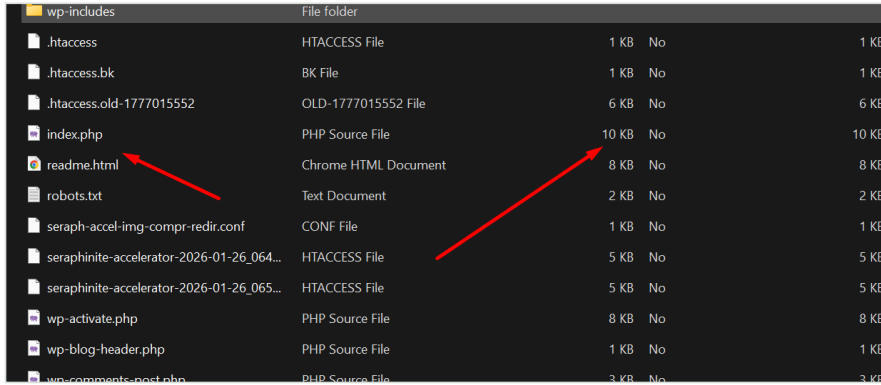



*Figure 1.1. The unusual size was a clue. The difference from a trusted WordPress* `index.php` *and the behavior of the replacement code were the evidence.*

Another client reported an almost identical blank screen. This time the site was not silently serving a malware loader. The error log pointed to the active theme's `functions.php`. A closing `?>` tag sat at the end of the file with blank whitespace after it. That invisible output interfered with WordPress header handling. When the closing tag and trailing whitespace were removed, the frontend and dashboard returned.[3]


*Figure 1.2. This real white-screen case was a coding fault, not proof of malware. WordPress recommends omitting the closing PHP tag at the end of PHP-only files.*

The symptom did not tell us which case we had. Evidence did.

That is why error logs matter before a malware hunt. WordPress can send errors, warnings, and notices to a log with `WP_DEBUG_LOG`; its documentation strongly recommends that errors not be displayed to public visitors.[4]

Debugging configuration should be changed carefully, preferably on a staging or local copy after a backup.

The lesson is not “blank page means malware” or “blank page means plugin conflict.” The lesson is to ask what failed, where it failed, and what evidence would distinguish the possibilities.

# Visible signs and hidden signs

Some attacks are loud. Others are designed to stay outside the owner's normal view.

## Signs a visitor may notice

- The browser opens an unrelated or scam website.
- A fake “Verify you are human,” CAPTCHA, or browser-update prompt appears.
- The site is blank, broken, or returns an unexpected `403` or `500` response.
- A browser, antivirus product, search engine, host, or network filter shows a security warning.
- Search results contain casino, Japanese-keyword, pharmacy, or other unrelated pages.
- Checkout fields, payment behavior, or order confirmation suddenly change.

These symptoms justify investigation. They do not all describe the same incident.

## Signs that may remain hidden

- Only phone users or visitors arriving from search engines are redirected.
- The behavior appears on the first visit but not the second.
- Logged-in administrators see a clean page while logged-out visitors do not.
- Page Source or the Network panel contains an unfamiliar external script.
- The Users screen and the database contain different accounts.
- Google knows about posts that the WordPress dashboard does not show.

- A deleted file, user, or injection returns.
- DNS points visitors to an unexpected server while the expected WordPress files remain clean.

Attackers filter traffic for practical reasons. If the owner does not see the payload, the infection may survive longer. If bots, administrators, or repeat visitors receive a normal page, routine checks may report nothing.

This does not mean every hidden attack is sophisticated. The mobile redirect in the opening case used a short rewrite rule. It survived because the owner tested from the wrong device, not because the code was advanced.

## Rule out ordinary failures without explaining the symptom away

Before searching for malware, record when the problem started and what changed around that time: a WordPress update, plugin release, PHP change, migration, restore, DNS edit, CDN rule, payment integration, deployment, outage, or resource limit.

A legitimate change can explain the symptom. It can also be the moment a security problem became visible. Keep both possibilities open.

Watch the address bar. If the URL changes, record every destination. If it does not change but the page does, consider replacement HTML, an iframe, a wrong document root, `index.html` overriding WordPress, a service worker, or a remote content loader.

Treat status codes the same way. A `403` can come from a legitimate firewall or permissions error; malicious `.htaccess` rules can also block normal PHP while leaving attacker-selected files reachable. A `500` can follow a normal PHP failure, but it can also appear after damaged or hostile configuration. The number is the start of the question, not the answer.

