TECHNICAL WHITEPAPER: Forensic Analysis, Initial Entry Vector & Remediation of a Multi-Layered WordPress Persistence Campaign

1. Executive Summary

This whitepaper documents the investigation, forensic analysis, initial entry vector determination, and successful remediation of a persistent, multi-layered compromise affecting a WordPress production environment (`eduard.dk`).

The attack vector relied on **cascading redundancy** combined with **anti-forensic defense evasion**: the attacker deployed multiple independent persistence mechanisms across the file system, database, and third-party search engine integrations, while using **timestomping** and **evasive filename formatting** (such as space-injected filenames) to evade casual inspection.

Standard file-level cleanups repeatedly failed because hidden database hooks, automated verification routines, and an undetected stored administrative script continuously regenerated deleted payloads.

By deploying a systematic, ground-up incident response strategy, the initial compromise mechanism, all eight discrete backdoor layers, and associated defense evasion tactics were identified, neutralised, and permanently eradicated.

2. Incident Context & Initial Symptoms

- **Primary Symptoms:** Spurious SEO spam URLs appearing in index reports (e.g., `/bambu-lab//slot-gacor`), unapproved owner accounts registered in Google Search Console, and automated regeneration of Google HTML verification files (`google*.html`) within minutes of deletion.
- **Pre-Breach Anomaly:** A surge of unreadable, abnormally long comment submissions following the silent deactivation/failure of an anti-comment plugin during a rebranding/update failure.
- **Core Challenge:** File-system-only scans provided a false sense of security. Though core files were refreshed, background triggers executed via database options and web shells immediately restored malicious access.
- **Defense Evasion Symptoms:** File dates in directory listings appeared completely normal due to forged metadata, and critical configuration files were disarmed or concealed using non-standard naming schemes (e.g., inserted spaces).

3. Forensic Analysis: Initial Entry Vector

Primary Entry Vector: Stored Cross-Site Scripting (XSS) via Unfiltered Comment Fields

The primary entry vector was an administrative session hijacking chain executed via **Stored Cross-Site Scripting (XSS)** in the WordPress comment system:

1. **Protection System Failure:** A third-party comment-disabling module failed to auto-update due to a plugin rebranding/rename, leading to its silent deactivation and opening default comment handling on the site.
2. **Payload Injection:** Automated spam bots flooded the comment queue with entries containing payload strings hidden inside the `Author` or `URL` fields (e.g., `<script src="../exploit.js"></script>`).
3. **Admin Trigger (Session Hijacking):** When an authenticated administrator logged into `wp-admin` to manually view and purge the spam queue, the browser rendered the unescaped script tags.
4. **Automated Escalation:** Executing under the active administrator's cookie session, the client-side JavaScript silently sent REST API / AJAX calls to:
    1. Inject the site-verification `<meta>` tag into database plugin settings (`ihaf_insert_header`).
    2. Create the stealth user filter (`__ga_hidden_users`) and insert the hidden administrative account (`sys_monitor27546e2d`).
    3. Trigger internal file-editor APIs to write the primary web shell (`Mode Pulut`) and obfuscated engine (`wsOBF.php`).

Secondary Contributing Attack Surface

- **Inactive Plugin Exposure (`Filester`):** Inactive plugins retain executable PHP files on disk. Outdated file managers or utility plugins allowed secondary direct-request execution paths.
- **Backup Probing:** Server logs showed active reconnaissance scanning for legacy backups (e.g., `/wp-config.php.bak`, `/wp-config.php.old`).

4. Forensic Analysis: The 8-Layer Persistence Architecture

The attacker constructed a self-healing operational loop spread across three distinct environments: **File System**, **Database**, and **Search Engine Console**, backed by explicit anti-forensic concealment.

```
        ├──> 1. Web Shell & Payload Engines (Mode Pulut / wsOBF.php)
        ├──> 2. Database Admin Escalation (__ga_hidden_users)
        ├──> 3. Search Console Verification (ihaf_insert_header -> google*.html)
        └──> 4. Evasion Tactics (Timestomping & Space-Injected Filenames)
```

Layer 1: Obfuscated Payload Engine (`wsOBF.php`)

- **Location:** `/wp-content/wsOBF.php`
- **Mechanism:** A 6.5 MB obfuscated script designed to dynamically generate spam content, handle proxy routing, and act as a local storage base for additional malware scripts.

Layer 2: Interactive Web Shell (`Mode Pulut`)

- **Location:** Root directory / subdirectories.
- **Mechanism:** An active PHP web shell allowing remote file navigation, database querying, arbitrary code execution, and privilege elevation directly from a browser interface.

