Deep Packet Anonymization

Michael Foukarakis, Demetres Antoniades, Michalis Polychronakis
Institute of Computer Science
Foundation for Research and Technology—Hellas
Heraklion, Crete, Greece
{mfukar,danton,mikepo}@ics.forth.gr

ABSTRACT

Network traces of Internet attacks are among the most valuable resources for network analysts and security researchers. However, organizations and researchers are usually reluctant to share their network data, as network packets may contain private or sensitive information. To alleviate the problem of information leakage, network traces are often anonymized before being shared. Typical anonymization approaches sanitize, or in some cases completely remove, certain packet header fields, higher-level protocol fields, or even payload information that could reveal the source and destination of an attack incident.

Although there exists a variety of network trace anonymization techniques, in this paper we show that in certain cases they are proven inadequate, because attack traces may contain sensitive information not only in the packet headers and the packet payload, which are both exposed "on the wire," but also in the encrypted payload of the self-decrypting shellcode carried in the attack vector of code-injection attacks. To overcome this limitation, we extend an existing network trace anonymization framework to identify and anonymize sensitive information contained in the shellcode of code-injection attack packets. Our approach takes advantage of the certain structure of widely used shellcode decryption schemes to produce fully anonymized attack traces.

1. INTRODUCTION

Logs and traces of network activity are a fundamental resource for security professionals, network analysts, and researchers. They provide the means for understanding network operations and threats, enhance the operational and security policies of an organization, and help in deploying and evaluating new algorithms and applications. Thus, it is widely recognized by the academic and research community that it is both desirable and beneficial to share network data for research purposes.

Due to the rapid growth of the Internet and the multitude of new services and protocols, there have been reported large numbers of cases of misuse and attacks on those services, and more continue being reported each year [10].

Sharing traces of network attacks is a useful practice that promotes research and undoubtedly helps in developing defense mechanisms against current and future threats. However, extensive access to network data and activity logs may also help attackers perform reconnaissance attacks in a network of hosts, e.g., by knowing which hosts of an organization are active, which network services they use, and so on. To reduce such exposure without sacrificing the ability to share useful information, network and system administrators often wish to anonymize network traces and logs before sharing them.

The multitude of application layer protocols that are being used today are well documented and, given the right conditions, they can be easily identified, parsed, and subsequently anonymized. Starting from the Ethernet and IP headers up to higher level protocols, all sensitive fields are known and can be sanitized according to the appropriate anonymization policy. MAC and IP addresses can be mapped to non-existent or randomly chosen addresses, while any payload data that reveal network or system information can be sanitized. For example, the HTTP `Host` filed can be changed to a fake address:

```
Host: 10.123.12.123\r\n
```

while various SMB or DCERPC fields that contain IP addresses, host names, or other identifiers can be sanitized:

```
principal: xxxxxx$@XXXXXX.XXX
Server NetBIOS Name: XXXXXX
Domain DNS Name: xxxxxx.xxx
Path: \\10.123.12.12\IPC$
```

However, when it comes to network traces of code injection attacks, the attack code itself may contain sensitive information that can expose the identities of the attacking or victim hosts. In a code injection attack, the code that is executed after exploiting a vulnerable service is provided as part of the attack vector. Although the typical action of the injected code in the early versions of code injection attacks was to spawn a shell, hereby dubbed shellcode, the typical operation of the shellcode used by malware is to connect back to the previous victim or some seeding server, download the main malware binary, and execute it. The server or previous infected host is directly identifiable once the payloads' behaviour is uncovered, since its IP address, hostname, or URL is usually hardcoded in the shellcode.

EUROSEC '09 Nuremberg, Germany

Preferably, any information about the seeding host should also be anonymized, since it identifies an infected or malicious system—a system that is definitively known to host and spread malware. An attacker could exploit the information contained in publicly available traces of code injection attacks to learn about unpatched or vulnerable systems. In turn, he could launch further attacks against these systems using the network information contained in the shellcode of the released attack traces.

More importantly, revealing host-identifying information about infected systems can raise legal or social concerns. The seeding host—an infected computer—might belong to a third-party organization or a high profile company which would not want the public to know that hosts systems serving malware.

Traces of code injection attacks are invaluable for the security research community, thus identifying and anonymizing any sensitive information contained in the shellcode of code injection attacks is of crucial importance for making such attack traces publicly available. In this paper, we demonstrate an extensible framework for the identification of executable payloads using signatures in the form of regular expressions, and provides users with the ability to anonymize potentially sensitive information contained in the shellcode.

Our main contributions are: a novel approach to identify sensitive information in IP attack packet traces; we describe the system design, demonstrate the feasibility of our approach by providing a prototype implementation which provides the means to hide information that is present inside attack traces, and evaluate its operation on traces of real attacks spanning a time period of one year. This work is a first step towards promoting sharing of packet traces containing malicious or generally executable payloads between organizations without exposing potentially sensitive information carried within the packets' payload.

