

# LARGE-SCALE MALWARE EXPERIMENTS: WHY, HOW, AND SO WHAT?

Joan Calvet, Jose M. Fernandez  
École Polytechnique de Montréal, Montréal, Canada  
Email {joan.calvet, jose.fernandez}@polymtl.ca

Pierre-Marc Bureau  
ESET, Montréal, Canada  
Email pbureau@eset.com

Jean-Yves Marion  
LORIA, Nancy, France  
Email jean-yves.marion@loria.fr

## ABSTRACT

One of the most popular research areas in the anti-malware industry (second only to detection) is to document malware characteristics and understand their operations. Most initiatives are based on reverse engineering of malicious binaries so as to understand a threat's features. In order to fully understand the challenges faced by a malware operator, it is sometimes necessary to reproduce a scenario where researchers have to manage thousands of infected computers in order to reach a set of objectives.

In this paper, we first discuss the reasons why one would want to replicate a botnet and perform experiments while managing it. In our case, our objective was to emulate the Waledac botnet and assess the performance of a mitigation scheme against its peer-to-peer infrastructure. We then present our experimental methodology and explain the technical decisions we took to perform our experiments. Finally, we explain our results, both in terms of the attacks against the Waledac botnet and the challenges we faced while creating our experimental environment.

## 1. WHY?

There have been numerous studies on botnets where modelling and simulation were used to evaluate both the size and the communication infrastructure of networks of zombies. On the other hand, very limited work has been done to actually create a botnet in order to observe its reaction to external stimuli and, most of all, understand the challenges faced by its operator.

Thus our goal is to build a framework to make this type of experiment feasible in a controlled environment. We think there are several good reasons to investigate this research area:

- It is a unique opportunity to learn about the problems faced by malware operators when managing thousands of infected computers.
- Running *in vitro* lab experiments lets us understand the various phases of a botnet operation: its creation, its growth, its updates, defence mechanisms, etc.

- Unlike with in-the-wild experiments [1], there are fewer ethical or legal issues to deal with than when performing arbitrary attacks against infected computers.
- Having an *in vitro* environment provides us with a way to conduct computer security research in a scientific way: we can reproduce experiments and test the effect of various independent variables.

We decided to use the Waledac botnet as a first experiment for the following reasons:

- Thanks to prior reverse engineering [2], we had in-depth knowledge of this threat family.
- This malware does not replicate, thus limiting the risk of running an experiment that might get out of control.
- There exists a set of vulnerabilities in Waledac's peer-to-peer protocol that were worth investigating. We wanted to evaluate the impact of a mitigation scheme against the botnet.

## 1.1 The Waledac case study

The architecture of the Waledac botnet is split into four layers. The first layer contains infected hosts with private IP addresses that are referred to as *spammers*. They are essentially the 'worker' bots and constitute approximately 80% of the botnet. The second layer is composed of bots with public IP addresses which we call *repeaters*. Their main duty is to relay the command and control (C&C) traffic to the upper layer. At the third layer, we have the *protectors*, *Linux* servers responsible for relaying traffic to and from the last layer, which is the *command and control server*, also known as the 'mother ship'.

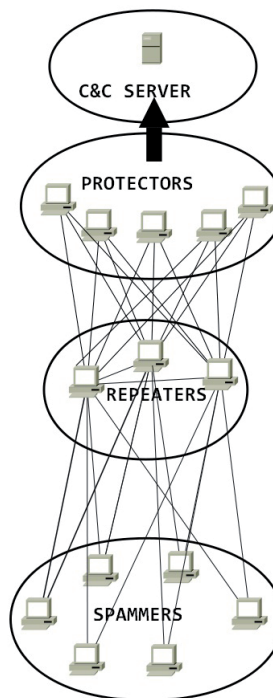


Figure 1: Waledac botnet network architecture.