Layer 3: Core File Injections

- **Files Affected:** `cache.php` , `about.php` , `index.php` .
- **Mechanism:** Code snippets injected into root and secondary PHP entry points designed to execute on standard page visits, maintaining backdoor connectivity even if primary shells were removed.

Layer 4: Database Stealth Admin (`__ga_hidden_users`)

- **Database Table:** `www_options`
- **Mechanism:** An option key named `__ga_hidden_users` containing a JSON array (`["sys_monitor27546e2d"]`). This option hooked into WordPress user query filters (`pre_user_query`) to completely hide the rogue administrator account (`sys_monitor27546e2d`) from the WP-Admin UI while keeping the account active in `www_users` .

Layer 5: Database Header Injection (`ihaf_insert_header`)

- **Database Table:** `www_options`
- **Mechanism:** The attacker populated plugin header settings with:
  `<meta name="google-site-verification" content="..." />`
  This ensured that even if verification files were deleted from disk, Google would continuously re-verify the attacker's ownership via header scraping.

Layer 6: Search Console Ownership Hijacking

- **Target:** Google Search Console property (`[https://www.eduard.dk/](https://www.eduard.dk/)`).
- **Mechanism:** The attacker linked `capjaya006@gmail.com` as a verified Owner, allowing them to submit sitemaps, monitor indexing of casino/gambling spam, and abuse search domain authority.

Layer 7: Verification File Auto-Spawning

- **Files Affected:** `google001db6e1772aa97d.html` , `googlec6492041986fe5c0.html` , etc.
- **Mechanism:** When Google or an automated bot checked for property ownership, the active header/database hooks intercepted the request and auto-created the requested `.html` token files back onto the file system in real time.

Layer 8: Expanded Attack Surface (Unused Plugins & Stray Folders)

- **Locations:** `wp-content/shipmondo/` , `wp-content/plugins/filester/` , and 12 inactive plugins.
- **Mechanism:** Unmaintained plugins and non-standard root folders provided unmonitored execution paths for exploit scripts.

⚠️ Anti-Forensic & Defense Evasion Techniques

1. **Timestomping (Timestamp Forgery):**
    1. **Tactics:** Modified system modification times (`mtime`) on malicious scripts to match legitimate historical file dates (e.g., back-dating files to 2017, 2024, or early installation dates).
    2. **Impact:** Standard directory sorting by "Date Modified" failed to highlight newly created or recently modified malicious files, rendering basic visual scanning ineffective.
2. **Evasive Filename Formatting & Space Injections:**
    1. **Tactics:** Utilizing intentional space characters and fake extension suffixes (e.g., `.htaccess bk` instead of `.htaccess`).
    2. **Impact:** Space-injected configuration files were ignored by Apache server processing, silently breaking security rewrite rules without throwing fatal web server errors. Furthermore, filenames designed to look like routine backups or harmless system files distracted administrators from noticing missing active rule sets.

5. Step-by-Step Remediation Framework

To break the re-infection loop and defeat the evasion tactics, remediation was executed in strict logical order: **Isolate -> Purge Files -> Sanitize Database -> Lock Down Server -> Revoke External Access.**

Phase 1: Access Isolation & Credential Rotation

1. **Hosting 2FA Enabled:** Implemented Two-Factor Authentication on the hosting account control panel (One.com).
2. **Database Passwords Updated:** Rotated MySQL credentials to sever unauthorized persistent connections.
3. **WordPress Salts Rotated:** Regenerated all secret keys in `wp-config.php` to immediately invalidate all existing user sessions and cookies.

Phase 2: File System Sanitization & Integrity Overwrite

1. **Core Directory Replacement:** Completely replaced `/wp-admin/` and `/wp-includes/` with pristine files from the official WordPress distribution, bypassing forged file timestamps.
2. **Root Folder Purge:** Removed all unauthorized `.php` files (`cache.php` , `about.php` , `wsOBF.php`) and all rogue web shells (`Mode Pulut`).
3. **Space-Injected File Fixes:** Audited hidden dotfiles, cleaned space-injected filenames (`.htaccess bk` -> `.htaccess`), and verified active rule parsing by Apache.
4. **Content Sanitization:** Inspected `/wp-content/` and removed orphaned/non-standard directories (e.g., loose `shipmondo/` folder), deleted inactive plugins (`filester` , etc.), and cleared unapproved `.html` verification tokens.