Unfamiliar code deserves the same discipline. `base64_decode`, long strings, minified JavaScript, recent modification times, and PHP in writable locations can be suspicious, but appearance is not enough. Ask what the code does:

- Does it accept unauthenticated input?
- Does it download or execute remote content?
- Does it create users or change privileges?
- Does it hide users, posts, or plugins?
- Does it redirect only selected visitors?
- Does it write or restore executable files?
- Does that behavior match the stated purpose of the component?

Behavior and context are stronger than a frightening function name.

## Check the site as different visitors see it

Testing the homepage once from your normal browser is one of the weakest checks you can perform. Your browser may be logged in, cached, allowlisted, or carrying a cookie that changes the result.

Use a small comparison instead.

| Condition | Compare |
|---|---|
| Login state | Logged in and logged out |
| Device | Desktop browser and a real phone |
| Browser state | Normal window and a private window |
| Network | Usual connection and a second connection such as mobile data |
| Entry path | Direct URL, search result, social link, and a deep page |
| Important page | Homepage, login, product, checkout, or the reported page |

You do not need every combination. Recreate the condition in the report. If a customer says the redirect follows a Google result, use that path. If it affects phones, try a real phone before relying only on browser emulation. If it appears once, start with a browser or device that has not visited the site.

Private browsing reduces stored state, but it is not a perfect clean room. The same IP address, DNS resolver, device traits, or CDN cache may still influence the response.

## Do not interact with a malicious prompt

If the page asks you to copy a command, install a browser update, allow notifications, download a file, or complete a strange CAPTCHA, stop. Capture a screenshot or screen recording if it is safe, but do not follow the prompt.

Use a separate test device or isolated environment for deeper analysis. Do not investigate a suspected download on the computer that holds your hosting, registrar, email, or password-manager access.

## The mobile request changed the case

The opening redirect became evidence when the complaint was reproduced. The case record showed a desktop request returning `200`, a mobile-user-agent request returning `302`, and a rule above the normal WordPress block that selected mobile device words and an external destination.

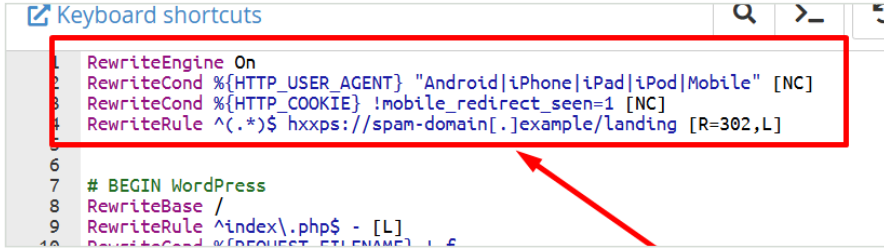



*Figure 1.3. The condition explains why the owner saw a normal site while phone visitors were redirected. Chapter 8 follows this kind of redirect through to its source and removal.*

The status comparison did not tell us how the attacker entered. It answered a smaller and more useful first question: **was the server treating those visitors differently?**

That is stronger than “a customer said it redirected,” but it is still not the whole incident.

# Follow what the browser reveals

The browser is not merely where the symptom appears. It can show how the page was assembled and where the next clue lives.

Start with the main document in the Network panel. Record its status and any redirect chain. Then look for unexpected scripts, iframes, XHR or fetch requests, and WebSocket connections. Chrome's Network tools can show status, resource type, initiator, headers, response, size, and timing.[5]

An unfamiliar domain is not automatically malicious. Modern sites contact analytics services, CDNs, payment providers, fonts, consent tools, chat systems, and advertising networks. The useful question is: **what initiated this request, and does the site have a legitimate reason to make it?**

Page Source and the rendered page answer different questions. “View Page Source” shows the HTML response received for the main document. The Elements panel shows the document after JavaScript has had time to modify it. A block in Page Source was probably generated before or during the response. A clean source followed by a new script in the rendered page points toward runtime execution, an extension, a service worker, or another later source.