## 1.2 Communication protocols

All Waledac bots need to know some repeaters, to be able to contact the C&C server. Initially, binaries are hardcoded with a list containing the contact information of some repeaters. This list – which is referred to as an *RList* – is stored as an XML file, in a registry key. An RList has a global Unix timestamp and contains between 100 and 500 records. Each record has the following fields: a 16-byte ID, an IP address, a port number and a local Unix-style timestamp:

```
<lm>
<localtime>1244053204</localtime>
<nodes>
<node ip="W.X.Y.Z" port="80" time="1244053204">
469abea004710c1ac0022489cef03183</node>
<node ip="A.B.C.D" port="80" time="1244053102">
691775154c03424d9f12c17fdf4b640b</node>
...
</nodes>
</lm>
```

The list is sorted in descending order of timestamp (i.e. the record with the most current local timestamp is at the top of the list). The RLists play a key role in facilitating Waledac's operations:

- The RList allows each node to 'know' a small subset of the botnet nodes. A spammer, for example, contacts the repeaters in its RList at frequent intervals, and requests tasks to perform. The repeater will, in turn, forward the job request to a protector, which will subsequently forward the request to the C&C server, and each agent will forward the information down the chain.
- The RList provides a means of propagating identification information for repeaters which have recently joined the botnet. All bots regularly send update messages to repeaters. A bot (bot B) that wishes to send an update message, if it is a spammer, will extract 100 records from its RList and send this extract to a randomly selected repeater. If bot B is a repeater, it selects 99 records from its RList, adds its record containing its identification and a timestamp to the top of the list and sends the list to the randomly selected repeater. When the recipient (bot R) receives the list, it reciprocates the process, i.e. it sends back a list containing 100 repeaters it knows of, to bot B. A recipient of an update list uses it to update its own RList, following an algorithm described below.

Finally, Waledac has a fallback mechanism which allows bots to maintain connection to the botnet even if none of the repeaters listed in its RList are reachable. The fallback mechanism works as follows: if a bot makes 10 consecutive attempts to contact a repeater that are unsuccessful, it connects to an HTTP server (the URL for the server is hardcoded in the Waledac binary) and downloads an up-to-date RList. These lists are updated every 10 minutes by the C&C server and contain the most recently 'seen' repeaters. Waledac uses the DNS double fast flux technique [3] to maintain its domain names, i.e. multiple IP addresses are assigned to them. These addresses correspond to some repeaters which relay the traffic for the domain names to the protectors, like they do with the C&C traffic.

In addition to an RList, each repeater also has a digitally signed protector list, containing identification information for the protectors. The repeaters regularly exchange lists of protectors among themselves in an effort to maintain connectivity with the mother ship.

## 1.3 Attack scenarios

The purpose of our experiment was to test attack scenarios against an as-realistic-as-possible botnet infrastructure. We had identified a vulnerability in Waledac's peer-to-peer protocol [2, 4] which we wanted to exploit. This attack is based on two facts. First, the IP address of a bot does not need to be unique: bots are primarily identified by their 16-byte ID. Consequently one can generate a large number of fake bots (*sybils*) – with different IDs but with the same IP address – thus gaining access to the botnet.

Second, and as explained in section 1.2, a bot uses the update message it receives to update its RList. The update algorithm is as follows: for each entry  $i$  in the update list, it computes a new timestamp ( $NewTS_i$ ):

$$NewTS_i = CurrentTS - |UpdateTS - TS_i|$$

where  $CurrentTS$  is the current timestamp,  $UpdateTS$  is the global timestamp of the update list received, and  $TS_i$  is the timestamp for this specific entry  $i$  in the update. The recipient inserts the entry in its RList at the correct location based on its provided timestamp. If the updated list contains more than 500 entries, all records behind the 500th position are deleted. Thus we can easily overwrite the whole list by setting the  $TS_i$  of all 500 sybil entries to a value that is identical to  $UpdateTS$ . This guarantees that  $NewTS$  for all the sybil entries will be equal to  $CurrentTS$ ; consequently these 500 sybil entries will be placed at the top of the recipient's RList and all the other entries will be removed.

The extent to which a bot can be controlled and isolated from other bots when it receives such an update message from a sybil, depends on its role in the botnet:

- *Repeater*: After it receives the message from the sybil, there is a race condition. Since the repeater's identity information is likely to be in other bots' RLists, other bots will send it update messages, whose entries can replace some of the sybil entries in the repeater's RList. Thus to maximize the chance that a repeater will eventually be isolated from other bots, the sybils need to continue re-sending the repeater update messages at short time intervals.
- *Spammer*: The result of the attack is more effective, since spammers cannot be contacted directly. When a spammer receives an update from a sybil, the spammer will be completely isolated from other bots. It should be noted though, that in order to infiltrate a spammer's RList, the sybils first need to infiltrate the RLists of repeaters whose identity information is in the RList of the spammer.

