phpList der zuverlässige Traktor

phpList lässt sich am besten als ein altmodisches, aber unzerstörbares Arbeitstier beschreiben. Es verzichtet bewusst auf modernen Schnickschnack und konzentriert sich rein auf das Wesentliche: den reinen E-Mail-Versand ohne ablenkende Zusatzfunktionen.

## Bedenken (Der "Altmodisch"-Faktor)

- Veraltete UI/UX
- Keine Marketing-Automation

## Vorteile

- Echte Arbeitserleichterung
- Robustes Arbeitstier
- Hervorragendes Rückläufermanagement (Erkennen und Aussortieren von unzustellbaren E-Mails (Bounces) gehört zu den besten am Markt)
- Leichte Installation
- Minimale Serverlast
- Statistiken sind extrem präzise und tiefgehend

# phpList Newsletter-Anmeldung: Einrichtung, Bot-Schutz & Automatisierung

Allgemeine Anleitung auf Basis einer erfolgreich umgesetzten phpList-Installation. Alle domain- und serverspezifischen Angaben sind durch Platzhalter ersetzt.

**Platzhalter in dieser Anleitung:**

- `DEINE-DOMAIN.de` – deine Haupt-Website
- `nl.DEINE-DOMAIN.de` – deine phpList-Installation
- `DEIN-SERVER` – dein Hostname/Server
- `LISTEN_ID` / `SEITEN_ID` – Listen- bzw. Subscribe-Page-ID (unabhängige Werte!)
- `DEIN_DB_…` – deine Datenbank-Zugangsdaten
- `/pfad/zu/private/` – ein Verzeichnis außerhalb des Web-Roots

Die vier Automatisierungs-Skripte werden direkt aus unserem Gitea-Repository synchronisiert (Repo: **phpList-Bot-Schutz**), sodass Änderungen dort immer sofort auch hier im Wiki aktuell sind.

## 1. Ausgangslage

Eigenes, in die Website eingebundenes Anmeldeformular (statt der nativen phpList-Anmeldeseite), das nur die E-Mail-Adresse abfragt und an die phpList-Installation übergibt.

# 2. Funktionierendes Anmeldeformular

## 2.1 Wichtige phpList-Parameter

| Parameter | Bedeutung |
|---|---|
| Listen-ID | Ziel-Liste für Neuanmeldungen (**nicht** identisch mit der Subscribe-Page-ID!) |
| Subscribe-Page-ID | ID der Anmeldeseite (`?p=subscribe&id=SEITEN_ID`) |
| `action` | Muss den kompletten Query-String enthalten – phpList postet intern per `action=„“` auf sich selbst |
| Feldname Liste | Format `list[LISTEN_ID]`, nicht nur `list` |
| Feldname E-Mail | `email` (Standard) |

**Tipp:** Geh in phpList zu Konfiguration → Anmeldeseiten, öffne eine Seite und schau im Reiter „HTML-Formular„ nach dem selbst generierten Code – dort stehen die für deine Installation korrekten IDs.

## 2.2 Formularvorlage

```
<div id="newsletter">
<form name="subscribeform" id="subscribeform" method="post"
action="https://nl.DEINE-DOMAIN.de/lists/?p=subscribe&id=SEITEN_ID"
onsubmit="return forceSubmit();">

    <input name="htmlemail" value="1" type="hidden" />
    <input name="list[LISTEN_ID]" value="signup" type="hidden" />

    <label for="subscriber_email" id="email">E-Mail Adresse</label>
    <input name="email" id="subscriber_email" size="28" style="border: 1px
solid gray; margin-top: 10px;" maxlength="64" type="email" required />
    <br><br>
    <div style="display: none !important; visibility: hidden; position:
absolute; left: -9999px;" aria-hidden="true">
        <label for="honeypot_feld">Bitte dieses Feld leer lassen, falls Sie
es sehen:</label>
        <input type="text" id="honeypot_feld" name="attribute3" value=""
tabindex="-1" autocomplete="off">
    </div>
    <input name="subscribe" class="submit" value="Abonnieren" type="submit"
/>
</form>
</div>
<script type="text/javascript">
function forceSubmit() {
    var honeypot = document.getElementById('honeypot_feld').value;
    if (honeypot !== "") {
        alert("Spam-Verdacht blockiert.");
        return false;
    }
    return true;
```

```
}
</script>
```

### 2.3 Typische Stolpersteine

- **POST wird zu GET:** Fehlt der abschließende Slash in der `action`-URL, löst der Server einen Redirect aus, der POST in GET umwandelt.
- **Verwechslung Listen-ID / Subscribe-Page-ID:** unabhängige Zahlen aus unterschiedlichen phpList-Tabellen.
- **Fehlender Query-String in der Action:** muss `?p=subscribe&id=SEITEN_ID` enthalten.

