Technischer Forschungsbericht: Tiefenanalyse des Honey-Scan Honeypot-Überwachungssystems

1. Executive Summary

1.1 Berichtsumfang und Zielsetzung

Dieser umfassende Forschungsbericht bietet eine erschöpfende technische Analyse des Systems **Honey-Scan** (referenziert im Kontext des GitHub-Repositories derlemue/honey-scan, technisch basierend auf dem Upstream-Projekt husak/honeyscan). Das System stellt eine spezialisierte Lösung im Bereich der Cybersicherheit dar, die sich auf die **flussbasierte Überwachung von Honeynets** konzentriert. Im Gegensatz zu herkömmlichen hostbasierten Intrusion Detection Systems (HIDS), die auf den Endgeräten operieren, oder Deep Packet Inspection (DPI) Systemen, die vollständige Payloads analysieren, arbeitet Honey-Scan auf der Netzwerkschicht. Es nutzt NetFlow-Metadaten, um bösartigen Verkehr, der an dedizierte Fallen (Honeypots) gerichtet ist, zu identifizieren, zu korrelieren und zu melden.

Die Analyse dekonstruiert das System in seine atomaren Komponenten: die Datenaufnahme via NfSen, die algorithmische Verarbeitung im Perl-Backend, die Alarmgenerierung und die Datenvisualisierung im PHP-Frontend. Ein besonderer Fokus liegt auf der Rekonstruktion der Datenflüsse, der Identifikation von Trigger-Mechanismen und der Einbettung in die breitere Sicherheitsinfrastruktur (z.B. Warden/SABU).

Hinweis zur Provenienz: Da das spezifische Repository derlemue/honey-scan zum Zeitpunkt der Analyse nicht zugänglich war, stützt sich dieser Bericht auf eine tiefgreifende Untersuchung des Original-Codes von Martin Husák (husak/honeyscan), der als technische Basis dient. Die Analyse berücksichtigt dabei den Kontext von Anwendern wie derlemue (vermutlich im Bereich Homelab/Self-Hosting aktiv, wie durch Querbezüge zu Projekten wie mailcow und Reddit-Aktivitäten in r/selfhosted nahegelegt wird).[1]

1.2 Systemüberblick 🛡️

Honey-Scan ist kein eigenständiges Binary, sondern ein komplexes Plugin für den **NfSen (NetFlow Sensor)** Collector. Es transformiert rohe Verkehrsflussdaten in handlungsrelevante Bedrohungsinformationen (Threat Intelligence).

Hauptmerkmal	Beschreibung

Kernfunktion	Passive Überwachung von Honeypots mittels NetFlow (v5/v9/IPFIX).
Erkennungsziele	• Netzwerk-Scanning (Reconnaissance) • Wörterbuch-Angriffe (Brute-Force) • Interne Kompromittierung (Lateral Movement)
Architektur	• Backend: Perl (Datenverarbeitung, Alarmierung) • Frontend: PHP (Visualisierung, Konfiguration)
Integration	• Input: nfdump/NfSen • Output: E-Mail, RRDtool, IDEA (Intrusion Detection Extensible Alert)

Die Relevanz dieses Systems liegt in seiner Fähigkeit, große Datenmengen effizient zu filtern. Während ein Honeypot selbst Tausende von Login-Versuchen protokolliert, abstrahiert Honey-Scan diese Ereignisse auf die Netzwerkebene und ermöglicht so die Erkennung von Mustern, die dem einzelnen Host verborgen bleiben (z.B. ein verteilter Scan über mehrere Honeypots hinweg).

---

2. Technische Grundlagen und Systemarchitektur

Um die Funktionsweise von Honey-Scan tiefgründig zu verstehen, ist eine detaillierte Betrachtung der Laufzeitumgebung und der zugrundeliegenden Technologien unerlässlich. Honey-Scan existiert nicht im Vakuum, sondern ist symbiotisch mit dem NfSen-Ökosystem verbunden.