## A checkout page that looked normal

During a review of a WooCommerce store, the first important clue appeared in the rendered source: a compact JavaScript block among normal site code. It used encoded values, opened a WebSocket, and could execute JavaScript received from the remote endpoint. Further investigation connected the output to a database setting used by a legitimate header-code feature.[6]

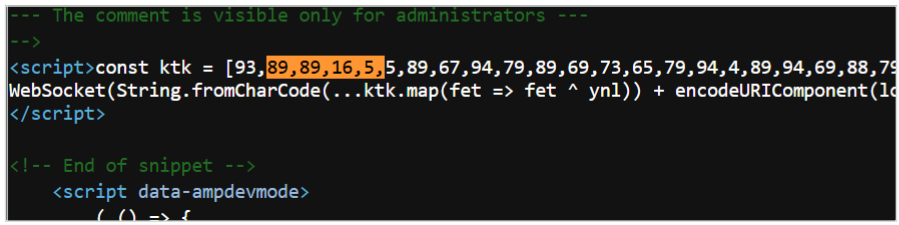



*Figure 1.4. The public preview uses a cropped source view. The code was visible in the browser before its database storage location was known.*

The loader selected checkout-related URLs. That made the case serious: remote code running in a checkout page could manipulate the form, show a false overlay, redirect the customer, or attempt to read fields that existed directly in the page.

But the historical second-stage payload was not captured. The evidence proves a checkout-targeted remote-code capability; it does not prove that card data was actually stolen. That distinction is not caution for its own sake. It is what separates an investigation from a dramatic guess.

It also demonstrates another lesson that typical “signs of a hack” lists miss: the place that stores malware is not necessarily the way the attacker entered. A legitimate custom-code feature can be abused after an administrator account, hosting credential, vulnerable component, or earlier backdoor has already been compromised.

Chapter 12 returns to the checkout and follows the questions this first look cannot answer: what loaded, where it was stored, what it could access, and what evidence would be required before claiming payment-data theft.

## Search engines may show the site you cannot see

Open Google and search for the domain with `site:example.com`. Then review the queries, indexed pages, and Security Issues report in Google Search Console if you control the property.

You are not looking for a large number by itself. A large legitimate archive or store may have many URLs. You are looking for a mismatch between the business and what search engines associate with it: Japanese product titles on an English service site, casino pages on a school website, pharmaceutical terms on a portfolio, or a sudden popular query unrelated to the organization.

In one recovery, Google displayed Japanese product spam under a compromised domain. In another, Search Console showed a surge around an unrelated streaming keyword; the traffic looked like good news until the query was read. A separate site eventually revealed hundreds of casino-titled posts only after the code that hid them from the dashboard was removed.[7]

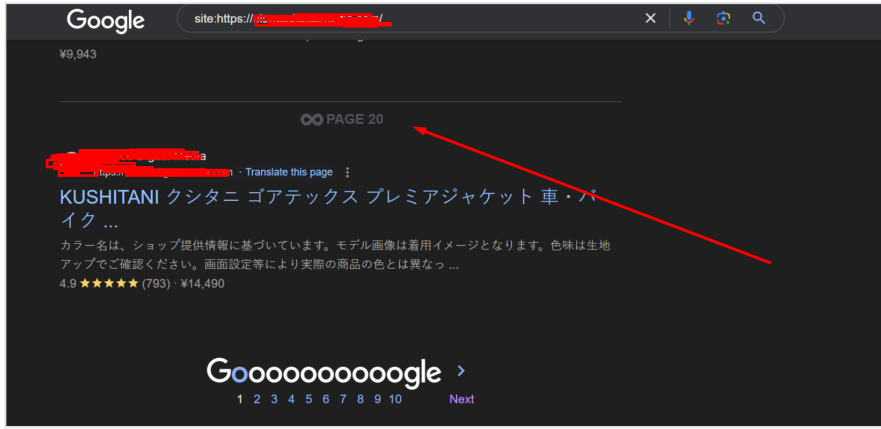



*Figure 1.5. The domain is redacted, but the mismatch is visible. Chapter 9 separates removing the spam from WordPress from removing its remaining URLs from Google.*

Google can preserve evidence after the visible site has changed. It can also show stale pages, old snippets, or legitimate content that the owner forgot. Search results are a lead, not proof of the current mechanism. Record examples before changing the site, then find out whether WordPress still stores or generates them and what response those URLs return now.

## When the numbers do not add up

Sometimes the most valuable clue is not code. It is arithmetic.

In one hidden-administrator case, the built-in WordPress filters reported one user and one administrator. A separate two-factor-authentication counter said there were two inactive accounts. Only one row appeared in the table.

