# Experiments with P2P Botnet Detection

## Experimente zur Entdeckung von P2P Botnetzen

Lionel Rivière, Sven Dietrich, Stevens Institute of Technology, Hoboken, USA

**Summary** Botnets, which are used to perform various malicious activities, have become a major threat in recent years. Spamming, phishing, stealing sensitive information, conducting distributed denial of service (DDoS) attacks, scanning to find more hosts to compromise using malware are the goals of many botnets, sometimes of low-profile botnets such as the Nugache botnet [1] which used a peer-to-peer (P2P) structure. Some botnets hide their network activities for many months (and maybe years) before being noticed. Networks might contain more deceptive or dormant bots which haven't been exposed yet. Here we apply an a posteriori detection approach based on mutual contacts peers exchange in a network, called the dye-pumping algorithm [2]. After briefly recalling typical botnet operations, we will talk further about the dye-pumping algorithm (DPA) mechanism and implementation, its input data structures, and then give a short analysis of the results of our experiment. ►►► **Zusammenfassung** Botnetze, die zum Ausführen von verschiedenen kriminellen Aktivitäten benutzt werden, sind zu einer großen Bedrohung geworden. Spam, Phishing, Datendiebstahl, DDoS-Attacken, Scanning, um neue Opfer zu finden, sind die Ziele vieler Botnetze, zum Beispiel das Nugache-Botnetz, das eine Peer-to-Peer-Struktur (P2P) verwendet hat. Einige Botnetze verstecken ihre Aktivitäten monatelang (manchmal auch jahrelang), bevor sie bemerkt werden. Netzwerke können Schläfer-Bots enthalten, die noch nicht entdeckt worden sind. Hier wenden wir a posteriori eine Methode an, die auf gemeinsame Netzwerk-Kontakte basiert, den sogenannten Dye-Pumping Algorithm (DPA, etwa: Farb- oder Tinten-Pump-Algorithmus). Nach einer Übersicht von typischen Botnetz-Verhalten besprechen wir den DPA an sich, seine Implementation und Datenstrukturen, und geben eine kurze Analyse der Versuchsergebnisse.

**Keywords** K.6.5 [Computing Milieux: Management of Computing and Information Systems: Security and Protection] P2P Botnet, IDS, Network Security ►►► **Schlagwörter** P2P Botnetze, Netzwerksicherheit

## 1 Introduction

In the Internet environment, security threats can take on the form of malicious software, also known as *m*alware. As a subcategory of malware, bots have played a significant role in network security, ranging from enabling attacks on high-profile websites in the early 2000s to compromising critical infrastructure systems in the early 2010s. The bots constitute the extension of the botmasters arm, making the host infected with the bot do whatever the botmaster wants it to do, such as performing denial of service, collecting keystrokes, spreading malware, hosting phishing sites, or serving pirated software. The coalition of bots forms a botnet, which can be hundreds, thousands, or even millions of hosts strong.

### 1.1 Botnet Context

A bot participates in a command-and-control (C&C) network, through which the bot receives commands that cause the bot to carry out the aforementioned attacks.

A typical bot implementation consists of two independent engines: a C&C communication protocol processor and a command interpreter. The interpreter allows bots to execute commands and defines its protocol. A bot command is generally encapsulated as the payload of a C&C communications protocol message. Its syntax encodes the actions the bot can perform as well as the ways in which each can be invoked, i. e., the parameters.

Bot control is achieved through a C&C network, which consists of the C&C protocol that defines the communication format and the network topology. The latter

identifies who talks to whom. In a centralized-type botnet, the C&C protocol also defines a central location to which commands are delivered. Historically speaking, botnets have been tightly controlled: the botmaster sends a command that is received and executed by all listening bots more or less immediately. The use of a non-centralized botnet topology for C&C, as seen in P2P bots (e. g., Nugache [1]), demonstrates a looser control model which has higher latency, since commands percolate through a distributed network as bots poll for them.

The use of P2P botnet C&C has challenged the detection schemes due to the lack of a central coordination point.