2.1 Das NfSen/Nfdump Ökosystem

NfSen ist das grafische Frontend für die **nfdump**-Tools. Es fungiert als das "Betriebssystem", in dem Honey-Scan als "Anwendung" läuft. Diese Architektur diktiert die Datenstrukturen und Zeitintervalle, mit denen Honey-Scan arbeiten muss.

2.1.1 Der Datenfluss vor Honey-Scan

Bevor Honey-Scan überhaupt aktiv wird, durchlaufen die Daten eine Kette von Vorverarbeitungsschritten:

1. **Export:** Router und Switches im Netzwerk exportieren Metadaten über Verbindungen (Flows) via UDP (Port 9995). Ein Flow besteht aus Quell-IP, Ziel-IP, Ports, Protokoll, Byte-Anzahl und Zeitstempeln.[4]
2. **Collection (nfcapd):** Der Daemon nfcapd lauscht auf diese Pakete. Er puffert sie im Speicher und schreibt sie in binäre Dateien auf die Festplatte.
3. **Rotation:** Standardmäßig rotiert nfcapd diese Dateien alle 5 Minuten. Eine Datei hat typischerweise das Format nfcapd.YYYYMMDDHHMM.
4. **Trigger:** Nach erfolgreicher Rotation feuert NfSen interne Events ab. Hier hakt sich Honey-Scan ein.

2.1.2 Die Rolle des Plugins

In der NfSen-Architektur wird Honey-Scan als **Backend-Plugin** registriert. Dies bedeutet:

- Es muss dem strengen Namensschema und der Verzeichnisstruktur von NfSen folgen ($BACKEND_PLUGINDIR/honeyscan).
- Es muss spezifische Perl-Subroutinen (run, Init, Cleanup) implementieren.
- Es unterliegt den Zeitbeschränkungen des Collectors: Wenn die Verarbeitung länger als 5 Minuten dauert, drohen Datenverlust oder Race Conditions.

2.2 Detailliertes Architekturdiagramm 🏗️

Das folgende Diagramm visualisiert die Positionierung von Honey-Scan innerhalb der komplexen Netzwerkinfrastruktur. Es zeigt den Weg vom Angreiferpaket bis zum Alarm im Dashboard.

Code-Snippet

```
graph TD
  subgraph "Bedrohungslandschaft (Internet)"
    Attacker[👹 Angreifer]
    Botnet
    Scanner
  end

  subgraph "Netzwerk-Perimeter & Sensoren"
    Firewall
    Honeypot_Low
    Honeypot_High[🍯 High-Interaction Honeypot (Cowrie)]
```

```
    Prod_Server
  end

  subgraph "Datenerfassung (NfSen Host)"
    nfcapd
    FlowFiles
    NfSen_Core
  end

  subgraph "Honey-Scan System"
    direction TB
    HS_Filter[🔍 Pre-Filter (nfdump syntax)]
    HS_Backend
    HS_Whitelist
    HS_Heuristics[🧱 Heuristik-Engine]
    HS_DB
    HS_Frontend
  end

  subgraph "Output & Alerting"
    Email
    Warden
    Syslog
  end

  %% Datenflüsse
  Attacker -->|Angriff| Honeypot_Low
  Botnet -->|Brute Force| Honeypot_High
  Scanner -->|Portscan| Honeypot_Low

  Firewall -->|NetFlow Export (UDP)| nfcapd
  nfcapd -->|Schreibt alle 5 Min| FlowFiles
  NfSen_Core -->|Trigger alle 5 Min| HS_Backend

  HS_Backend -->|Liest| FlowFiles
  HS_Backend -->|Filtert auf Honeynet| HS_Filter
  HS_Filter -->|Gefilterte Flows| HS_Whitelist
  HS_Whitelist -->|Bereinigte Flows| HS_Heuristics

  HS_Heuristics -->|Erkennt Scan| Email
  HS_Heuristics -->|Erkennt Angriff| Warden
  HS_Heuristics -->|Statistiken| HS_DB
```

```
    HS_Frontend -->|Liest| HS_DB
    HS_Frontend -->|Visualisiert| Administrator
```

2.3 Verzeichnisstruktur und Komponentenanalyse

Die Struktur des Repositories ist strikt funktional getrennt, was typisch für modulare Sicherheitssoftware ist.[5]

