Medium Interaction Honeypots

Georg Wicherski

April 7, 2006

Abstract

Autonomously spreading malware has been a global threat to the Internet Community ever since the existence of the Internet as a large-scale computer network. A specialization of this threat are botnets; recent trends towards commercialization of botnets made the situation even worse. This document outlines the weaknesses of different existing approaches to catch malware – especially bots – and shows how Medium Interaction Honeypots solved these problems. It evaluates the success of Medium Interaction Honeypots so far and additionally points out some other related work.

1 The Problem: Botnets

One of the biggest problems the Internet is facing today – besides spam – is autonomously spreading malware. Some malware was written solely for proof of concept or education of the author, other malware was written with solely destructive intentions in mind. The biggest threat is however posed by remotely controllable backdoors. They not only allow commercial industry espionage on a highly advanced level, recent threats have become critical even to the end customer's computer.

Controllable networks of many infected nodes are currently referred to as *botnets* [1] as most of these are even today still based upon the IRC control and an early term for non-human, controllable IRC servants (even though not malicious) was *bot*. However, new protocols are now emerging for command and control of such botnets, with the most widespread alternative to IRC being HTTP. Another interesting trend is to use DNS as command and control protocol, although this is not in widespread use yet.

Botnets pose a severe threat to today's Internet community for two main reasons: first of all, the sum of resources available by a single botnet is so immense that they can cause severe damages. A botnet of 5,000 DSL 6MBit bots with a common upstream of 576 kbit/s has a total theoretical bandwidth of 2812.5 MBit/s. Now that there are botnets controling up to 50,000 hosts which yields to a practical bandwidth of 20 GBit/s, it is obvious that any webhoster or even upstream provider can be taken down with a Distributed Denial of Service attack. This type of abuse through botnets was the most common from 1997 through 2003 as it did not require any real technical sophistication. Another critical risk created by the control of so many resources is the sending and delivering huge amounts of spam.

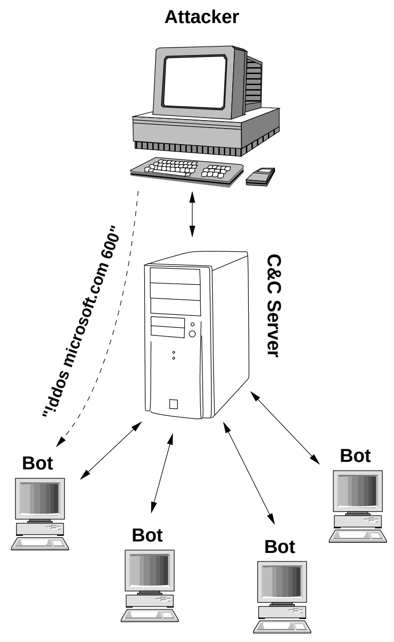

Figure 1: Typical IRC Botnet layout

The other reason is that botnets offer great potential to steal personal data as each bot is a local backdoor on one single user's computer. Bots nowadays usually carry a keylogger with them and of course allow to install additional backdoor executables, e.g. for intercepting bank transfers before they actually reach the bank's webserver. Thus, botnets allow for identity theft of end-customers; credit card trading is already a well established business between bot herders and their customers.

Therefore it is of great importance to the Internet's security to track and then shutdown botnets. To achieve these goals, we need certain information about the botnets. In the following, we will evaluate how this intelligence can be obtained.

1.1 Netflow based detection

Netflow based detection can be implemented by either observing uplinks of big networks on the transport or the application layer. While the transport layer observations do not require as much resources as application layer observations, they do lead to less information.

Although it is possible via statistical analysis to get an overview of which nodes are infected and attempt to spread, it is not possible to get detailed information about new botnets or the exploits the bots use to spread. Furthermore, statistical analysis requires huge datasets and is prone to false positives. The outbreak of a new autonomously spreading worm cannot be determined at the outset of the epidemic but only when already a large set of hosts is infected and generate sufficient noise to be recognized.

