Attack of the 50 Foot Botnet

Ryan Vogt and John Aycock
Department of Computer Science, University of Calgary
2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4
{vogt,aycock}@cpsc.ucalgary.ca

TR 2006-840-33, August 2006

Abstract

The trend toward smaller botnets may be *more* dangerous in terms of large-scale attacks like distributed denials of service. We examine the possibility of "super-botnets," networks of independent botnets that can be coordinated for attacks of unprecedented scale. For an adversary, super-botnets would also be extremely versatile and resistant to countermeasures. Our simulation results shed light on the feasibility and structure of super-botnets and some properties of their command-and-control mechanism. Possible defenses against the threat of super-botnets are suggested.

1 Introduction

Big botnets are big news. Botnets involving over 100,000 zombie computers have been claimed [5, 6, 16], and there was even one case involving 1.5 million compromised computers [13]. However, big botnets are bad from the standpoint of survivability: someone is likely to notice a big botnet and take steps to dismantle it.

The recent trend is toward smaller botnets with only several hundred to several thousand zombies [4]. This may reflect better defenses – the malware creating new zombies may not be as effective – but it may be a conscious decision by adversaries[1] to limit botnet size, and try to avoid detection. It has also been suggested that the wider availability of broadband access makes smaller botnets as capable as the larger botnets of old [4].

We suggest that there is a new threat posed by smaller botnets. An adversary can create a large number of small, independent botnets. By themselves, the smaller botnets can be exploited by the adversary in the usual way, such as being rented to spammers. The discovery and disabling of some of the adversary's botnets is not a concern, either, because the botnets are independent and numerous.

The new threat arises if the botnets are designed to be coordinated into a network of botnets, which we call a *super-botnet*. For example, an adversary could command the super-botnet to launch a massive DDoS attack on a chosen target, or to pummel a critical piece of the Internet's infrastructure like the DNS. A super-botnet design potentially allows an adversary to surreptitiously amass enough machines for attacks of enormous scale. Super-botnets have additional applications in information warfare and cyberterrorism, where an adversary is able to prepare for an attack in advance.

To the best of our knowledge, the idea of super-botnets is new, although a related idea has been mentioned in passing for worms [9]. Super-botnets have not yet been examined, nor have they been seen in the wild. However, given that smaller botnets are already a reality, and different command-and-control (C&C) mechanisms are being seen and suggested [4, 7], super-botnets must be considered as a possible future evolution of today's botnets.

In the remainder of this paper we explore super-botnets and potential defenses against them. Section 2 examines the feasibility and creation of super-botnets. Section 3 considers adding inter-botnet communication, and looks at some communication properties that result.

[1] In this paper, we generically refer to people creating and using botnets as adversaries.

2 Super-Botnet Feasibility

Are super-botnets feasible? To answer this question, we must consider how a super-botnet can be constructed. We begin by abstracting away some details:

1. For simplicity, we assume that the super-botnet is constructed using only one worm. This does not imply that the worm's code is uniform across infections, of course, because the worm could be polymorphic or metamorphic in nature.

   More than one worm may be involved in practice. A single adversary could create multiple worms, or worm variants; multiple adversaries could conspire to create multiple worms based on a common super-botnet specification. Neither possibility is farfetched. For some malware, creating variants is practically a cottage industry – Spybot, for example, has thousands of variants [3]. Adversaries have collaborated in the past on malware [14], and *N*-version programming [2] of worms by adversaries within a single organization is definitely possible in the context of information warfare. Implementation diversity can make worm detection more difficult for signature-based detection methods.

2. The infection vector(s) the worm uses are not relevant to the analysis and are not considered. We also ignore failures to propagate in our simulations. While these considerations are important in real worm propagation, we are only interested in the resulting super-botnet structure.

3. The exact C&C mechanism(s) used within individual botnets does not matter. We assume a centralized IRC-based C&C for concreteness, but it could just as easily be a peer-to-peer architecture or some other method.

One way to establish a super-botnet is with a two-phase process. In the first phase, a worm is released which makes every new infection a C&C machine for a new, independent botnet. In the second phase, after enough C&C machines have been created, new infections populate the C&C machines' botnets. This method is risky for an adversary, because the "backbone" of the super-botnet's C&C infrastructure is all established by direct infections; discovery of one C&C machine can easily lead to others using information from firewall logs, for example.