## 3. Honeypot-Schutz (Formular-Ebene)

- Verstecktes Zusatzfeld (hier: `attribute3`, gemappt auf ein selbst angelegtes phpList-Attribut)
- Per CSS versteckt, von einfachen Bots erkennbar
- JavaScript prüft vor dem Absenden, ob das Feld befüllt ist

**Attribut anlegen:** Konfiguration → Attribute definieren → neues Text-Attribut.

**Auch auf der nativen Anmeldeseite verstecken** (Konfiguration → Anmeldeseiten → Custom-CSS-Bereich):

```
<style>
#attribute3, label[for="attribute3"] {
    display: none !important;
    visibility: hidden !important;
    position: absolute !important;
    left: -9999px !important;
}
</style>
```

**Wichtige Einschränkung:** Ein Teil fortgeschrittener Bots lässt das Honeypot-Feld bewusst leer, um es zu umgehen. Der Honeypot allein reicht daher nicht als alleiniger Schutz (siehe Kapitel 7–9).

## 4. Server-seitiger Zusatzschutz

### 4.1 phpList-Konfiguration (config.php)

Teil der zentralen Konfigurationssammlung, siehe Abschnitt config_snippets.php weiter unten, Block 1.

### 4.2 BotBouncer-Plugin

Installierbar über Konfiguration → Plugins, Quelle: `github.com/bramley/phplist-plugin-botbouncer`

- Prüft E-Mail-Adressen bei jeder Anmeldung gegen Stop Forum Spam
- Wichtige Einstellung: „Whether to validate email address submitted on the subscribe page“ = Ja (gilt für externes und natives Formular)

### 4.3 Hinweis zu Captcha-Plugins

Können zu Redirect-Problemen führen, wenn ein externes Formular ohne Captcha-Token an denselben Endpunkt postet wie die native Anmeldeseite.

## 5. Automatisierte Spam-Meldung an Stop Forum Spam

### 5.1 Konzept

Honeypot-Treffer werden automatisiert mit der echten, protokollierten IP an Stop Forum Spam gemeldet und in phpList gesperrt.

### 5.2 Benötigte phpList-Attribute

| Name (Beispiel) | Typ | Zweck |
|---|---|---|
| Honeypot-Feld | Text | Verstecktes Fangfeld |
| „Als Spam bereits gemeldet„ | Checkbox | Verhindert doppelte Meldung |

**Wichtig:** phpList speichert eine angehakte Checkbox als Wert `on` (nicht `1`).

### 5.3 Relevante Datenbank-Tabellen (phpList 3.x, Standardpräfix phplist_)

| Tabelle | Zweck |
|---|---|
| phplist_user_user | Haupttabelle der Abonnenten |
| phplist_user_user_attribute | Werte aller Attribute (userid, attributeid, value) |
| phplist_user_attribute | Attribut-Definitionen – keine Werte |
| phplist_user_user_history | Protokoll je Abonnent inkl. ip, date, summary |

Struktur vorab per `SHOW TABLES;` / `DESCRIBE tabellenname;` prüfen – kann je nach Version abweichen.

### 5.4 Stop Forum Spam: Vorbereitung

1. Account unter `stopforumspam.com/signup`, API-Key unter „User Panel → Get API Key“
2. Optionen „Public„ und „Reportable“ i. d. R. deaktiviert lassen

### 5.5 Skript: spam_report.php

Außerhalb des Web-Roots ablegen bzw. per Serverregel gegen Web-Zugriff sperren.

```php
<?php
/**
 * spam_report.php
 * Meldet Honeypot-Treffer (verstecktes Formularfeld befuellt) automatisiert
 * an Stop Forum Spam und sperrt den Abonnenten in phpList.
 *
 * Voraussetzungen:
 * - phpList-Attribut "Honeypot" (Text) - ID unten eintragen
 * - phpList-Attribut "Als Spam bereits gemeldet" (Checkbox) - ID unten
eintragen
 * - Stop-Forum-Spam-Account + API-Key
 *
 * Aufruf: per Cron, z. B. taeglich nachts
 */

$dbHost = '127.0.0.1';   // ggf. TCP statt Socket noetig
$dbName = 'DEINE_DB';
$dbUser = 'DEIN_DB_USER';
$dbPass = 'DEIN_DB_PASSWORT';
$sfsApiKey = 'DEIN_STOPFORUMSPAM_API_KEY';

$pdo = new PDO("mysql:host=$dbHost;dbname=$dbName;charset=utf8mb4", $dbUser,
$dbPass);

$attrHoneypotId = X;   // ID deines Honeypot-Attributs
$attrReportedId = Y;   // ID deines "Bereits gemeldet"-Attributs

$sql = "SELECT u.id, u.email
        FROM phplist_user_user u
        JOIN phplist_user_user_attribute uah ON uah.userid = u.id AND
uah.attributeid = :honeypotId AND uah.value != ''
        LEFT JOIN phplist_user_user_attribute uar ON uar.userid = u.id AND
uar.attributeid = :reportedId
        WHERE (uar.value IS NULL OR uar.value = '')";

$stmt = $pdo->prepare($sql);
$stmt->execute(['honeypotId' => $attrHoneypotId, 'reportedId' =>
$attrReportedId]);
$subscribers = $stmt->fetchAll(PDO::FETCH_ASSOC);

function getSubscriberIp($pdo, $userid) {
    $stmt = $pdo->prepare("SELECT ip FROM phplist_user_user_history
                           WHERE userid = :uid AND ip IS NOT NULL AND ip !=
''
                           ORDER BY date ASC LIMIT 1");
    $stmt->execute(['uid' => $userid]);
    return $stmt->fetchColumn();
}

foreach ($subscribers as $sub) {
    $botIp = getSubscriberIp($pdo, $sub['id']);
```

```
    if (!$botIp) {
        echo "Uebersprungen (keine IP in History): {$sub['email']}\n";
        continue;
    }

    $data = http_build_query([
        'email'    => $sub['email'],
        'ip_addr'  => $botIp,
        'username' => $sub['email'],
        'api_key'  => $sfsApiKey,
        'evidence' => 'Automatisch erkannt: Honeypot-Feld auf phpList-
Anmeldeseite befuellt',
    ]);

    $ch = curl_init('https://www.stopforumspam.com/add.php');
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    $response = curl_exec($ch);
    curl_close($ch);

    if (strpos($response, 'success') !== false) {
        $upd = $pdo->prepare("INSERT INTO phplist_user_user_attribute
(userid, attributeid, value)
                              VALUES (:uid, :aid, 'on')
                              ON DUPLICATE KEY UPDATE value = 'on'");
        $upd->execute(['uid' => $sub['id'], 'aid' => $attrReportedId]);

        $blk = $pdo->prepare("UPDATE phplist_user_user SET blacklisted = 1
WHERE id = :uid");
        $blk->execute(['uid' => $sub['id']]);

        echo "Gemeldet & geblacklistet: {$sub['email']} (IP: $botIp)\n";
    } else {
        echo "Fehler bei: {$sub['email']} - $response\n";
    }
    sleep(1);
}
```