Application layer observations can provide much more information, but on the contrary require detailed processing for each individual protocol and additionally huge and therefore expensive amounts of computing power to observe big data links. However, application layer observations do not only provide information about the infected nodes but also about the command and control structure in the case there is a control structure and not only a purely static payload. The required knowledge about the protocol and the flowing network data make netflow based detection useless in case of 0day intrusion vectors.

Obviously it can be concluded that netflow based detection can be used as an additional measure to secure your networks but is not well suited to detect emerging botnets and find new command and control mechanisms. Deploying netflow based detection for botnet finding would only turn your production network into a super honeypot.

1.2 Honeypot based detection

There were already existing technologies to learn from attackers and to recognize new and recent trends in Blackhat activities. Honeypots have been well established since the late nineties and present a good way to obtain knowledge about current Blackhat technology. The ability to gain information about botnets and autonomously spreading malware with Honeypots will be discussed in the following.

1.2.1 Low Interaction Honeypots

Low Interaction Honeypots are similar to transport layer netflow observations in terms of botnet detection. Low Interaction Honeypots only simulate the transport layer of large networks on a single physical host, but can hardly be used to gain information on the application layer. Therefore when it comes to the detection of new botnets and learning about emerging malware technologies, the same restrictions apply here as with application layer based netflow observation.

1.2.2 High Interaction Honeypots

High Interaction Honeypots are real, vulnerable systems, often running in a virtual machine environment and behind a rate limiting firewall. Due to the nature of High Interaction Honeypots, they can be used to detect 0day attack vectors and automatically adapt to any new command and control protocol. Therefore at first glance, it seems well suited to detect botnets, since after infection, the honeypot even automatically connects to the command and control framework, e.g. a central IRC server. Therefore there is even no need to differentiate between bot acquisition and botnet monitoring.

However, there are some major disadvantages in using High Interaction Honeypots for the capture of autonomously spreading malware and later on observing the control facilities. First of all, most exploits used by malware utilize certain system dependent data and are not reliable across all OS versions and patch levels. This means that system services can be crashed by wrong offsets and such used in exploits, ultimately resulting in a reboot of the honeypot. Therefore, performance is extremely limited as for example during the Sasser spread, a Windows 2000 Honeypot usually rebooted at least every 3 minutes.

Additionally it becomes very unhandy to run High Interaction Honeypots once you need to extract the sample, e.g. in order to submit it to a virus researcher. Most recent malware tries to hide using rootkit-like techniques or is simply that abusive so it slows your honeypot down so much, you cannot handle it. Sometimes it is even hard to obtain a sample from a clean directory view when the virtual harddrive is mounted outside of the virtual machine.

Even if one argues that High Interaction Honeypots can be used for immediate observation of the command and control structure since they automatically connect after infection, you would need an increasing amount of honeypots. Each honeypot could only be infected once and could not be reverted before the botnet was taken down or it was decided that monitoring is not necessary anymore.

2 Medium Interaction Honeypots

Medium Interaction Honeypots try to combine the benefits of both approaches in regards to botnet detection and malware collection while removing their shortcomings.

The key feature of Medium Interaction Honeypots is application layer virtualization. These kind of honeypots do not aim at fully simulating a fully operational system environment, nor do they implement all details of an application protocol. All that these kind of honeypots do is to provide sufficient responses that known exploits await on certain ports that will trick them into sending their payload.

Once this payload has been received, the shellcode is extracted and analyzed somehow. The Medium Interaction Honeypot then emulates the actions the shellcode would perform to download the Malware. Therefore the Honeypot has to provide some virtual filesystem as well as virtual standard Windows download utilities. The Honeypot can then download the Malware from the serving location and store it locally or submit it somewhere else for analysis.

2.1 Existing Implementations

Several implementations of this principle have been recently published, some being Open Source software and others being strictly commercially distributed. We will discuss the most widespread and popular implementations in the following.