Instead, we use the worm pseudocode shown in Figure 1. It is controlled by three parameters: `HOSTS_PER_BOTNET` is, not surprisingly, the number of hosts in each botnet; `BOTNETS` is the total number of botnets to create; `SPREAD` is the number of infections a given worm instance will perform. The infections follow a tree-structured pattern, as shown in Figure 2. A narrower breadth of tree, i.e., a smaller `SPREAD` value, can reduce suspicious traffic and help avoid rate limiting defenses [11].

Each infection passes along information to its progeny about the number of new infections they must contribute to the current botnet, the number of botnets left to create, and the location of the current botnet's C&C server. This propagation of information does not make the growing super-botnet as vulnerable to countermeasures as it might first seem. Locally, a C&C server can count the number of connections to it, in order to verify that the botnet it controls is indeed the desired size. Intercepting a worm close to an initial infection can prune a large number of botnets in this pseudocode, but in practice, an adversary can seed multiple initial infections, and use worm self-stopping mechanisms [8] to control super-botnet growth instead of relying on information being propagated from worm to worm.

When a new botnet/C&C server is created, all information about the previous botnet is discarded, leaving each botnet isolated. Furthermore, new C&C servers are placed as far away from one another as the balanced tree structure allows: for nontrivial values of `HOSTS_PER_BOTNET`, C&C servers are separated by at least $\lfloor \log_{\texttt{SPREAD}} \texttt{HOSTS_PER_BOTNET} - 1 \rfloor$ intermediate infections.

The adversary does not have a direct way to issue commands to the super-botnet and, as shown, the adversary will not even know how to locate individual botnets. In Section 3, we explore advanced communication methods. However, for now we assume that the adversary will command the super-botnet by external means. For instance, botnets may periodically poll an information source that is unlikely to be blocked or raise suspicion, such as accessing a web site (perhaps located via a web search engine), or making a DNS request to a domain under the adversary's control [7].

We simulated the worm pseudocode, allowing one infection per machine per time step. Starting with a single seed, the simulation established 15,000 botnets with 100 machines apiece – a super-botnet 1.5 million strong. Figure 3 charts the growth of infections and C&C servers over time for a `SPREAD` value of 2; Figure 4 uses a more generous value of 25 for `SPREAD`. One interesting anomaly is that, even though

```
# distribute x fairly into y slots

def D(x, y):
   let L be an empty list
   for i in [0, y):
      append x/y to L
   for i in [0, x mod y):
      L[i] += 1
   return L

# propagate worm

def P(nti, bts, c&c):
   # nti ≡ number to infect
   # bts ≡ botnets to start
   # c&c ≡ IP address of C&C server

   if nti == 0:
      # establish new C&C server
      bts -= 1
      start IRC server
      c&c = my IP address
      nti = HOSTS_PER_BOTNET
   else:
      # use existing C&C server
      connect to c&c

   nti_child = D(nti-1, SPREAD)
   bts_child = D(bts, SPREAD)

   for i in [0, SPREAD):
      if nti_child[i] == 0 and bts_child[i] == 0:
         continue
      pick target host to infect
      infect target
      target runs P(nti_child[i], bts_child[i], c&c)

# invocation for initial infection

P(0, BOTNETS, 0.0.0.0)
```

Figure 1: Worm pseudocode for super-botnet creation.

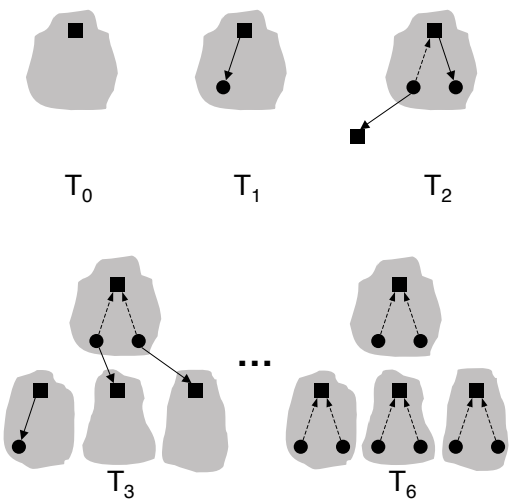

Figure 2: Worm infection pattern with 3 hosts/botnet, 4 botnets, and a spread of 2. Shown are C&C servers (■), non-C&C infected machines (•), new infections (solid arrows), links to C&C servers (dashed arrows), and botnets formed (grey blobs).