2026/08/28 21:09

**Fragen & Feedback**

Haben Sie Fehler gefunden, Verbesserungsvorschläge oder Fragen zu diesem Skript? Hier ein neues Gitea-Issue öffnen

## 5.6 Typische Stolpersteine

| Problem | Ursache | Lösung |
|---|---|---|
| SQLSTATE[HY000] [2002] No such file or directory | PDO versucht Unix-Socket-Verbindung über localhost | ''$dbHost = '127.0.0.1' '' |
| Table 'db.xyz' doesn't exist | Tabellenpräfix nicht berücksichtigt | Präfix per SHOW TABLES; ermitteln |
| Falsche/verwechselte Attribut- bzw. Werte-Tabelle | phpList trennt Definitions- und Werte-Tabellen | Struktur vorher per DESCRIBE prüfen |
| IP Address … cannot be added [XYZ-Infrastruktur] | Kein ip_addr übergeben → SFS nutzt automatisch die anfragende Server-IP, ggf. als geteilte Infrastruktur abgelehnt | Echte Bot-IP aus der History-Tabelle explizit mitschicken |
| SFS verlangt zusätzlich username | API-Anforderung | E-Mail zusätzlich als username mitschicken |
| Checkbox bleibt in Admin-UI leer trotz korrektem DB-Wert | phpList speichert on, nicht 1 | Insert-Wert auf 'on' setzen |

# 6. Bounce-Verarbeitung (unzustellbare Adressen)

## 6.1 Hintergrund

Viele Bot-Anmeldungen nutzen geratene, nicht existierende Adressen bei echten Organisationen. phpLists eingebautes Bounce-Management erkennt das automatisiert über ein IMAP/POP3-Postfach.

## 6.2 Dediziertes Bounce-Postfach

- Eigene Adresse (z. B. `bounces@nl.DEINE-DOMAIN.de`), getrennt vom persönlichen Postfach
- **Kein Spamfilter** (Bounce-Mails werden sonst fälschlich aussortiert, phpList braucht Zugriff auf alle eingehenden Mails)
- POP3 über SSL, „Nach Abruf löschen„ aktiviert, damit das Postfach schlank bleibt

## 6.3 Konfiguration (config.php)

Siehe config_snippets.php, Block 2.

## 6.4 Fehlende native PHP-IMAP-Unterstützung

Ab PHP 8.4 ist die `imap`-Erweiterung nicht mehr standardmäßig im PHP-Kern enthalten. Falls kein Root-Zugriff zur Nachinstallation besteht: Plugin **phplist-plugin-imap2** (`github.com/bramley/phplist-plugin-imap2`) installieren, das IMAP/POP3-Kommunikation in reinem PHP nachbildet.

## 6.5 CLI-Aufruf vs. Remote-Call

Ein direkter CLI-Aufruf von `processbounces.php` kann an mehreren Hürden scheitern:

- Fehlender Init-Kontext (PHPLISTINIT nicht definiert) → Aufruf muss über index.php mit passenden Parametern erfolgen
- Datenbankverbindung im CLI-Kontext (bei eingeschränkten SSH-Umgebungen/Jails sieht die Shell u. U. nicht denselben MySQL-Socket wie der Webserver)
- Fehlende IMAP-Unterstützung (siehe 6.4)

**Praktikabler Weg, falls CLI nicht funktioniert:** Remote-Call über die Weboberfläche (dort läuft PHP-FPM, das über ein IMAP-Plugin funktionierenden Postfachzugriff haben kann, auch wenn CLI-PHP das nicht hat).