2.1.1 mwcollectd

mwcollectd was the first Open Source Medium Interaction Honeypot available. It was published together with the paper ”Sammeln von Malware in nicht-nativer Umgebung” [2] by Georg Wicherski as the underlying proof of concept. In September 2005, version 3.0 – a complete rewrite – was released which was even more modular and improved some shortcomings of the old mwcollectd version.

mwcollectd was merged with the nepenthes project in February 2006 to reduce redundant development and join forces. Since both projects were already licensed under the GNU Public License, there were no further licensing issues. The nepenthes code was used as new code-base as it contained more vulnerability modules and therefore had a higher success rate deployed in the wild. Hence the project name of nepenthes was used further.

2.1.2 Multipot

Multipot [3] is a graphical Medium Interaction Honeypot for Windows and therefore has limited scalability for distributed and dedicated deployment. Initially released in July 2005 by iDefense, it was limited to catching Bagle variants, but has developed somewhat since then. It has the proof of concept character, the first mwcollect versions had.

2.1.3 nepenthes

nepenthes was released in June 2005 by Paul Bächer and Markus Kötter, implementing the design of mwcollect 3.0, which was already published by Georg Wicherski at that time, but not yet implemented. Both projects co-existed until the above mentioned project merge in February 2006. nepenthes is now the de-facto industry standard in Open Source Medium-Interaction-Honeypots.

nepenthes is now published by the mwcollect.org development crew, consisting of all three, Paul Bächer, Markus Kötter and Georg Wicherski. Since new vulnerabilities, shellcodes and bots are emerging every day, development is steadily continued.

3 Inside View on nepenthes

We will now have a closer look at the internals of the nepenthes daemon and it's modules. The nepenthes team aimed at developing a standalone console daemon designed to be deployed on different UNIX sensors distributed about a diverse IP space. Therefore the configuration is done via files in a configuration directory and minimally the command line. The daemon is POSIX compliant and *autotools* is used for compilation, thus it runs on a wide variety of UNIXes, ranging from Gentoo Linux to Solaris SPARC-64.

nepenthes is designed as a totally single-threaded daemon. All I/O, including network receives and send are buffered in the daemon itself. As long as there is no incoming data, the main-loop sleeps using the *poll* system call. Because of this, the daemon does not consume any resources as long as there is no malicious input given. A single-threaded design additionally eliminates the risk of deadlocks and the need for mutexing, which would have become very complex if the whole nepenthes daemon was multi-threaded.

3.1 Modularization

The nepenthes program is divided into the core daemon and multiple modules performing different tasks. This modularity ensures an easily maintainable source code of the program. Additionally it allows turning off certain features or parts at runtime or in the configuration files.

The heart of the program is the actual daemon itself, linked as an executable ELF binary. It provides the main function wrapping around an OOP daemon core, which is responsible for loading all modules, running the main loop and most importantly providing the network interfacing API as well as inter module communication facilities.

To satisfy overlapping port binding requirements, the nepenthes daemon creates it's own wrappers around the UNIX socket functions. Therefore multiple vulnerability modules can subscribe to connections on the same port and have a look at the incoming data until one module decides that this data is meant to exploit a vulnerability specifically simulated by that module. The other modules will then be dropped from the data subscription.

Modules are dynamically linked libraries (*.so* on most UNIXes), providing a single startup and cleanup function, which wrap around the creation and deletion of an OOP module class, respectively. There are different types of modules, discussed in the following.

3.1.1 Vulnerability Modules

Vulnerability Modules pose the entry point of malware into the Honeypot. They provide the simulation of certain vulnerable services to the extent that known exploits need to be lured into sending their payload.

Once this payload is received, the shellcode is extracted and passed to a shellcode handler module. Since the exploits are known, it is not required to utilize statistic means to find the shellcode. Often it is sufficient to do some offset calculation or look for specific cookies.

3.1.2 Shellcode Handlers

These modules analyze given binary dumps of the shellcodes coming over the wire, independent from the exploit they were delivered with. How the shellcode handler modules determine the actions, the shellcode would have performed on a real machine, is completely up to the modules' authors. The core simply provides them with a set of functions used to emulate the possible actions a shellcode could take.

