A Fast Worm Scan Detection Tool for VPN Congestion Avoidance

Arno Wagner*, Thomas Dübendorfer, Roman Hiestand, Christoph Göldi, Bernhard Plattner
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology, ETH-Zentrum, CH-8092 Zurich
{*wagner, duebendorfer, plattner*}*@tik.ee.ethz.ch,*
roman.hiestand@alumni.ethz.ch, christoph@goeldi.org

Contact author: Arno Wagner

Abstract

Finding the cause for congested virtual private network (VPN) links that connect an office network over the Internet to remote subsidiaries can be a hassle. Scan traffic of worm infected hosts is one important possible cause. We developed a scan detection tool, which continuously monitors network traffic on VPN gateway(s) and that reliably detects and reports worm infected hosts by tracking anomalous TCP, UDP and ICMP traffic. Our tool is not sensitive to most P2P software and was successfully tested on real production traffic as well as with traces of captured real and simulated worm traffic. Our various tests demonstrated a low false positive rate and a high detection rate. Our open source tool is an extension to the free intrusion detection system Bro. It was developed jointly by ETH Zurich and Open Systems, a company offering managed security services, one of which is based on the presented worm scan detection tool.

Keywords: scan, detection, worm, VPN, gateway, Bro

1 Introduction

Many enterprises connect their subsidiaries over the Internet or through leased lines using Virtual Private Network (VPN) links. One frequent problem in the operation of such an overlay network is detection, diagnosis and correction of link congestion problems. An important source of such problems is malicious traffic such as scan traffic from worm-infected hosts. Typically infected machines are brought in from outside, e.g. by employees that have that work on their laptops both in the organisational LAN and at home, but have insufficient security measures in place.

*Partially funded by the Swiss National Science Foundation under grant 200021-102026/1 and SWITCH.

Depending on the worm in question, the scanning rate can be latency-limited, for example in the case of the Code Red worm [8] that tries to open a TCP connection to each target. A latency-limited worm generates less traffic for links with high latency. The scanning rate can also be bandwidth-limited, as in the case of the Witty worm [17], which uses a single packet UDP exploit and exhausts the available uplink bandwidth of the infected host with worm traffic, since it does not have to wait for answers from the target hosts.

In both cases manual analysis of the traffic in the link has to be done first to determine the nature of the traffic degrading link performance. This is made more difficult by the presence of varying normal, non-attack traffic. In a second step, the infected hosts have to be identified and the site operator has to be contacted and asked to stop the hosts from generating scan traffic, usually by shutting them down.

This process is labour intensive and can take hours when done manually. This paper presents an automatic scan detection system that resides on the VPN gateways (which can be ordinary computers running Linux). The system presented in this paper is capable of detecting worm scans for TCP, UDP and ICMP based scanning strategies. It reports scan characteristics and infected hosts to the overlay network operator. This dramatically increases response time and reduces the effort needed in dealing with this type of problem significantly. Note that the primary goal of using such a system is not the detection of infected hosts, but the protection of the VPN connectivity against degradation due to the scan traffic.

The chosen approach is based on the observation that for TCP normal connections are bidirectional, i.e. connection attempts are typically successful. In scan traffic generated by a TCP based worm many scans try to connect to IP addresses that are not assigned to a host and therefore fail. Furthermore infected hosts try to connect to many different

hosts within a short time, while uninfected hosts typically connect to a comparatively small number of other hosts in the same time interval.

In order to detect UDP or ICMP worm scans, the method is modified with different threshold values and the importance of failed connection-attempts is de-emphasised. For ICMP scans answers of type ICMP *destination unreachable* are counted as failed connections. Generally, detecting UDP and ICMP scan traffic takes longer than detecting TCP scan traffic, but the detection times are still within acceptable limits.

One important characteristic of any intrusion detection system (IDS) is a low rate of false positives. The presented scan detection method has been validated on real VPN links. The impact of scans that P2P filesharing applications generate during start-up has been studied and it was found that these applications usually do not trigger our scan detector. The method is also able to distinguish between worm scans and Denial-of-Service (DoS) attacks that flood a single target or a small number of targets with a stream of scan-like traffic.

The scan detection tool was developed in a cooperation between the Communication Systems Group at the Swiss Federal Institute of Technology Zurich (ETHZ) and OpenSystems, Zurich, a company that offers, among other security services, managed VPN networks to companies worldwide. The administration of the VPN links and thereby also the operation of our present worm scan detector is done remotely from the central OpenSystems network operations centre (NOC) in Zurich.