In our attack scenario, we target repeaters and, as a side effect, we affect the spammers. Collecting the repeaters' IP addresses could be done in the wild in at least two ways: first, as previously said, they are used as HTTP proxies for the Waledac domain names, so simply brute-forcing the domain names

would give us repeaters' addresses. Secondly, joining the botnet plus storing RLists gives us a reliable way to collect IP addresses.

It should also be noted that, in order for the affected bots to remain isolated, the sybils need to remain active until all the Waledac domains that are encoded for in the affected bots are deactivated. This is necessary to prevent the bots from resorting to the failback mechanism we described above and downloading a 'clean' RList from a Waledac HTTP server. The ultimate purpose of the attack is to propagate sybil information and isolate bots from their C&C server and botmaster.

## 2. HOW?

Preparing and running a large-scale malware experiment in a controlled environment is a hard task. Multiple aspects have to be taken into account including the physical infrastructure, software requirements, and experimental methodology.

### 2.1 Physical infrastructure

Our set-up is based on a computer cluster containing 98 blades. For security reasons, this equipment is completely isolated and does not have any access to the Internet or any other network. Each cluster blade has a quad-core processor, 137GB hard drives, 8GB of RAM and a network card with four-gigabit Ethernet ports. The blades are placed in a blade chassis in groups of 14. The first ports from each blade are connected to access switches that are in turn connected to a core switch, forming a separate control network. The other three ports are connected to separate access and core switches, forming the experiment network. The control network is used to control the VMs on which the botnet code will run and to control the botnet code itself, while the experiment network is used to bridge the virtual network cards that the botnet code is using in its 'normal' botnet activity. The *Extreme Cloud Administration Toolkit (xCAT)* is a popular, open-source data centre control toolkit that is used to manage virtual machines on the cluster. The *xCAT* server is responsible for assigning a MAC address and an IP address to each newly deployed VM. In addition to the blades in the cluster, we also use a standalone *Linux* machine which offers DNS and SMTP services to the VMs.

### 2.2 Setting up the botnet

For our experiment, we created two *VMware* virtual machine templates; one for the spammers and another for the repeaters, both of them using *Windows XP* and infected by an unpacked Waledac binary. Using an unpacked version gives us the insurance of binary stability, e.g. no VM-detection, and also an easy way to debug the binary directly on the bot if necessary, without losing time in the unpacking process.

Then we deployed 30 VMs on each blade, which gave us a total of about 3,000 bots (2,500 spammers/500 repeaters).

In order to enable easy remote control of the bots, we created a Python script that is able to receive commands through the control network. It runs on all virtual machines and has the following abilities:

- Infection and disinfection. The disinfection process is easy in Waledac's case: we simply need to remove the registry key responsible for the malware execution.
- Cleaning. Initially the RList contains the repeaters' IP addresses with the more recent timestamp. The script is able to erase the current RList and return it to its initial state.
- Dump RList. This allows us to track the progress of an attack.

For the protector layers we use some simple HTTP proxies. Since the protectors' list is signed, and since we did not want to modify the binary, we chose to use the exact same IP addresses for the protectors as the ones in the wild. For the last layer of the botnet, we built a fake Waledac C&C server, responsible for answering all the C&C requests it receives. Based on our observations of the real botnet, we were able to construct a special VM running a Python script to perform this task.

As with Waledac [2], our C&C server uses 1024-bit RSA and 128-bit AES keys to provide confidentiality for the C&C traffic. Moreover we programmed our server to respond in a similar manner to Waledac's C&C server. For example, we observed that spam orders contain usually between 500 and 1,000 email addresses; we mimicked this functionality by creating five different spam order messages, each containing between 500 and 1,000 email addresses, and programmed the C&C to send these spam orders to bots that request spam jobs.