Opening of a shell, downloading of files and similar actions are then directly mapped to the corresponding modules by the dispatching mechanisms described below.

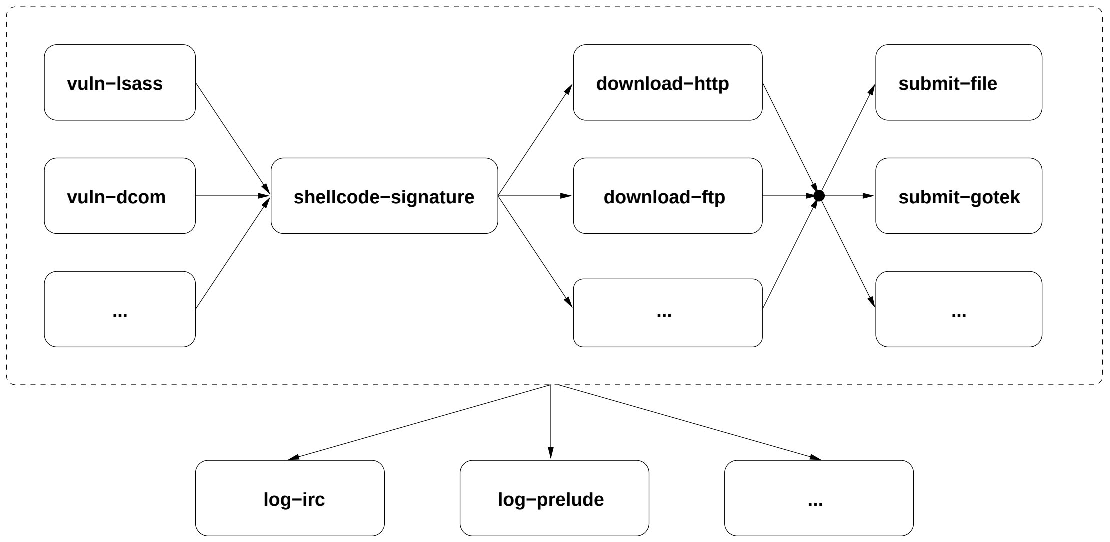

Figure 2: Module call graph for common exploitation attempt

3.1.3 Shell Emulations

Most malware spreading in-the-wild today is so unsophisticated that it still uses plain bind or reverse shell shellcodes, therefore the core provides a rudimentary shell emulation with a virtual filesystem. Each exploitation attempt triggers a new instance of this filesystem to prevent corruption of the honeypot in run-time by virtual filesystem corruption.

Shell Emulation modules provide command emulations for the virtual shells. The implementation of a Windows shell in the core for example only contains rudimentary commands like *echo* or *START*. Download clients like Windows' *ftp.exe* command line client have to be emulated by such a Shell Emulation module. However, it is important to note that the module does not perform the actual download, it only emulates the executable's run-time behavior and extracts the information from the given input. The actual download is then passed to a Download Module.

Some malware also requires relatively complex emulation of the Windows shell to infect the honeypot correctly as can be seen in figure 3.

3.1.4 Download Modules

Download Modules register themselves as protocol handler for a specific type of URLs at the core. They then get informed once a Shell Emulation or Shellcode Handler detects the download of an URL and download the specified URL. Such URLs are not only limited to *http://* but can also contain uncommon protocols as *tftp://* or *rcp://*.

Certain download mechanisms like such as the non-standardized protocol *linkbot* uses (*link://* protocol handler) require additional information about the transfer, as in this case an authentication key. This information is then encoded in the URL as well.

3.1.5 Submission Modules

Once a binary has been downloaded, it is passed to the submission modules for further processing. The most intuitive and obvious action to take upon a binary is to store it on the hard drive for later analysis – this is what the *submit-file* module does.