### 6.6 Remote Processing Secret einrichten

phpList generiert das Secret standardmäßig bei jedem Aufruf neu, sofern kein fester Wert gesetzt ist – für Cron ungeeignet. Fester Wert nötig:

```
openssl rand -hex 20
```

**Wichtig:** Muss als `$GLOBALS['config'][…]` gesetzt werden, eine einfache Variable wird von `getConfig()` nicht erkannt (siehe config_snippets.php, Block 3).

### 6.7 Remote-Aufruf-URL

```
https://nl.DEINE-DOMAIN.de/lists/admin/?page=processbounces&secret=DEIN_GENERIERTES_SECRET
```

## 7. Erweiterte Bot-Erkennung: „Blitzbestätigung" und Adress-Scan

### 7.1 Beobachtetes Muster

Manche Bots tragen geratene Namen bei echten Firmen-/Behörden-Domains ein, um herauszufinden, welche Adressen dort existieren (Directory-Harvest-Angriff über ein fremdes Formular als Umweg). Erkennbar an:

- Bounce-Mail bei ungültiger Adresse
- Ungewöhnlich schnelle Bestätigung nach der Anmeldung (z. B. 2–18 Sekunden)

### 7.2 Wichtige Lektion: reine Zeitspanne als Kriterium reicht nicht aus

Eine erste Version des Erkennungsscripts filterte ausschließlich nach Zeitspanne (< 120 Sekunden zwischen Anmeldung und Bestätigung). Im praktischen Einsatz erfasste das dadurch fälschlich auch **echte, nur besonders schnell reagierende Menschen** (in einem dokumentierten Fall genügten einem echten Bekannten des Betreibers nur 35 Sekunden).

**Der entscheidende Unterschied:** Bei den fälschlich erfassten echten Personen war die IP-Adresse

bei Anmeldung und Bestätigung identisch. Bei den tatsächlichen Bot-Fällen unterschieden sich die IPs dagegen durchgehend (unterschiedliche Maschinen im Botnetz für Anmeldung und automatisiertes Bestätigen).

## 7.3 Verbessertes Kriterium: kurze Zeit UND unterschiedliche IP

**Benötigtes Attribut:**

| Name (Beispiel) | Typ | Zweck |
|---|---|---|
| „Blitzbestätigung — eher Bots“ | Checkbox | Duplikatschutz |

**Skript: blitzbestaetigung_process.php**

```
<?php
/**
 * blitzbestaetigung_process.php
 * Erkennt Bots, die einen Bestaetigungslink ungewoehnlich schnell nach der
 * Anmeldung anklicken (z. B. automatisiertes Adress-Scanning bei fremden
 * Organisationen). Kriterium: kurze Zeitspanne UND abweichende IP zwischen
 * Anmeldung und Bestaetigung (siehe Wiki-Kapitel 7.2 fuer die Begruendung,
 * warum reine Zeitspanne allein NICHT ausreicht).
 *
 * Verhalten bei Treffer:
 * - Abonnent wird auf "unbestaetigt" zurueckgesetzt
 * - IP wird zur Sperrliste vorgemerkt (Attribut "IP sperren")
 * - NUR die IP wird an Stop Forum Spam gemeldet, NICHT die E-Mail-Adresse
 *   (koennte einem unbeteiligten Dritten gehoeren, siehe Wiki-Kapitel 7.4)
 *
 * Voraussetzungen:
 * - phpList-Attribut "IP sperren" (Checkbox, siehe blocklist_update.php)
 * - phpList-Attribut "Blitzbestaetigung - eher Bots" (Checkbox,
Duplikatschutz)
 *
 * WICHTIG: Vor produktivem Einsatz zunaechst nur mit SELECT testen
 * (siehe Wiki-Kapitel 7.5)!
 *
 * Aufruf: per Cron, z. B. stuendlich
 */

$dbHost = '127.0.0.1';
$dbName = 'DEINE_DB';
$dbUser = 'DEIN_DB_USER';
$dbPass = 'DEIN_DB_PASSWORT';
$sfsApiKey = 'DEIN_STOPFORUMSPAM_API_KEY';

$pdo = new PDO("mysql:host=$dbHost;dbname=$dbName;charset=utf8mb4", $dbUser,
$dbPass);

$attrIpBlockId     = X;   // ID des "IP sperren"-Attributs
```

```
$attrBlitzId       = Y;   // ID des "Blitzbestaetigung"-Attributs
$thresholdSeconds  = 120; // < 2 Minuten zwischen Anmeldung und Bestaetigung
= verdaechtig

$sql = "SELECT u.id, u.email,
               sub.ip AS sub_ip, conf.ip AS conf_ip,
               TIMESTAMPDIFF(SECOND, sub.date, conf.date) AS diff_seconds
        FROM phplist_user_user u
        JOIN phplist_user_user_history sub ON sub.userid = u.id AND
sub.summary IN ('Subscription','Re-Subscription')
        JOIN phplist_user_user_history conf ON conf.userid = u.id AND
conf.summary = 'Confirmation'
        LEFT JOIN phplist_user_user_attribute uab ON uab.userid = u.id AND
uab.attributeid = :blitzId
        WHERE u.confirmed = 1
          AND TIMESTAMPDIFF(SECOND, sub.date, conf.date) BETWEEN 0 AND
:threshold
          AND sub.ip != conf.ip
          AND (uab.value IS NULL OR uab.value = '')
        GROUP BY u.id";

$stmt = $pdo->prepare($sql);
$stmt->execute(['blitzId' => $attrBlitzId, 'threshold' =>
$thresholdSeconds]);
$candidates = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($candidates as $c) {
    // 1. Abonnent auf "unbestaetigt" zuruecksetzen
    $reset = $pdo->prepare("UPDATE phplist_user_user SET confirmed = 0 WHERE
id = :uid");
    $reset->execute(['uid' => $c['id']]);

    // 2. IP zur Sperrliste vormerken (nutzt bestehende Infrastruktur aus
blocklist_update.php)
    $blk = $pdo->prepare("INSERT INTO phplist_user_user_attribute (userid,
attributeid, value)
                          VALUES (:uid, :aid, 'on')
                          ON DUPLICATE KEY UPDATE value = 'on'");
    $blk->execute(['uid' => $c['id'], 'aid' => $attrIpBlockId]);

    // 3. Nur die IP an SFS melden - NICHT die moeglicherweise fremde E-
Mail-Adresse
    $data = http_build_query([
        'ip_addr'  => $c['conf_ip'],
        'username' => 'blitzbestaetigung-bot',
        'api_key'  => $sfsApiKey,
        'evidence' => "Automatisiert erkannt: Bestaetigung nach nur
{$c['diff_seconds']}s von abweichender IP (Anmelde-IP: {$c['sub_ip']})",
    ]);
    $ch = curl_init('https://www.stopforumspam.com/add.php');
    curl_setopt($ch, CURLOPT_POST, true);
```

```
    curl_setopt($ch, CURLOPT_POSTFIELDS, $data);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_exec($ch);
    curl_close($ch);

    // 4. Als bearbeitet markieren (Duplikatschutz)
    $mark = $pdo->prepare("INSERT INTO phplist_user_user_attribute (userid,
attributeid, value)
                           VALUES (:uid, :aid, 'on')
                           ON DUPLICATE KEY UPDATE value = 'on'");
    $mark->execute(['uid' => $c['id'], 'aid' => $attrBlitzId]);

    echo "Zurueckgesetzt & IP vorgemerkt: userid {$c['id']}
({$c['diff_seconds']}s, sub_ip: {$c['sub_ip']}, conf_ip:
{$c['conf_ip']})\n";
    sleep(1);
}
```

