Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation

Jan Goebel
Center for Computing and Communication
RWTH Aachen University, Germany
goebel@rz.rwth-aachen.de

Thorsten Holz
Laboratory for Dependable Distributed Systems
University of Mannheim, Germany
thorsten.holz@informatik.uni-mannheim.de

Abstract

In this paper, we describe a simple, yet effective method to detect bot-infected machines within a given network that relies on detection of the communication channel between bot and Command & Control server (C&C server). The presented techniques are mainly based on passively monitoring network traffic for unusual or suspicious IRC nicknames, IRC servers, and uncommon server ports. By using n-gram analysis and a scoring system, we are able to detect bots that use uncommon communication channels, which are commonly not detected by classical intrusion detection systems. Upon detection, it is possible to determine the IP address of the C&C server, as well as, the channels a bot joined and the additional parameters which were set. The software *Rishi* implements the mentioned features and is able to automatically generate warning emails to report infected machines to an administrator. Within the 10 GBit network of RWTH Aachen university, we detected 82 bot-infected machines within two weeks, some of them using communication channels not picked up by other intrusion detection systems.

1 Introduction and Motivation

Nowadays, remotely controllable computers (*bots*) are one of the most dangerous threats in the Internet, as they combine most of the malicious actions usually performed by worms, rootkits, and Trojan horses. Therefore, it is vital for the protection of hosts in the network to detect and defang infected machines in a very early stage. Currently, one basic action that is performed to stop a given botnet is to *disable* the communication channel for the bots by shutting down or changing the DNS entry for the Command & Control (C&C) server. By doing so, the bots can no longer connect to the C&C server and thus the botherder can no longer send commands to the infected machine.

However, the hosts stay infected and are in most cases still backdoored, allowing an attacker to reclaim the machine at any time. Thus, it is necessary to find a way to not only disable the communication channel, but to also detect and inform the owners of contaminated systems. Within a university environment, this is rather hard to achieve: this kind of networks have traditionally a rather open security policy. Students and faculty staff often have unlimited access to the Internet and firewalls are sometimes only used in the sensitives parts of the network, whereas most of the network is not very well controlled. In such an environment – and also in a lot of networks with higher security standards – mobile users pose a severe threat: these users have their own laptop and connect to the network via some kind of authentication, for example with a central VPN server. The problem is that the laptops are often not secured at all, for example important security patches are missing or the antivirus software is outdated. Such systems are often infected with some kind of autonomous spreading malware in the form of worms or bots. When the user now connects to the network, the malware tries to propagate further within the closed network and poses a threat to other users. Detecting and cleaning up one of theses devices is a hard task. In this paper, we present an accurate, yet simple method to *detect* bot-infected machines in a given network that relies on the fact that the bot contacts his Command & Control server (C&C server) directly after an infection. This communication channel has some unique features (e.g., similarity of nicknames used by the bot and characteristic substrings) that we try to detect.

The paper is outlined as followed: In Section 2, we introduce the background for our work on detecting infected machines. Section 3 presents a method based on n-gram analysis and a scoring system to detect infected hosts within a given network and we discuss the limitations in Section 4. Evaluation results are given in Section 5 and we conclude the paper in Section 6.

2 Background and Related Work

In this section, we introduce the basics of our detection mechanism and discuss related work.

2.1 Internet Relay Chat

Internet Relay Chat (IRC) is a concept that allows users to communicate with each other in real time [12]. There exist several separate networks of so called IRC servers, which provide users with a connection to IRC. These networks often have several thousand users online at the same time. The users connect via IRC client programs to a server on one of the networks (Figure 1). The server relays information to and from other servers on the same IRC network. Each of the different servers hosts a number of *different* chat rooms, called *channels*, which a user can join to discuss certain topics. Every user connected to an IRC server has its own *unique* username, called *nickname*. Conversations within the channels can either be private, client-to-client, or public, so that everyone in the channel can read the messages. The channel names can be freely chosen, but usually start with the character "#" or "&". Channels with the latter prefix are not shared by all IRC servers on the network, but exist only on a single server. Each user of the IRC network can create new private or public channels for others to join.

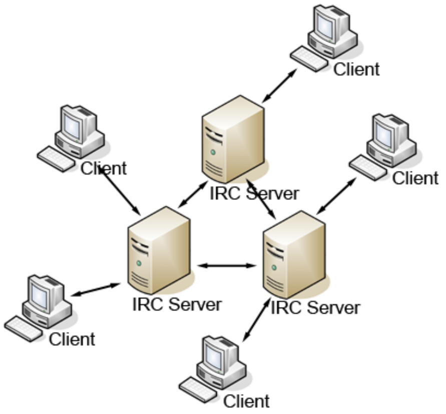

Figure 1: Setup of an IRC network

IRC is a common method to remotely command and control bots, which is described in more detail in the next section.

2.2 Bots and Botnets

The term *bot* is derived from the word *robot* and refers to a program which can, to some degree, act in an autonomous manner. A computer system that can be remotely controlled by an attacker is called a *bot* or *zombie*. Bots started off as little helper tools, especially in the IRC community, to keep control of a private channel or as a quiz robot, randomly posting questions. In the context of malware, bots are harmful programs, designed to do damage to other hosts on the network. Moreover, bots can be grouped to form so called *botnets*, consisting of several hundreds up to thousands of hosts, whose combined power can be utilized by the owner of the botnet to perform powerful attacks. One of these powerful attacks are *Distributed Denial of Service* (DDoS) attacks, which overwhelm the victim with a large number of service requests. DDoS attacks are described in more detail in the work by Mirkovic and Reiher [15]. Other abuses of bots can be identity theft, sending of spam emails and similar nefarious purposes [9].