2 Related Work

We tested several existing algorithms for worm and traffic anomaly detection by applying them to network traffic captured on productive VPN gateways. Due to the nature of a worm an infected computer needs to scan many others in order to propagate. One main characteristic of scans is that many connections that an infected hosts tries to establish fail due to filtering, non-existence of the target host or service.

For TCP two common cases of a failed connection attempt occur, which are (1) no answer within a predefined timeout and (2) a TCP RST packet as answer to the TCP SYN packet. UDP is not connection oriented and consequently the receiving host does not have to send an answer. Nevertheless ICMP "Destination Unreachable" messages in response to UDP packets can be interpreted as unsuccessful UDP connection attempts.

In 1990, the Network Security Monitor [10] was one of the first intrusion detection tools that implemented the "connection counter" algorithm of the University of California in Davis. It counts the number of connections and can give an idea when a worm is active but also reports benign hosts as infected, if they are more active than they have been before. This method can therefore not be applied to a dynamic network environment.

The "failed connection counter" [15] counts the failed connection attempts. It has shown useful results but is not able to distinct between failed connection attempts resulting from scans and from benign programs.

The "sequential hypothesis test" [12] produced too many false positives in our network setting and we found it not to be well suited for an office environment.

The "entropy" [19] algorithm is based on the idea that the entropy of the source and destination IP addresses and port numbers seen in IP packet headers increases or respectively decreases during an attack. To make entropy calculation fast, an estimation based on the compressibility of the IP header fields is used. This algorithm is known to be a good and economical worm detection algorithm for high speed links, but would have to be adjusted for traffic in an office network. We consider an entropy based algorithm a possible future extension of our scan detection system.

We have implemented the algorithms mentioned above and tested them with simulated and injected real worm traffic combined with benign traffic in a productive office environment of a VPN site. We found that a combination of the different approaches proved most successful for a reliable worm scan detection algorithm, which we describe in the next section.

3 Approach: Scan Traffic Detection in VPN Links

Our algorithm uses network traffic captured on a VPN gateway connecting an office network to its remote subsidiaries over the Internet. The idea is to detect worms on the basis of their typical scan traffic. Many worms search for random targets in the internal subnet or in the whole IP range by sending thousands of packets when scanning for vulnerable targets. Others try to propagate via emails sent to every email address found on the infected host. Our algorithm is based on the detection of those characteristic traffic anomalies.

We define the requirements our algorithm has to fulfill:

- High recognition (true positive) rate
- Very low false positive rate
- Economical use of system resources
- Scalability to variable network sizes
- Adequate response time after infection