2026/08/28 21:09

**Fragen & Feedback**

Haben Sie Fehler gefunden, Verbesserungsvorschläge oder Fragen zu diesem Skript? Hier ein neues Gitea-Issue öffnen

### 7.4 Wichtiger ethischer Punkt: bewusste Nichtmeldung der E-Mail-Adresse

Bei diesem Angriffstyp kann die eingetragene E-Mail-Adresse die eines **unbeteiligten Dritten** sein (Opfer, nicht Täter). Eine Meldung dieser Adresse an eine Spam-Datenbank wäre unfair und sachlich falsch. Deshalb: nur die IP wird gemeldet, nicht die Adresse selbst (siehe Skript, Schritt 3).

### 7.5 Empfehlung: vor produktivem Einsatz zunächst nur per SELECT testen

Bevor ein solches Skript scharf geschaltet wird, empfiehlt sich ein reiner Prüf-Lauf (nur SELECT, keine UPDATE/INSERT-Anweisungen), um zu verifizieren, dass ausschließlich plausible Bot-Fälle erfasst würden — insbesondere nach jeder Anpassung des Erkennungskriteriums.

## 8. Automatisierte IP-Sperrliste

### 8.1 Konzept

Bekannte Bot-IPs (aus Honeypot-Treffern oder manueller Markierung) werden dauerhaft direkt auf Anwendungsebene gesperrt.

## 8.2 Attribute

| Name (Beispiel) | Typ |
|---|---|
| „IP sperren„ | Checkbox (manuelle Markierung) |
| „IP bereits gesperrt“ | Checkbox (Duplikatschutz) |