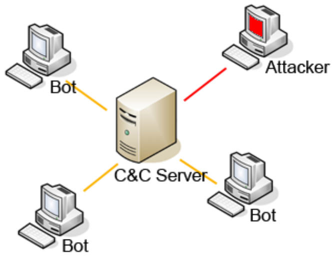

Figure 2: *Setup of a botnet with a central server for Command & Control*

To control a large group of bots, a *Command and Control* (C&C) server is utilized, which all zombie machines connect to and receive instructions from (Figure 2). A common method of an attacker to communicate with the botnet is to use IRC, which we described in the previous section. In this case, the infected machines automatically join a specific channel on a public or private IRC server, to receive further instructions. This could be for example to perform an attack against a specified victim or to scan certain network ranges for hosts with known vulnerabilities. It is not even necessary for a bot to join a channel, there are also bots which use private messages to receive instructions from the botnet owner. Thus, a connection to the C&C server suffices. Besides IRC, other protocols are used more and more by bots. For example, HTTP is more and more common as communication channel. In this case, the bot periodically polls the C&C server and interprets the responses as commands. Several bots also use Peer-to-Peer based protocols (see for example the analysis of Storm worm by Stewart [19]) to avoid a central C&C server and this form of communication could become more and more prevalent in the future.

Since bots are able to autonomously propagate further across the network and feature keylogging and backdoor functionalities as well, they can be seen as a combination of worms, rootkits and Trojan horses. A more detailed description of botnets and their attack features is provided in the work by the Honeynet Project [20].

2.3 Related Work

One of the earliest works related to our approach is by Kristoff [13]. In his presentation, he mentions that signs of rogue IRC server are suspicious nicknames, topic and channel names. We extend this idea and evaluate whether or not the similarity in nicknames used by bots can be used to detect an infected machine.

Botnets also commonly use the same IRC channel for bots. This observation is used by Binkley and Singh to detect suspicious IRC servers [5, 4]. They combine TCP-based anomaly detection with IRC-based protocol analysis and are able to detect botnets efficiently. The system collects statistics over a complete day and aggregates the collected information. In contrast, our method works near real-time and can detect an infected machine often earlier then our other intrusion detection system.

Chen presented a system that tries to detect botnet traffic at edge network routers and gateways [7]. This is similar to our approach since our system is also best deployed at these observation points. Chen presented preliminary statistics like mean packet length of IRC packets and the distribution of IRC messages like JOIN or PING/PONG, but did not give statistics about the success route of the approach. Strayer et al. use a similar approach to examine flow characteristics such as bandwidth, duration, and packet timing [21].

Livadas et al. use machine learning techniques to identify the C&C traffic of IRC-based botnets [14]. Their approach could be combined with ours since both are orthogonal: we use characteristics of the IRC protocol itself and similarity measurements to known botnets, whereas Livadas et al. observer characteristics of the communication channel.

Another approach to detect bot-infected machines is *behavior-based detection*. One characteristic of bots is for example that they are idle most of the time when they wait for a command from the botherder [16]. Moreover, bots would respond faster than a human upon receiving of a command. Racine proposed a system that tries to find such characteristics on Netflow traffic, but the system had a rather high false-positive rate.

3 Communication Channel Detection

Since bots pose a severe threat in today's Internet, we need to develop ways to detect infected machines within a given network. We often can not rely on common intrusion detection systems: bots can for example compromise victims via channels like email (malicious attachment to an email) or drive-by downloads (malicious websites exploiting a vulnerability in a browser), which are often not detected by these systems. In addition, bots can stay calm on an infected machine and only become active at certain dates or under specific conditions. Our approach focuses on detecting the *communication channel* between the bot and the botnet controller. This is the earliest step right after the infection, so we try to detect the compromised machine as fast as possible.

3.1 Motivation

All bots have one characteristic in common: they need a *communication channel* in order to receive commands or report status information to the botnet owner. This is the main differentiation between a *worm* and a *bot*: both kinds of malware propagate autonomously, but a worm does not offer a remote control channel to the attacker

Currently, the most common method for botnet herders to communicate with the zombie hosts and to issue commands is to utilize IRC servers for command and control. Bots connect to these servers and usually join a certain channel to receive instructions on how to proceed. However, there are bots which utilize other communication methods like HTTP or Peer-to-Peer protocols [19]. Since the detection method described here is based on IRC nicknames, the main focus lies on the IRC protocol. However, our method is also applicable to other protocols which have the property that at least some bytes in each message between bots and botherder stays constant. HTTP bots for example tend to have some common strings in the URL of the botnet server, thus they can also be detected using our method.

The main disadvantage when using IRC as communication source – from the botnet owner´s point of view – is that one loses control over the bots as soon as the central IRC server is not reachable anymore. Thus, a common method in botnet fighting is to shutdown known C&C servers to prevent infected machines from receiving further commands. Although the botnet is successfully disabled, the zombie hosts remain infected and vulnerable. Therefore, we use a different approach, which not only reveals the C&C server, but also the infected hosts. As a result, the owner of a contaminated machine can be informed and is able to clean it, before valuable information leaks or the host is compromised again. Additionally, we are able collect valuable information about the C&C server itself, with which it is possible to infiltrate and monitor the botnet prior to shutting it down.

The use of a standardized protocol like IRC allows an easy detection of hosts joining the IRC network, since it is well documented [12]. One of the first commands issued when connecting to an IRC server is *NICK*, followed by the nickname by which the host should be identified within the IRC network. As it is not allowed to have duplicate nicknames within the network, each bot has to join utilizing a different name. This is what we take advantage of when detecting bot infected machines. A common method used by bots to avoid duplicate nicknames is to concatenate a certain word with a random number. For example the Worm/Rbot.210944 [1] utilizes nicknames which are constructed as follows: *country abbreviation|nine-digit number* (e.g., `USA|016887436` or `DE|028509327`). Some other bots use the opposite approach and concatenate a random word to a constant number. The Worm/Korgo.F.var [2], for example, uses "`_13`" as a constant suffix, prefixed by a random number of letters (e.g. `bmdut_13`).

