HIVE: an Open Infrastructure for Malware Collection and Analysis

Davide Cavalca[1] and Emanuele Goldoni[2]

[1] University of Pavia
Department of Computer Engineering and Systems Science
davide.cavalca01@ateneopv.it

[2] University of Pavia
Department of Electronics
emanuele.goldoni@unipv.it

Abstract. Honeynets have become an important tool for researchers and network operators. However, the lack of a unified honeynet data model has impeded their effectiveness, resulting in multiple unrelated data sources, each with its own proprietary access method and format. Moreover, the deployment and management of a honeynet is a time-consuming activity and the interpretation of collected data is far from trivial. In this paper we propose HIVE (Honeynet Infrastructure in Virtualized Environment), a new highly scalable automated data collection and analysis architecture, which is built on top of proven FLOSS (Free, Libre and Open Source) solutions integrated and extended with new tools we developed. We use virtualization to ease honeypot management and deployment, combining both high-interaction and low-interaction sensors in a common infrastructure. We address the need for rapid comprehension and detailed data analysis by harnessing the power of a relational database system, which provides centralized storage and access to the collected data while ensuring its constant integrity. Finally, we present some techniques for the active monitoring of centralized botnets we integrated in our infrastructure.

Key words: botnet, honeynet, honeypot, threat monitoring, malware

1 Introduction

In last years there has been a dramatic increase in malware activity on the Internet: according to [29], the last two years saw a 1500% increase in threat volume, with major contributions from botnet creation and expansion. A constant monitoring of criminal activity on the network is needed to identify, prevent and actively counteract the impeding menaces. To be able to successfully defend itself, every networked company has to carry the burden of threat monitoring, building and maintaining a dedicated infrastructure. Due to the recent shift from mass attacks (mostly represented by viruses) to targeted attacks, which are performed for a specific purpose (data theft, disruption of operations, etc.), threat control has become even more compelling.

Thanks to The Honeynet Project [1] and similar initiatives, the employment of honeypots for threat monitoring has grown into wide usage. There are now several honeynets monitoring the Internet, deployed both by academic and commercial organizations, for research and defense purposes. Most of these honeynets are independent and disconnected: they are owned and maintained by a single organization and they serve a specific purpose; the data they collect is often kept confidential, especially for commercial organizations. Moreover, each and every honeynet uses a different, usually homegrown, data collection and analysis system, impairing interoperability with external systems. Even if a company *wants* to share its honeynet data, it is often difficult to integrate different data sources in a unified system. The mwcollect Alliance [2] has taken a step in the right direction, building a single infrastructure for the collection of honeynet data and the monitoring of threats in a collaborative fashion. But the Alliance is a centralized system, essentially closed, for safety reasons: there is no public information on their infrastructure.

We believe there is a need for an open honeynet infrastructure, to lower the barrier needed to setup and operate an honeypot network and to ease data collection and analysis. The openness needs to be both structural and material: open and common data schemas make the sharing of information effortless, thus encouraging it. On the other hand, the use of Open Source software to implement the actual infrastructure makes it easy for everyone to study, build, audit and improve it, allowing the whole community to benefit from individual efforts.

The remainder of this paper is organized as follows: in Section 2 we discuss previous works in this area and its relation with our research. The architecture and implementation of HIVE (Honeynet Infrastructure in Virtualized Environment), our proposed honeynet infrastructure, is described in detail in Section 3. We present some preliminary results from our testbed system in Section 4. Finally, in Section 5 we conclude and lay out the grounds for some future works.