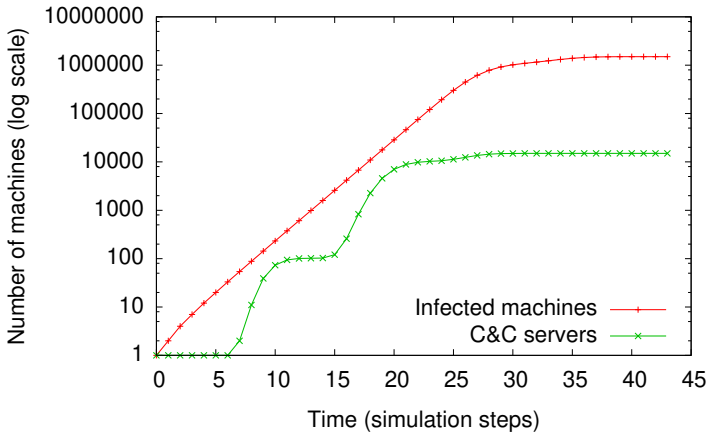

Figure 3: Worm simulation, spread of 2.

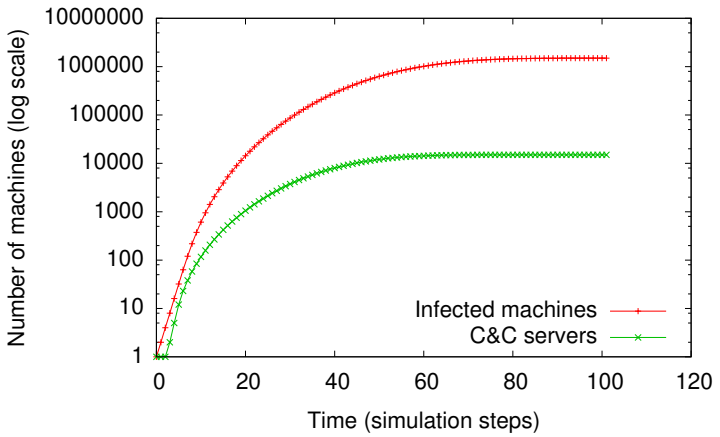

Figure 4: Worm simulation, spread of 25.

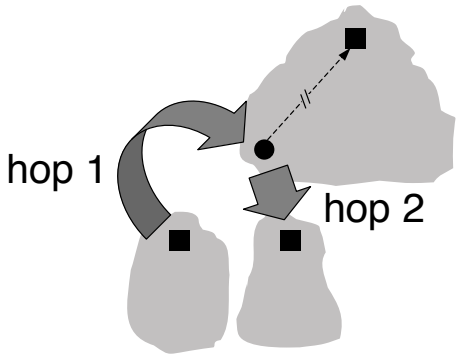

Figure 5: Tracking infections backwards: the two-hop weakness.

the C&C growth curve is the least smooth for a spread of 2, a spread of 3 infects 1.5 million machines the fastest. In all cases, it is clear that a super-botnet can be established in short order.

We now turn to defense. There would be no sense for an adversary to build a super-botnet for immediate use; a traditional worm would be more effective in that scenario. It is reasonable to assume, therefore, that super-botnets would be deployed in advance of an attack. Traditional anti-virus software would thus be useful against super-botnets, as anti-virus software would have time for updates and detection.

Besides using traditional defenses to eradicate, block, or contain the super-botnet worm, there are four other goals for defense:

1. Locate or identify the adversary. The adversary is vulnerable to detection when they issue commands to the super-botnet. (Recall that super-botnet commands are sent via some external channel.) Defenders may not know the nature of the adversary's commands in advance, but assuming some of the botnets have been detected and analyzed, the source of the adversary's commands will be known and alarms can be triggered when commands are sent. For example, a defender may know that super-botnet commands will be found by botnets periodically performing Google searches for "haggis gargling" and sifting through the avalanche of results for the adversary's commands. Monitoring when and where Google finds such a web site may provide a lead as to the whereabouts of the adversary. In turn, an adversary may try to obfuscate their trail by issuing super-botnet commands through proxies or anonymity networks like Tor [15].