We have also implemented support for Waledac's failback mechanism: every 10 minutes the C&C server creates an RList containing the identification information of the most active repeaters; this list is placed on an HTTP server. All HTTP requests from the Waledac binaries to the failback domains are directed to this host (see below). To configure the botnet correctly, we programmed our DHCP server to give private IP addresses to spammers and public ones to repeaters. Moreover, our DNS server is responsible for providing the IP address of our SMTP server.

### 2.3 Setting up the attack

We use the following three entities for our sybil attack:

- *Fake repeaters (sybils)*. The role of these is to passively monitor the network and poison the RList of nodes that contact them. They do this by accepting connections coming from real bots and answering with the specially crafted lists we previously described.
- *Attackers*. These agents actively target a specific number of real repeaters and send them the crafted update messages. We assume they have the ability to discover repeaters' IP addresses, because, as previously said, there exist at least two easy ways to collect them: DNS brute forcing and botnet crawling. The records in the update messages all contain sybil identification information. The attackers send the target repeaters one update message per minute. We observed that the Waledac bots in the wild send between two and five update messages during a two-minute time period. Thus this message frequency is in line with that of in-the-wild bots.

- *Sybil C&C server*: The role of this server is to prevent bots from activating their failback mechanism. To do this, it will send harmless orders to the ‘controlled’ bots. To perform this task, we used the following feature of Waledac: spammers are supplied with a special SMTP server IP address that they are required to use to test if they are capable of sending spams. Before sending spam, spammers try to connect to this special SMTP server; if they succeed in connecting to the server, they send the spams, otherwise they do nothing. Our sybil C&C server therefore sends the address of an unreachable SMTP server to the bots.

In our experiment, we used three VMs to launch the sybils, one VM to launch the attackers and another one to run the sybil C&C server.

## 2.4 Measurements

We used the following metrics to assess the effectiveness of our attack:

- *Spam output*. We measured the spam output of the botnet, over a fixed period of time, before and after we launched the attack. To facilitate the spam output measurement, we programmed our *C&C server* to send spam orders with email addresses that belong to the same domain as the networks used for the experiment. Consequently, when the spammers send the spam, they are all collected by our SMTP server, providing us with an easy way to measure the botnet’s output.
- *Connectivity of the botnet*. We wanted to evaluate the extent to which the sybil attack causes the connectivity of the botnet to decrease. To accomplish this task, we counted the number of NOTIFY messages the C&C server receives over a period of time, before and after we launched the sybil attack. NOTIFY messages are the second message that a bot sends during its dialogues with the C&C server. By counting only the NOTIFY messages, we filter out the failed connection requests, thus giving us a good indication of how many bots can reach the C&C server and hence do the botmaster’s bidding.
- *Percentage of sybils in RList*. The goal of the attack is to replace the entries of the bots’ RLists with sybil records. In so doing the bots will be isolated from the botnet. To collect this measurement, we use our remote control script to dump the RList and count the number of sybils.

We wanted to evaluate the success of our attack against a subset of known repeaters. Thus, we performed three sets of experiments in each of which we directly targeted 200, 100 or 25 repeaters.

## 2.5 Running the experiments

The work involved in running these experiments can be divided into two phases as follows:

**Preparation phase** (done only once to set up the cluster):

1. *Prepare the virtual machine templates*. See sections 2.2 and 2.3.

2. *Deploy the VM templates on the cluster blades*. This step takes about four hours for our 3,000 machines.

### Experimental phase:

1. *Start the virtual machines on each node with xCAT scripts*. At this moment the VMs are not infected by Waledac.
2. *Start the malware binaries on each node*. For this we used our Python script running on the bots. However, it turned out to be trickier than we thought it would be. Indeed, by starting the whole botnet at the same time, a working cycle effect is created. All the bots contact the C&C during the same short period of time and thus the C&C traffic follows a sine-wave-like curve that is not natural (this would not normally happen on a real bot). So we had to add some randomization to the starting process in order to counter that effect. With this, it took one hour to launch the whole botnet and converge to a stationary state without cycle effect.
3. *Start monitoring the experiment*. Activate measurement processes and start gathering statistics on both the SMTP and the C&C server.
4. *Start the attack with the desired number of direct targets*. This is done by remotely activating the sybil and attacker code in their respective VMs. The targets are chosen randomly because, as said before, we can easily gather their IP addresses.
5. *Gather data until steady state is reached*. The steady state is reached when the botnet spam output is essentially constant.
6. *‘Clean’ machines*. By this we mean disinfect the bots and put back the initial RList (with real repeater addresses), and not ‘disinfection’ in the real-world sense (remove/deactivate malicious code). This allows us to revert back to the initial conditions for the next experiment, and this without tainting results.