## 2 Approach

Our contribution focuses on the detection of P2P C&C traffic for several botnet classes, and on the tools associated with the detection scheme, namely implementing the algorithm efficiently and integrating it with network flow analysis tools.

### 2.1 Detection

In this work, we refine the detection scheme presented in the *Friends of An Enemy* paper [2], which shows that once we have detected a single P2P bot in a local (or edge) network, also referred to as the seed bot, we can efficiently identify other members of the same botnet in the same network. In contrast, there are approaches that work on the backbone, such as Botgrep [16]. The detection mechanism in [2] is based on an analysis of connections made by the hosts in the local network. To avoid regular and redundant peer connections, bots can select their peers randomly and independently (i. e., creating an unstructured topology). If so, in a same network, a given pair of P2P bots communicate with at least one mutual peer outside the network with a surprisingly high probability. This, along with the low probability of any other host communicating with this mutual peer, allows us to link together local nodes belonging to a P2P botnet.

### 2.2 Contribution

Initially, this scheme was built to identify potential members of an unstructured P2P botnet in a network starting from a known peer. We propose a refinement and an effective implementation of that method. We keep approaching the problem as a graph problem and mathematically analyze a solution using an iterative algorithm. The proposed scheme requires only flow records captured at network borders. We analyze the efficiency of the proposed scheme using real botnet data, including data obtained from both observing and crawling the Nugache botnet.

### 2.3 Monitoring Limitations

Network monitoring gives us plenty of information showing how traffic is received, forwarded, or sent. By analyzing large amounts of data, one could more or less specifically model traffic behavior on a network without directly using or even reading the actual content of the data packets. To create a model for a specific botnet simply based on monitoring, one needs to analyze the network data flows and logs over a long period of time.

What if we cannot "measure" traffic because a bot is too deceptive or simply dormant? One approach is to characterize the C&C channel from traffic of a recently discovered bot and then identify hosts that exhibit similar C&C characteristics. Characterizing the C&C channel is possible in a centralized botnet topology by focusing on the central control servers, the main source of the C&C channel. However, a C&C infrastructure with an unstructured topology doesn't have a central server, as any node of its network can inject C&C messages or commands. Hence, no obvious source of C&C messages can be observed and therefore neither can links to possible dormant bots.

Packet sizes and timing analysis, such as packets per flow, bytes per flow, flows per hour, may not be useful in characterizing a C&C channel, since botmasters can easily randomize such features, thereby obtaining different feature values for each bot. Furthermore, botnets such as Nugache, Storm, Waledac and Conficker use advanced encryption mechanisms making analysis based on packet content inspection impossible.

### 2.4 Existing Tools

Despite all these limitations, to achieve our purpose we need to monitor network traffic and observe who is connecting to whom and how (e. g., looking at IP addresses, ports, and their frequency). The proposed technique is based on the simple observation that peers of a P2P botnet communicate with other peers in order to receive commands and updates. Here are some software and tools we used in the implementation and testing of the technique.

The Packet Capture library libpcap [3] provides a high-level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. Snort is one example of a libpcap-based packet sniffer. On the other hand, Bro [4] is an intrusion detection system that works by passively watching traffic seen on a network link in real time. It is built around an event engine that pieces network packets into events that reflect different types of activity.

More sophisticated tools [5; 6] are beyond the scope of this article, but related methods for P2P botnet detection can be found in the *Friends of An Enemy* paper [2]. The dye-pumping algorithm (DPA) [2] relies on network data, typically in the form of simplified pcap data, Cisco NetFlow, or derivative formats such as IPFIX [7] or SiLK [8].

## 3 Dye-Pumping Algorithm

The DPA provides a list of hosts that belong to the same P2P botnet as the discovered bot, with a given degree

of certainty. Network administrators can use this list as a starting point of their investigation and potentially identify more bots in their network once they have discovered the first one. This relies on the premise that in order to receive commands and updates, peers of a P2P botnet communicate with other peers.

