Honeycomb

Automated NIDS Signature Creation using Honeypots

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Christian Kreibich
christian.kreibich@cl.cam.ac.uk

http://www.cl.cam.ac.uk/~cpk25/honeycomb

The Problem

Creating signatures for Network Intrusion Detection Systems is difficult for a number of reasons:

- The process is **manual, slow and error-prone**, leading to signatures that often are either too narrow (causing false negatives) or too loose (causing false positives).
- Good signatures require **detailed knowledge** of the specific traffic phenomenon they are designed to capture.

Our Approach

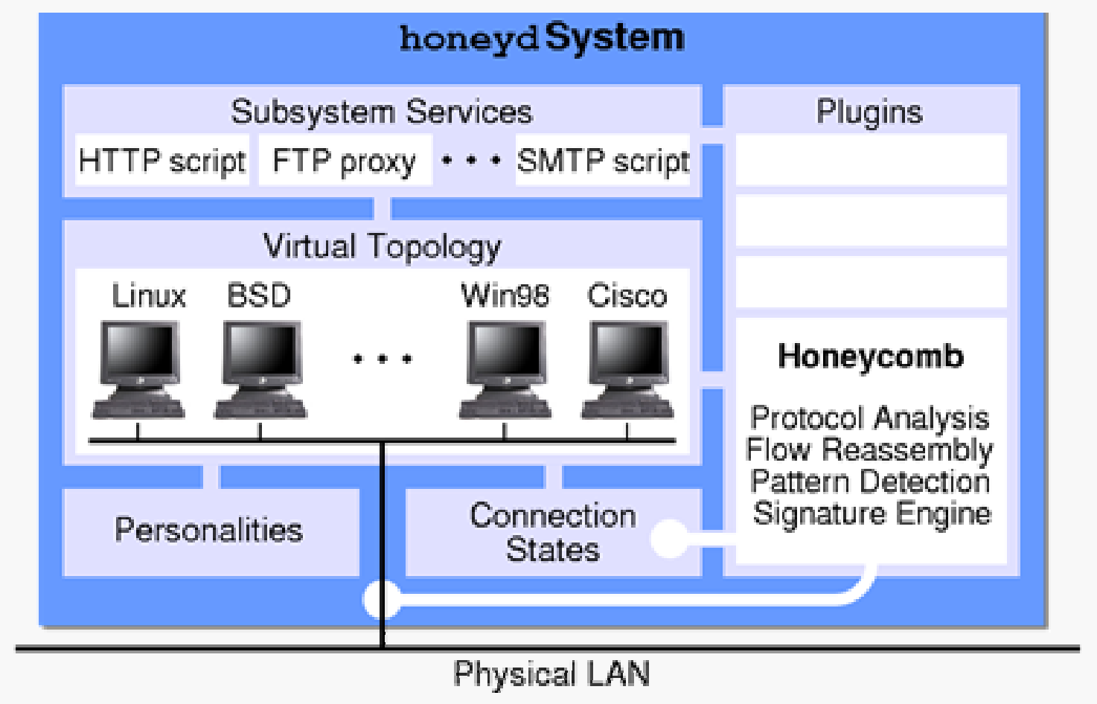
Honeycomb applies protocol analysis and pattern detection techniques to network traffic on honeypots, without hardcoding any application-specific knowledge. This approach has the following benefits:

- Traffic on a honeypot can be assumed to be **malicious**.
- Traffic volumes are **manageable** as honeypots see comparatively little traffic.

The results are automatically-generated, precise signatures for malicious traffic.

System Design

Our system is a pluggable extension to the open-source honeypot honeyd. The Honeycomb plugin runs within honeyd and hooks itself into the connection state engine and the traffic entering and leaving honeyd.