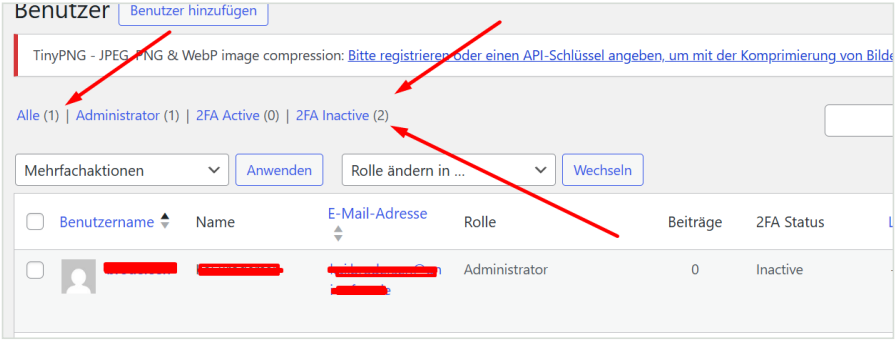



*Figure 1.6. The mismatch did not identify the hidden account, but it proved that the Users screen was not the only view that mattered.*

The database contained another user. Malicious code altered the query and count shown in the dashboard. Once the hiding code was removed, the account became visible.[8]

Removing that account first would have treated the symptom. The code that hid and protected it could have remained. Chapter 10 examines fake plugins and other components before removal; Chapter 11 follows the account into the database and the concealment hooks without trusting the dashboard alone.

The same logic applies to posts and plugins. If WordPress says 978 posts exist but only eleven belong to the owner, or the server contains more plugin folders than the Dashboard lists, the mismatch deserves an independent comparison. The number is not the malware. It is the crack through which you see that one interface is withholding part of the state.

## Let scanners answer a narrow question

A remote scanner can tell you whether it saw a suspicious public response. An internal WordPress scanner may compare files, signatures, repository versions, and content it has permission and configuration to inspect. A hosting scanner may see a different filesystem scope again.

Record the exact URL or path, time, settings when available, and what the report actually found. Test the affected product, post, checkout, or visitor path - not only the homepage.

A clean result means that tool did not report a problem in the view it examined at that time. It does not erase a reproducible redirect, an unfamiliar WebSocket request, impossible user counts, a hostile DNS change, or a customer screenshot. Use the report as evidence, not as the verdict.

# Follow the clue beyond the screen

Once the symptom is reproducible, let it decide where you look next.

If a core file differs from a trusted WordPress copy, preserve its path, modification time, size, owner, and relevant code before replacing it. If a plugin exists on disk but not in the Dashboard, compare the folder and header with the installed-plugin view. If a script appears in Page Source, search for the exact comment, hostname, function, or short distinctive fragment in the database and files.

Ask a specific question instead of searching everything for the word "malware":

- Which component printed this exact script?
- Which record stores this external domain?
- Which rule produced this redirect?
- Which user ID is absent from the dashboard query?
- Which process or request wrote this file again?
- Which DNS record sends visitors away from the expected server?

Specific questions create evidence you can connect.

## Logs can reveal the second event

In one case, a malicious `index.php` returned after removal. The returning file was the visible event. An access log supplied another: a direct request to an unfamiliar PHP file shortly before the change.[9]

That request did not, by itself, prove the entire reinfection chain. It gave the investigation a path, time, requester, and response to compare with file modification times. Chapter 4 shows how to map connected infection

instead of stopping at the first payload; Chapter 7 follows server logs, scheduled work, hosting state, and other control layers.

Error logs answer a different question. They can explain a blank page, plugin failure, missing file, permissions problem, or memory exhaustion. Access logs show what was requested and how the server responded. Neither log should be made to answer a question it does not contain.