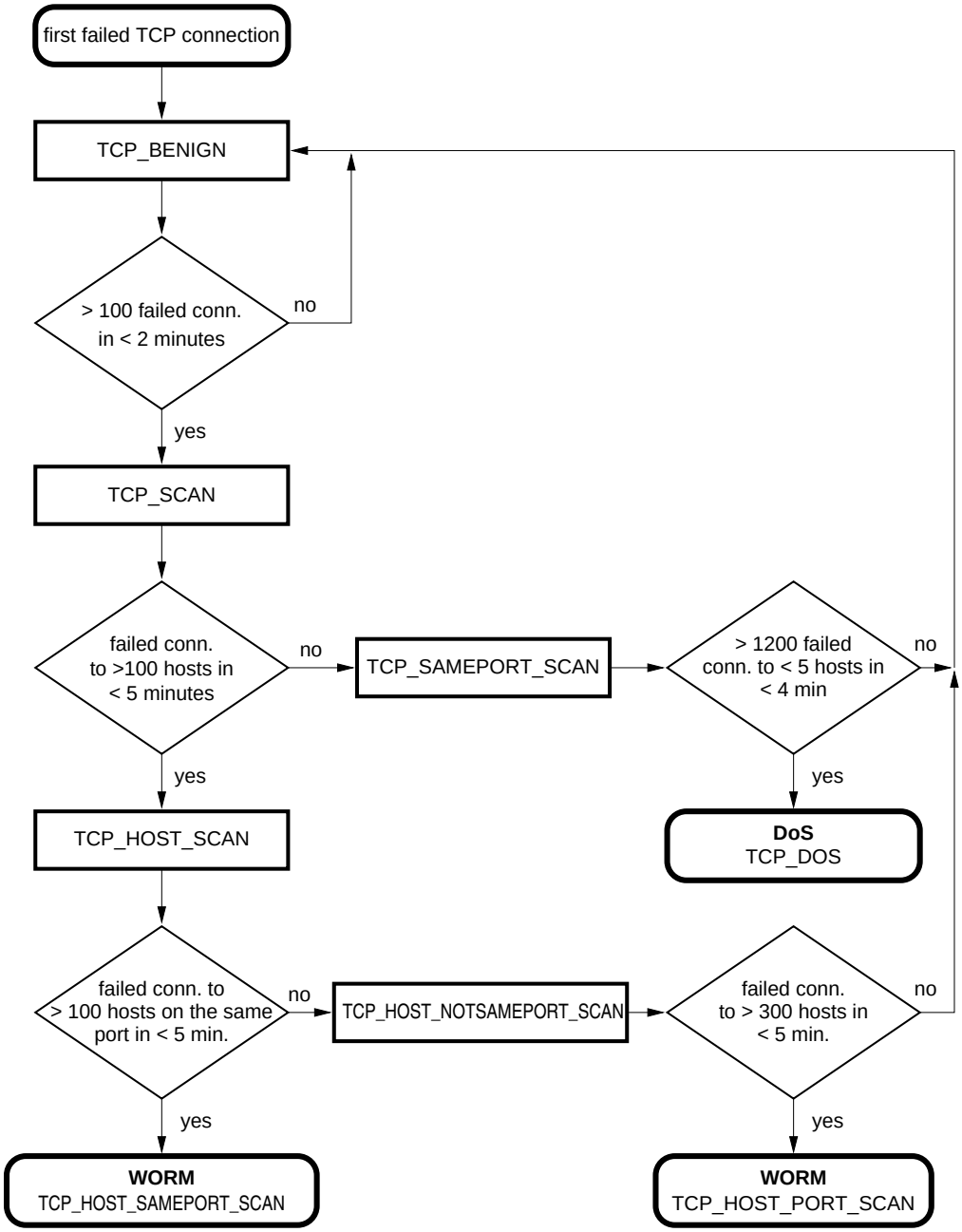

Figure 1. Finite state diagram for TCP scan detection per host

3.1 Adaptive Algorithm

Although our approach is based on similar worm indicators as the other detection methods as discussed in Section 2, its design is different. We use a multi-level approach which employs different views of the network and of single hosts, whith different level of detail. Only hosts which appear to be infected using low-cost checks have to be investigated closer. This adaptive detection method allows to save valuable system resources.

The suggested detection method monitors each host that tries to initiate a connection individually. At first, all of the hosts are monitored and tested for a possibly appearing traffic anomaly. If this general low-cost test marks a host as suspicious, it will be monitored in a second step in more detail. These steps lead to more and more specific tests which