Signature Creation Algorithm

The algorithm triggers on two major events:

- **Packet interception**

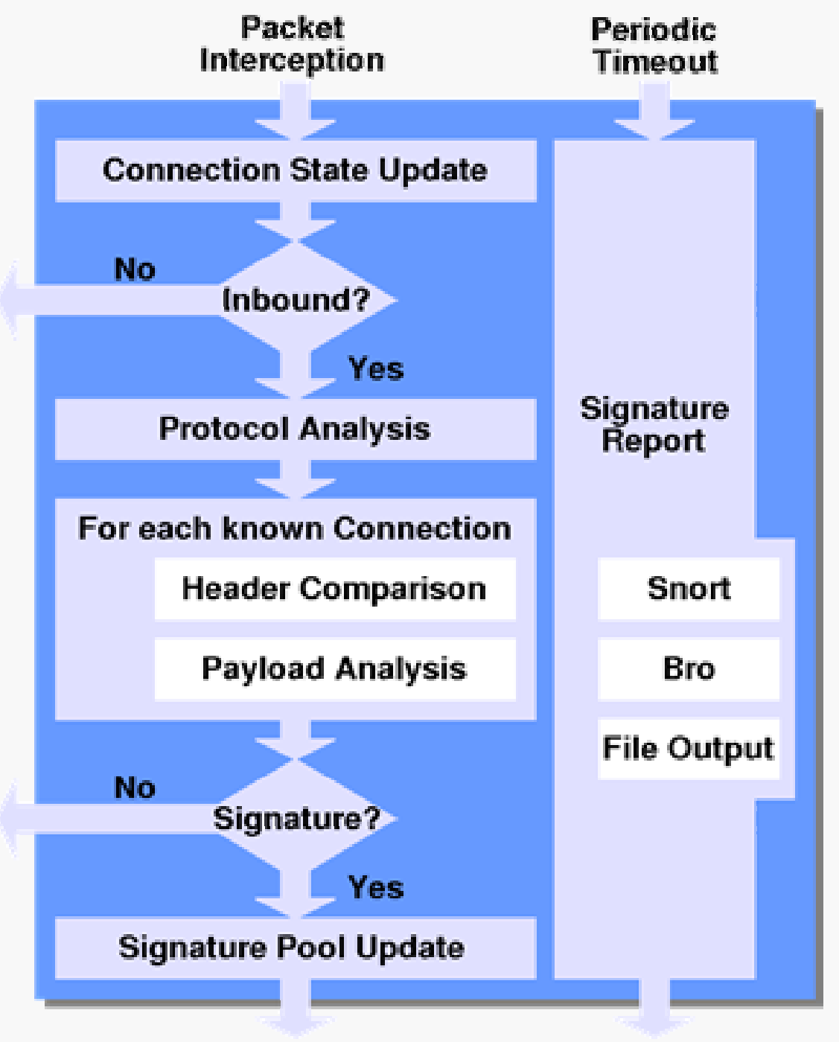
  In- and outgoing packets are intercepted and analyzed in two phases:

  - **Protocol Analysis** tests headers for protocol compliance.
  - **Payload Analysis** looks for repeated patterns within flow data.

  New signatures are added to a signature pool, dropped if they are duplicates, or used to augment existing signatures.