Phase 3: Database Forensic Sanitization

Executed targeted SQL queries in phpMyAdmin to eradicate stealth administrative hooks and injected meta tags:

SQL

```sql
/* 1. Delete the stealth user display filter */
DELETE FROM `www_options` WHERE `option_name` = '__ga_hidden_users';

/* 2. Clear injected verification tags from header settings */
UPDATE `www_options` SET `option_value` = '' WHERE `option_name` = 'ihaf_insert_heade

/* 3. Delete rogue administrator accounts */
DELETE FROM `www_users` WHERE `user_login` = 'sys_monitor27546e2d';
```

```
/* 4. Clean orphaned user metadata */
DELETE FROM `www_usermeta` WHERE `user_id` NOT IN (SELECT `ID` FROM `www_users`);
```

Phase 4: Web Server Firewall (`.htaccess`) Hardening

Deployed a hardened `.htaccess` configuration designed to block automated exploit probes, prevent script execution in upload zones, limit administrative access, and manage maintenance routing:

Apache

```
# Force default index to maintenance page during updates
DirectoryIndex index.html index.php

# BEGIN WordPress Standard Routing
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
RewriteBase /

# Allow asset loading for static maintenance screen
RewriteCond %{REQUEST_URI} ^/index\.html$ [OR]
RewriteCond %{REQUEST_URI} ^/images/ [NC]
RewriteRule ^ - [L]

# Preserve administrator dashboard access
RewriteCond %{REQUEST_URI} ^/wp-admin [NC,OR]
RewriteCond %{REQUEST_URI} ^/wp-login\.php [NC]
RewriteRule ^ - [L]

# Route general traffic to static maintenance page
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule . /index.html [L]
</IfModule>
# END WordPress

# SECURITY: Restrict login page to authorized IP
<Files wp-login.php>
    Require all denied
    Require ip [YOUR_MANAGEMENT_IP]
</Files>

# SECURITY: Disable XML-RPC pingback/brute-force vector
<Files xmlrpc.php>
    Require all denied
```

```
</Files>

# SECURITY: Block execution of PHP scripts within Uploads directory
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteCond %{REQUEST_URI} ^/wp-content/uploads/.*\.php$ [NC]
RewriteRule .* - [F,L]
</IfModule>
```

Phase 5: Search Engine Ownership Revocation & SEO Recovery

1. **Removed Unauthorized Users:** Revoked access for `capjaya006@gmail.com` under **Google Search Console -> Settings -> Users and Permissions**.
2. **Removed Unused Ownership Tokens:** Cleared old tokens to prevent automated re-verification.
3. **Resubmitted Clean Sitemaps:** Submitted primary XML sitemap (`/sitemap.xml` / `/wp-sitemap.xml`) to prompt Googlebot to recrawl clean pages and register `404 Not Found` status for deleted spam paths (`/bambu-lab/`).

6. Final Verification & Results

1. **Malware Scan:** A High-Sensitivity **Wordfence Scan** evaluated **27,581 files** and **44,263 URLs**, returning **0 infections / 0 integrity issues**.
2. **File System Stability:** Verified zero re-appearance of `google*.html` verification files over extended monitoring periods.
3. **Search Engine Health:** Search Console reports confirmed property ownership is restricted strictly to legitimate site administrators, with invalid spam URLs cleanly dropping from Google indexation.

7. Key Lessons for System Administrators

- **Beware of Admin Panel Rendering (Stored XSS):** Viewing unmoderated comments, form submissions, or log entries in an admin dashboard can execute malicious scripts in the context of an authenticated admin session. Sanitize inputs and disable public submission systems if unmonitored.
- **Never Trust File Modification Dates (Timestomping):** Attackers easily manipulate file creation and modification timestamps (`mtime`). Always rely on cryptographic hash verification against official checksums rather than file modification dates.
- **Watch for Filename Manipulation:** Inspect filenames carefully for spaces, typos, or double extensions (e.g., `.htaccess bk` or `index .php`). A single inserted space breaks server parsing while concealing malicious or disarmed configurations from automated filters.

- **Deletion is Not Clean-Up:** Deleting a malicious file without checking database options (like `ihaf_insert_header` or `__ga_hidden_users`) leaves the door open for automated script regeneration.
- **Audit Database Query Filters:** Malware frequently uses core WordPress hooks (like `pre_user_query`) saved in `wp_options` to hide rogue admin users from standard UI dashboard lists.
- **Lock Down Upload Directories:** Prevent web server execution of `.php` files inside `/wp-content/uploads/` via web server rules; this neutralizes over 80% of dropped web shell payloads.
- **Minimize Attack Surface:** Inactive plugins are not benign. Unused file managers (e.g., `Filester`), backup scripts, and outdated extensions should be deleted entirely, not merely deactivated.