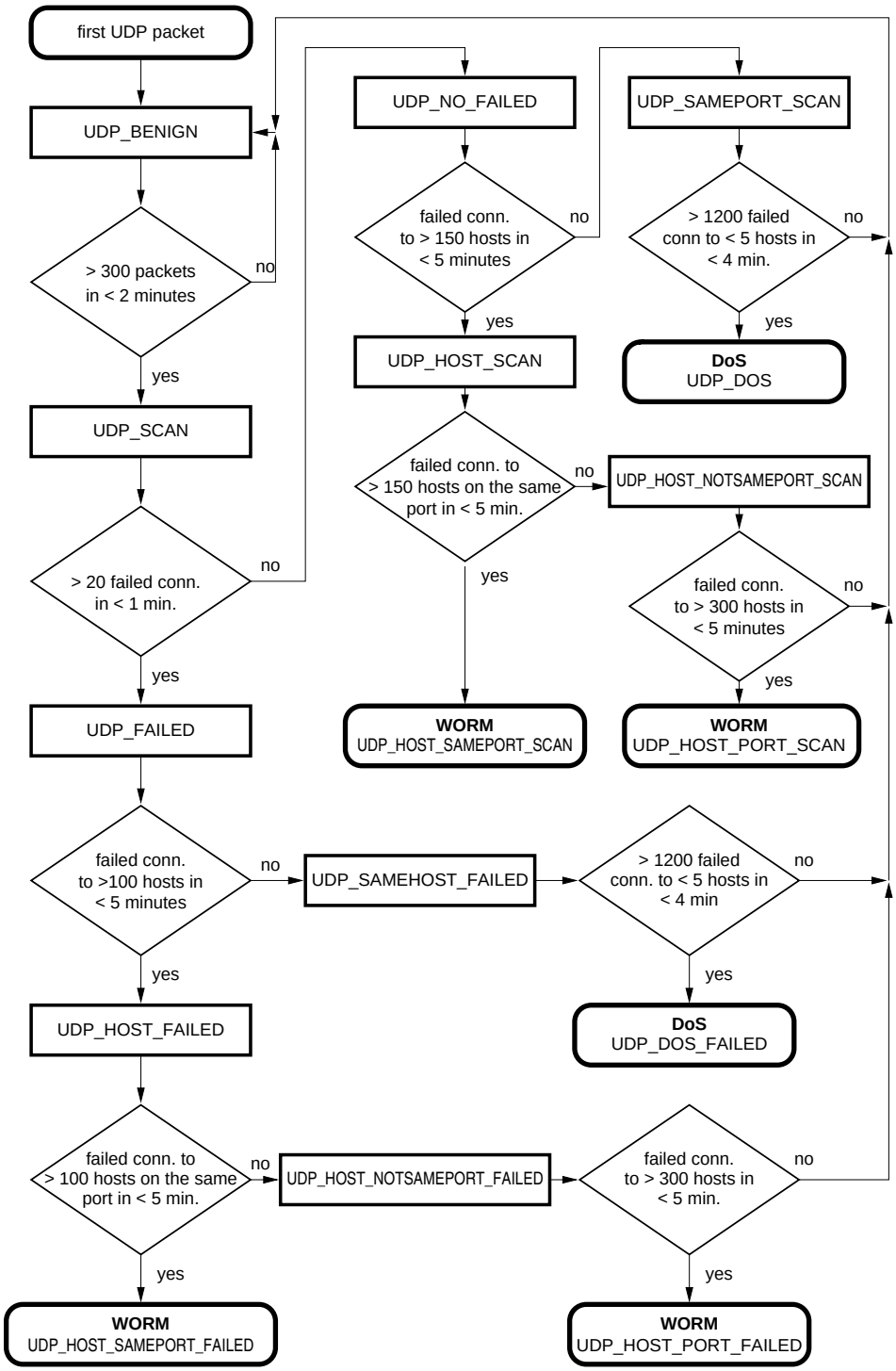

Figure 2. Finite state diagram for UDP scan detection per host

analyse the behaviour of this host. The test can be done with new measurements in each decision node, leading to higher detection latency, or by evaluation all tests on the same data, leading to higher resource consumption. See Section 3.5 for details.

3.2 Algorithm for TCP

Because of the differences between worms, it is necessary to discuss TCP, UDP and ICMP separately. We illustrate the multi-level approach with a flowchart diagram in Figure 1. It shows the shows the states an individual host can be in (rectangles), the tests performed in order to determine state-changes (diamonds) and the default state and final states (rectangles with rounded corners) for TCP scan detection. Hosts without failed connection attempts are not tracked and have no state.

A host that originates a failed connection goes into the TCP_BENIGN state. The number of its failed connections is then counted for up to the given time limit. When the enough failed connections are seen, the new host state is TCP_SCAN. The more detailed tests which follow take into consideration whether packets are sent to one or multiple targets. Multiple targets are typical for a worm, while a single target could be a denial of service attack. The last tests regard the destination port. Scans to the same and to different ports have two different threshold values. Consequently, a host sending packets to different hosts but on the same port reaches the state TCP_HOST_SAMEPORT_SCAN and one scanning to different ports reaches the state TCP_HOST_PORT_SCAN.

3.3 Algorithm for UDP

UDP is not connection-oriented and we cannot expect that each transmitted packet causes the receiving host to answer with a packet. Nevertheless a packet that is not answered by a packet in the other direction with reversed port addresses may still indicate a situation similar to a failed connection. But still we cannot simply count the number of unanswered packets to get the number of failed connections, but have to use a more sophisticated classification scheme. Figure 2 gives the extended state diagram used for tracking hosts receiving UDP packets.

In a first step we monitor all hosts which have sent UDP packets and only consider those further which have sent packets with a rate of at least 5 packets per second. Due to the fact that some firewall or hosts reject UDP (i.e. send an ICMP "Destination Unreachable" packet) packets to nonexisting host or services, while others just ignore them (i.e. drop them quietly), we have to consider these two cases separately in the next levels of checking the sending host for a possible worm infection. The proximate tests are similar to the ones done with TCP but differ based on the firewall behaviour. The detection of a UDP worm with the same scan rate as a TCP worm takes longer because there are more tests done until a final decision about an infection can be made. On the other hand UDP worms tend to be faster, since they often use single-packet exploits, which is basically impossible for TCP worms.

3.4 Algorithm for ICMP

In the past, ICMP "Echo Request" scans have been used by some worms to find out if a target exists and therefore we have to detect these scans too. For unreachable hosts, some firewalls reply to ICMP "Echo Request" packets with ICMP "Destination Unreachable" while others just drop them. We monitor both cases and call them ICMP failed attempts. The second case uses a timeout value. The flowchart diagram for ICMP is similar to TCP, but it is simpler since there is not need for port handling with ICMP. Consequently, a host can only reach one worm state.