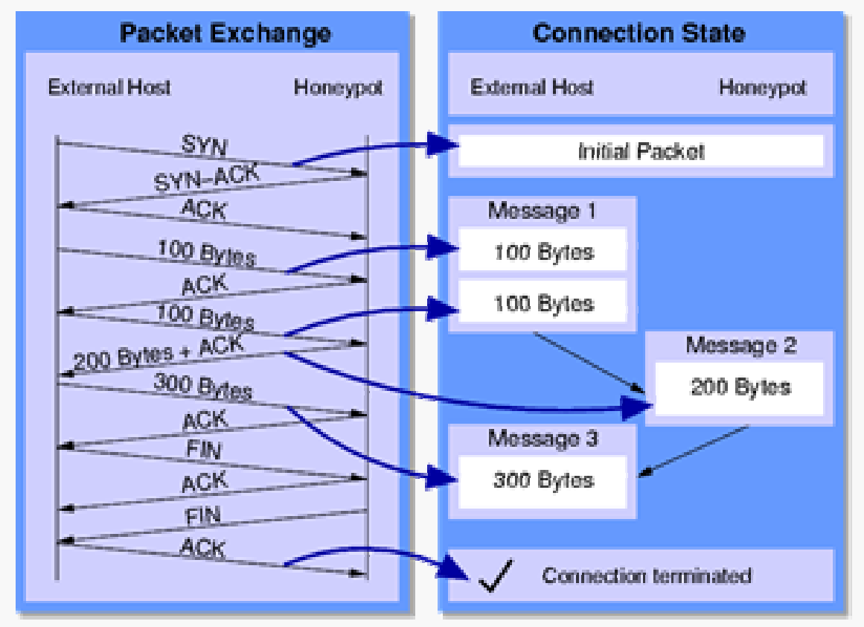
- **Periodic timeouts**

  The signature pool is periodically reported to configurable output mechanisms, currently producing Bro or Snort signatures.



Flow Reassembly

Honeycomb performs per-direction flow reassembly, creating connection state as a **sequence of messages**. Terminated connections are marked but not immediately released, as the system uses them to look for traffic patterns later on.