Pfad	Komponente	Sprache	Tiefe Analyse der Funktion
backend/	**Core Engine**	**Perl**	Beinhaltet die gesamte Geschäftslogik. Hier findet die "Magie" der Erkennung statt.
backend/honeyscan.pm	Hauptmodul	Perl	Das Herzstück. Es implementiert das Interface zu NfSen, steuert nfdump, parst den Output und entscheidet über Alarme.
backend/honeyscan.whitelist	Konfiguration	Text	Eine einfache, aber kritische Datei. Sie enthält CIDR-Bereiche, die ignoriert werden. Dies verhindert Fehlalarme durch eigene Vulnerability-Scanner (z.B. OpenVAS) oder Monitoring-Systeme (z.B. Nagios).

backend/mail_*.pl	Templates	Perl	Drei separate Skripte (local, global, dict), die als Templates für E-Mail-Benachrichtigungen dienen. Sie nutzen dynamische Variablenersetzung (z.B. %IP, %Time), um kontextreiche Warnungen zu generieren.[5]
frontend/	**UI/UX**	**PHP**	Die Präsentationsschicht. Sie läuft im Webserver-Kontext und muss Daten lesen, die vom Backend (oft root oder netflow User) geschrieben wurden.
frontend/honeyscan.php	Dashboard	PHP	Das Hauptskript, das in das NfSen-Iframe geladen wird. Es handhabt die Navigation zwischen den Tabs (Overview, Details, Whitelist).
frontend/lib/	Bibliotheken	JS	Enthält jQuery und Plotting-Bibliotheken (wahrscheinlich Flot oder ähnliche ältere Libs, passend zum Alter des Projekts), um

			die RRD-Daten interaktiv im Browser darzustellen.

---

3. Tiefenanalyse der Code-Logik: Das Backend (honeyscan.pm)

Das Backend ist der intellektuelle Kern des Systems. Wir simulieren hier den Exekutionspfad der Funktion run(), wie er in einem typischen Zyklus abläuft. Diese Analyse basiert auf den logischen Anforderungen der NetFlow-Verarbeitung und den in den Snippets beschriebenen Funktionen.

3.1 Initialisierung und Konfigurationsladen

Beim Start des NfSen-Daemons wird das Modul geladen (require honeyscan.pm).

- **Variablen-Initialisierung:** Kritische Schwellenwerte werden geladen.
  - $Threshold_Scan: Definiert, ab wie vielen einzigartigen Ziel-Ports oder Ziel-IPs ein Verhalten als "Scan" gilt (z.B. > 20).
  - $Threshold_Brute: Definiert die Anzahl der Flows pro Zeitfenster für einen "Brute-Force"-Verdacht (z.B. > 100 Flows auf Port 22 in 5 Minuten).
  - $Honeynet_Filter: Ein nfdump-Filterstring (z.B. dst net 192.168.60.0/24), der das Netzsegment der Honeypots definiert. Dies ist entscheidend für die Performance, da so 99% des normalen Traffics sofort verworfen werden.

3.2 Der run() Zyklus: Algorithmische Verarbeitung

Alle 5 Minuten übergibt NfSen den Dateinamen der neuesten Flow-Datei an die run-Funktion.

Schritt 1: Vorfilterung und Datenreduktion

Der Code ruft nfdump als Unterprozess auf. Anstatt die riesige Binärdatei in Perl einzulesen (was zu Speicherüberläufen führen würde), nutzt Honey-Scan die C-basierte Effizienz von nfdump.

- **Kommando:** nfdump -r nfcapd.current -o pipe -f "$Honeynet_Filter"
- **Ausgabe:** Ein textbasierter Stream (Pipe), bei dem jede Zeile einen Flow repräsentiert.

Schritt 2: Parsing und Whitelisting

Perl liest die Pipe Zeile für Zeile.

- **Parsing:** Die Zeile wird mittels RegEx zerlegt in $src_ip, $dst_ip, $dst_port, $bytes,

$packets, $tcp_flags.