2 Background and related work

The use of honeypots for malware detection and capture is now a well-established field. Two major kinds of honeypots have been developed: low-interaction honeypots simulate vulnerable services or environments to lure malware into attacking them and capture its sample. They are easy to deploy and require low maintenance, but the simulation is imperfect and they can often be detected or circumvented. The major Open Source products in this area are Nepenthes [3], HoneyTrap [30] and Amun [7]. High-interaction honeypots, on the other hand, use a full fledged system as a bait; the honeypot machine is periodically analyzed, malware is collected and the system is rebuilt to a clean state. This kind of honeypot can potentially capture more malware samples,

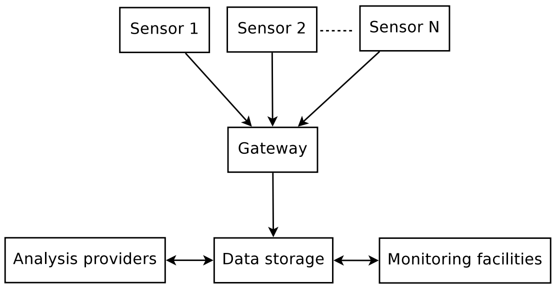

Fig. 1. Block diagram of HIVE architecture.

but is also much more expensive to deploy and maintain. There are also legal liability concerns: we have to avoid that the honeypot participates or starts a potentially dangerous attack to external machines (e.g. a denial of service). Nowadays, high-interaction honeypots are often implemented using virtualization, which eases most of their drawbacks but potentially makes them easier to detect and thus vulnerable to targeted attacks [11, 34].

The HoneyBow project [33] deployed a sensor infrastructure using virtualized VMWare systems combined with Nepenthes [3] sensors to capture malware samples. While part of the software they developed is freely available at [24] it has not received any significant update in the last two years. The HoneyBow project also fails to detail their data storage and analysis system, focusing instead on the collection infrastructure and the analysis results of collected data.

In [19] Rajab et al. developed a botnet measurement infrastructure using Nepenthes sensors and a physical honeynet. They focused on IRC bots, developing a series of tools to study and infiltrate these kinds of botnets. While they make the collected data available to the research community, to our knowledge the developed software has not been released.

3 Implementation of HIVE

We chose to structure HIVE on a three-tier architecture, shown in Fig. 1, to fully decouple data acquisition from data memorization. This decision has a twofold advantage: it is easier to build and scale the whole system, and if a honeypot sensor is compromised there is no way it can harm (or even see) the data store.

The first stage is the actual honeynet, which is implemented using virtualization and features a combination of low-interaction and high-interaction sensors. Malware samples acquired from the honeypots are sent to a *gateway*, which validates and preprocesses them; samples are then sent to the data store for storage and analysis. Malware samples are analyzed with the help of external services, which provide reports on the samples' behavior to help classification.

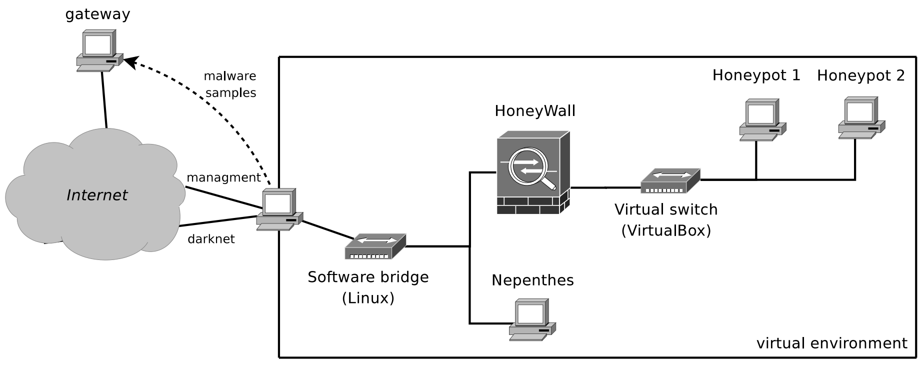

Fig. 2. Our virtual honeynet structure.

Monitoring facilities query the data store to obtain threat information (e.g. botnet controller servers' addresses) and oversee their development.

The architecture is fully scalable: adding more sensors is only a matter of creating new virtual machines or adding other physical systems. If a honeypot is damaged or compromised it cannot affect other sensors and it can be easily rebuilt or replaced. Structure and implementation of the honeynet and its sensors is described in Section 3.1. The gateway (which itself can be replicated with no effort) allows preliminary filtering of honeypot data: we can potentially remove bogus samples before sending them to the data store, avoiding to waste system resources on their analysis. The data store, finally, can be made redundant using database replication, and itself uses multiple providers for the samples' analysis, to achieve better accuracy and lower the samples turnaround. This part of HIVE, which can be considered its core, is presented in Section 3.2.