3.5 Efficiency Considerations

As an implementation choice, each test can be done sequentailly based on a new measurement. This is the approach we use in our implementation. In each decision node in Figures 1 and 2 a new measurement is done, running not longer than the time stated in the decision node. This leads to low memory needs since each observed connection attempt can be processed immediately and then discarded. Only the state of each observed host and the counter for the test currently done for it needs to be kept in memory. The maximum detection time is the sum of all individual measurement times on a path to a final state in the flowchart. For TCP detection time would be up to 17 minutes to reach the state on the bottom, right in Figure 1 from the point where the first scan traffic is observed from a host. For UDP the maximum detection time is 18 minutes to reach the lower or upper right state in Figure 2 after the first connection from an infected host was seen.

Keep in mind that the conditions are evaluated incrementially, i.e. a test can be sucessfully evaluated when either the number of specific failed connections has been observed or the time limit has been exceeded. Especially for hosts that generate a lot of scan traffic, detection is significantly faster than the upper limits. This measn that the greater the amount of scan-traffic from a host, and hence the potential impact on the VPN link, the faster the scanning host will be identified. For this reason the maximum detection latency is a secondary concern. Since most active hosts never leave the state TCP_BENIGN or UDP_BENINGN respectively, this approach is very memory efficient. As a result our implementation is especially suitable to run on VPN gateway nodes with limited resources.

Alternatively, input data could be stored and re-evaluated for each decision node in the flowcharts. This would require storing up to 5 minutes of observed network data for each host that enters state TCP_BENIGN or UDP_BENINGN respectively. This data interval could then be used to run

through the complete flowchart in an incremental fashion, i.e. whenever a test cannot be evaluated conclusively, additional data is recorded until it can. The disadvantage is that a lot of data has to be kept in memory, while there is only a moderate speed gain. Still, if maximum detection latency is the most important consideration, this approach could be used to implement our detection algorithm.

4 Implementation

We use the freely available Bro IDS Framework [1] to implement our scan detection algorithm. Bro is designed for high-speed monitoring of network traffic and real-time notification. The architecture of Bro allows integrating own algorithms utilizing all the functionality which Bro provides. This can be done using the scripting language Bro provides. The so called policy script interpreter translates all scripts to C code when the program is started.

Bro is based on libpcap library, which makes it highly portable and lets Bro run on recorded tcpdump files as an alternative to monitoring live traffic on a network interface. Additionally, libpcap can be instructed to pass only specific packets to Bro and thereby reduce the traffic load which Bro has to process.

4.1 Scan Detection Policy Scripts

The implementation of our algorithm has been done by writing policy script files for the Bro IDS framework. The scan detection architecture and the corresponding Bro policy script files which contain the implementation are shown in Figure 3. The main file `osag-sd.bro` adjusts Bro internal settings and contains global variables and tables. Further, it includes all parameters which control the scan detection and loads the files shown in the lowest layer in Figure 3.

The four files `osag-tcp.bro, osag-smtp.bro, osag-udp.bro` and `osag-icmp.bro` implement the scan detection for the different protocols. This includes the interception of events, the subsequent calling of the corresponding functions and the finite state machines which control the state of each source host seen.

The functions for counting events and checking the behaviour of the source hosts are implemented in the files shown in the third line of Figure 3. The exceedance of thresholds will be recognized here.

The notification when a host changes its state requires additional functions, which are implemented in the file `osag-notification.bro`. This file provides functions for saving information about the behaviour of a suspicious host and for writing this information to files or for passing it to syslog [16]. Different types of messages can be written to syslog depending on the type of the occurring event. Hosts which never reach a worm state but several time pass the first test are reported as *suspicious*.

The script file `osag-infection.bro` provides functions which are called when a host is recognised as infected. A list of these hosts is maintained. An infected host remains 24 hours in this list and during this time packets from this host are not observed any more. A Bro mechanism allows us to update the libpcap filter and block the packet stream of such an infected host on a lower level. This can save a lot of system resources when an infected host has been detected.

4.2 IP Spoofing

An infected host could send packets with faked IP source addresses. This behaviour is known as IP spoofing. In our solution we store the state and several table entries for each source IP address. Consequently, a scanning host sending packets which all have different source IP addresses causes a high memory and CPU consumption. Therefore, we have to deal with this issue.

The length of each host state table is tracked and warnings are written to syslog if a table exceeds a predefined length (e.g. number of actual hosts in an observed office network). Additionally, an external process can be started to observe the CPU and memory usage of Bro and to restart Bro if CPU and/or memory usage exceed a certain limit.