Different bots which are members of the same botnet and located in the same edge network may communicate with different external peers. As demonstrated in the *Friends of An Enemy* paper, in the case of P2P botnets with an unstructured topology, if bots randomly pick peers to communicate with, then there is a high probability that any given pair of P2P bots communicate with at least one common external bot during a given time window. In other words, there is a significant probability a pair of bots within a network have a **mutual contact**.

From packet capture data translated into network flows, we construct the network graph where every node is characterized with its IP address and port. This way, the same host identified by its IP address, but using different ports, will generate different nodes in the network graph. Each time a node $x$ connects to a node $y$, the node $y$ is added to the lists of active and known peers of node $x$. As soon as the connection gets inactive, the node $y$ is deleted from the active peers list of $x$, but stays in the known peers list.

From the network graph, we then extract the mutual contact graph, where every host is a node and two nodes share a bidirectional edge, if they share at least one mutual contact during a given time window. The weight or "capacity" on an edge is the number of mutual contacts shared between the corresponding hosts incident on the edge. Then, given a discovered bot which we will call the *seed bot*, we use the mutual contact graph as input to the DPA, which identifies other potential members of the botnet by iteratively computing a level of confidence for each host in the graph. We declare the hosts with confidence levels higher than a particular *threshold* to be potential members of the same P2P botnet as the *seed bot*.

The DPA evaluates the "private mutual contact" principle as illustrated in Figs. 1 and 2: considering nodes A and B in a network and an external host X, if A is linked to X and X is linked to B then X is a private mutual contact of A and B. This way we could assume that A can reach B in the network. Graphs are a natural representation of the network and allow us to model the mutual contact concept. In our mutual contact analysis we restrict ourselves to private mutual-contacts, which are mutual contacts communicating with less than $k$ internal hosts during a given observation window (vs. other mutual contacts such as DNS servers). It is very unlikely that external peers, which are members of the same botnet, will be communicating with many internal hosts that do not belong to the botnet. Therefore, private mutual-contacts can be considered strong indicators of peer relationships among hosts within a botnet. In the

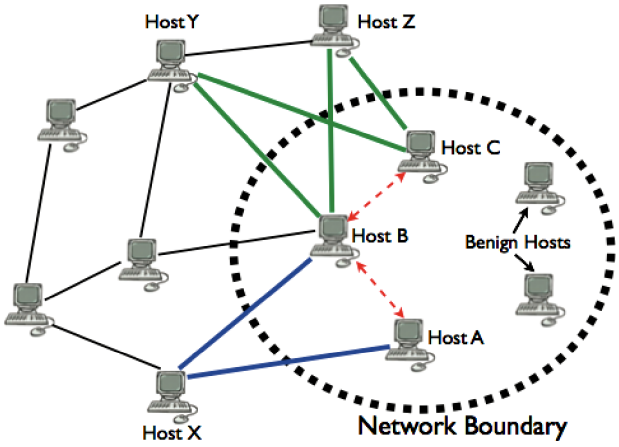


**Figure 1** Botnet communication (from [2]).

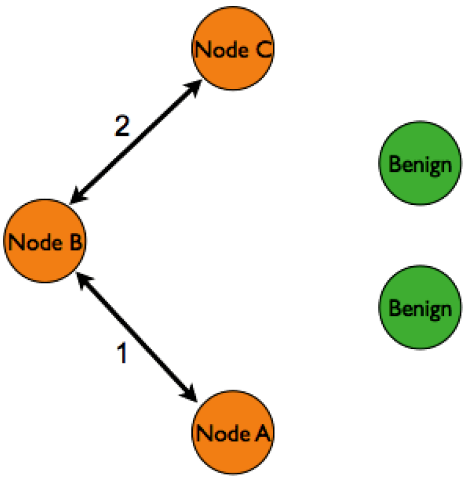


**Figure 2** Mutual contact graph (from [2]).

rest of this paper, we use the term mutual-contacts to mean private mutual-contacts.

### 3.1 Iterative Algorithm Principle

Once the mutual-contact graph $\Gamma$ is structured and generated, the DPA is run to compute the confidence levels of potential membership of a host in the botnet. In the mutual-contact graph, the DPA iteratively pumps dye from the *seed bot* node and distributes it to other nodes. Then, the DPA picks the nodes which accumulate more dye than a particular *threshold* during the round.