3.1 Honeynet sensors

We chose to implement our honeypot sensors using virtualization. As noted before in [33], virtualization dramatically cuts down the efforts needed to deploy and manage the honeynet. Moreover, the ability to simulate an arbitrary network topology on a single physical machine allows us to dynamically reconfigure the honeynet. This choice is not mandatory: our infrastructure works equally well with virtual and physical honeypots, but in the latter case restoring the machine to a clean state will require considerably more efforts.

We based our implementation on VirtualBox [22], an Open Source virtualization product. VirtualBox has similar performance to its more prominent competitor VMWare, which has been used in honeypot research [33], is multiplatform and features some advantages from our point of view. Besides being Open Source, it's fully scriptable using a command-line interface or, in the last 1.6.0 version, a standards-compliant web service. The latter feature allows to easily automate the honeypots management operations using a few shell scripts.

Compared to Xen, another leading Open Source virtualization product, VirtualBox does not require dedicated hardware support to execute unmodified guests (such as Microsoft closed-souce operating systems). Our back-end is, of course, product-agnostic and is not tied to this particular choice.

We believe that using a combination of low-interaction and high-interaction honeypots allows us to get the best of both worlds; through the use of virtualization, we are able to nullify the traditional problems (difficult and expensive deployment and maintenance) associated with high-interaction systems. We are currently using Nepenthes [3] for the low interaction sensors and a Gen-III Windows honeynet [4] for the high-interaction. An HoneyWall [25] 'Roo' gateway, itself running in a VM (Virtual Machine), monitors incoming and outgoing traffic to the high-interaction honeynet. It is important to limit outgoing traffic, to avoid being part of an attack or a denial-of-service and thus preserving legal liability [11].

The Windows honeypots are automatically rebuilt twice a day from a clean snapshot. Before the rebuild, their disk contents are analyzed: we register the differences with a clean installation and send to the gateway the new executables found — potential malware samples — for further analysis. The deployment and rebuilding of VMs is currently implemented with the VirtualBox command-line utility (*VBoxManage*). Our rebuild script powers the VM off and mounts the virtual disk on the host system; the directory tree is then compared *post-mortem* to a clean image using the standard Unix tool *diff*. We found this procedure to be the most reliable, although rather expensive. Preliminary testing of client-based collection tools, such as MwWatcher and MwFetcher from the HoneyBow project [24], showed poor reliability and a large number of missed samples. Using the VirtualBox snapshotting features was also not an option — snapshots are stored in a proprietary format and they cannot be easily mounted for analysis. Once the samples have been collected, we clone the clean image into a new virtual disk (using the built-in VirtualBox cloning facility) which is then attached to the VM, replacing the old one. Our script finally restarts the machine, which becomes ready for a new collection round. The collected samples are fed to a Python script, which submits them to the gateway with a properly formatted HTTP request.

The honeynet structure is shown in Fig. 2. Linux software bridging [26] interconnects HoneyWall and the Nepenthes sensors with the darknet; the Windows honeypots are connected through a VirtualBox internal network (an isolated virtual switch) to HoneyWall internal interface.

3.2 Core infrastructure

Once malware samples have been acquired, they must be processed, stored and analyzed. The life cycle of a malware sample in HIVE is shown in Fig. 3. The gateway exposes a web service to receive samples from the sensors; every sample has some metadata attached (file details, name of the sensor, date of collection