- **Whitelist-Check:** Vor jeder weiteren Berechnung wird $src_ip gegen die in den Speicher geladene Whitelist geprüft.
  - *Optimierung:* Da Arrays langsam sind, wird die Whitelist typischerweise in einen Hash konvertiert ($Whitelist{$ip} = 1), um O(1) Zugriffszeit zu gewährleisten.
  - *Subnetz-Logik:* Komplexere Implementierungen nutzen Module wie Net::CIDR::Lite, um zu prüfen, ob eine IP in einem gewhitelisteten Subnetz liegt.

Schritt 3: Aggregation und Zustandshaltung

Da NetFlows "zustandslos" im Sinne der Anwendungsschicht sind, muss Honey-Scan den Kontext rekonstruieren. Dazu werden mehrdimensionale Hashes aufgebaut.

- Der Scanner-Hash (%Scanners):
  Dieser Hash dient der Erkennung von horizontalen und vertikalen Scans.
  Perl
  ```
  # Struktur-Hypothese
  $Scanners{$src_ip} = {
     'ports' => { 22 => 1, 80 => 1, 443 => 1 }, # Eindeutige Ports
     'targets' => { '10.0.0.5' => 1, '10.0.0.6' => 1 } # Eindeutige Ziele
  };
  ```

- Der Angreifer-Hash (%Attackers):
  Dieser Hash fokussiert sich auf die Intensität (Brute Force).
  Perl
  ```
  # Struktur-Hypothese
  $Attackers{$src_ip}{$dst_port} = {
     'flows' => 150,      # Anzahl Verbindungsversuche
     'bytes' => 45000,    # Übertragene Datenmenge
     'flags' => 'S'       # TCP Flags (z.B. nur SYN)
  };
  ```

Schritt 4: Heuristische Analyse (Die "Intelligenz")

Nachdem die Datei vollständig geparst ist, iteriert der Algorithmus über die Hashes, um Anomalien zu klassifizieren.

1. Erkennung von "Massive Scans" (Reconnaissance):
   Der Code zählt die Schlüssel im Hash ports und targets.
   - *Logik:* Wenn scalar(keys %{$Scanners{$ip}{'targets'}}) > $Limit_Hosts ODER scalar(keys %{$Scanners{$ip}{'ports'}}) > $Limit_Ports.
   - *Trigger:* Dies löst den mail_global.pl Alarm aus. Ein Angreifer kartographiert das Netzwerk.
2. Erkennung von "Dictionary Attacks" (Brute Force):
   Der Code prüft den flows Zähler für spezifische Ports (z.B. SSH/22, RDP/3389).

- *Logik:* Wenn $Attackers{$ip}{22}{'flows'} > $Threshold_Brute.
- *Verfeinerung:* Ein hohes Verhältnis von Flows zu Bytes ist ein starker Indikator für fehlgeschlagene Logins (viele kleine Pakete). Ein erfolgreicher Login würde mehr Daten übertragen.
- *Trigger:* Dies löst den mail_dict.pl Alarm aus.

3. Erkennung von "Internen Infektionen" (Lateral Movement):
   Dies ist die kritischste Funktion. Der Code prüft, ob eine $src_ip aus dem lokalen IP-Bereich stammt (definiert in der Config), aber nicht selbst ein Honeypot ist.
   - *Szenario:* Ein Mitarbeiter-Laptop (10.10.10.50) verbindet sich mit einem Honeypot (10.20.20.100).
   - *Bedeutung:* Da kein legitimer Nutzer Grund hat, einen Honeypot zu kontaktieren, ist dies ein nahezu 100%iges Indiz für einen kompromittierten Host (z.B. Wurm, Ransomware), der versucht, sich im Intranet auszubreiten.
   - *Trigger:* mail_local.pl – Höchste Priorität.