The principle behind our approach is simple: the nickname must contain a random component to avoid bots being unable to join the IRC network due to duplicate usernames. Besides the random part, bot names usually contain an additional constant part, which for example holds certain information about the kind of bot (e.g., `RBOT|XP|48124`), the kind of operating system running on the bot, or the location of the contaminated machine, like `DEU` for Germany or `CHN` for China. These constant parts of the nickname form a valuable starting point for the detection of bot infected machines.

3.2 Project Rishi

Since one of the first actions of a newly infected machine is to establish a connection to the botnet server to receive further commands, it is possible to detect a bot even before it performs any malicious actions. Therefore, *Rishi*, our proof-of-concept implementation, monitors captured TCP packets for the occurrence of one of the following IRC commands: *NICK*, *JOIN*, *USER*, *QUIT*, and *MODE*. The parameters given with these commands are extracted and stored to be further analysed by the program. The analysis focuses on the nickname we extracted, all other parameters are just stored to collect additional information about the botnet, e.g., for tracking purposes.

Figure 3 illustrates the setup of *Rishi* and the process of an attacker infecting a vulnerable machine, which in turn connects to the C&C server to receive further commands. Instead of monitoring the network traffic for malicious commands issued by the attacker, *Rishi* listens for the connections of infected machines to the IRC servers hosting the botnet.

Rishi is a Python script consisting of about 1700 lines of code, which receives its data from a running *ngrep* [17] instance. With the help of *ngrep*, we are able to filter certain network packets. With the following command, we can for example extract all network packets containing IRC-related information in the header:

```
# ngrep [...] 'JOIN$|$NICK$|$MODE$| \
    $USER$|$QUIT' 'tcp[((tcp[12:1] \
    \& 0xf0) $>>$ 2):4] = 0x4e49434b \
    and [...]'
```

Every captured packet is then further analysed by the script, which extracts the following information (if available):

- Time of suspicious connection
- IP address and port of suspected source host
- IP address and port of destination IRC server
- Channels joined
- Utilized nickname

For each IRC connection a *connection object* is created, which stores the above mentioned information, plus an additional identifier. The identifier consists of the source and destination IP address and the destination port. With the help of the identifier it is possible to update an already existing object with new parameters. For example, if a new channel is joined, no new object is created, but the existing one is updated. To minimize the amount of memory consumed by *Rishi*, the objects are stored in a queue. Objects which are updated move to the beginning of the queue, so they are not removed from the queue as quickly as objects which do not receive any updates. Additionally, objects belonging to connections which issue the *QUIT* command are removed from the queue. The basic concept of *Rishi* is illustrated in Figure 4. It shows the connection object, which stores the various information, the dynamic whitelist (Section 3.2.3) and blacklist (Section 3.2.4) to maintain nicknames and the queue containing a certain number of objects currently monitored.

3.2.1 Scoring Function

After an appropriate packet has been captured and all necessary information have been extracted, the gathered nickname is passed to an analysis function. The analysis function implements a *scoring function* in order to estimate whether or not a given suspicious host is infected with a bot or not. The function checks for the occurrence of several criteria, like for example suspicious substrings, special characters, or long numbers. For each successful test, a certain number of points is added to the overall score the particular nickname has already received. Currently the scoring function uses a rather adhoc approach based on experimental data, but we will

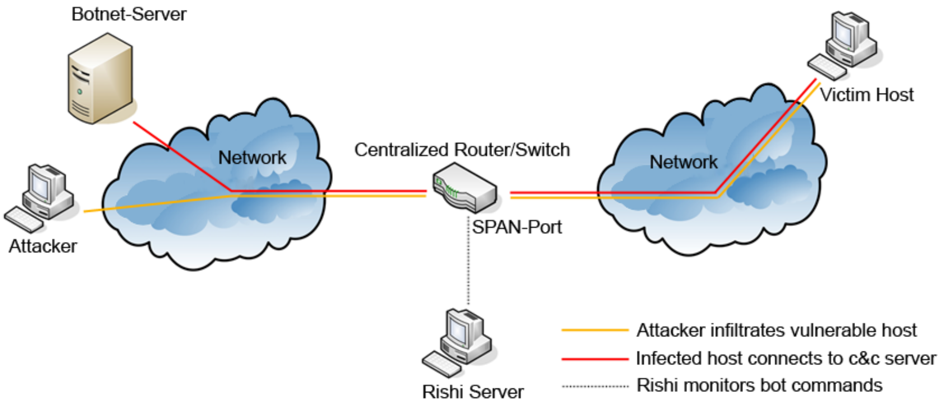
Botnet-Server
Attacker
Network
Centralized Router/Switch
SPAN-Port
Network
Victim Host
Rishi Server
Attacker infiltrates vulnerable host
Infected host connects to c&c server
Rishi monitors bot commands

Figure 3: Network setup of *Rishi*

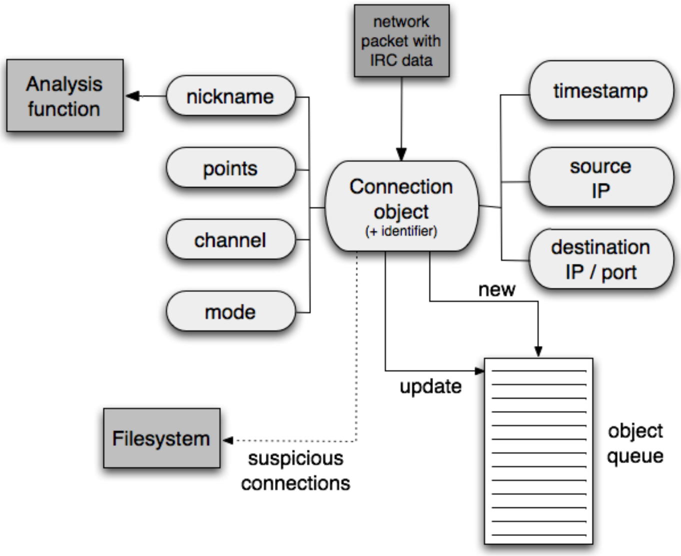

Figure 4: Basic concept behind *Rishi*, an approach to detect IRC characteristics of infected machines

further examine the influence of the scoring parameters to the detection rate in the future.