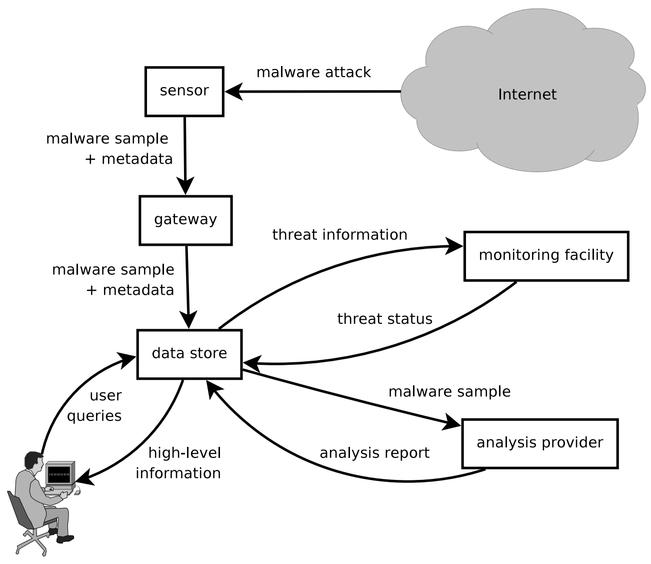

Fig. 3. Life cycle of a malware sample in HIVE.

and cryptographic hash). We have currently implemented the web service using a PHP script which receives data via HTTP POST.

A simplified version of the Entity-Relationship diagram for the database schema is shown in Fig. 4. We chose to rely on PostgreSQL 8.3, an Open Source object-relational DataBase Management System (DBMS): it supports most of the major SQL:2003 features [28], including referential integrity, transactions, views, stored procedures and triggers. We made extensive use of foreign keys constrains to ensure data integrity and implemented most of the analysis logic using stored procedures and triggers, in a mixture of pl/pgSQL and pl/Python.

Our data model is centered around *samples*, which are uniquely identified using their MD5 digest and stored in binary fields in the database. The addition of a new sample triggers its submission to external services, which analyze it and return a *report*. We use the reports to categorize the samples and extract useful data.

HIVE currently relies on the external services offered by Anubis [12, 5] and CWSandbox [23, 32] for the analysis of malware samples. Both services run the malware in a controlled environment, tracing its actions and logging network connections to provide a *behavioral analysis*. We use CWSandbox, which provides a machine-parsable XML report and a PCAP [15] network trace, as our main provider and rely on Anubis (whose reports are human-readable web pages) for verification and human cross-checking. CWSandbox provides also a VirusTotal [10] report we use to identify the malware according to antivirus classification. Our infrastructure can be integrated, in principle, with any analysis service available, either in-house or outsourced; we are currently evaluating integration with Joebox [13] and Norman SandBox [16].

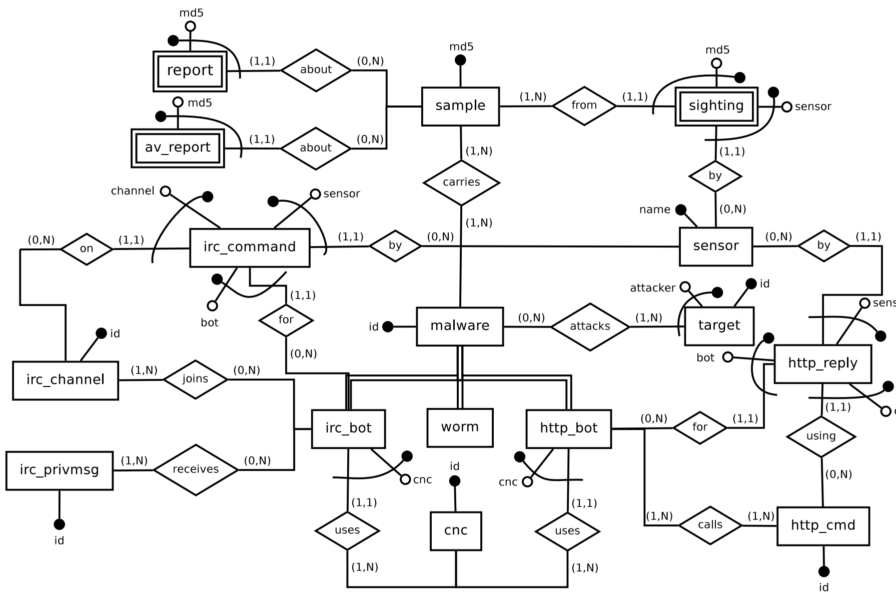

Fig. 4. Simplified Entity-Relationship diagram for HIVE database.