Another submission module contained in the nepenthes main distribution automatically submits the downloaded binary to the Norman Sandbox Live Demo on the Internet. A virtualized sandboxing is then performed on the binary and a report, what the malware would do on a real computer is sent to a specified email address.

Furthermore, there is a submission module that directly submits downloaded samples to the *mwcollect Alliance*, which is discussed in detail further below. A similar module submits binaries to a central

```
cmd.exe /c echo open static984.amdwebhost.com 21 >appmr.dll &
  user majic 6AVlEMzJ >>appmr.dll &
  echo binary >>appmr.dll &
  echo get asn.exe >>appmr.dll &
  echo bye >>appmr.dll &
  ftp.exe -n -s:appmr.dll &
  del appmr.dll &
  asn.exe
```

Figure 3: Example shell command set used by a rBot

server via XML-RPC, but also provides additional information about the download process.

3.1.6 Log Modules

Log modules implement the very simple task of logging to different targets than a file on the hard disk – the hard disk logging is part of the core. Possible targets at the moment are a syslog daemon and an IRC channel. Both provide convenient ways of observing your sensors in real-time.

3.1.7 Miscellaneous Modules

Since all kinds of modules appear the same to the core and it exports all its interfaces to all modules, there is no need to restrict the design of a module to one of the above types. On the one hand, multiple types can be combined in one module. A shellcode handler for a shellcode that only appears together with one certain exploit could be handled in the appropriate vulnerability module. On the other hand, modules not fitting in the above scheme at all could be created.

One example is the *portwatch* module contained in the current nepenthes main distribution. Its sole purpose is to log connections to a certain port in order to determine, whether it is required to create additional vulnerability modules for these ports based upon demand.

Some other institutions wrote non-public modules which automatically add hosts actively spreading malware to a firewall blacklist, ultimately turning nepenthes into an Intrusion Prevention System.

3.2 Events and Dispatchers

The core glues all loaded modules together by providing different means of inter module communication. Since a module cannot know which other modules are loaded at run-time, dynamic relinking at module load is not possible.

The generic way for inter module communication is the *EventManager* class, the nepenthes core provides. It allows modules to register for the handling of certain event types or a new, module defined event type. Additionally, modules can fire events of a predefined or already registered types to the event manager. It then distributes these events to all handlers masked for the event type. Events can contain arbitrary data, including binary data.

Dispatchers are a more specialized way for inter module communication. They are used to delegate analysis of certain tasks to other modules. Therefore there exists a dispatcher for shellcodes, one for downloads and one for submissions. Internally they work similar to the event dispatcher, without specific event types.

This differentiation between these similar models originates from the initial mwcollectd design where there was no event manager yet. Future plans include the substitution of the three dispatchers with an event type for each of them.

3.3 Case Study: rBot Infection

In the following, we will see how a classical – maybe modded – rBot tries to infect our nepenthes honeypot. The infected host first establishes a TCP connection to port :445 of our honeypot to exploit the PNP vulnerability, *Zotob.A* became famous for as well. It sends a shellcode as payload, which will call the `CreateProcess` function to execute arbitary commands packed into the shellcode.

The bot issues the commands shown in Figure 3. It first creates an *ftp.exe* instruction file by issuing Windows shell echo commands and redirecting them into a file. It then launches *ftp.exe* in silent batch mode to download the bot's binary. All this information is turned into the simple URL `ftp://majic:6AVlEMzJ @static984.amdwebhost.com:21/asn.exe` by the shell emulation modules, which support even more cryptic shell command sequences.

nepenthes then connects to the given FTP server and downloads the specified binary. After the transfer has been completed successfully, the binary is passed to the submission modules. Since the *submit-norman* module sent the bot to the Norman sandbox, we soon have more information about this malware specimen in our inbox. After a short read over the report, it turns out that this bot connects to the IRC server `dynamic1086.amdwebhost.com` on port 6667 and joins the channel `#asn3` with the keyword `NdrVyk5D`.