At some point, some nodes will accumulate more dye than others, thanks to the distribution based on the dye-attraction heuristic which estimates the confidence of node $i$ being a P2P bot given that node $j$ is a P2P bot. The dye-attraction coefficient indicates what portion of the dye arriving at node $j$ will be distributed to node $i$ in the next iteration. At each iteration each node of $\Gamma$ updates its dye accumulation level. However as dye will be pumped from the *seed bot* node, to avoid interference with "legitimate" P2P traffic, our dye-attraction heuristic computes the dye-attraction level from node $j$ to node $i$ as the ratio of the edge capacity between node $i$ and node

$j$ by the degree of node $i$. The node degree is defined as the number of neighbors or edges that the node has.

The more private mutual-contacts two nodes $i$ and $j$ share, the higher the confidence level is between them. Also, the confidence is reduced if node $i$ shares mutual-contacts with many other nodes in the graph. Different applications other than botnet C&C can interfere with this approach, so if a host shares mutual-contacts with many other hosts, then these mutual-contacts are not necessarily linked to a C&C botnet activity.

In practice, the DPA takes three inputs: the mutual contact graph, the index $s$ of the *seeder bot* node and the number of iteration called *maxIter*. The DPA first computes the dye-attraction coefficient of each node based on the edges capacities. Each one of these coefficients is stored in a $n \times n$ square matrix indexed by node 1 to node $n$ (where $n$ is the number of nodes in the mutual-contact graph). After a stochastic normalization of the matrix, the DPA pumps dye from the given *seeder bot* node. The pumping mechanism first fills the *seed bot* node with dye and leaves all the other nodes unchanged. Then, the dye is pumped from the *seed bot* node across the mutual-contact graph proportionally to the edge capacities. During the first round, except the *seed bot* node, all nodes have zero dye accumulated, and at the end of each round, this value is updated.

Once the algorithm reaches *maxIter* rounds, we have the output $L$, a vector such that each $L[i]$ indicates the confidence level of node $i$ being a member of the same P2P botnet of the *seeder bot* node set as input. A null dye-level accumulation in $L[i]$ or below a certain *threshold*, indicates a low confidence level for the corresponding hosts belonging to the same botnet as the *seeder bot* node. We reduce the vector $L$ to only nodes having a sufficient dye-level (greater than a fixed *threshold*) to produce the DPA output.

### 3.2 DPA Input Formatting with SiLK

As mentioned before, the DPA takes a graph representation of the network connections as input. To build the network graph from which we extract the mutual-contact graph, we used the SiLK Tools Suite [8]. We used the SiLK output format as input for our algorithm. SiLK is composed as a modular framework that allows to easily implement plugins and is ideally suited for analyzing traffic on the backbone or border of a large, distributed enterprise or mid-sized ISP. For the testing phase, the SiLK setup is reduced to using only a small network, but in the long term we are designing our algorithm implementation to perform on larger networks.

In order to have a broader view on peer connections, we observe only network flows rather than packets. Similarly to the NetFlow Protocol [9], IPFIX considers a flow to be any number of packets observed in a specific time slot and sharing a number of properties. Keeping in mind to take advantage of SiLK tool suite, we use Yet Another Flow-meter (YAF) to translate raw packet capture (pcap) data to the SiLK network flow format, which helps preserve flow integrity.

We tweak the output logs using the *rwfilter* function which can then be parsed by our DPA implementation. In the case of very large-scale networks, tera-logs will be generated and require high computational resources and disk space. Since SiLK network flow files are much smaller than packet capture files, the pcap log files could be deleted right after SiLK filtering and conversion, by taking advantage of automated log organization tools of the SiLK Tools Suite in order to preserve disk space.

## 4 Experiment

For our experiment, we set up a dedicated hardware (physical network topology) and software infrastructure (Virtual Machines and Virtual Network Interfaces), so that we could manage and control all the traffic generated by every VM performing malicious code and bots from the host system.