We have also implemented a pre-filtering stage, which allows HIVE to preliminary screen the samples at gateway or analysis layer for known malware, thus reducing the load on external services. The analysis currently relies on the Open Source antivirus scanner ClamAV [21], but could be extended to support other antivirus engines.

3.3 Data analysis and active monitoring

Our database schema currently accounts for IRC and HTTP centralized botnets and generic network worms. We make an extensive use of database views to aggregate data and present it in a coherent fashion. Views allow also to disclose useful data without compromising the necessary anonymity on the location of honeypot sensors and attack targets. We currently lack a full-fledged reporting interface: data analysis is currently done directly querying the database.

We wrote a Google Maps mashup to graphically show the geographic distribution of botnet C&C (Command and Control) centers; the centers' IP addresses were geolocated using MaxMind GeoLite City [14]. This example shows how easy is to access data and mangle it in the needed way.

Using the data collected during analysis we have all the information needed to connect to the botnet C&C and impersonate a bot. To study IRC botnets, we extended the Infiltrator Open Source software [8] by Jan Göbel to interface with our database. Infiltrator connects to the C&C and logs every communication on the channel, providing insight in the botnet operations and targets. To achieve a similar analysis for HTTP botnets, we wrote httpmole, a Python program which periodically connects to the command center and reads its reply. Since the reply is arbitrary and different botnets use different ad hoc control software, some form of fingerprinting is needed to be able to parse these messages in a useful way. While the software is still not ready for production, we are currently working in this direction.

3.4 Caveats and pitfalls

The use of virtual machines for the honeypot implementation opens them to a risk (albeit small) of being detected. Virtualization detection is rather easy and several works have been published [11]; nevertheless, we are not aware of any widespread malware checking for a virtualized environment. As studied by [17], virtualization may pose a security risk: if an attacker is able to exploit a weakness in the VM software, he may be able to execute arbitrary code on the host system, potentially breaking into it. An automated periodic rebuild of the host systems would address this concern. On the other hand, the eventual compromise of a honeypot sensors is not a catastrophic event: while the sensor could send bogus data, it has no way to directly access the database. In case of a suspect break-in, it is rather trivial to setup the gateway to ignore traffic from a specific sensor until the situation is investigated.

The strategic weakness of every honeypot is its secrecy: if an attacker were to discover the honeypot's address, he could blacklist it or send targeted attacks trying to disable, poison or subvert it. As shown by [34], honeypots can potentially be detected by specially crafted malwares; refer to [6] for a HoneyWall targeted example. While there are no reports of current malware in the wild exploiting such techniques, this is a legitimate concern.

To extract meaningful statistics, it's important to have an even distribution of honeynets, both in geographical and in IP address space. In order to acquire a good understanding of Internet threats a very large number of sensors would be necessary. On the other hand, a global knowledge must be completed by a good understanding of local malicious activities — as shown by [18], it's essential to deploy geographically local sensors.

The critical point of HIVE is the database — the DBMS server represents a single point of failure, which can be overcome using replication techniques to duplicate it on multiple machines. Slony [27] is an Open Source replication system for PostgreSQL which could be useful for this purpose.

4 Preliminary results

We built a simple testbed system to evaluate HIVE feasibility. We used only two physical machines for simplicity: one ran the virtual honeynet, the other implemented the gateway and the data store. Our honeynet was composed of

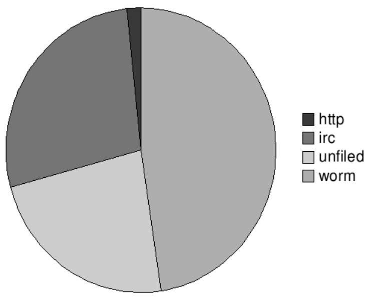

Fig. 5. Distribution of acquired samples by type.