2. Reveal all the infected machines. Again, if the botnets are polling a known location for the adversary's commands, then the polling activity will reveal infected machines. Meeting this goal and the previous one may require cooperation between law enforcement and the private sector.

   There may be weaknesses in the super-botnet structure as well that can reveal infected machines. Our worm pseudocode, for example, spaces out C&C servers along the (tree-structured) infection path. If infections can be tracked backwards from a C&C server to the machine that infected it, then there are up to `SPREAD` − 1 other C&C servers two hops away (Figure 5). The adversary has a clear incentive to choose small spread values and to destroy any information that might link different botnets.

3. Command the super-botnet. A defender could attempt to send a command to the super-botnet to shut it down. A related concern for the adversary is that *another* adversary may try to usurp control of the super-botnet. For these reasons, the adversary must digitally sign or encrypt super-botnet commands using asymmetric encryption [10], so that the compromise of a botnet does not reveal a secret key. Generating a public/private key pair in advance, the adversary could send the public key along with the worm, solving the key distribution problem.

4. Disrupt super-botnet commands. The adversary's commands can be intentionally garbled by a defender regardless of whether or not they are encrypted or signed – changing a few bits is sufficient to achieve this goal. An adversary can thus establish a super-botnet but be denied the ability to control it. This defense can be applied locally, e.g., by IPS/firewall rules, or globally if a defender has enough access to the super-botnet's command source.

```
def exchange(A, B):
   if M > 1:
      let N ∈ [1, M-1] be randomly chosen
   else:
      N = 1
   A randomly chooses N slots in M_A
   B randomly chooses N slots in M_B
   A and B exchange routing information
    from their chosen slots
```

Figure 6: Routing information exchange in the hockey card algorithm.

The conclusion? Independent botnets do present some advantages to the adversary: individual botnets can be farmed out for spamming and other traditional uses, the loss of an individual botnet is not catastrophic, and the number of botnets used in a super-botnet attack can be scaled appropriately to the target. External command of the super-botnet maintains the independence of individual botnets, but it quickly becomes a liability for the adversary.

3 Inter-Botnet Communication

The adversary may try to send commands along the super-botnet itself, to avoid revealing information or having externally-issued commands be subject to attack. In this section we consider how this is accomplished.

We assume that an adversary will want to encrypt communications both within each botnet and between botnets. When a new C&C server is created during worm propagation, it can generate a new public/private key pair. The private key is passed along to new infections within the botnet, and is used as a key for symmetric encryption within the botnet. The public key is used to asymmetrically-encrypt messages sent to the botnet from other botnets, so the *routing information* needed for botnet A to talk to botnet B is the pair

$$(\text{public key}_B, \text{C\&C IP address}_B)$$

The adversary's problem: propagating routing information around the super-botnet. No one botnet can be allowed to have complete information about the super-botnet, because the complete structure of the super-botnet would be revealed if a botnet with complete routing information were compromised by a defender. Each botnet can have partial routing information instead, and know how to contact a small finite set of its "neighbors." But how can partial information be gathered by a botnet?

One obvious time to gather partial routing information is when a new C&C server is created. One less-obvious time is when a worm tries to infect an already-infected machine. Normally this is considered something an adversary would want to avoid – worm contention is a waste of effort – but, for super-botnets, this is an ideal opportunity to exchange routing information. (A similar idea for worms was briefly mentioned in [12].) Such exchanges would presumably only occur while the super-botnet was being constructed, so that a defender could not trivially extract routing information later by pretending to be a worm, or flood the super-botnet with routing information pointing to a honeypot.

Note that we abstractly consider C&C servers to be performing the routing information exchange here. In practice, a C&C server's clients would be performing routing information exchanges too, and sending new information to their C&C server.

In the remainder of this section, we simulate and analyze a routing information-exchange algorithm over a population of 1.5 million vulnerable machines. The worm pseudocode from Figure 1 is used, assuming infection targets are chosen randomly, i.e., a random-scanning worm is simulated. Routing information exchange occurs in both of the above scenarios: creation of a new C&C server, and one infected machine locating another during its random scan. We denote botnet A's and botnet B's routing information by a and b, respectively.