### Sometimes the request never reaches WordPress

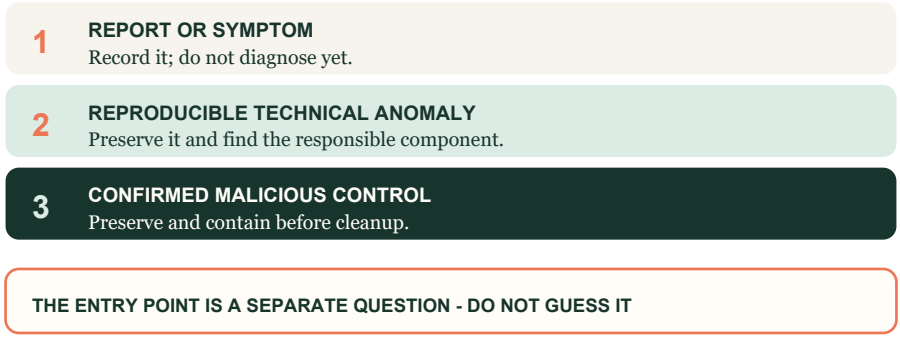
I investigated an e-commerce redirect where the expected WordPress files could be checked repeatedly without finding the public behavior. The visitors were being sent elsewhere by a changed DNS A record. The clean-looking origin and the public response belonged to different paths.[10]

That case changes the diagnostic question from “which WordPress file redirects the visitor?” to this:

> **Is the public request reaching the server I think it is reaching?**

If public DNS, the CDN, and the expected origin do not agree, another WordPress scan cannot resolve the contradiction. Chapter 7 handles the hosting, DNS, and account side of that recovery. Chapter 2 asks the harder question that remains after any payload is found: how did the attacker gain control in the first place?

## Grade the evidence before making a decision




Evidence becomes stronger as you move from a report to a reproducible anomaly and then to confirmed malicious behavior or unauthorized control.

## Level 1: A report or symptom

Examples:

- a customer says the site redirected;
- traffic dropped;
- the host sent a warning;
- the homepage is blank.

Record it. Do not dismiss it, and do not declare malware yet.

## Level 2: A reproducible technical anomaly

Examples:

- a clean phone redirects while desktop does not;
- a browser warning appears for the same URL repeatedly;
- the Network panel shows an unfamiliar request when the pop-up appears;
- the Users screen and account counters disagree;
- Google shows URLs the owner did not create;
- a recently modified file contains visitor-filtering logic.

Preserve the conditions, collect the evidence, and find the component that stores, generates, or applies the behavior.

### Level 3: Confirmed malicious behavior or unauthorized control

Examples:

- `.htaccess` sends selected visitors to an attacker-controlled destination;
- a database value executes code received from an external WebSocket;
- an unauthorized administrator exists in the database;
- a fake plugin hides itself and changes dashboard results;
- an audit log records an unauthorized DNS change.

Treat the site as compromised. Preserve what you need and contain immediate harm before cleanup.

### The entry point remains a separate question

A database record may be malicious without proving who uploaded it. A plugin may deliver a payload without proving that plugin was vulnerable. A hidden administrator may exist without proving which credential or vulnerability created it.

“Unknown” is more useful than a confident story with no evidence.

## Decide what to do next

If the symptom cannot be reproduced and no technical indicator is found, do not declare the site clean forever. Record what was tested, ask for the exact URL, device, time, and screenshot, then try again under the reported conditions.

If the symptom is reproducible but the mechanism is unknown, treat the site as a possible incident. Preserve the current state and avoid random deletion. Expand the investigation according to the symptom: files, database, users, logs, DNS, checkout, or search visibility.

If malicious code, an unauthorized account, or unauthorized traffic control is confirmed, do not submit a blacklist review or announce that the site is clean. Chapter 3 explains the preparation, containment, access, backup, evidence, and rollback decisions that should happen before removal.

Stop DIY work when you do not control the relevant accounts, the site processes payments or sensitive information, evidence suggests credential theft or checkout manipulation, several sites are affected, malware returns, or you do not understand the effect of a command or database change.

Knowing when to stop is part of a safe diagnosis.

## First-response checklist

Before changing the site:

1. Write down the exact symptom, URL, time, device, network, entry path, and login state.
2. Capture a safe screenshot or recording. Do not interact with a strange prompt or download.
3. Reproduce the report from another relevant visitor condition.
4. Record the main response, redirects, unexpected browser requests, and their initiators.
5. Check search results, Search Console, and any browser, antivirus, hosting, or network warning.
6. Use scanner results as supporting evidence and record what each scan actually examined.
7. Follow the clue to the relevant file, database record, user source, log, or DNS layer without deleting it yet.
8. Grade the evidence. If compromise is confirmed or strongly suspected, move to the pre-cleaning process in Chapter 3.

The goal is not to prove that a favorite tool is right or wrong. The goal is to explain what the site served, under which conditions, where that behavior came from, and what the evidence does not yet tell you.

## What this first investigation leaves unanswered

You now know more than a list of “ten signs” can tell you.