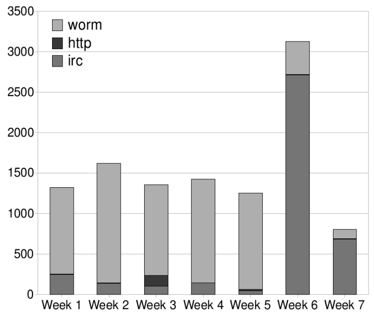

Fig. 6. Weekly collected samples per type.

two Windows systems and a Nepenthes sensor, on a small darknet of three previously unused IP addresses on a commercial network. We provided the high-interaction honeypots with 2000 Server and XP Professional, both unpatched and with default configurations. A vast majority of malware targets these Microsoft systems: XP is probably the most widespread client system, while the presence of 2000 Server allows us to capture samples attacking server-grade services (such as IIS).

During a month of operation, we detected more than 14,000 malware samples (about 13,000 unique), classified as in Fig. 5. It looks like HTTP bots are a negligible part, compared to other threats. The 'worm' samples — about a half of the total — are due to the Allaple polymorphic worm, which spreads through Microsoft systems using network share and buffer overflow vulnerabilities [20]. The 'unfiled' samples are currently not classified: they may be corrupt samples (which are screened but currently not included in results) or types of malware still unknown to our system or to the analysis services we rely on.

Fig. 6 shows the weekly samples breakdown by threat type. There has been a spike in the sixth week, due to a specific IRC bot, which spread over 2,000 samples on one honeypot, all tied to a single C&C center. We also noticed a slight decrease in Allaple infections in the last two weeks; we would need to collect more data in the following weeks to be able to confirm the trend. Due to the strong locality of our sensors, we would require data from other honeynets to be able to draw conclusive results.

In Fig. 7 and Fig. 8 we present the preliminary results from our monitoring infrastructure. We tracked about 50 different IRC botnets for two weeks, logging the activity on the control channel. The majority of botnet control is performed with private messages (PRIVMSG), and most commands issued are network scans and botnet expansion.

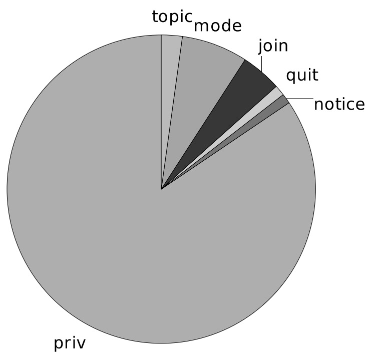

Fig. 7. Distribution of the IRC commands issued on the controlling channels.

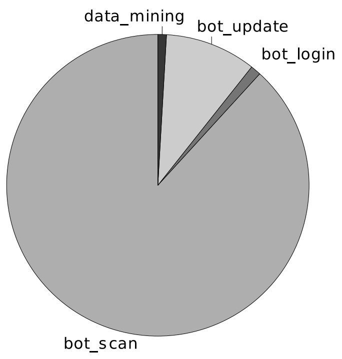

Fig. 8. Breakdown of IRC commands by type.

5 Conclusions and future works

We have proposed a new three-stage open honeypot architecture for malware data collection and analysis. We believe our solution, if widely deployed, could significantly ease the sharing of collected data. Our architecture is fully open: anyone can implement it using Open Source software.

In the future, we plan to integrate our database with the attack information collected by HoneyWall, to correlate attack vectors with malware samples. We are also investigating PE Hunter [31] as a possible additional sensor. Our honeypot management tools currently rely on the VirtualBox command-line interface; we plan to rewrite them to use the web service facility introduced in VirtualBox 1.6.0.

Our database schema is currently focused on centralized botnets; there is no support for P2P botnets [9], which are an increasing threat. We plan to add at least some basic support for P2P botnets in the future.

HIVE currently has a strong dependency on CWSandbox — it's our main source of analysis data on captured samples. In the future, we plan to integrate more tightly several other analysis services, to provide a level of redundancy and cross-checking on analysis results and spread the load on multiple systems.

Finally, we will write a reporting interface to expose interesting data and trends in a user friendly way — currently, obtaining statistical data requires querying the database using SQL. We believe that the easy availability of aggregate data on malware threats will greatly help developing new countermeasures.

Availability

Implementation details for the HIVE platform, the database DDL and our programs source code are all available at `http://netlab-mn.unipv.it/hive` or contacting the authors. Future developments and data reports will be published at the same location.