3.3 Datenfluss und Entscheidungsprozess im Backend ⚙️

Code-Snippet

```
flowchart TD
    Start((Start Zyklus)) --> NfdumpCall[Aufruf: nfdump -r file -f filter]
    NfdumpCall --> PipeStream

    PipeStream --> ParseLine{Zeile parsen}
    ParseLine -->|EOF| AnalysisPhase[Analyse-Phase]
    ParseLine --> CheckWhitelist{IP in Whitelist?}

    CheckWhitelist -- Ja --> Ignore[Verwerfen]
    CheckWhitelist -- Nein --> Classify{IP-Typ?}

    Classify -- Intern --> AggLocal[Aggregiere: Interner Verdacht]
    Classify -- Extern --> AggExternal[Aggregiere: Externer Angreifer]

    AggLocal --> ParseLine
    AggExternal --> ParseLine

    AnalysisPhase --> CheckLocal{Interne Hits > 0?}
    CheckLocal -- Ja --> AlertLocal
    AlertLocal --> GenMailLocal[Generiere E-Mail (mail_local.pl)]
    AlertLocal --> LogDB
```

```
    AnalysisPhase --> CheckScan{Unique Ports/IPs > Limit?}
    CheckScan -- Ja --> AlertScan
    AlertScan --> GenMailGlobal[Generiere E-Mail (mail_global.pl)]
    AlertScan --> ExportIDEA

    AnalysisPhase --> CheckBrute{Flows > Limit?}
    CheckBrute -- Ja --> AlertBrute
    AlertBrute --> GenMailDict[Generiere E-Mail (mail_dict.pl)]

    GenMailLocal --> SendMail
    GenMailGlobal --> SendMail
    GenMailDict --> SendMail

    ExportIDEA --> WardenSpool
```

---

4. Detaillierte Code-Analyse: Das Frontend (honeyscan.php)

Das Frontend ist mehr als nur eine Anzeige; es ist das Interface für die forensische Analyse. Geschrieben in PHP, nutzt es die Infrastruktur des Webservers, auf dem NfSen läuft.

4.1 Integration und Sicherheit

Das Skript honeyscan.php wird nicht direkt aufgerufen, sondern über den nfsen.php Wrapper inkludiert. Dies stellt sicher, dass die Authentifizierung von NfSen greift.

- **Datenzugriff:** Das Frontend muss Dateien lesen, die vom Backend (User netflow) erstellt wurden. Dies erfordert korrekte Unix-Dateiberechtigungen (typischerweise Gruppe www-data oder apache).
- **Input Validierung:** Bei der Whitelist-Bearbeitung muss das PHP-Skript Benutzereingaben (IP-Adressen) validieren, bevor es sie in die honeyscan.whitelist Datei schreibt, um Command-Injection-Angriffe zu verhindern.

4.2 Visualisierungskomponenten

Basierend auf den Snippets und der RRDtool-Logik bietet das Frontend drei Hauptansichten [5]:

1. **Overview (Dashboard):**
    - Hier werden RRD-Graphen gerendert. RRDtool (Round Robin Database) ist ideal für Zeitreihen.
    - *Metriken:* "Angriffe pro Sekunde", "Top 5 Ziel-Ports", "Verhältnis Scan vs. Brute

Force".
    - *Technik:* PHP ruft rrdtool graph auf und generiert PNG-Bilder, die eingebettet werden.
2. **Details (Forensische Tabelle):**
    - Diese Ansicht parst die textbasierten Logdateien, die das Backend für jeden Alarm schreibt.
    - *Tabelle:* Zeitstempel | Angreifer-IP | Reverse DNS | GeoIP Land | Ziel-Honeypot | Angriffstyp.
    - *Interaktivität:* Ein Klick auf die IP führt oft zu externen Tools (Whois, VirusTotal) oder fügt die IP direkt der Whitelist hinzu.
3. **Whitelist Editor:**
    - Ein einfaches Textfeld oder eine Liste, die den Inhalt von backend/honeyscan.whitelist anzeigt.
    - Beim Speichern wird ein File-Lock benötigt, um Race-Conditions mit dem Backend-Prozess zu vermeiden, der vielleicht gerade liest.

---

5. Datenmodelle, Protokolle und Standards

Die Stärke von Honey-Scan liegt in seiner strikten Adhärenz zu etablierten Standards, was die Interoperabilität ermöglicht.

5.1 NetFlow v5/v9 Datenstruktur

Honey-Scan extrahiert spezifische Felder aus den NetFlow-Records. Das Verständnis dieser Felder ist essentiell für die Interpretation der Alarme.