When the analysis is finished, the final number of points for the nickname is stored along with the other information in the connection object. The higher the score a nicknames receives, the more likely it is a bot infected machine trying to contact its C&C server. If a certain threshold is reached, *Rishi* triggers an alarm. In this case, the object is marked as a possible bot and all information about the connection are stored in a separate file on disk. In addition, we also generate a warning email containing all gathered information about the incident. This email is sent to one of the network administrators. Since we are still in the development phase, emails need to be manually investigated for false positives first, before forwarding them to the responsible person.

Currently, we us a threshold of 10 points, any value above is considered contaminated. A value of zero is considered clean and the nickname is added to the dynamically changing whitelist (Section 3.2.3).

In the following, we describe the scoring function in more detail. The first test checks for the occurrence of suspicious substrings in the nickname. This can for example be the name of a bot (e.g., `RBOT` or `l33t-`), the country abbreviations (e.g., `DEU`, `GBR`, or `USA`), or the operation system (e.g., `XP` or `2K`). For each substring found, the final score of the nickname is raised by one point. Second, each occurrence of special characters like `[`, `]`, and | increases the overall points by one. A third criteria is whether or not the nickname consists of many digits: for each two consecutive digits, the score is raised by one point.

The reason for the rather low increase of score is that these strings and characters are not a true sign for a bot infected machine. For example, many IRC users playing online games belong to certain clans or groups, which tend to have a so called "clan tag" in their nickname. Thus, nicknames of clan members often use the characters `[` and `]` to surround their clan tag or abbreviation. To avoid false positives, we are thus rather conservative with these soft indicators of bot-related nicknames. As noted earlier, the score values and the weighting is currently rather ad-hoc, but we are experimenting with other scoring parameters in order to find a good justification of the weighting.

To reduce the number of false positives, only true signs for an infected host raise the final score by more than one point. True signs are:

- a match with one of the regular expressions (Section 3.2.2)
- a connection to a blacklisted server (Section 3.2.4)
- or the use of a blacklisted nickname (Section 3.2.4)

3.2.2 Regular Expression

Each nickname is tested against several regular expressions, which match known bot names. Currently, the configuration file contains 52 different regular expressions to match several hundred nicknames known to be used by bots. These regular expressions were generated by analyzing more than 4,000 different bots and their corresponding nicknames. To avoid false alarms, the regular expression are very specialized, each matching only a few usernames. For example the following expression: `\[[0-9]\|[0-9]{4,}` matches nicknames like `[0|1234]` and another expression matches names like `|1234`. Although both regular expression could be merged to one, we kept them separated to be more flexible with fine tuning. Another example is an expression like `^\[[0-9]{1, 2}\|[A-Z]{2,3}\|[0-9]{4,}\]$` which matches common bot nicknames like `[00|DEU|597660]`, `[03|USA|147700]`, or `[0|KOR|43724]` As all regular expressions are kept in a separate configuration file, existing ones can be easily adjusted or new ones can be added to keep up with the ever-increasing number of bots.

If a bot matches one of the regular expressions, the final score is raised by the minimum number of points needed to trigger an alarm. Thus, in our current configuration, another 10 points would be added.

3.2.3 Whitelisting

To prevent certain machines from being accidently detected by *Rishi*, the software utilizes a hard coded whitelist, which can be adjusted via the configuration file. With the help of the whitelist it is possible to exclude certain hosts from the analysis process. Whitelisted hosts are identified either by their source IP address, the destination IP address, or the IRC nickname they use. Currently, this static whitelist contains 11 source IP addresses, 13 destination IP addresses (9 overlapping with source IP addresses), and 29 IRC nicknames. The IP addresses belong to well-known servers within the network and the nicknames from known humans or false positives we observed.

Additionally, *Rishi* operates a dynamic whitelist, which automatically adds and removes nicknames according to their final score as returned by the analysis function. Each nickname, which receives zero points by the analysis function, is added to the dynamic whitelist. However, if the score of a whitelisted nickname raises above half of the points needed to trigger an alarm, it is removed from the list. If it exceeds the threshold of 10 points, the nickname is automatically added to the dynamic blacklist, which is described later on.

During the analysis phase nicknames are compared against both the hard coded and the dynamic whitelist. Thus, a nickname listed in either one of the whitelists will always receive zero points by the analysis function. Furthermore, *Rishi* checks for similarity of nicknames to names on the whitelists. The technique used for similarity checks is called *n-gram analysis* [6]. N-gram analysis uses a sliding window character sequence to extract common features of a given data stream. The n-gram analysis disassembles two nicknames, which are to be compared, into parts, each containing two characters. With these 2-grams, each part is compared with the parts of the other nickname and the number of congruities is counted. The more parts are identical, the more likely both nicknames are the same or at least very similar.

With this technique we are able to automatically determine if a given nickname is similar or equal to a nickname already stored in one of the whitelists and react accordingly. As a result, nicknames similar to a name on the hard coded or dynamic whitelist are automatically added to the dynamic whitelist. For example, if a user changes his whitelisted nickname from `myNickname` to `myNickname_away`, the new nickname will still be similar enough to the already whitelisted nickname to also receive zero points by the analysis function. Thus, it is not necessary to place all known good nicknames on the hard coded whitelist by editing the configuration file, but let *Rishi* decide.

3.2.4 Blacklisting

The same concept is used by *Rishi* to maintain nicknames on one of two blacklists: the first blacklist is hard coded in the configuration file and can be adjusted manually. The second one is a dynamic list, with nicknames added to it automatically according to the final score received by the analysis function. That means, each nickname, for which the analysis function returns more points than needed to trigger the alarm, is added to the dynamic blacklist. Additionally, during the analysis phase, nicknames are compared against all names stored in either one of the blacklists so far. In case we have a match, the minimum number of points needed to reach the alarm threshold is added to the final score. Furthermore, if a nickname is found to be similar enough to a name on one of the lists with the help of the n-gram analysis, it is added to the dynamic list, too. As a result, it is possible to detect bot names which would not receive enough points from the analysis function, e.g., due to missing regular expressions, but are similar enough to an already known bot name stored in one of the blacklists and will therefore be detected.