References

1. The honeynet project. `http://www.honeynet.org`.
2. The mwcollect alliance. `http://alliance.mwcollect.org`.
3. P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The Nepenthes platform: An efficient approach to collect malware. In Springer, editor, *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection (RAID)*, pages 165–184, Sept. 2006.
4. E. Balas and C. Viecco. Towards a third generation data capture architecture for honeynets. In *Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005. Proceedings from the Sixth Annual IEEE*, pages 21–28, June 15–17, 2005.
5. U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2(1):67–77, Aug. 2006.
6. M. Dornseif, T. Holz, and C. N. Klein. NoSEBrEaK - attacking honeynets. In *Information Assurance Workshop, 2004. Proceedings from the Fifth Annual IEEE SMC*, pages 123–129, June 2004.
7. J. G. Göbel. Amun: Python honeypot. `http://amunhoney.sourceforge.net`.
8. J. G. Göbel. Infiltrator v0.1. `http://zeroq.kulando.de/post/2007/11/15/infiltrator_v01`, Nov. 2007.
9. J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: overview and case study. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 1–1, Berkeley, CA, USA, 2007. USENIX Association.
10. Hispasec Sistemas. Virustotal. `http://www.virustotal.com`.
11. T. Holz and F. Raynal. Detecting honeypots and other suspicious environments. In *Information Assurance Workshop, 2005. IAW '05. Proceedings from the Sixth Annual IEEE SMC*, pages 29–36, June 2005.
12. International Secure Systems Lab. Anubis: Analyzing unknown binaries. `http://anubis.iseclab.org`.
13. Joe Security. Joebox. `http://www.joebox.org/`.
14. MaxMind. GeoLite City. `http://www.maxmind.com/app/geolitecity`.
15. S. McCanne, C. Leres, and V. Jacobson. libpcap. `http://www.tcpdump.org`, 2008.
16. Norman. SandBox information center. `http://sandbox.norman.no/`.
17. T. Ormandy. An empirical study into the security exposure to hosts of hostile virtualized environments. Technical report, Google, Inc., Apr. 2007.
18. F. Pouget, M. Dacier, and V. H. Pham. Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference, 29-30th March 2005, Monaco*, Mar. 2005.

19. M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 41–52, New York, NY, USA, 2006. ACM.
20. Sophos. W32/Allaple-B. `http://www.sophos.com/security/analyses/viruses-and-spyware/w32allapleb.html`, 2008.
21. Sourcefire, Inc. Clam antiVirus. `http://www.clamav.net`.
22. SUN Microsystems. VirtualBox. `http://www.virtualbox.org/`.
23. Sunbelt. Cwsandbox. `http://www.cwsandbox.org/`.
24. The Artemis Team. HoneyBow. `http://honeybow.mwcollect.org/`.
25. The Honeynet project. Know your enemy: Honeywall cdrom roo. `http://www.honeynet.org/papers/cdrom/roo/index.html`, Aug. 2005.
26. The Linux Foundation. Net:bridge. `http://www.linux-foundation.org/en/Net:Bridge`.
27. The PostgreSQL Global Development Group. Slony-I: enterprise-level replication system. `http://slony.info/`.
28. The PostgreSQL Global Development Group. PostgreSQL 8.3.1 documentation. appendix D. SQL conformance. `http://www.postgresql.org/docs/8.3/static/features.html`, Mar. 2008.
29. Trend Micro. 2007 threat report and forecast. Technical report, Trend Micro, 2007.
30. T. Werner. Honeytrap. `http://honeytrap.mwcollect.org/`.
31. T. Werner. PE hunter. `http://honeytrap.mwcollect.org/pehunter`.
32. C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security & Privacy Magazine*, 5(2):32–39, Mar./Apr. 2007.
33. J. Zhuge, T. Holz, X. Han, C. Song, and W. Zou. Collecting autonomous spreading malware using high-interaction honeypots. In *ICICS 2007*, pages 438–451, 2007.
34. C. C. Zou and R. Cunningham. Honeypot-aware Advanced Botnet Construction and Maintenance. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 199–208, Philadelphia, PA, 2006.