We generated real botnet traffic data to act as input to the DPA. Our monitoring locations were varied: in the VM, on the gateway used by the VM, and as the host is a machine in a network itself, the whole local network, assuming knowledge of the topology and proper access to the network routing and gateway devices.

### 4.1 Experiment Protocol

We ran the DPA on several distinct botnets other than Nugache [10], for which we had recorded network flow data from the last 4 years. We used packet capture to monitor peer communication taking place in several Qemu-based VMs running Windows XP guests. This dedicated VM infrastructure constitutes the core of our botnet zoo. To build the mutual contact graph, our "who talks to whom" method can measure sequence, frequency and packet size to provide partial network knowledge on any information or command exchange.

We validated our algorithm on P2P botnets (Koobface v1 and v2, Waledac v1 and v2) and non-P2P botnets (Zeus v5 and v10) during several time windows. We computed the mutual-contact graph of our network to locate other members of each botnet thanks to one known *seed bot* peer for each one of these bots.

As explained in previous work, we don't look at the content of the packet, but instead at who is connected or related to whom. Knowledge of a peer IP address in the botnet is what allows us to know whether a probable peer, returned by the algorithm, is a false positive or not, giving us ground truth. For Nugache, the data records only contained network flows of confirmed bots, so to obtain a more realistic situation we had to blend the botnet data with background traffic. For the other bots, as they are all active in different VMs behind a NAT, each VM generated both bot traffic and user traffic (such as contacting Web 2.0 pages, checking for mail updates, etc.).

### 4.2 Results and Analysis

Details on the Nugache botnet can be found in [11]. In summary, the C&C protocol of Nugache is P2P-based, without a central C&C server. A Waledac [12] node uses HTTP requests and responses to communicate between peers and to update its spam campaigns. Details on the Koobface and Zeus network botnet can be found in [13] and [14].

In Fig. 3, we consider the tweaking of the threshold parameter, the one that gives us higher confidence in the resulting hosts being other bots, namely the friends of the enemy. Selecting a low threshold generates many false positives. On the other hand, setting a high threshold generates less potential hosts. This is due to the more restrictive filtering on the background traffic, bypassing a large portion of the benign hosts. So below 10%, the algorithm finds a large amount of probable botnet members. Combining a very low threshold with very few iterations of the dye-pumping algorithm (two, as shown here), led to a weak precision by generating false positives. Beyond 80% another plateau indicates results closer to ground truth.

For Waledac 1, it turns out the bot was detected and quarantined by the Windows VM environment. For Waledac 2 and Koobface, not that many IP addresses were involved and collected (up to 24 at most). We can deduce that this class of bots uses many ports on the same machine. This is what [12] explains: in its communication phase, to establish a connection with a node, Waledac randomly opens a local port on the compromised computer and attempts to connect to port 80 of the remote Waledac relay node. In order to connect to a malicious malformed PHP page, Koobface bots exhibit a similar behavior.

For Koobface, as seen in Fig. 4, running one or 32 rounds is almost the same in terms of output. In the first round we detected almost every probable peer involved in the botnet. The Zeus curve looks very similar to the Koobface one, and for this time window, after 8 iterations we detected almost every probable member of the botnet. Even if Waledac 2 requires some extra rounds