4.3 Resource Consumption

The resource consumption of the detection tool is an important issue since we want to run it on the VPN gateway together with firewall and other services. The consumption of CPU and memory mainly depends on the amount of scan traffic and the number of infected hosts in the observed network. We have tested the scan detection tool with one and with several infected hosts which were scanning for targets with a high rate. The scan detection algorithm was running on a VPN gateway with an Intel x86 Pentium 4 2.4 GHz processor, 1 GB RAM, two 100 Mbit/s and two 1 Gbit/sec network interfaces. The VPN gateway is running on a highly customized Linux (Kernel 2.4).

The number of infected hosts has a big impact on the resource consumption of the scan detection of Bro and therefore, we simulated different numbers of infected hosts. To simulate the worm attacks we have used the MACE [13] worm simulation tool. The performance tests showed nearly the same results for UDP and TCP. Therefore, all the following conclusions which are presented for TCP also hold for UDP.

Figure 4 shows the CPU and memory usage when four hosts are infected. If up to four hosts are infected, the scan detection needs less than 1% CPU time and less than 8 MB of memory.

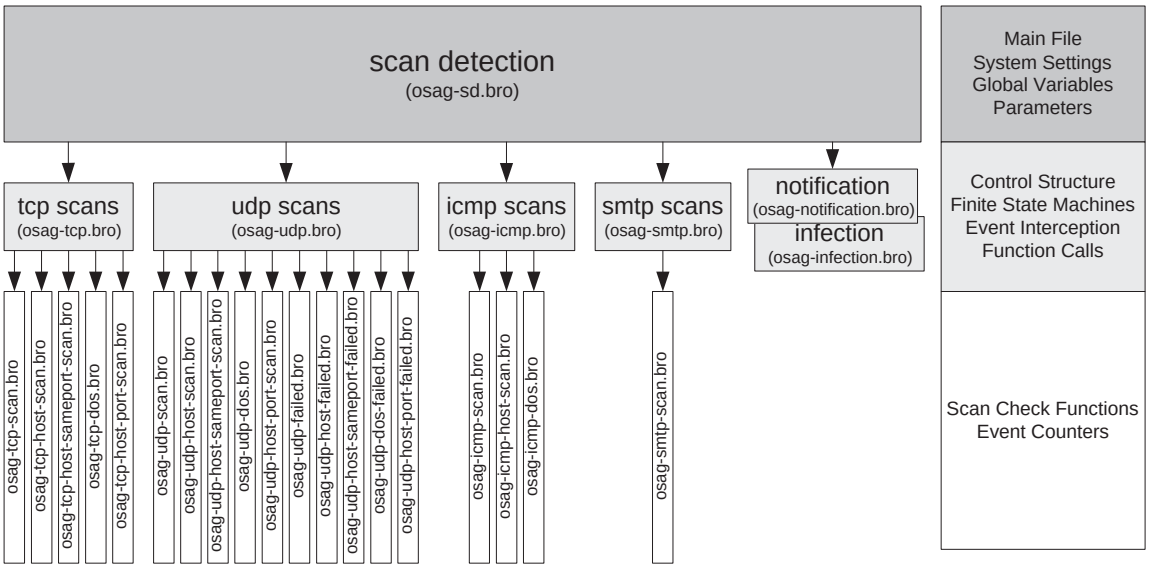

Figure 3. Scan detection architecture and corresponding implementation files

The detection of 252 infected hosts uses up to 75% CPU during one minute. Figure 5 shows that after 2.5 minutes when all 252 infected hosts have been detected, the CPU usage falls off as the traffic of these hosts is now excluded from capturing by libpcap. The memory usage does not exceed 20 MB.

Consequently, we can summarise that the scan detection tool does not exceed a memory usage of 20 MB and the CPU usage is quite low during normal operation. CPU load increases in case of many infected host, but only for a short time.

4.4 P2P Traffic

P2P overlay networks and their clients are used to e.g. share files, make phone calls or exchange instant messages over the Internet. We found that such P2P clients are still in frequent use also within companies.

Many P2P clients scan for other clients using host lists. These ”contact” lists are built over time and often contain many hosts which are unreachable. Seome do not run the specific P2P client anymore, while others are simply offline or dynamically assigned IP addresses that refer to changing hosts. The scan for these clients is similar to a worm scan for targets. Depending on the list length and the scan rate hosts with P2P clients can cause the scan detection tool to generate suspicion messages or warnings.