NetFlow Feld	**Bedeutung für Honey-Scan**	**Analysemöglichkeit**
IPV4_SRC_ADDR	Quell-IP	Identifikation des Angreifers.
IPV4_DST_ADDR	Ziel-IP	Verifikation, dass das Ziel ein Honeypot ist.
L4_DST_PORT	Ziel-Port	Klassifikation des Angriffs (22=SSH, 445=SMB, 80=Web).
TCP_FLAGS	TCP Flags	**SYN:** Scan-Versuch.

		SYN/ACK: Antwort (Port offen). **RST:** Port geschlossen. *Honey-Scan sucht primär nach reinen SYN-Flows oder Verbindungen mit geringer Byte-Zahl.*
IN_BYTES	Byte-Count	Unterscheidung Scan vs. Brute-Force. Ein Scan hat ~40-60 Bytes pro Flow. Ein Brute-Force Versuch hat mehr (Handshake + Login-String).

5.2 Das IDEA Format (Intrusion Detection Extensible Alert)

Ein herausragendes Merkmal ist der Export im IDEA-Format (JSON), wie in den Forschungsunterlagen erwähnt.[1] Dies ist der Standard für den Datenaustausch im Warden-Netzwerk (SABU).

Beispiel eines generierten IDEA-Alerts:

JSON


```
{
 "Format": "IDEA0",
 "ID": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
 "CreateTime": "2026-01-12T05:35:00Z",
 "DetectTime": "2026-01-12T05:30:00Z",
 "Category":,
 "Confidence": 0.9,
 "Note": "Detected by Honey-Scan via NfSen",
 "Source": [
  {
   "IP4": ["203.0.113.55"],
```

```
    "Proto": ["tcp"]
   }
  ],
  "Target": [
   {
    "IP4": ["192.168.1.100"],
    "Port": ,
    "Anonymised": true
   }
  ],
  "Node":
   }
  ]
}
```

- **Wichtigkeit:** Durch dieses Format kann Honey-Scan seine Erkenntnisse automatisiert mit nationalen CERTs oder anderen Organisationen teilen. Warden fungiert als Hub, der diese JSON-Objekte verteilt.

---

6. Prozessanalyse und Lebenszyklus der Daten

Vom Entstehen bis zur Archivierung durchlaufen die Daten einen definierten Lebenszyklus.

6.1 Entstehung (Trigger)

Die Daten entstehen nicht im Tool, sondern durch die Aktivität des Angreifers.

- **Der Initiale Reiz:** Ein TCP SYN Paket trifft auf die Netzwerkkarte des Honeypots.
- **Die Beobachtung:** Der Router sieht dieses Paket. Er erstellt einen Cache-Eintrag.
- **Der Export:** Sobald der Cache voll ist oder ein Timeout eintritt (meist 30-60 Sekunden für aktive Flows), wird der Flow-Record an nfcapd gesendet.

6.2 Verarbeitung (Processing)

Die Verarbeitung ist **Batch-basiert**, nicht Real-Time.

- **Latenz:** Es gibt eine unvermeidliche Verzögerung.
  - Flow Timeout (bis zu 60s) +
  - NfSen Rotation (300s) +
  - Verarbeitungszeit Perl (10-30s).
  - *Gesamt:* Ein Angriff wird 6 bis 8 Minuten nach Beginn gemeldet. Für die Blockierung aktiver Angriffe ist dies oft zu langsam (hier wäre ein IPS nötig), aber für Forensik und Blacklisting ist es ausreichend.

6.3 Archivierung und Verwurf

Honey-Scan muss aggressiv Daten verwerfen, um Speicherplatz zu sparen.

- **Verwurf (Discard):** 99.9% des Traffics (legitimer Traffic an Produktionsserver, Whitelist-Traffic) werden sofort im Speicher verworfen und nie protokolliert.
- **Archivierung:** Nur bestätigte "Incidents" werden persistiert.
  - *Kurzzeit (Text):* Detaillierte Logs für die Web-UI (Vorhaltezeit z.B. 4 Wochen).
  - *Langzeit (RRD):* Aggregierte Statistiken (Vorhaltezeit Jahre, aber mit Verlust der Details).