## 3. SO WHAT? – EXPERIMENTAL RESULTS

### 3.1 Baseline

The first sets of experiments we conducted were used to evaluate the performance of the botnet during its normal operation. We performed three experiment runs. For each run, we let the botnet activity reach a steady state, then we let the botnet run for 10 hours and we calculated the following two measures: the number of emails the botnet outputs per minute, and the number of NOTIFY messages the C&C server receives per minute.

The average values were 13,200 emails per minute and 120 NOTIFY messages per minute. It should be noted that this number of emails only represents the ideal value that the botnet could obtain in the wild under ideal conditions where all email addresses are reachable.

### 3.2 Measurements

We performed three experimental runs for each set of parameters (25, 100 and 200 direct targets) and calculated the

average values for the runs. The standard deviation values for the experiment runs were relatively small; therefore we believed that three runs were enough to obtain statistically significant values for the metrics we measured.

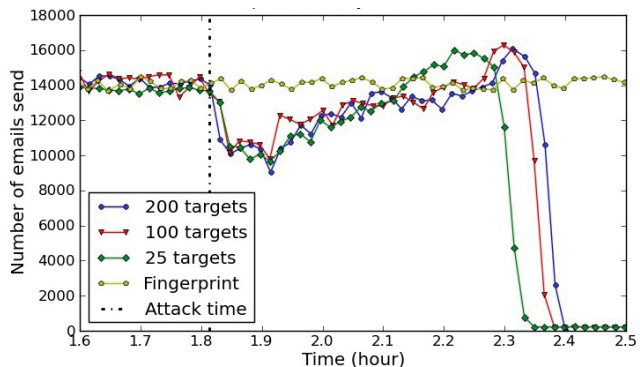


Figure 2: Spam sent by the botnet.

Figure 2 shows the spam output of the botnet before and after the sybil attack begins (dotted vertical line). The yellow line represents the behaviour of the botnet during its undisturbed activity. The graph shows that the attack is a success and that the spam output falls drastically after less than an hour. We also observe that after the sybil attack begins, there is an initial decrease in the spam output, with a subsequent gradual return to its original level and even rising significantly above it before the final fall. Furthermore, we can see that the time taken to reach the final fall is longer with 200 and 100 targets than it is with 25.

We think these two behaviours are caused by the load on the C&C servers (both the sybil and the malicious one). The C&C servers are overloaded and cannot keep up with the computing time required for cryptographic operations. As mentioned in [2], the Waledac botnet uses RSA with 1024-bit key pairs and AES 128. Through our observation of the Waledac botnet in the wild, we discovered that the C&C server used the same AES session key for all bots, for approximately 10 months. We initially thought that this was a design error made by the bad guys, but when implementing the Waledac C&C server it turns out that it was impossible to generate a session key for each bot, because it overloads the server with cryptographic computation. Waledac bots are too verbose and if good availability of the C&C server is desired, there is no choice but to keep the same session key for all active connections (at least for several minutes) and give bots the same set of encrypted orders. Hence it is likely that this was not a mistake, but rather a conscious design choice by Waledac's creators.

Shortly after the sybil attack begins the bots that get controlled by the sybils begin to send job requests to them, which are relayed to the sybil C&C server. This causes a significant decrease in spam output as these bots only receive harmless orders, but at the same time this initial burst of connection requests overwhelms the sybil C&C server because of the required initial RSA-1024 encryption. This results in some of the requests not being answered, thus causing some of the infected bots to resort to the fallback mechanism after 10 successive attempts to contact the sybil C&C server. As a

consequence, these previously sybil-controlled machines now revert back to the real C&C server and the control of the 'legitimate' botmaster.