Most of the common P2P clients* like Freenet [4], Kazaa Lite [5], DC++ [2] and Limewire [6] have not caused the scan detection tool to generate any messages.

eMule† [3] caused the algorithm to generate a suspicion warning if the client could not connect to the P2P network because all the traffic was blocked by the firewall. When an eMule client was connected to the network and searches were performed, it sent hundreds of packets to hosts which did not reply and therefore it was detected as a worm. This case could be reported as a false positive pertaining to worm detection. However, from a network operations perspective, the detection of eMule can be rated as a success because of the large amount of (unwanted) scan traffic it generates.

4.5 Worm Detection Validation

Our tool was tested with real worms - among others with Blaster [7] and SQL Slammer [14].

According to our specification a host has to send 300 TCP packets on the same port until it is detected as infected. Blaster was reported to scan with $\sim$11 scan packets/s and therefore, we expect to detect it within approximately 27 seconds. Because the Blaster worm in our setting started to scan at a much lower rate with $\sim$3 scan packets/s the detection took longer and we detected this worm 57.4 seconds after the infected host had sent its first scan packet.

The detection of a SQL Slammer infected SQL server is highly dependent on the firewall settings. We tested it with a firewall that does not send any ICMP unreachable packets and detected it therefore rather late after 74.86 seconds. In an environment with ICMP unreachable packets it would be detected within less than 10 seconds.

*We tested Freenet v0.5.2.8, Kazaa Lite v2.61d, DC++ v0.668 and Limewire v4.2.6.

†We tested eMule v0.44d.

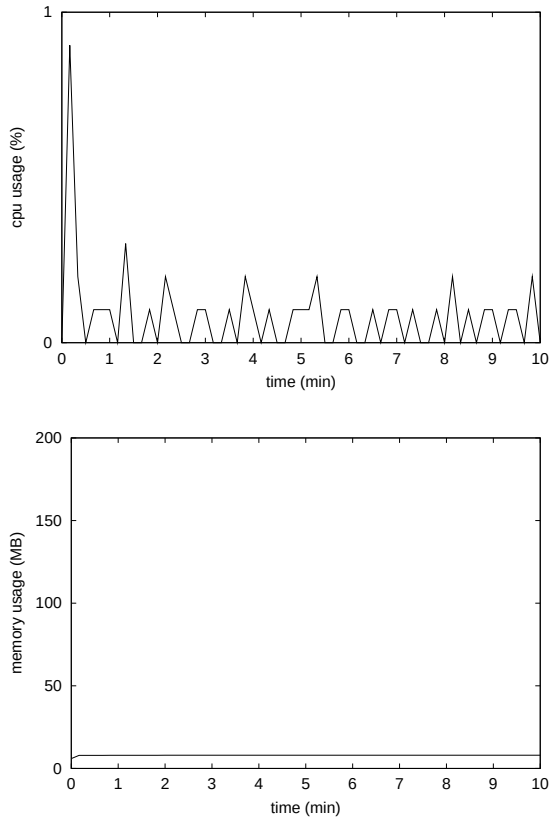

Figure 4. CPU and memory usage with four infected hosts (TCP worm)

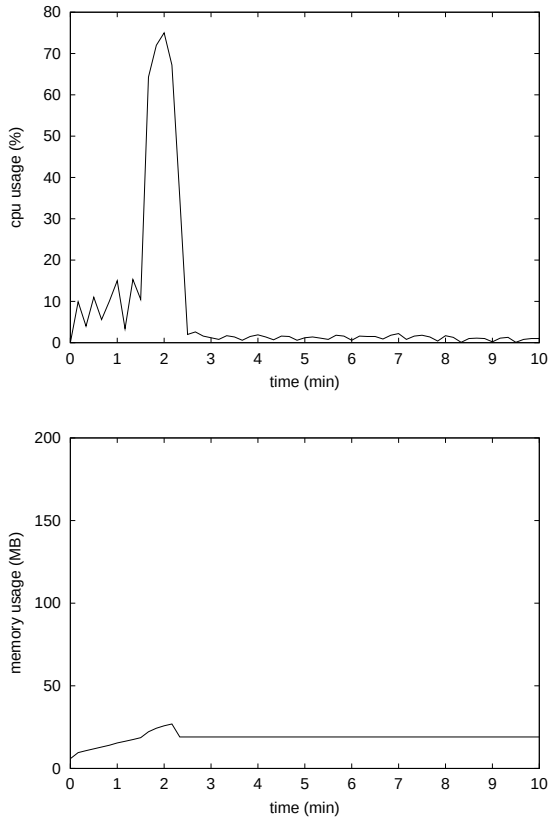

Figure 5. CPU and memory usage with 252 infected hosts (TCP worm)

The scan detection tool has detected all the worms within a reasonable time. Further we have tested the tool with more than 22 hours of productive office traffic at 15 different sites of various companies worldwide and the algorithm has not caused any false positives in all these tests. The tool has proven to have a very low false positive rate, with the one exception of sensitivity searches done with the eMule P2P client when it is not connected to the eMule network.