Rishi also maintains a server blacklist for known C&C servers. If a connection to one of those servers is established, the final score is raised to the minimum number of points needed to trigger an alarm. Furthermore, if the destination port is not within the common list of IRC ports (currently 6666 and 6667), another point is added to the score.

3.2.5 Example

As an example, we want to describe the whole analysis process. Imagine that the nickname `RBOT|DEU|XP-1234` was added to the blacklist due to a match of one of the regular expressions. For some reason, the regular expression only matches the following country abbreviations for this kind of nickname: `DEU`, `USA`, and `GBR`. The next captured IRC connection contains the nickname `RBOT|CHN|XP-5678` and it is thus missed by the regular expression. From the analysis function, the name would receive 7 points:

- 1 point each due to the suspicious substrings `RBOT`, `CHN`, and `XP`
- 1 points each due to the two occurrences of the special character |
- 1 point each due to two occurrences of consecutive digits

Since the number of points is lower than the threshold, it would not trigger any alarm. However, due to the n-gram analysis against already stored nicknames, *Rishi* will notice a more than 50% congruence with a name stored on the dynamic blacklist and will therefore add another 10 points. As a result, the analysis function returns 17 points as a final score and thus triggers an alarm.

4 Limitations

Due to the fact that *Rishi* depends on regular expressions as signatures to automatically identify bot infected machines, the software is limited to only detect bots for which an expression exists. To circumvent this limitation *Rishi* also considers certain substrings, characters, number of digits, and destination port when evaluating a nickname. Thus, a botname is more likely to have a higher final score than a benign IRC name. However, there exist bots which use common names undistinguishable from a real name, which cannot be detected with the methods described. For example, the trojan Zapchast.AU [10], which is distributed by email, utilizes a list of 32.398 different nicknames to choose from when connecting to the botnet server. All of them look like common names with a two digit number attached to the end. For this kind of bots, it is almost impossible to detect them with the help of our approach. We could add the whole list of nicknames used by this bot to the blacklist, but this would presumably lead to a much higher number of false positives.

Another limitation of the software is the monitoring of protocol commands to determine a nickname or a joined channel. Thus in case a botherder utilizes a customized protocol, there is little chance for our approach to detect infected machines. Since bots move away from using IRC-based protocols to advanced techniques or even Peer-to-Peer-based communication, the basic concept behind *Rishi* has to evolve further.

However, as long as some protocol commands are still used, there is a chance to detect the botnet. We exemplify this with the help of an incident from December 2006. During that time we observed that *Rishi* logged several channel joins to an IRC server on port 54932, without any further information like nickname or user mode. Fortunately, we could extract the destination IP address from the *Rishi* log files. We started a separated packet capture instance to analyse network traffic to and from the suspicious IRC server. As a result, we noticed that the bot utilized its own IRC protocol by changing a few basic commands to customized ones. The command NICK was changed to SENDN, USER was changed to SENDU, and PRIVMSG was changed to SENDM. So in this case we were lucky as the bot missed to also change the JOIN command which triggered our botnet detection software. In the case where all IRC commands are customized, there is almost no chance for *Rishi* to detect an infected host at this time, as it is the case with many signature based detection mechanisms.

Another problem could be applet frontends to IRC, which let new users join a channel and learn what IRC is all about. These user names often follow a characteristic schema with patterns that could generate false positives. We have not yet had any problems regarding these web-based IRC clients For example, the ICQ network hosts an web-based IRC client, which is accessible even for users who do not have their own ICQ account. Those users without a valid ICQ account can still use the IRC service and get a nickname like `Guest_979`, `LeaP_195` or `onn_5201`. Since their is no regular expression which matches these names, the overall scoring value is typically around 3-4 points:

- 1 point for the special character: _
- 1 point for an uncommon destination port (7012)
- 1-2 points due to the occurrences of consecutive digits

Since the destination IP address and destination port are well known, this information can be added to the static whitelist. So far we did not experience any problems or falsely suspected hosts while examining such web-based IRC applications with random nickname generation.

Besides the above mentioned software limitations, we are also reaching the limits of the utilized hardware. The backbone in our testing environment uses 10 GBit Ethernet with traffic peaks of up to 3 GBit/s. Due to huge amount of traffic passing through the centralized router of RWTH Aachen, we are experiencing packet loss and corrupt packets since we are using a COTS system for packet capture and no dedicated hardware solution. As a result, *Rishi* can miss packets containing IRC specific information. Thus, it is very likely that some bot infected machines are completely undetected, as the packets never reach our software.

5 Results and Evaluation

We use *Rishi* within the network of RWTH Aachen university. With about 30,000 computer users to support, this network represents a typical university environment. *Rishi* runs on a Quad-CPU Intel Xeon 3,2Ghz system with 3GB of memory installed. As we are monitoring a 10 GBit network, the software consumes one complete CPU. Memory usage is rather low with 0.2%, as the different dynamic lists are limited to 400 entries each.

5.1 Detection Rates

Figure 5 illustrates the distribution of final scores of the analysis function for a number of nicknames which were evaluated by *Rishi* during one day of February 2007. Nicknames with a score equal to zero are not shown in this figure. Four bots are detected: three of them follow the scheme *[P00|country code|five digits]* and the fourth one is `br_yZFNprk`. The score for them is clearly above the threshold of 10 points, whereas all other nicknames receive a score of less then five. During this particular day in February 2007, another 101 nicknames were added to the dynamic whitelist, as they received zero points by the evaluation function, plus an additional 12 due to a more than 50% match with already whitelisted nicknames.