You know that two visitors can receive different sites. You know that the same blank screen can point to malware or a harmless coding fault. You know that finding the storage location of a payload does not prove the entry point. You know that a clean scan is only one observation, and that a disagreement between screens can be evidence in its own right.

But the most important questions remain deliberately open:

- How did the attacker get in, and what if that cannot be proven?
- What must be preserved before cleanup begins?
- How do you find connected malware instead of deleting the first file?
- When should WordPress core be replaced, and when must custom code be read?
- What survives in the database after the files look clean?
- What can persist in hosting, scheduled jobs, DNS, or compromised accounts?
- How do you remove a conditional redirect without missing its trigger?
- How do you clean hidden SEO spam and then remove its remaining URLs from Google?
- What did a fake plugin create, hide, or restore before it was removed?
- How do you expose and remove an administrator that WordPress refuses to show?
- What evidence separates a suspicious checkout loader from confirmed card theft?

Those are the investigations the rest of this book follows.

A site can be visibly broken without being hacked. It can also look perfectly normal while selected visitors are redirected, remote JavaScript is loaded, administrators are hidden, spam pages are published, or DNS sends customers to another server.

Good diagnosis keeps both possibilities open until the evidence closes one of them.

# Source notes

**[1]** MD Pabel, “How I Found and Fixed a WordPress Mobile Redirect Hack Using Access Logs,” archived from mdpabel.com, published April 10, 2026. The retained `.htaccess` artifact supports the conditional mobile rule.

**[2]** MD Pabel, “WordPress Site Showing a Blank Page? I Found Malware in index.php,” archived from mdpabel.com, published April 25, 2026.

**[3]** MD Pabel, retained four-page client incident report, “Technical Incident Report: Root Cause Analysis & Photographic Evidence”; WordPress.org, “Custom Functionality (functions.php),” Theme Handbook, https://developer.wordpress.org/themes/core-concepts/custom-functionality/; and “PHP Coding Standards,” https://developer.wordpress.org/coding-standards/wordpress-coding-standards/php/.

**[4]** WordPress.org, “wp-config.php,” Common APIs Handbook, https://developer.wordpress.org/apis/wp-config-php/, and “Debugging in WordPress,” Advanced Administration Handbook, https://developer.wordpress.org/advanced-administration/debug/debug-wordpress/.

**[5]** Chrome for Developers, “Inspect network activity,” https://developer.chrome.com/docs/devtools/network.

**[6]** 3Zero Digital, “WooCommerce Checkout Malware Hidden in the Database: WebSocket Loader Case Study,” published July 28, 2026, https://www.3zerodigital.com/case-studies/woocommerce-checkout-malware-websocket-loader/.

**[7]** MD Pabel, “Remove WordPress SEO Spam and Hacked URLs From Google,” retained case archive; supporting cases include Japanese product spam, unrelated Search Console queries, and hidden casino posts. The visible artifacts support the stated mismatches but do not establish one common writer or entry point.

**[8]** MD Pabel, “How to Find and Remove Hidden Admin Users in WordPress: Malware Analysis,” https://www.mdpabel.com/blog/how-to-find-and-remove-hidden-admin-users-in-wordpress-malware-analysis/, and the retained user-count, database, concealment-code, and after-cleanup artifacts.

**[9]** MD Pabel, “Case Study: Fix Regenerating index.php Malware in WordPress,” archived from mdpabel.com. The retained access-log image supports a direct request to an unfamiliar PHP path; the complete writer chain remains a separate conclusion.

**[10]** MD Pabel, “E-Commerce Site Redirected to Gambling for Two Days: How a Hijacked Cloudflare Account Hides Where Malware Scanners Can't Look,” archived from mdpabel.com, published October 4, 2025. The chapter uses the public-origin mismatch and changed-record narrative without relying on exact timing, revenue, or initial-access claims.

THE BOOK CONTINUES

# The Symptom Was Only the Doorway

The next chapters follow the questions Chapter 1 deliberately leaves open: how access was gained, what must be preserved, how connected infections are found, and how files, databases, hosting, DNS, redirects, SEO spam, fake plugins, hidden administrators, and checkout malware are recovered safely.

> **The goal is not merely to remove a suspicious file. It is to understand what changed, remove every connected control path, and verify that the attacker cannot restore it.**

## Publication updates: mdpabel.com

Written by MD Pabel, founder of 3Zero Digital.

Preview edition - August 2026