4 Evaluation

Medium Interaction honeypots have proved to be very efficient in the collection of autonomously spreading malware. The vast majority of malware spreads using similar patterns, including common exploits, common shellcodes and common shell scripts. There have been no known counter reactions towards medium interaction honeypots yet.

Medium Interaction Honeypots can be detected relatively easily. Although DNAT could make it possible, it is very rare for a machine to serve both Windows RPC ports and a Linux Apache at once. Additionally, nepenthes sensors can be fingerprinted relatively easily by testing for non emulated service portions as only the necessary parts to receive a payload are implemented. Another possibility, although risky when fingerprinting production environments, would be to exploit a host twice, leaving a trace on the filesystem and checking for it upon the second exploitation.

nepenthes has proved to be stable and has a good performance in deployments of large networks, ranging up to a single machine serving a whole /18 subnet. The majority of vulnerabilities currently being exploited in-the-wild can be emulated by nepenthes. Additionally, the daemon can be used in a lot of different ways due to its modularity, ranging from plain collection to hard drive over geographical mapping of attacks to Intrusion Prevention mechanisms.

The daemon is already deployed on several sites on the Internet and is considered to be stable production software.

5 Related Work

Several different projects have emerged around Medium Interaction Honeypots and are now related to the nepenthes project. Via modularization and customizable nature, several different web interfaces have been developed individually by owners of nepenthes sensors. Some law enforcement organizations and other governmental organizations are known to deploy nepenthes sensor on the Internet and in internal networks.

5.1 mwcollect Alliance

Initially derived from the mwcollectd project, the mwcollect Alliance [4] is a non-profit organization aiming at malware collection. This closed community deploys nepenthes sensors on the Internet and combines this data together with other malware sources in a central database. Rough analysis of the data is performed on the Alliance's server in real-time.

The database's license ensures that members of the Alliance do not distribute any samples obtained from the database to any other third parties. Membership to the Alliance can be applied for via email, at which point a careful examination of the applicant is then performed.. Membership is free, but limited to people or organizations steadily providing the Alliance with new samples, e.g. by deploying a nepenthes sensor.

5.1.1 gotekd Daemon

gotekd is the daemon handling the binary submissions of nepenthes sensors or the respective command line client on the Alliance's server in real-time. It was developed by Georg Wicherski and is considered stable now as well. It feeds the Alliance's MySQL database with samples and additional information.

The underlying protocol is binary and uses a scheme similar to *HMAC* with *SHA-512* for authentication at the server. Existence of binary is also tested based upon the *SHA-512* hashes of the binaries. Once a new binary is detected by one of the sensors, it is streamed to the server over an additional data connection.

The daemon is published under the Gnu Public License and freely available to the public.

5.1.2 Webinterface

The Alliance utilizes a closed internal webinterface for communication and coordination as well as data aggregation. This webinterface is written in PHP and MySQL and based upon wordpress, DokuWiki and custom code for the malware database. Its sourcecode is not publicly available.

5.2 Surfnet IDS

The Surfnet IDS [5] project aims at creating a simple to manage, distributed intrusion detection system for the Dutch university ISP Surfnet. nepenthes is deployed on one single physical workhorse and sensors are linked to it by an OpenVPN tunnel. Therefore the sensor can be contained on a single USB stick to boot from and does not change in configuration or in software as only the central nepenthes sensor has to be configured and updated.

6 Acknowledgments

Thanks go to Markus Kötter and Paul Bächer for technical review as well as Andre DiMino for language review.

This paper was written as a part of a school work in computer science, supervised by Werner Voss.

References

[1] Honeynet Alliance (2005); Know your Enemy: Tracking Botnets; http://www.honeynet.org/papers/bots/

[2] Georg Wicherski (2005); Sammeln von Malware in nicht-nativer Umgebung

[3] Multipot Honeypot; http://labs.idefense.com/labs-software.php?show=9

[4] The mwcollect Alliance; https://alliance.mwcollect.org/

[5] Surfnet IDS Project; http://ids.surfnet.nl/