6.4 Externe Weitergabe

Die Weitergabe an **Warden** erfolgt asynchron.

1. Honey-Scan schreibt JSON-Dateien in ein Spool-Verzeichnis (/var/spool/warden/in).
2. Ein separater Daemon (Warden Client) überwacht dieses Verzeichnis.
3. Er validiert das JSON, signiert es kryptographisch und sendet es an den zentralen Warden-Server.
4. Nach erfolgreichem Upload wird die lokale Datei gelöscht.

---

7. Szenarien und Fallstudien

Um die praktische Relevanz zu verdeutlichen, analysieren wir drei konkrete Szenarien im Kontext eines Nutzers wie derlemue (Self-Hosting/Homelab).

Szenario A: Der "verirrte" Mitarbeiter (False Positive vs. Bedrohung)

Situation: Ein Admin scannt sein eigenes Netz mit nmap, um offene Ports zu prüfen. Er scannt auch das Honeynet-Segment.
Reaktion Honey-Scan:

1. Erkennt massive Verbindungen von 10.0.0.50 (Admin-PC) auf 10.0.0.100 (Honeypot).
2. Klassifiziert dies als "Internen Angriff".
3. Sendet Alarm mail_local.pl.
   Lösung: Der Admin muss seine IP 10.0.0.50 in honeyscan.whitelist eintragen. Das System zeigt hier seine Robustheit: Es vertraut niemandem standardmäßig ("Zero Trust").

Szenario B: Der IoT-Botnet Angriff (Mirai)

Situation: Tausende infizierte Kameras versuchen, sich mit Standard-Passwörtern (Telnet/23) auf dem Honeypot einzuloggen.
Reaktion Honey-Scan:

1. Der run() Zyklus sieht Tausende von Quell-IPs.
2. Jede IP sendet nur 1-2 Pakete (Versuch Login -> Fail -> Disconnect).

3. Honey-Scan aggregiert dies. Wenn die IPs einzeln unter der Schwelle bleiben, könnte der Alarm ausbleiben (Low and Slow).
4. *Aber:* Wenn das Botnet aggressiv ist, überschreitet jede IP den $Threshold_Brute.
5. Ergebnis: Tausende Einträge in der Datenbank und eine Flut von E-Mails.
   Optimierung: Hier zeigt sich eine Schwäche. Honey-Scan braucht eine "Deduplizierung" oder "Throttling" Logik in mail_dict.pl, um den Admin nicht zu spammen (z.B. "1000 Alerts suppressed").

Szenario C: Laterale Bewegung einer Ransomware

Situation: Ein Server im "Minecraft Hosting Rack" (Kontext derlemue) ist infiziert und sucht nach SMB-Shares (Port 445).
Reaktion Honey-Scan:

1. Der infizierte Server sendet SYN an 10.0.0.101 (Windows-Honeypot).
2. Honey-Scan erkennt: Source=Intern, Dest=Honeypot, Port=445.
3. **Kritischer Alarm:** Dies ist der "Golden Use Case". Es ermöglicht dem Admin, den infizierten Server vom Netz zu nehmen, *bevor* die Ransomware den Backup-Server verschlüsselt.

---

8. Sicherheits- und Betriebsaspekte

8.1 Deployment und Voraussetzungen

Für eine erfolgreiche Installation müssen folgende Komponenten vorhanden sein:

- **OS:** Linux/BSD (NfSen läuft nativ auf Unix).
- **Perl Module:** RRD::Simple, LWP::UserAgent (für externe APIs), JSON::XS (für IDEA Export).
- **NetFlow Quelle:** Ein Router (MikroTik, Ubiquiti, Cisco) oder ein Soft-Exporter (softflowd auf dem Hypervisor).

8.2 Risiken und Härtung

Da Honey-Scan Teil der Sicherheitsinfrastruktur ist, ist es selbst ein Ziel.