However, as the sybil attack progresses, the sybil C&C server has fewer cryptographic operations to perform because we use exactly the same strategy as the Waledac C&C server in the wild: we use the same session key and pre-encrypt the work orders in batch mode before they need to be sent. Thus as the attack progresses, the sybil C&C server availability does not decrease too significantly since it does not have to encrypt any orders, hence allowing it to adequately handle requests from and control an increasing number of bots.

It is important to note that there is a delay between the moment we completely infect a spammer's RList and the moment it stops sending spam: until it has finished its current task, a bot will not contact the C&C server. For example, it can be completely infected and still have several hundred email addresses to spam.

As the attack progresses and we gain power within the botnet, we decrease the load on the real C&C server, which becomes more available for the non-poisoned bots. Thus, these bots receive orders every time they ask (which is not a normal situation, even in the wild) and continue to spam in a more efficient way than in a normal state. During that short interim time period, the botnet is more efficient under attack than in its normal state. It should be noted that if we had given more resources to the C&C server to start with this effect would probably be less important, but as we attributed four processors and 8GB RAM for its VM, we think it is realistic to assume that this effect would also be observed in the wild.

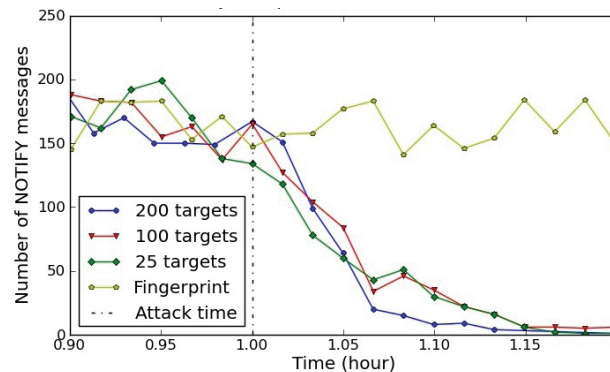


Figure 3: Number of job requests that reach the C&C server.

Figure 3 shows the number of NOTIFY messages the C&C server receives before and after the sybil attack begins. The number of NOTIFY messages the C&C server receives is essentially a measure of the connectivity of the non-sybil controlled portion of the botnet. The figure indicates that, as expected, there was a gradual decrease in the number of messages that reach the C&C server, after the sybil attack begins.

After the attack begins, the most efficient attack is the one with 25 targets. This is also a consequence of the load on the sybil C&C server. Because it is overwhelmed by the more aggressive attacks, it refuses connections. After a transition period, the 100

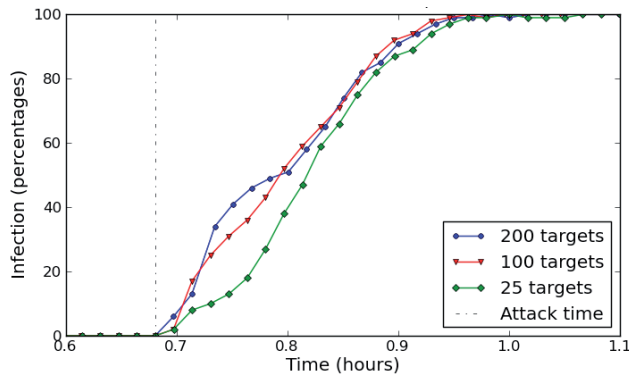


Figure 4: RList infections for all the repeaters.

and 200 target attacks eventually become more effective as the sybil C&C server has fewer cryptographic computations to perform.

Figure 4 shows the percentage of sybil entries in all the repeaters' RLists (targets and non-targets). After an initial transitory phase where the more aggressive attacks seem more efficient, we reach a stage of equivalent linear growth in the number of sybil-controlled machines in the RList. This is due to the fact that propagation of sybil records in the RList of non-sybil bots is dependent on the rate of RList updates between non-targeted real bots, which is the same for all attacks.

Figure 5 shows the percentage of poisoning on the targeted repeaters only. We can observe that it is quicker to fully control 25 direct targets than 100 or 200, because of the race condition faced by these direct targets: the more bots we target, the higher the chance that we lose some races and have sybil records replaced by real ones.