The results are similar for other days. Benign IRC nicknames receive a value ranging from zero to six points, whereas bot names almost always exceed fifteen points. Thus choosing ten as a first experimental threshold seemed to be a good mean value to sort out bot infected machines with little false alarms. We are currently evaluating how the threshold influences the number of false positives and false negatives.

We are running *Rishi* for about three months now and managed to detect more than 300 different bot infected machines within the campus network, already in the early development stages. In some cases, detection happened even prior to the detection by other intrusion detection mechanisms we have deployed. Furthermore, we were able to monitor botnets utilizing non-standard IRC commands for bot communication, which was described in the previous section. As a result, upcoming releases of *Rishi* will be able to monitor network traffic for commands which can be specified in the configuration file.

Estimating the ratio of detected bots within the complete network is hard since we do not know the total number of bots within the network. However, we can compare the number of bots detected by *Rishi* with the number of infected machines detected by other intrusion detection systems deployed. Within the university network, we use a system called *Blast-o-Mat* [11]. Blast-o-Mat relies on three detection approaches:

1. Detection of scanning machines via a threshold on SYN packets a host is allowed to send out for a specific time interval
2. Detection of machines sending out spam due to a threshold on emails allowed to be sent within a specific time interval
3. Detection of propagation attempts with the help of honeypots

Blast-o-Mat is the only intrusion detection system running within the network, thus we can only compare these two approaches.

Running nonstop for 14 days, we were able to detect 82 bot infected machines with *Rishi*. Of these hosts, only 34 were also detected by Blast-o-Mat. The remaining 48 were not detected since they used ways to propagate which are not monitored (e.g., email or drive-by downloads) or remained stealth on the compromised machines. Blast-o-Mat detected 20 additional hosts which were not picked up by *Rishi*. Thus it seems like both approaches should be combined to have an additional burglar alarm within the network.

During this time period, an additional 5 hosts were falsely suspected due to a few not too specific regular expressions.

5.2 Observed Botnet Characteristics

Figure 6 illustrates the different ports utilized for command and control servers. Port 80 is currently the most

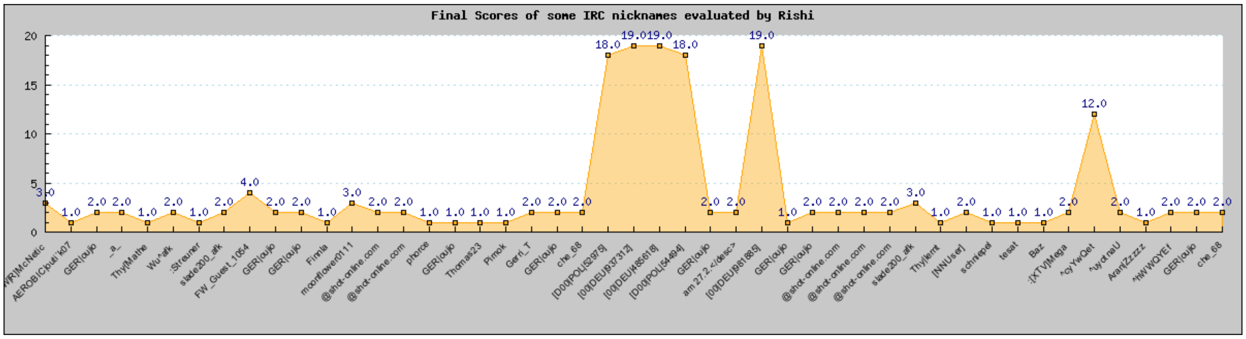

Figure 5: *Final scores returned by the analysis function for several nicknames*

frequently used port for more stealthy botnets, as it is commonly not blocked by firewalls. We also monitored an increase in HTTP based bots, which do not use IRC as a protocol to communicate with the botnet herder. Such infected machines could also be detected by payload inspection. The basic concept behind *Rishi* can be applied to this kind of communication channel. In the current implementation, we are already experimenting with HTTP-based bot detection. For this purpose, we also analyse traffic for URLs containing suspicious strings such as "`cnt=DEU`" in combination with "`cmd.php`". Due to the huge amount of HTTP traffic, we have limited *Rishi* to listen only on port 80 for HTTP bots. However, inspecting HTTP payloads will become impossible as soon as SSL encryption is involved.

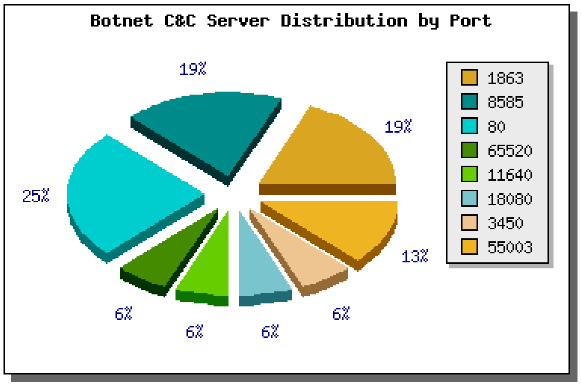

Figure 6: Botnet C&C server ports for botnets detected by *Rishi*

The detected bots connected to 16 different command and control servers located in countries all around the world. Figure 7 illustrates the distribution of C&C servers detected with *Rishi*.

Although the method presented here cannot be fully automatized, the prefiltering of suspicious IRC activity achieves a good and accurate detection ratio, with little manual interaction. As described in the next case study, it is possible to detect new unknown bots, which might become active hours after the first infection and prevent them from spreading in the network.