## 8.3 Skript: blocklist_update.php

```
<?php
/**
 * blocklist_update.php
 * Pflegt eine lokale IP-Sperrliste (PHP-Datei), gespeist aus Honeypot-
Treffern
 * und/oder manueller Markierung ("IP sperren") in phpList.
 * Die Sperrliste wird von config.php bei jedem Seitenaufruf ausgelesen
 * (siehe Wiki-Kapitel 8.4 - Lese-Check).
 *
 * Voraussetzungen:
 * - phpList-Attribut "Honeypot" (Text)
 * - phpList-Attribut "IP sperren" (Checkbox, manuelle Markierung)
 * - phpList-Attribut "IP bereits gesperrt" (Checkbox, Duplikatschutz)
 *
 * Aufruf: per Cron, z. B. alle 15 Minuten
 */

$dbHost = '127.0.0.1';
$dbName = 'DEINE_DB';
$dbUser = 'DEIN_DB_USER';
$dbPass = 'DEIN_DB_PASSWORT';

$pdo = new PDO("mysql:host=$dbHost;dbname=$dbName;charset=utf8mb4", $dbUser,
$dbPass);

$attrHoneypotId = X;
$attrIpBlockId  = Y;
$attrBlockedId  = Z;

$blocklistFile = '/pfad/zu/private/ip_blocklist.php';

$sql = "SELECT DISTINCT u.id
        FROM phplist_user_user u
        LEFT JOIN phplist_user_user_attribute uaip ON uaip.userid = u.id AND
uaip.attributeid = :ipBlockId AND uaip.value = 'on'
        LEFT JOIN phplist_user_user_attribute uah ON uah.userid = u.id AND
uah.attributeid = :honeypotId AND uah.value != ''
        LEFT JOIN phplist_user_user_attribute uab ON uab.userid = u.id AND
uab.attributeid = :blockedId
        WHERE (uaip.value = 'on' OR uah.value IS NOT NULL)
          AND (uab.value IS NULL OR uab.value = '')";
```

```
$stmt = $pdo->prepare($sql);
$stmt->execute(['ipBlockId' => $attrIpBlockId, 'honeypotId' =>
$attrHoneypotId, 'blockedId' => $attrBlockedId]);
$candidates = $stmt->fetchAll(PDO::FETCH_COLUMN);

function getSubscriberIp($pdo, $userid) {
    $stmt = $pdo->prepare("SELECT ip FROM phplist_user_user_history
                           WHERE userid = :uid AND ip IS NOT NULL AND ip !=
''
                           ORDER BY date ASC LIMIT 1");
    $stmt->execute(['uid' => $userid]);
    return $stmt->fetchColumn();
}

$currentBlocklist = file_exists($blocklistFile) ? include $blocklistFile :
[];

$newlyBlocked = [];
foreach ($candidates as $userid) {
    $ip = getSubscriberIp($pdo, $userid);
    if ($ip && !in_array($ip, $currentBlocklist, true)) {
        $currentBlocklist[] = $ip;
        $newlyBlocked[] = $ip;
    }
    $upd = $pdo->prepare("INSERT INTO phplist_user_user_attribute (userid,
attributeid, value)
                          VALUES (:uid, :aid, 'on')
                          ON DUPLICATE KEY UPDATE value = 'on'");
    $upd->execute(['uid' => $userid, 'aid' => $attrBlockedId]);
}

if (!empty($newlyBlocked)) {
    $currentBlocklist = array_values(array_unique($currentBlocklist));
    $content = "<?php\nreturn " . var_export($currentBlocklist, true) .
";\n";
    file_put_contents($blocklistFile, $content, LOCK_EX);
    echo "Neu gesperrt: " . implode(', ', $newlyBlocked) . "\n";
} else {
    echo "Keine neuen IPs zu sperren.\n";
}
```

2026/08/28 21:09

### Fragen & Feedback

Haben Sie Fehler gefunden, Verbesserungsvorschläge oder Fragen zu diesem Skript? Hier ein neues Gitea-Issue öffnen

### 8.4 Lese-Check in config.php

Siehe config_snippets.php, Block 4.

**Wichtig:** Als PHP-Datei statt SQLite/separater Datenbank umgesetzt, da config.php bei jedem Seitenaufruf geladen wird – ein include ist deutlich performanter als eine zusätzliche Datenbankabfrage bei jedem Request.

### 8.5 Grenze dieses Ansatzes

Bei Angriffen mit **ständig wechselnden IPs** gegen dieselbe Zieladresse (siehe Kapitel 9) bremst eine IP-Sperre einzelner Adressen nicht wirksam.

## 9. Subscription-Bombing-Schutz (E-Mail-basiertes Rate-Limiting)

### 9.1 Angriffsmuster

Manche Angreifer reichen dieselbe fremde Zieladresse wiederholt über das Formular ein – jedes Mal von einer anderen IP (rotierendes Proxy-/Botnetz). Zeitliche Abstände können von wenigen Minuten bis zu mehreren Stunden reichen.

**Wirkung:** Das System verschickt bei jeder Einreichung erneut eine Bestätigungsmail an die (oft nicht existierende oder unbeteiligte) Zieladresse – bekannt als „Subscription Bombing„, Missbrauch fremder Formulare, um eine dritte Person mit E-Mails zu belästigen.

**Weder Honeypot noch IP-Sperre greifen zuverlässig:** Honeypot-Feld bleibt leer, IP wechselt bei jedem Versuch.

### 9.2 Lösung: Rate-Limiting pro Zieladresse statt pro IP

Siehe config_snippets.php, Block 5.

**Empfohlene Parameter:** 24-Stunden-Zeitfenster, Blockierung ab dem 3. Versuch derselben Adresse. Ein kürzeres Fenster (z. B. 1 Stunde) reicht bei hartnäckigen Angreifern u. U. nicht aus, da Abstände zwischen Versuchen auch mal 1–2 Stunden betragen können.

**Abwägung:** In seltenen Fällen könnte ein echter Mensch betroffen sein (z. B. mehrfache legitime Neuanmeldung). Das lässt sich durch einen Hinweis auf dem Formular abfedern („Bei Problemen bitte direkt Kontakt aufnehmen“) – das Restrisiko einer versehentlichen Blockade wird gegen den Schutz vor Missbrauch abgewogen.