#### 4. CONCLUSIONS AND FUTURE WORK

As a first conclusion based on our results, we think it is important to note that, with respect to certain performance measures, directly targeting a lot of repeaters is not more efficient than targeting a smaller number. This is primarily due to a load-balancing problem. Furthermore, there is clearly an optimal number of targets depending on the attacker resources. This not-so-obvious trade-off could not have been discovered without performing this kind of experiment, and we think that speaks clearly in support of this kind of experimental approach. Thus, we put forward that the only way to have an idea of the resources needed to mount an efficient sybil attack (that is, one that will take down the botnet most quickly and thus not alert the botmasters), is to actually run the botnet.

This experimental approach also allows researchers to understand the limitations malware operators have to deal with. In the case of Waledac, before actually running the botnet we did not realize why the bad guys were using cryptography in such a 'bad' way. After the first experiment, however, the reason became clear: they cannot afford the more secure single-use session-key approach if they want to maintain good availability of their C&C server. Moreover, we have also learned some other facts, for example that simultaneously

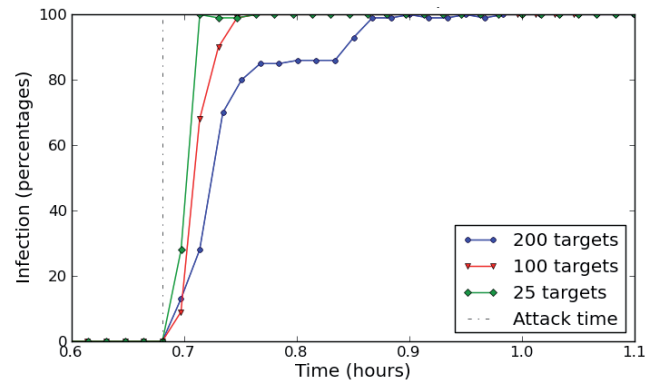


Figure 5: RList infections for the target repeaters.

bootstrapping a whole botnet within a short time frame is not so easy.

On the other hand, it must be pointed out that running such a large-scale malware experiment is not an easy task. Multiple aspects have to be taken into consideration including the security, physical infrastructure, software requirements, experimental methodology and information gathering. The Devil is in the detail and especially when undertaking experiments to study the Devil! A thousand details have to be taken care of in order to set up and conduct such experimental work properly, from handling of software licences on thousands of machines, to predicting and controlling server-room temperature increases, and optimizing set-up and deployment times between experiments. Nonetheless, we hope to have convinced the reader that the benefits in terms of gaining new, otherwise unattainable knowledge, brought about by this kind of experimentation are definitely worth such efforts.

As a final remark and open question, it should be noted that our experiment is far from perfect: our environment is limited in 'realism' in that it does not reproduce many of the characteristics found in botnets in the wild, such as network effects like varying latency of communication between bots, the diurnal effect by which some bots become on- or off-line according to the time of day, the infection/disinfection race between botmaster recruiting campaigns and disinfection efforts by AV products and careful owners/administrators, etc. Understanding and modelling such effects in a complex Internet-wide system is another area of research in itself, and how to reproduce them in a laboratory environment is, as far as we know, a wide-open research problem. Finally, the efficacy of our attack as measured is probably an overestimation of its performance on a real botnet, as it did not have to fight an intelligent and aware botmaster that could have fought back by a number of reactive measures (new protector lists, actively contacting bots with new RList updates, DDoS-ing sybil C&C servers, etc.). This falls within another very interesting and novel area of research, i.e. the application of Game Theory to computer security.

#### ACKNOWLEDGEMENTS

We would like to thank profusely Pier-Luc St-Onge and Wadie Guizani, the lab analysts in Montréal and Nancy respectively,

without whose help and dedication these experiments would have never been possible. Thanks also to Greg Sinclair, another Waledac expert, for providing us with very valuable insights into how the Waledac code and botnet worked and was being used.

## REFERENCES

- [1] Stone-Gross, B. et al. Your botnet is my botnet: analysis of a botnet takeover. ACM CCS, 2009, pp. 635-647.
- [2] Calvet, J.; Bureau, P.M.; Davis, C. Malware authors don't learn, and that's good! Malware, 2009.
- [3] Riden, J. Know your Enemy: fast-flux service networks. <http://www.honeynet.org/papers/ff/> (last visited 7 June 2010).
- [4] Sinclair, G. et al. The Waledac protocol: the how and why. Malware, 2009.