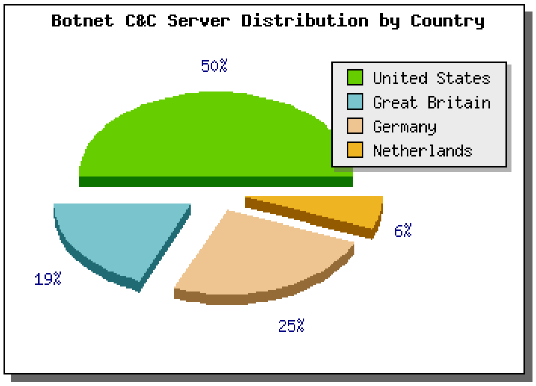

Figure 7: Geographical distribution of botnet C&C server detected via *Rishi*

5.3 Case Study: Detecting Spam-Bots

As a case study, we want to show an example of how we can detect special kinds of bots in an early stage. We take a closer look at *spam bots*, bots which a designed to send out large amounts of spam messages via the compromised machines. If these bots do not send out spam mails, they are commonly either propagating or in *sleep mode*, i.e., idling in the channel. Normal IRC bots also do nothing if they do not receive commands from the botherder. Due to the low volume of messages sent in this mode, it remains a challenge to detect this kind of stealthy bots.

With the help of the information collected by *Rishi*, we spotted several hosts infected with the Trojan *Troj/Loot-BH* [18], also known as *Trojan.Zlob.Gen*, which at that time was not detected by the antivirus software we were running. Thus, the machine owners were not aware of their systems being infected.

This type of bot utilizes nicknames which look like the following examples: `jece-1_9143_1019`, `jaal-1_4923_1178`, or `jeck-1_5120_1586`. The only two constant parts of this names are the "`j`" at the beginning and the substring "-1_" in the middle. This small amount of constant parts, together with the rather unusual large number of digits used in the nickname, was enough to raise an alarm in the analysis phase.

With the help of the information collected with *Rishi* about the botnet (e.g., C&C DNS entry, channel, and nickname), we were able to start tracking the botnet. We also managed to get our hands on a copy of the bot software itself, which we immediately transmitted to the antivirus company to upgrade their signatures. A total of 15 different hosts belonging to the RWTH Aachen network were infected, and could all be successfully detected, informed, and cleaned. Furthermore, by monitoring the botnet, we discovered that the bots receive an update to their software about every two days to avoid basic signature detection. However, they never changed the way their nicknames were generated and therefore it was easy to spot it among the usual IRC traffic.

After an infected host connects to the IRC-based C&C server, the bot does not join any channel to receive additional commands. Instead, orders are directly transmitted via private messages to each host connecting with a correct bot nickname. By default, this type of bot also does not try to propagate further and thus can not be detected due to aggressive scanning behavior.

The transmitted private messages start with the command *exec* followed by an URL leading to an update of the bot software or templates for spam messages. Figure 8 shows an example for the header part of the template and the variables which are replaced by the bot. Within the spam templates, the bot only needs to modify a few parameters before sending out large amounts of spam emails.

```
Received: by 192.168.54.34 with SMTP id nacZcMBB;
for <%MAIL_TO>; Wed, 30 Aug 2006 01:40:03 -0700
Message-ID: <000001c6cc0fe36f84702236a8c0@amjit>
Reply-To: ”%NAME_FROM” <%MAIL_FROM>
From: ”%NAME_FROM” <%MAIL_FROM>
To: %MAIL_TO
Subject: Re: tiRXda
Date: Wed, 30 Aug 2006 01:40:03 -0700
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary=”—-=_NextPart_000_0001_01C6CBD5.3710AC70”
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1106
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1106
```

Figure 8: Extract from the SPAM email template

The Trojan opens a backdoor on the compromised machines, which allows an adversary to access the machine and send out anonymous spam emails directly.

Due to the spam-sending behaviour, the infected hosts showed up in Blast-o-Mat, our custom intrusion detection system, some time later, too. However, the time between the first connection to the IRC server and the sending of spam could easily exceed a few hours, thus we were able to detect, inform, and react on the incident in a very early stage.

5.4 Case Study: Spotting Botnet Tracking

Along with the "regular" bots, we have also discovered some botnet tracking hosts with our approach. A botnet tracker is a machine which is dedicated to connect to known C&C servers to monitor and store all information exchanged via the IRC channel. Therefore it is possible to get information about updates to the bots binary, but also retrieve knowledge about new targets a botnet is going to attack.

To prevent being the target of Distributed Denial of Service (DDoS) attacks upon detection by the botnet herder, some botnet tracking groups use the Tor [8] service to anonymize their origin. Tor is a freely available service to anonymize the true origin of a machine. The basic principle is to route the traffic encrypted through several different routers, a so called *circuit*. For each request which does not happen within a short time period, a new circuit is constructed. As none of these servers stores any connection based information, it is very hard to reconstruct from which machine a certain request originated.

Within the network of RWTH Aachen university, several TOR servers are located. One of these servers is also an *exit node*, i.e., it can be the end point of a circuit and sends data to other hosts in the Internet. This node frequently showed up in the logfiles since the traffic from this host contained suspicious IRC activities.

Following are two examples for botnet trackers which were detected by *Rishi* during a very early development stage:

```
[2006/10/29 16:05:52]
Nick: [0||116823] Value: 17 srcIP: x.y.143.131
dstIP: xxx.xxx.124.236  dstPort: 4280
Channel: []
User: ['USER XP-9345 0 0 :[0||116823]']
Mode: ['MODE [0||116823] -x+i']

[2006/10/29 16:39:22]
Nick: [0||360830] Value: 17 srcIP: x.y.143.131
dstIP: xxx.xxx.166.38    dstPort: 55555
Channel: ['JOIN ##rx-noleggio## noleggius']
User: ['USER tilrcwa 0 0 :ESP|523075']
Mode: ['MODE ESP|523075 -xt', 'MODE [0||360830] -x+i']
```

The logfile should be self-explanatory, basically it shows the information stored in a connection object. The suspected source host itself is running Linux and thus it was very unlikely that this host was infected with a bot. In addition, it did not show any additional malicious activities like propagation attempts. A closer examination revealed that these connections were caused by botnet tracking hosts.