===== 10. Zentrale Konfigurationssammlung: config_snippets.php ===== config_snippets

Alle oben referenzierten Ergänzungen für die phpList `config.php` an einer Stelle gesammelt (keine eigenständig lauffähige Datei, sondern Referenz zum Übernehmen).

```php
<?php
/**
 * config_snippets.php
 *
 * Sammlung aller Ergaenzungen fuer die phpList config.php.
 * Dies ist KEIN eigenstaendig lauffaehiges Skript, sondern eine
 * Referenzsammlung - die einzelnen Bloecke werden in die produktive
 * config.php uebernommen, nicht diese Datei selbst eingebunden.
 *
 * Reihenfolge in diesem Dokument = empfohlene Reihenfolge beim Einfuegen.
 */


// ============================================================
// 1) Server-seitiger Basis-Spamschutz (Wiki-Kapitel 4.1)
// ============================================================
define('USE_SPAM_BLOCK', 1);
define('NOTIFY_SPAM', 1);


// ============================================================
// 2) Bounce-Postfach-Konfiguration (Wiki-Kapitel 6.3)
// ============================================================
$bounce_mailbox_host = 'DEIN-SERVER';
$bounce_mailbox_user = 'bounces@nl.DEINE-DOMAIN.de';
$bounce_mailbox_password = 'DEIN_PASSWORT';
$bounce_mailbox_port = "995/pop3/ssl/novalidate-cert";
$bounce_mailbox_purge = 1;
$bounce_mailbox_purge_unprocessed = 1;
$bounce_unsubscribe_threshold = 5;

// Wichtig: Envelope-Absender auf die Bounce-Adresse setzen,
// sonst laufen Bounces weiterhin beim persoenlichen Postfach auf
$message_envelope = 'bounces@nl.DEINE-DOMAIN.de';


// ============================================================
// 3) Remote Processing Secret fuer Bounce-Cron (Wiki-Kapitel 6.6)
// Generieren mit: openssl rand -hex 20
// WICHTIG: Muss als $GLOBALS['config'][...] gesetzt werden,
// eine einfache Variable wird von getConfig() NICHT erkannt!
// ============================================================
$GLOBALS['config']['remote_processing_secret'] = 'DEIN_GENERIERTES_SECRET';


// ============================================================
// 4) IP-Sperrliste: Lese-Check (Wiki-Kapitel 8.4)
// Wird von blocklist_update.php befuellt.
// ============================================================
$blocklistFile = '/pfad/zu/private/ip_blocklist.php';
if (file_exists($blocklistFile)) {
```

```
    $blocked_ips = include $blocklistFile;
    $remote_ip = $_SERVER['REMOTE_ADDR'] ?? '';
    if (in_array($remote_ip, $blocked_ips, true)) {
        header('HTTP/1.1 403 Forbidden');
        exit('Access denied.');
    }
}


// ============================================================
// 5) Subscription-Bombing-Schutz (Wiki-Kapitel 9.2)
// Rate-Limiting pro Zieladresse statt pro IP - wirksam gegen
// Angreifer, die dieselbe (oft fremde) Zieladresse ueber
// wechselnde IPs wiederholt einreichen.
// ============================================================
if (isset($_GET['p']) && $_GET['p'] === 'subscribe' &&
!empty($_POST['email'])) {
    $targetEmail = trim($_POST['email']);

    try {
        $bpdo = new PDO(
            "mysql:host=127.0.0.1;dbname=DEINE_DB;charset=utf8mb4",
            'DEIN_DB_USER',
            'DEIN_DB_PASSWORT'
        );
        $stmt = $bpdo->prepare("
            SELECT COUNT(*) FROM phplist_user_user_history h
            JOIN phplist_user_user u ON u.id = h.userid
            WHERE u.email = :email
              AND h.summary IN ('Subscription', 'Re-Subscription')
              AND h.date >= (NOW() - INTERVAL 24 HOUR)
        ");
        $stmt->execute(['email' => $targetEmail]);
        $recentAttempts = (int) $stmt->fetchColumn();

        if ($recentAttempts >= 2) {
            header('HTTP/1.1 429 Too Many Requests');
            exit('Please wait before trying again.');
        }
    } catch (Exception $e) {
        // Bei DB-Fehler nicht blockieren, normal weiterlaufen lassen
    }
}
```

2026/08/28 21:09

### **Fragen & Feedback**

Haben Sie Fehler gefunden, Verbesserungsvorschläge oder Fragen zu dieser Konfigurationssammlung? Hier ein neues Gitea-Issue öffnen

# 11. Cron-Jobs

| # | Zweck | Empfohlene Frequenz |
|---|---|---|
| 1 | Spam-Report an Stop Forum Spam | täglich (z. B. nachts) |
| 2 | IP-Sperrliste aktualisieren | alle 15–30 Minuten |
| 3 | Bounce-Verarbeitung | alle 1–2 Stunden |
| 4 | Blitzbestätigung erkennen & zurücksetzen | stündlich |

**Bei Hosting-Panels mit eigener Cron-Verwaltung (z. B. ISPConfig):**

- PHP-Skripte als „Full"/Shell-Cronjob mit vollem Pfad zum PHP-Interpreter
- Reine URL-Aufrufe (Remote-Call für Bounce-Verarbeitung) ggf. als eigener „URL„-Crontyp, der intern per wget arbeitet – dort keine Shell-Umleitung (», 2>&1) in die URL einbauen, das Logging übernimmt meist das Panel selbst

**Falls direkter SSH-Zugriff auf crontab fehlt** (Berechtigungseinschränkung): Cron-Jobs über die Hosting-Oberfläche einrichten.