The rest of the paper is organised as follows. We first discuss the problem and the motivation behind our work in Section 2, and the research related to this work in Section 3. In Section 4 we present the design of our approach and its implementation in *anontool*, a generic network trace anonymization framework. We present results obtained from use of a prototype implementation in Section 5, and conclude in Section 6.

2. BACKGROUND

Network traces of attacks containing executable code are collected and produced by a variety of network monitoring applications. Packet traces can be collected from intrusion detection systems (IDS) [23] containing an attack that has passed through the network. Also, a source of this kind of traces can be deployed honeypots and honeypot networks [2–4, 8], as well as specialized code-injection attack detection tools [20].

In their effort to hide from simple payload-based signature matching identification, attackers often employ polymorphic or metamorphic techniques [5, 22, 24], which typically produce a self-decrypting version of the original shellcode. The initial shellcode is encrypted using a simple and easily reversible encryption algorithm, and a small decryption routine is prepended to the encrypted code. Once the attack succeeds and the program counter of the vulnerable application has been hijacked, the first piece of code executed is the decryption routine, which loops over the encrypted payload data to reveal the actual shellcode that is going to be executed.

```
00000836  8A06          mov al,[esi]
00000838  3C99          cmp al,0x99
0000083A  7505          jnz 0x841
0000083C  46            inc esi
0000083D  8A06          mov al,[esi]
0000083F  2C30          sub al,0x30
00000841  46            inc esi
00000842  3499          xor al,0x99
00000844  8807          mov [edi],al
00000846  47            inc edi
00000847  E2ED          loop 0x836
```

Figure 1: A typical XOR decoder, used by the Wuerzburg shellcode.

An example of such a shellcode is the Wuerzburg shellcode [1], shown in Figure 1, which contains an XOR decoder and a connect-back file transfer code segment that connects to a host and downloads a file named *ftpupd.exe*. In the Wuerzburg shellcode, the IP address and port are XOR'ed with a secondary key (which has the static value `0xAAAAAAAA`) inside the already encrypted shellcode.

In cases like the above, sharing the attack trace as-is may leak information that the organization or owner of the host considers confidential. Any individual can, with little effort considering the available tools, decrypt the payload and extract any host-identifying network information. This information may be used for a subsequent attack to the organization, or for discrediting the organization as hosting vulnerable systems.

Anonymizing the seeding host address is not straightforward. On the wire, the actual shellcode is encrypted, and thus the address of the seeding host cannot be anonymized simply by searching for it in the packet payload and sanitizing it—the address is not exposed in the packet payload at all. The actual address will be revealed only upon execution of the shellcode on the vulnerable system, i.e., after the decryption routine decrypts the encrypted payload.

We further explain the problem by observing an actual instance of the Wuerzburg shellcode from a real attack. Figure 2 shows the original payload of the attack as seen on the wire, captured by a `tcpdump` session. Due to the encryption, the whole payload appears as almost random bytes and seems to contain no interesting information. However, when the payload is decrypted(Figure 2), the actions of the attack can be clearly identified.

The payload first builds a small file containing FTP commands and uses this file as a parameter to the `ftp` program to download the malware binary. In this example, the IP [1] address of the FTP server (128.192.216.37) might be considered as information that the corresponding organization would not like to be released, in order not to expose vulnerable hosts or not be recognized as a vulnerable organization.

3. RELATED WORK

Although recent work on anonymization of network logs has come a long way in providing the essentials to hide sen-

[1]The address in this example has been anonymized by arbitrarily choosing a random IP address.

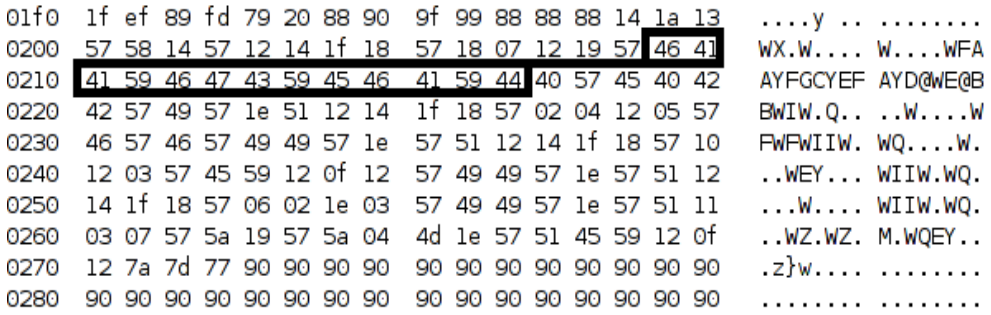

Figure 2: The encrypted part of the Wuerzburg shellcode, as seen on the wire. No sensitive information is seemingly exposed at this stage.