6 Conclusion

Detecting machines infected with a bot is often not easy: the bot can hide its presence on the machine and only become active under certain conditions. From a network point of view, it can be hard to detect the infection process, since this can happen via channels like emails or malicious websites. Due to the fact that bots need a communication channel back to the attacker, we have a way to detect an infected machine. In this paper we have explored a simple, yet effective way to detect IRC-based bots based on characteristics of the communication channel. We observe protocol messages and use n-gram analysis together with a scoring function and black-/whitelists to detect IRC characteristics.

Our proof-of-concept implementation *Rishi* has proven to be a useful extension to existing intrusion detection mechanisms. Besides the early detection of infected hosts, it is also possible to determine the IRC server the bots connect to. This information can then also be used to monitor the network traffic to find out more about the botnet and the actions it performs.

Currently, we are experimenting with the final score needed to trigger an alarm, as well as, the way points are distributed by each test performed on a nickname. However, a fully automated tool seems not to be possible without a rather large number of false positives on the one hand or completely undetected hosts on the other. Especially in the case where bots utilize nicknames composed out of random characters only, or if innocent people accidentally use a nickname containing suspicious strings, which trigger an alarm. Thus, it is required to have an administrator watch over the generated messages to manually filter out false alarms. Therefore, *Rishi* serves more as an extension to already deployed intrusion detection mechanisms to provide additional information, than a standalone software.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherds Michael Bailey and David Dagon for helpful feedback on earlier versions of this paper.

References

[1] Avira AntiVir. *Worm/Rbot.210944 - Worm*. 2004. `http://www.avira.com/en/threats/section/fulldetails/id_vir/3469/worm_rbot.210944.html`.

[2] Avira AntiVir. *Worm/Korgo.F.var - Worm*. 2005. `http://www.avira.com/de/threats/section/fulldetails/id_vir/1874/worm_korgo.f.var.html`.

[3] Avira. *AntiVir Personal Edition*. 2006. `http://www.free-av.de`.

[4] James R. Binkley. Anomaly-based botnet server detection. In *Proceedings of FloCon 2006 Analysis Workshop*, October 2006.

[5] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, pages 43–48, July 2006.

[6] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, Las Vegas, US, 1994.

[7] Yan Chen. Irc-based botnet detection on high-speed routers. In *ARO-DARPA-DHS Special Workshop on Botnets*, June 2006.

[8] Roger Dingledine and Nick Mathewson. Tor: The second-generation onion route. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[9] Felix Freiling, Thorsten Holz, and Georg Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *Proceedings of 10th European Symposium On Research In Computer Security (ESORICS05)*. Springer, July 2005.

[10] Jan Goebel. *A short visit to trojan Zapchast.AU*. 2006. `http://zeroq.kulando.de/resource/papers/download/Trojan_Zapchast.pdf`.

[11] Jan Goebel, Jens Hektor, and Thorsten Holz. Advanced honeypot-based intrusion detection. *USENIX ;login:*, 31(6), December 2006.

[12] C. Kalt. Internet relay chat: Architecture, April 2000. Request for Comments: RFC 2810.

[13] John Kristoff. Botnets. In *North American Network Operators' Group Meeting (NANOG32)*, October 2004.

[14] Carl Livadas, Bob Walsh, David Lapsley, and Tim Strayer. Using machine learning techniques to identify botnet traffic. In *Proceedings of 2nd IEEE LCN Workshop on Network Security*, November 2006.

[15] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attacks and defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2):39–54, April 2004.

[16] Stephane Racine. Analysis of internet relay chat usage by ddos zombies. Master's thesis, Swiss Federal Institute of Technologie, Zurich, April 2004.

[17] Jordan Ritter. ngrep – network grep, Accessed: February 2007. Internet: `http://ngrep.sourceforge.net/`.

[18] Sophos. *Troj/Loot-BH*. 2006. `http://www.sophos.com/security/analyses/trojlootbh.html`.

[19] Joe Stewart. Storm worm ddos attack. Internet: `http://www.secureworks.com/research/threats/storm-worm/?threat=storm-worm`, Accessed: February 2007.

[20] The Honeynet Project. Know Your Enemy: Tracking Botnets, March 2005. `http://www.honeynet.org/papers/bots/`.

[21] Detecting Botnets with Tight Command and Control. W. timothy strayer, robert walsh, carl livadas, and david lapsley. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, November 2006.

A Bot Nicknames

The following table shows an incomplete list of some variants of bot nicknames we have monitored using *Rishi*. The malware names were determined with the help of the antivirus software AntiVir [3]. The term "not found" means we could not find the name of the bot.

ID	Nickname	AV-scanner output
1	[00|DEU|172507]	Worm/Aimbot.AE.6
2	|DEU|2K|92193	not found
3	[XP]|6454734036	Worm/Rbot.171008.6
4	DEU|245500	Worm/Rbot.166912.5
5	LL-9985168738	Worm/Rbot.146432.10
6	ZX-44697595408	Worm/Rbot.166912.7
7	ASN-649955079	Worm/Rbot.90112.46
8	ZD-91817267335	Worm/Rbot.91136.62
9	[RAPEDv2]775571	Worm/Rbot.455680
10	{ripper}-310167	Worm/SdBot.56924.A
11	[SOUL]983586	Worm/SdBo.100864.22
12	[FUCK]-56507	Worm/Rbot.are
13	vjlr_13	W32/Parite
14	ezbxtju_12	Worm/Korgo.I
15	jeck-1_8887_1350	Trojan.Zlob.Gen
16	RBOT|DEU|XP-20366	not found
17	[XP-4848456]	not found
18	[M]DEU|973140123	not found
19	RBOT|F|USA|XP-54143	not found
20	MIR-[53681665]	not found
21	[MOO|USA|83868]	not found
22	DEU|XP-SPO-44733	not found
23	Ss-6098404166	Worm/Rbot.174080.8
24	POP3-3555035	Worm/Rbot.189440.5
25	H33-24054789964	Worm/IRCBot.QY
26	br_mxjDusy	not found