Automated NIDS Signature Creation using Honeypots

Christian Kreibich, Jon Crowcroft
University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom
{firstname.lastname}@cl.cam.ac.uk

***Abstract*— This paper describes Honeycomb, a system for automated generation of attack signatures for network intrusion detection systems (NIDSs). Our system applies pattern detection techniques and protocol conformance checks on multiple levels in the protocol hierarchy to network traffic captured on a honeypot system. While running Honeycomb on an unprotected cable modem connection for 24 hours, the system successfully created precise traffic signatures that otherwise would have required the skills and time of a security officer.**

Currently, the creation of NIDS signatures is a tedious manual process that requires detailed knowledge of the traffic characteristics of any phenomenon that is supposed to be detected by a new signature. Simplistic signatures tend to generate large numbers of false positives; overly specific ones cause false negatives. To address these issues, we present Honeycomb[1], a system that generates signatures for malicious network traffic automatically. Our system applies protocol analysis and pattern-detection techniques to traffic captured on honeypots. Honeypots are computer resources set up for the purpose of monitoring and logging activities of entities that probe, attack or compromise them [1][5][6]. Using traffic on honeypots has the major advantage of concentrating on traffic that can be considered malicious by definition, as they provide no production value.

We have extended the open-source honeypot `honeyd` [3] by a subsystem that inspects traffic inside the honeypot. Integrating our system with `honeyd` has advantages over a bump-in-the-wire approach: we avoid duplication of effort, as `honeyd` already uses libpcap to capture the relevant packets; also, we avoid cold-start issues common to devices like packet normalizers or NIDSs, since `honeyd` does not just passively listen to traffic but rather emulates hosts answering incoming requests. It hence knows exactly when a new connection is started or terminated.

The philosophy behind our approach is to keep the system free of knowledge specific to application layer protocols: Upon packet interception, the system first performs protocol analysis similar to traffic normalizers. However, instead of modifying packets, deviations from expected behaviour are registered in a signature. The system then performs flow reassembly and compares the current connection's flow with the connections for which state is kept, trying to detect similarities in the payloads. For this purpose, we have implemented a generic O(n) longest-common-substring (LCS) algorithm based on suffix trees, using the algorithm proposed by Ukkonen [7]. Any detected patterns are added to the signature.

Created signatures are stored in a signature pool that is periodically reported to an output module; currently either outputting Bro [2] or Snort [4] signatures. New signatures are added if they differ from all stored signatures, dropped if they are duplicates, and used to improve existing signatures whenever possible. Signatures that differ only in destination ports are aggregated to reduce the number of reported signatures.

```
alert udp any any -> 192.168.169.2/32 1434 (msg: "Honeycomb Fri Jul 18 11h46m33 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 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|"; )
```

Fig. 1. Signature Honeycomb created for the Slammer Worm.

Initial tests are encouraging; Honeycomb has created detailed signatures for the CodeRed II and Slammer worms (see Figure 1) and for a variety of portscanning techniques, while maintaining good response times (see Figure 2).

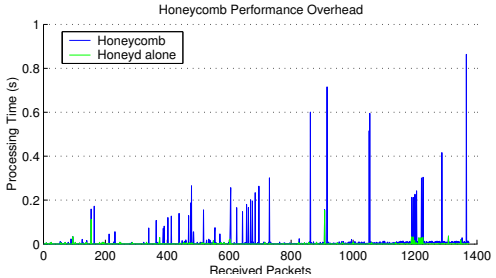

Fig. 2. Honeycomb's packet response times over 24-hour period.

In the future, we want to expose Honeycomb to more aggressive traffic patterns to better understand its performance. We are currently trying to minimise the amount of effort spent per arriving packet. Regarding the LCS algorithm, approximate matching schemes would allow us to create signatures that contain regular expressions. Also, applying Honeycomb to other traffic could be useful for deriving signatures for specific application traffic, or to verify existing signatures.

[1] `http://www.cl.cam.ac.uk/users/cpk25/honeycomb/`

References

[1] William R. Cheswick. An Evening with Berferd, in which a Cracker is lured, endured, and studied. In *Proceedings of the 1992 Winter USENIX Conference*, 1992.

[2] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23-24):2435–2463, 1998.

[3] Niels Provos. Honeyd - A Virtual Honeypot Daemon. In *10th DFN-CERT Workshop, Hamburg, Germany*, February 2003.

[4] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Conference on Systems Administration*, pages 229–238, 1999.

[5] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2003.

[6] Clifford Stoll. *The Cuckoo's Egg*. Addison-Wesley, 1986.

[7] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, (14):249–260, 1995.