Intuitively, our information-exchange algorithm mimics how children exchange sports trading cards; we refer to it as the *hockey card algorithm*. Each C&C server has M slots for routing information, and a newly-created C&C server for botnet A begins with all its slots (M_A) initialized to a. Thereafter, when either of the two opportunities arises to exchange information between botnets A and B, the pseudocode in Figure 6 is run – this happens immediately in the case of a new C&C server. This code maintains M links

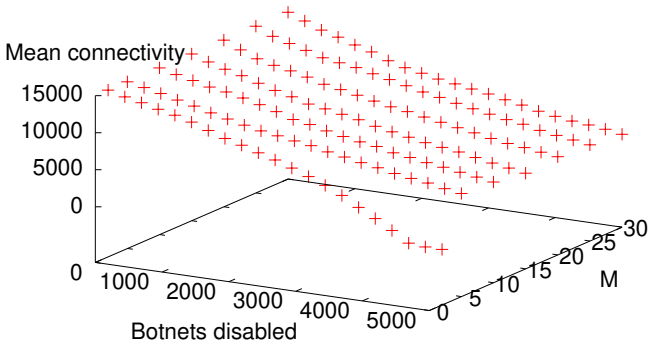

Figure 7: Mean connectivity decrease when high-degree botnets are targeted.

to each botnet within the super-botnet. Although a botnet may end up with duplicates and self-links, our simulations showed that this does not have any negative effects.

We assume that the adversary knows the addresses of seed infections, and that they will want to communicate through the super-botnet as surreptitiously as possible. In other words, the adversary will not broadcast their commands to all seeds, but will select one seed and send a super-botnet command through it. The important metric to the adversary is thus the amount of connectivity from some seed. How many botnets will a command reach?

We fixed three simulation parameters on the basis that the adversary would want a large, quickly-constructed super-botnet: 15,000 botnets, 100 seeds, and a spread of 3. Two parameters were adjusted, however. First, the value of M was varied from 1 to 30 in increments to ascertain its effect on connectivity. Second, the number of hosts per botnet was varied in increments from 25 to 100 to study different probabilities of finding other botnets during propagation. All simulations were repeated five times to minimize the effects of randomness.

The results were dramatic. In every simulation run where M was greater than 1, the amount of connectivity from any single seed was 100%. An adversary would be able to command all the botnets comprising the super-botnet using any one seed.

M does play a role in the resistance of the super-botnet to defensive measures. In particular, a defender may try to prevent an adversary from commanding their super-botnet by reducing connectivity. Here we define the *degree* of a botnet to be the sum of its useful in- and out-degrees, which are the links remaining once self-links and duplicate links are removed.

Following research on network attacks [1], we consider a defender who will try to disable a super-botnet using two different strategies. First, a defender may disable high-degree botnets first. This is an effort to reduce super-botnet connectivity more quickly than simply disabling botnets at random. It is important to stress that this is the absolute best case, where a defender has oracular knowledge of the super-botnet's connectivity. Second, a defender may disable botnets at random. This is perhaps a more likely case, where uncoordinated defenders would simply be shutting down botnets randomly upon discovery.

The results are shown in Figure 7 (high-degree first) and Figure 8 (random selection). We have only presented the data for 25 hosts per botnet here, because the results were almost identical for the different numbers of hosts per botnet we ran. Results were averaged over five runs to compensate for randomness.

The strategy used by the defender does not seem to matter. Once $M \geq 5$, a defender able to disable one-third of the botnets in a super-botnet still leaves the adversary able to contact one-third of their original botnets from some seed, on average. This is still enough for a sizeable attack to be launched with the super-botnet. Two conclusions can be drawn:

- The adversary would set M to at least 5. However, there is a tradeoff between robustness and disclosure. Too large a value for M would reveal the location of a large number of other botnets, if a defender discovers one botnet.

- Communication through a super-botnet is robust to a defender's countermeasures. Finding and dis-

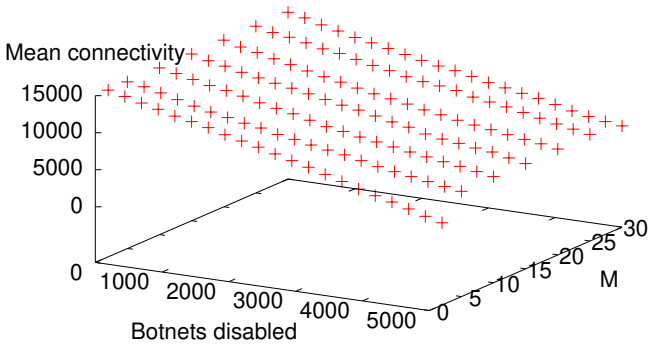

Figure 8: Mean connectivity decrease when botnets are randomly targeted.

abling 5000 botnets, randomly or otherwise, would be a gargantuan task. The only defense that is deployed widely enough to feasibly do this is anti-virus software.