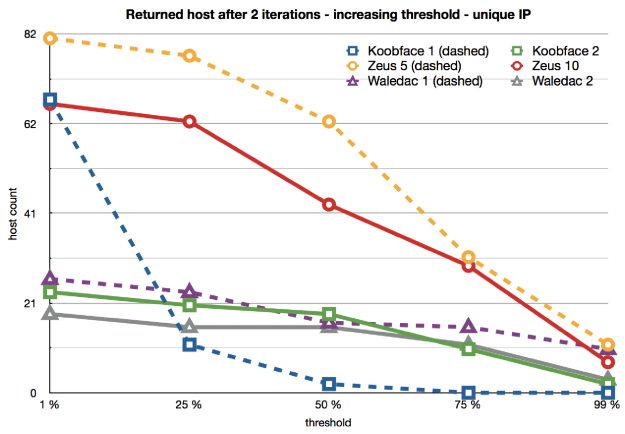


**Figure 3** Returned host count vs. unique IP.

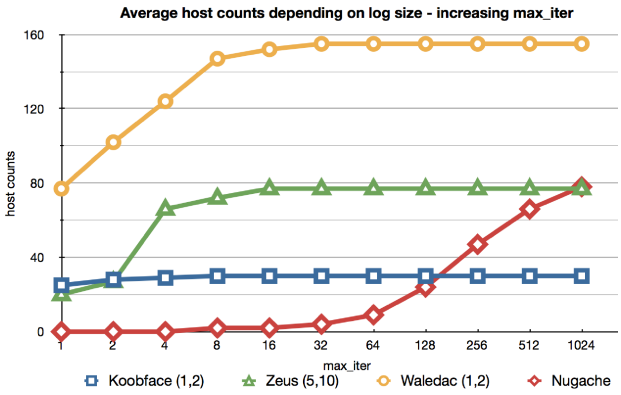


**Figure 4** Average host count vs. max_iter.

for the detection to be complete, it exhibits the same behavior as Waledac 1. Generally, depending on the nature of the bot, an increasing number of iterations led the DPA to produce more, if not as much results as with only one round. A different *maxIter* input parameter must be calibrated and then defined in order to avoid waste of computational resources and achieve a gain in efficiency. The naive implementation of DPA yields a complexity cubic in the number of nodes [2].

We can also notice that simply increasing the number of iterations doesn't significantly improve the results. At eight rounds, based on this data, most of the botnet detection results reach a ceiling. This is due to the small amount of data recorded up to that point. Indeed, as long as we get fed new records of network data, our tests show that the number of iterations has a bigger impact on the DPA output. This is demonstrated in Fig. 4, where we took the average of the output by bot family (Koobface, Zeus, Waledac) and compared to the results obtained with Nugache. First, we notice that Nugache is the most deceptive botnet in this set as we need almost 32 times more iterations before initial detection. Second, while the other botnets reach a ceiling due to the small amount of collected data, Nugache shows a different progression in function of *maxIter* thanks to the availability of multi-year flow records.

## 5 Conclusion

We implemented, tested and refined the *Friends of an Enemy* approach, specifically the Dye-Pumping Algorithm, on several botnets. The DPA requires network flow records and an identified *seed bot* peer. We provided some initial results in testing the parameters of the algorithm on larger data sets, both from previous and recent experiments with contained malware.

Using the network flow data analysis tools provided by SiLK, we can identify suspect traffic as almost all existing IDS [15] and NIDS do today, by signature for example. We can also launch the DPA on suspect peers, and model their dye expansion graph. According to the structure of this expansion graph, we can deduce whether it is a legitimate peer or not with a degree of certainty. We expect to refine this approach for more practical applications.

## References


[1] D. Dittrich and S. Dietrich. P2P as Botnet Command and Control: a Deeper Insight. In: *Proc. of the 3rd Int'l Conf. on Malicious and Unwanted Software*, Malware 2008, pages 46–63, 2008.

[2] B. Coskun, S. Dietrich, and N. Memon. Friends of An Enemy: Identifying Local Members of Peer-to-Peer Botnets Using Mutual Contacts. In: *Proc. of the Annual Computer Security Applications Conf.*, ACSAC'10, pages 131–140, 2010.

[3] V. Jacobson, C. Leres, and S. McCanne. *libpcap*. Lawrence Berkeley National Laboratory, 1994.

[4] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In: *Computer Networks*, 31(23–24):2435–2463, 1999.

[5] E. Stinson and J. C. Mitchell. Characterizing Bots' Remote Control Behavior. In: *Proc. of the 4th Detection of Intrusions, Malware and Vulnerability Assessment Conf.*, DIMVA'07, pages 89–108. LNCS 4579, Springer Verlag 2007.