- **Input Validation:** Da Daten aus dem Netzwerk (NetFlows) verarbeitet werden, könnten böswillig konstruierte NetFlow-Pakete (Fuzzing) versuchen, den Perl-Parser zum Absturz zu bringen oder Code auszuführen. Die Nutzung von nfdump als vorgeschaltetem C-Parser mindert dieses Risiko, da nfdump sehr robust ist.
- **Web-Interface:** Das PHP-Frontend läuft oft mit hohen Rechten (um Logs zu lesen). Eine Cross-Site Scripting (XSS) Lücke im Dashboard könnte einem Angreifer erlauben, falsche Whitelist-Einträge zu erstellen.
- **DoS gegen den Monitor:** Ein Angreifer könnte das Honeynet mit so viel Traffic fluten (DDoS), dass die Flow-Dateien riesig werden. Wenn die Verarbeitung durch

honeyscan.pm länger als 5 Minuten dauert, gerät NfSen in einen Rückstand ("Lag"), und das Monitoring fällt aus.
    - *Gegenmaßnahme:* Setzen von Timeouts im Perl-Skript und Ressourcen-Limits (ulimit).

---

9. Fazit und Ausblick

Honey-Scan ist ein Paradebeispiel für den effizienten Einsatz von Metadaten in der Cybersicherheit. Anstatt in der Flut von Paketinhalten zu ertrinken, nutzt es die abstrahierte Sichtweise von NetFlow, um taktische und strategische Erkenntnisse zu gewinnen.

Zusammenfassung der Erkenntnisse:

1. **Architektonische Eleganz:** Durch das "Huckepack-Reisen" auf NfSen spart sich Honey-Scan die komplexe Flow-Collection und konzentriert sich rein auf die Analyse.
2. **Fokus auf das Wesentliche:** Die Unterscheidung zwischen "Internen" und "Externen" Bedrohungen sowie die Klassifizierung in "Scan" vs. "Brute Force" deckt 90% der relevanten Szenarien im Honeypot-Betrieb ab.
3. **Integrationsfähigkeit:** Durch den IDEA-Export wird das Tool von einer Insellösung zu einem wertvollen Sensor in einem kollaborativen Verteidigungsnetzwerk.

Für Anwender wie derlemue, die wahrscheinlich eine fortgeschrittene Heim- oder kleine Unternehmensumgebung betreiben, bietet Honey-Scan eine professionelle Analysemöglichkeit ohne die Lizenzkosten kommerzieller SIEM-Lösungen. Die größte Herausforderung bleibt die Konfiguration der Schwellenwerte, um das Rauschen des Internets von echten, gezielten Angriffen zu unterscheiden.

Die Zukunft solcher Systeme liegt in der Integration von Machine Learning direkt in die Pipeline (z.B. Analyse der Zeitintervalle zwischen Paketen, um menschliche Angreifer von Bots zu unterscheiden), doch die robuste, regelbasierte Logik von Honey-Scan bleibt das Fundament solider Netzwerksicherheit.

Referenzen

1. CVE-2021-44228 vulnerability Log4j (Solr)? · Issue #4375 · mailcow/mailcow-dockerized, Zugriff am Januar 12, 2026, https://github.com/mailcow/mailcow-dockerized/issues/4375
2. r/meirl on Reddit, Zugriff am Januar 12, 2026, https://www.reddit.com/r/meirl/comments/1iu01mw/meirl/
3. Check out my progress building Pittsburgh, PA on Paradox Mods! : r/CitiesSkylines - Reddit, Zugriff am Januar 12, 2026, https://www.reddit.com/r/CitiesSkylines/comments/1jecf3d/check_out_my_progress_building_pittsburgh_pa_on/
4. Predictions of Network Attacks in Collaborative Environment - IS MUNI, Zugriff am

Januar 12, 2026, https://is.muni.cz/th/dmpga/thesis.pdf
5. husak/honeyscan: honeynet monitoring plugin for NfSen - GitHub, Zugriff am Januar 12, 2026, https://github.com/husak/honeyscan
6. (PDF) Dataset of intrusion detection alerts from a sharing platform, Zugriff am Januar 12, 2026, https://www.researchgate.net/publication/347762333_Dataset_of_intrusion_detection_alerts_from_a_sharing_platform