This simulation also assumes that an adversary will only communicate through one of the initial seeds. Botnets could announce their (encrypted) routing information to an adversary instead; this would give the adversary many more communication endpoints to use at the risk of leaking information to a defender. An adversary may also use a small number of seeds for communication, rather than just one. Either variation by the adversary should reduce the effectiveness of the defender's countermeasures.

These conclusions suggest that new, large-scale defenses are needed for the super-botnet threat. Revisiting the four goals for defense from the last section:

1. Locate or identify the adversary. The adversary may issue a command to the super-botnet in a wide variety of places. Barring sheer luck, the defender will not see a command entering the super-botnet.

2. Reveal all the infected machines. This goal is somewhat more promising: one possibility is to take advantage of information that would otherwise be thrown away. When anti-virus software locates a super-botnet infection with routing information, it could pass the routing information along to a central defense location. Given enough disinfections, this tactic should reveal a sizeable portion of the super-botnet's structure.

3. Command the super-botnet. As in the previous section, an adversary who signs or encrypts commands will effectively eliminate any possibility of defenders injecting their own commands into the super-botnet.

4. Disrupt super-botnet commands. Even if a defender compromises a botnet, garbling super-botnet commands is unlikely to work well. There is no guarantee that the adversary's commands must travel through the compromised botnet, so the majority of the super-botnet is likely to receive the adversary's commands intact.

The defensive situation thus appears to be worse for a super-botnet with inter-botnet communication.

4 Future Work and Conclusions

An underlying assumption is that an adversary must command the super-botnet. One disturbing possibility is that the super-botnet could be designed to automatically decide on targets and coordinate attacks – without any human adversary. Ways to counter this, and all other forms of super-botnet, are an area of future work.

In any form, super-botnets provide adversaries with an enormous amount of virtual firepower that is easy to construct yet hard to shut down. The trend toward smaller botnets can be seen as an evolutionary step in

the direction of super-botnets. This means that attacks by millions of machines on the Internet's infrastructure can appear from nowhere, as multiple small botnets join forces. Super-botnets must be considered a serious threat that must be defended against.

5 Acknowledgment

The authors' research is supported in part by grants from the Natural Sciences and Engineering Research Council of Canada.

References

[1] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[2] A. Avižienis. The *N*-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1501, 1985.

[3] J. Canavan. The evolution of malicious IRC bots. In *Virus Bulletin Conference*, pages 104–114, 2005.

[4] E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *USENIX SRUTI Workshop*, pages 39–44, 2005.

[5] D. Dagon, G. Gu, C. Zou, J. Grizzard, S. Dwivedi, W. Lee, and R. Lipton. A taxonomy of botnets. Unpublished paper, c. 2005.

[6] A. Householder and R. Danyliw. Increased activity targeting Windows shares. CERT Advisory CA-2003-08, 11 March 2003.

[7] N. Ianelli and A. Hackworth. *Botnets as a Vehicle for Online Crime*. CERT Coordination Center, 2005.

[8] J. Ma, G. M. Voelker, and S. Savage. Self-stopping worms. In *Proceedings of the 2005 ACM Workshop on Rapid Malcode*, pages 12–21, 2005.

[9] J. Nazario, J. Anderson, R. Wash, and C. Connelly. The future of Internet worms. In *Black Hat USA*, 2001.

[10] B. Schneier. *Applied Cryptography*. Wiley, 2nd edition, 1996.

[11] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode*, pages 33–42, 2004.

[12] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, 2002.

[13] T. Sterling. Prosecutors say Dutch suspects hacked 1.5 million computers worldwide. Associated Press, 20 October 2005.

[14] P. Ször and P. Ferrie. Hunting for metamorphic. In *Virus Bulletin Conference*, pages 123–144, 2001.

[15] Tor: An anonymous Internet communication system. http://tor.eff.org, last accessed 31 May 2006.

[16] United States v. Ancheta. Case CR05-1060, Indictment, U.S. District Court, Central District of California, February 2005.