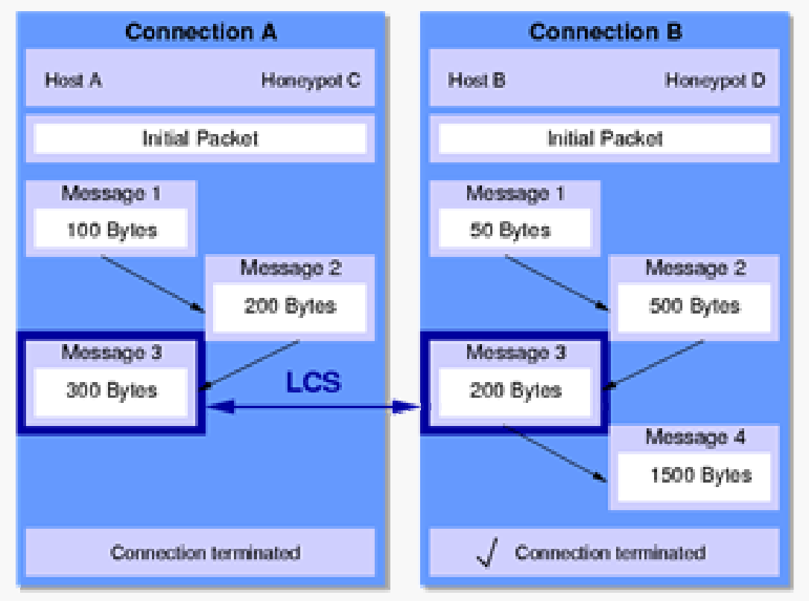
LCS Algorithm

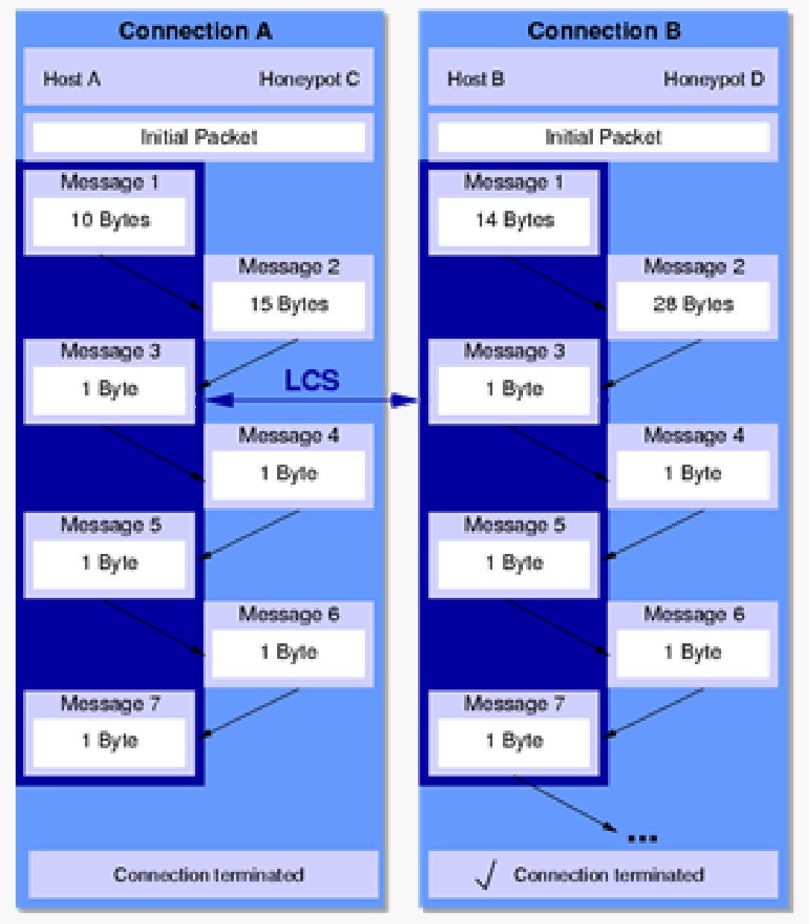
Honeycomb uses an **O(n) longest-common-substring algorithm** based on a suffix tree implementation to detect patterns in the flow messages.

Message Pattern Detection

Honeycomb employs two pattern detection strategies:

- **Horizontal Detection** applies LCS to individual messages at the same depth.
- **Vertical Detection** concatenates a number of messages before applying LCS. This improves detection in interactive sessions and masks TCP protocol dynamics.





Initial Results

Our tests have produced **encouraging results**, particularly for **worm detection**:

- The system generated **full signatures** for the SQL Slammer and Code Red II worms.

```
alert udp any any -> 192.168.169.2/32 1434 (msg: "Honeycomb Thu May  8 09h58m38 2003 "; content:
"|04 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 DC C9 B0|B|EB 0E 01 01 01 01 01 01 01|p|AE|B|01| p|AE|B|90 90 90 90 90 90 90 90|h|DC C9
B0|B|B8 01 01 01 01|1|C9 B1 18|P|E2 FD|5|01 01 01 05|P|89 E5|Qh.dllhel32hkernQhounthickChGetTf
|B9|llQh32.dhws2_f|B9|etQhsockf|B9|toQhsend|BE 18 10 AE|B|8D|E|D4|P|FF 16|P|8D|E|E0|P|8D|E|F0|P
|FF 16|P|BE 10 10 AE|B|8B 1E 8B 03|=U|8B EC|Qt|05 BE 1C 10 AE|B|FF 16 FF D0|1|C9|QQP|81 F1 03 01
04 9B 81 F1 01 01 01 01|Q|8D|E|CC|P|8B|E|C0|P|FF 16|j |11|j|02|j|02 FF D0|P|8D|E|C4|P|8B|E|C0|P
|FF 16 89 C6 09 DB 81 F3|<a|D9 FF 8B|E|B4 8D 0C|@|8D 14 88 C1 E2 04 01 C2 C1 E2 08|)|C2 8D 04 90
01 D8 89|E|B4|j|10 8D|E|B0|P1|C9|Qf|81 F1|x|01|Q|8D|E|03|P|8B|E|AC|P |FF D6 EB|"; )
```

- Aggregating identical signatures by destination ports reduces the number of signatures and is well-suited to capture **portscans**.

Summary

The system works and creates useful signatures. Future work will include **minimizing the per-packet overhead** and **approximate pattern detection** to allow generation of regular-expression type signatures.