5 Conclusions and Outlook

Our scan detection tool uses a new detection algorithm that is a combination of several different approaches for worm detection. Our tool was implemented for the intrusion detection system BRO [1] and installed on several dozens VPN gateways. We could successfully validate it on office network traffic. It reliably detected scan traffic of worm infected hosts while at the same time not being sensitive to P2P traffic, which results in a very low false positive rate.

The algorithm of our tool offers a powerful scan detection using low system resources and is still simple enough such that it can be understand in detail. The tool scales to larger company networks and is also applicable to networks with several hundred infected hosts that are scanning concurrently. Timely detection of maliciously scanning hosts has shown to improve reaction times of network administrators considerably as they were notified by our tool before the users called the helpdesk upon real worm infections. Installation of the tool is quick and thanks to syslog support, the tool's output can be tracked remotely and integrated in most network security information management suites.

The first version of the detection tool was developed in the context of the DDoSVax project [18] and the Master's thesis [11] of Roman Hiestand and Christoph Göldi, which was co-supervised by Open Systems and was awarded the prestigious Fritz Kutter-Preis in 2005 [9].

The source code of the presented scan detection system can be obtained free for non-commercial use by contacting Arno Wagner. Possible extensions to the tool are the support for worm specific traffic signatures in order to identify the exact cause for scan traffic detected or an incorporation of traffic policies (that e.g. state how much scan-like traffic

is tolerable) depending on time and other factors. Detecting P2P traffic that might also congest VPN links would be a complementary extension as well as incorporating additional promising detection algorithms (e.g. entropy based methods) for additional tests for suspicious host behaviour.

6 Acknowledgements

We thank Martin Bosshardt, Stefan Lampart, and Roel Vandewall from Open Systems for providing the possibility and support for this industry inspired research project and their co-supervision of the students. We thank Bernhard Tellenbach for valuable feedback on the paper.

References

[1] Bro intrusion detection system. http://www.bro-ids.org/, June 2005.

[2] Dc++ your files, your ways, no limits. http://dcplusplus.sourceforge.net, Jan. 2005.

[3] emule-project.net - official emule site. downloads, help, docu, news, ... http://www.emule-project.net, Jan. 2005.

[4] The freenet project - index - beginner. http://www.freenetproject.org, Jan. 2005.

[5] K++ / kazaa lite 2.6.1 deutsch - mp3 download software - [mpex.net]. http://www.mpex.net/software/details/kazaalite.html, Jan. 2005.

[6] Limewire.org - open source p2p file sharing. http://www.limewire.org, Jan. 2005.

[7] CERT. Security Advisory: MS.Blaster (CA-2003-20). http://www.cert.org/advisories/CA-2003-20.html, 2004.

[8] R. Danyliw and A. Householder. CERT Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer verflow. http://www.cert.org/advisories/CA-2001-19.html, 2001.

[9] ETHZ. Fritz-Kutter Preis. http://www.kutter-fonds.ethz.ch/preistr.html, 2005.

[10] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the IEEE Computer Society Symposium, Research in Security and Privacy*, pages 296–303, May 1990.

[11] R. Hiestand and C. G"oldi. Scan detection based identification of worm-infected hosts. Master's thesis, ETH Zurich, 2005.

[12] A. W. B. Jaeyeon Jung, Vern Paxson and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.

[13] P. B. Joel Sommers, Vinod Yegneswaran. A framework for malicious workload generation. http://www.cs.wisc.edu/~jsommers/pubs/p82-sommers.pdf, Oct. 2004.

[14] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security and Privacy*, 4(1):33–39, July 2003.

[15] V. Paxson. Bro: A system for detection network intruders in real-time. http://www.ece.cmu.edu/~adrian/731/readings/paxson99-bro.pdf, Jan. 1998.

[16] J. Schoenwaelder. syslog - write messages to the system logger. http://www.infodrom.org/projects/sysklogd/, Jan. 2001.

[17] US-CERT. Vulnerability Note: Witty (VU#947254). http://www.kb.cert.org/vuls/id/947254, 2004.

[18] A. Wagner, T. Dübendorfer, and B. Plattner. The DDoSVax project at ETH Zürich. http://www.tik.ee.ethz.ch/~ddosvax/, 2005.

[19] A. Wagner and B. Plattner. Entropy Based Worm and Anomaly Detection in Fast IP Networks. In *Proceedings of 14th IEEE WET ICE / STCA security workshop*. IEEE, June 2005.