[6] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha and J. C. Mitchell. A Layered Architecture for Detecting Malicious Behaviors. In: *Proc. of the 11th Int'l Symp. on Recent Advances in Intrusion Detection*, RAID'08, pages 78–97. LNCS 5230, Springer Verlag 2008.

[7] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken. *RFC5153: IP Flow Information Export (IPFIX) Implementation Guidelines*, IETF, April 2008.

[8] CERT. *The SiLK Reference Guide (SiLK 2.4.5)*. CERT Network Situational Awareness, http://tools.netsa.cert.org/silk/, Carnegie Mellon University, February 2011.

[9] B. Claise (ed). *RFC3954: Cisco Systems NetFlow Services Export Version 9*, IETF, October 2004.

[10] D. Dittrich and S. Dietrich. *Discovery techniques for P2P botnets*. Stevens CS Technical Report 2008-4, September 2008. Revised April 2009.

[11] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache trojans: P2P is here. In: USENIX ;login:, 32(6)8–17, 2007.

[12] G. Tenebro. W32.Waledac Threat Analysis. In: Symantec Security Response blog, http://www.symantec.com/connect/blogs/waledac-overview, August 2010.

[13] J. Baltazar, J. Costoya, and R. Flores. The Real Face of KOOBFACE: The Largest Web 2.0 Botnet Explained. In: *Trend Micro Threat Research*, http://us.trendmicro.com/imperia/md/content/us/trendwatch/researchandanalysis/the_real_face_of_koobface_jul2009.pdf, July 2009.

[14] K. Stevens and D. Jackson. *ZeuS Banking Trojan Report*, SecureWorks Counter Threat Unit, http://www.secureworks.com/research/threats/zeus/, March 2010.

[15] K. Scarfone and P. Mell. *Guide to Intrusion Detection and Prevention Systems (IDPS)*. Recommendations of the National Institute of Standards and Technology. Special Publication 800-94, February 2007.

[16] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. BotGrep: finding P2P bots with structured graph analysis. In: *Proc. of the 19th USENIX Security Conf.*, pages 95–110, 2010.


Received: September 11, 2011, accepted: December 4, 2011

**Lionel Rivière** was a Master's student in Computer Science from the University of Bordeaux, France when he was visiting Stevens Institute of Technology for the summer of 2011. He graduated in September 2011 with a Cryptology and Cybersecurity Research Masters Degree. His interests are in software and network security, network administration, cryptology, encryption algorithms and their applications to smartcard environments, network authentication mechanisms and e-voting.

Address: Stevens Institute of Technology, 07030 Hoboken, USA, e-mail: lriviere@cs.stevens.edu

**Dr. Sven Dietrich** is an assistant professor in the Computer Science department at the Stevens Institute of Technology. Prior to joining Stevens in August 2007, he was a Senior Member of the Technical Staff at CERT Research at Carnegie Mellon University and also held an appointment at the Carnegie Mellon University CyLab, a university-wide cybersecurity research and education initiative. He taught cryptography in the Mathematics and Computer Science Department at Duquesne University in Spring 2007. From 1997 to 2001, he was a senior security architect at the NASA Goddard Space Flight Center, where he observed and analyzed the first distributed denial-of-service attacks against the University of Minnesota in 1999. He taught Mathematics and Computer Science as adjunct faculty at Adelphi University, his alma mater, from 1991 to 1997. His research interests include computer and network security, anonymity, cryptographic protocols, and cryptography. His previous work has included a formal analysis of the secure sockets layer protocol (SSL), intrusion detection, analysis of distributed denial-of-service tools, and the security of IP communications in space. His publications include the book Internet Denial of Service: Attack and Defense Mechanisms (Prentice Hall, 2004), as well as recent articles on botnets. He has organized workshops on ethics in computer security research (2009 – present). Dr. Dietrich has a Doctor of Arts in Mathematics, a M. S. in Mathematics, and a B. S. in Computer Science and Mathematics from Adelphi University in Garden City, New York.

Address: Stevens Institute of Technology, 07030 Hoboken, USA, e-mail: spock@cs.stevens.edu