# 12. Bewusste Entscheidung: keine automatische Löschung unbestätigter/gesperrter Abonnenten

Unbestätigte, blacklistete oder als Bot erkannte Abonnenten sollten **nicht** automatisiert gelöscht werden, sondern dauerhaft in der Datenbank verbleiben – aus zwei Gründen:

1. **Löschen würde die eigene Abwehr schwächen:** Die Report-, Sperrlisten- und Rate-Limiting-Skripte (Kapitel 5, 8, 9) prüfen jeweils gegen bestehende Zeilen (Duplikat-Attribute, Anzahl bisheriger Anmeldeversuche in der History-Tabelle). Würde ein Datensatz gelöscht, könnte derselbe Bot mit derselben Adresse erneut von vorne beginnen, da sein Zähler zurückgesetzt wäre.
2. **Kein echter Speicherplatzvorteil:** Selbst mehrere tausend zusätzliche Zeilen sind für die Datenbankgröße vernachlässigbar; das Löschrisiko (versehentlich einen echten, nur verspätet reagierenden Interessenten zu treffen) steht in keinem Verhältnis zum Nutzen.

**Getrennt davon zu betrachten:** Eine echte Abmeldung eines Menschen (Sperrliste „nicht mehr kontaktieren") ist rechtlich zulässig und im Interesse der Person dauerhaft aufzubewahren – das ist ein anderer Fall als die hier beschriebene Bot-Bereinigung und wird nicht durch diese Überlegung berührt.

# 13. Monitoring: eigene Listung auf Spam-Blacklists überwachen

## 13.1 Hintergrund

Da automatisierte Skripte selbst Meldungen an Stop Forum Spam absetzen, besteht ein Restrisiko fehlerhafter Selbstmeldung (siehe Kapitel 7.2 — eine erste, noch fehlerhafte Skript-Version hätte

beinahe eine eigene, feste IP-Adresse gemeldet). Zusätzlich können Dritte unabhängig davon eine Meldung auslösen.

### 13.2 Empfohlenes Monitoring (z. B. mit Uptime Kuma oder ähnlichem Tool)

Drei Monitore, HTTP(s)-Keyword-Check mit „Invert Keyword„ (Alarm, sobald das Keyword auftaucht), Intervall z. B. alle 12 Stunden:

| Monitor | URL | Keyword |
|---|---|---|
| Eigene feste IP | https://api.stopforumspam.com/api?ip=DEINE_IP&json | „appears“:1 |
| Server-/Mailversand-IP | https://api.stopforumspam.com/api?ip=SERVER_IP&json | „appears“:1 |
| Allgemeine Blacklist-Prüfung | https://mxtoolbox.com/SuperTool.aspx?action=blacklist%3aSERVER_IP&run=toolpage | Listing-Keyword prüfen, ggf. auf JavaScript-Rendering der Seite achten |

### 13.3 Abwägung: Monitoring der Newsletter-Absenderadresse

Ein zusätzlicher Monitor für die tatsächliche Versandadresse des Newsletters (Schutz gegen Meldung durch Dritte, z. B. via Spam-Beschwerden von Empfängern) ist möglich, aber oft verzichtbar: Ein Blacklist-Monitoring der Server-IP deckt Domain-/Absenderreputation meist bereits mit ab, und das Risiko einer Selbstmeldung durch eigene Skripte besteht bei der Absenderadresse strukturell nicht, sofern diese – wie hier empfohlen – nie an SFS gemeldet wird. Abwägung zwischen zusätzlicher Absicherung und Wartungsaufwand treffen.

## 14. Wichtige Grundsätze

- **Fairness gegenüber Stop Forum Spam:** Nur eindeutig identifizierte Bots melden (klares Kriterium wie ein Honeypot-Treffer), nicht pauschal jede unbekannte Adresse.
- **Niemals ungeprüft die eigene Server-IP als „Spammer“ melden:** IP explizit mitschicken statt der automatischen Erkennung der anfragenden Verbindung zu vertrauen.
- **Bei Adress-Scan-/Bombing-Angriffen:** möglicherweise fremde/unbeteiligte E-Mail-Adressen nicht öffentlich als Spam melden – nur die Infrastruktur (IP) angehen.
- **Vor größeren Software-Updates:** eigenen Datenbank-Dump erstellen, auch wenn die Software selbst automatische Backups anlegt.
- **Eigene Skripte außerhalb des Web-Roots ablegen**, niemals im öffentlich erreichbaren Installationsverzeichnis.
- **Testdaten mit reservierten Dokumentations-Adressen** (z. B. IP nach RFC 5737 wie 203.0.113.1) statt echter eigener oder fremder Adressen anlegen.

---

*Allgemeine Anleitung, erstellt auf Basis einer erfolgreich umgesetzten phpList-Konfiguration. Skripte werden live aus Gitea (Repo: phpList-Bot-Schutz) synchronisiert.*