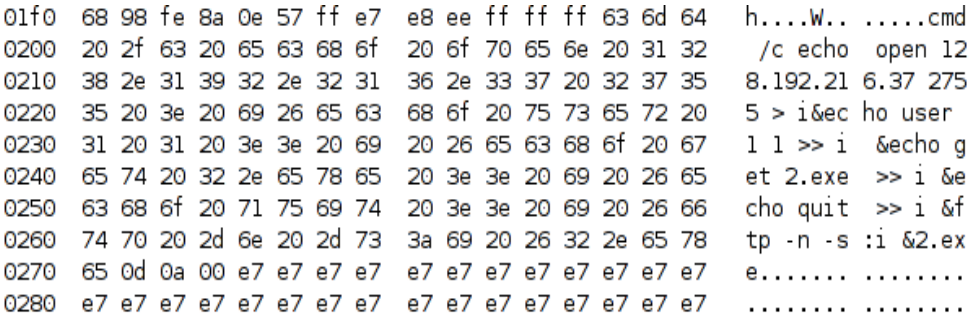

Figure 3: The same part of the Wuerzburg shellcode shown in Figure 2, after decryption. The IP address and port number of the seeding host in the FTP shell commands are now visible.

sitive information and promote information sharing, we are not aware of any automated or semi-automated method of masking sensitive information that may be contained in malicious code encountered in attack packet traces.

Tcpdpriv [14] is a well-known anonymization tool that takes as input traces written in `tcpdump` format and removes sensitive information from packet headers. The TCP or UDP payload is simply removed, while the entire IP payload is discarded for protocols other than TCP or UDP.

Paxson and Pang [19] introduce a way to anonymize the payload of a packet and remove sensitive information instead of removing entirely as the other approaches do. Packets are reconstructed into data stream flows and application level parsers modify the data streams as specified by a policy written in a high-level language. However they do not address the problem of sensitive information being encoded or obfuscated within the payload in binary form.

NetDuDe [16] is a GUI-based tool for interactive editing of packets in `tcpdump` files. NetDuDe itself does not perform parsing of application-level protocols in the payload, but offers the option for plug-ins to perform packet processing such as recomputing checksums. SCRUB-tcpdump [13] is a set of functions that are used to anonymize a packet trace in libpcap format so that it can be shared without jeopardizing the anonymity of the network represented by the captured trace. SCRUB-tcpdump does not perform payload inspection, or application-level protocol decoding.

Mogul et al. [17,18] propose that instead of sharing traces, researchers can send reduction agents to the site that hosts the network data. Despite the serious trust issues that are still present with source code being shipped for execution and the per-sharing request costs in verifying code manually, we believe our approach to be complementary. The kind of code that would be shipped could in fact be a policy in a well-defined language that allows for integrity and privacy checking, such as an XML-derivative.

`Anontool` is a complete and extensible command line tool which enables users to anonymize both live and stored traffic. Its functionality is based upon the Anonymization API (AAPI) [15]. AAPI allows users to write their own anonymization applications by defining which anonymization functions are applied on each field, having complete freedom in choosing their policy. The API provides a large set of anonymization primitives, from setting fields to constant or random values and performing basic mapping functions, to prefix-preserving anonymization and several hash functions and block ciphers, as well as support for regular expression matching and replacement. AAPI can operate on a wide variety of protocols, ranging from Ethernet to HTTP, FTP, and Netflow in the application layer. All protocol fields are being made available for the user application to access.

Modern shellcodes are compact, self-contained pieces of code that exploit a vulnerability in the target service, acquire superuser access privileges and usually connect back to a predefined host which is usually defined within the payload and transfer a rootkit or trojans, or anything that serves the will of the attacker. The methods used by attackers in order to obfuscate their shellcode are several and range from simple XOR encoding to metamorphic payloads. Ways to identify malicious payloads range from simple regular expressions of invariant strings present within the payload (as is most usually the case with Snort [23]) to complex taint

```
"\\xEB\\x27(..)(....)\\x5D\\x33\\xC9\\x66\\xB9..\\x8D"
"\\x75\\x05\\x8B\\xFE\\x8A\\x06\\x3C.\\x75\\x05"
"\\x46\\x8A\\x06\\x2C.\\x46\\x34.\\x88\\x07"
"\\x47\\xE2\\xED\\xEB\\x0A\\xE8\\xDA\\xFF\\xFF\\xFF";
```

Figure 4: A regular expression matching the Wuerzburg shellcode.

analysis within a sandbox environment, such as the Argos honeypot architecture [21].

It becomes clear that there is a gap to be filled when it comes to packet trace anonymization. There are major advantages to promoting attack trace sharing for the computer security research community and industry, and providing the means to perform thorough anonymization could lead to that direction. In the remainder of this paper we are going to relate our work to the modern methods that aim to identify binary code injection attacks, and describe our proposed anonymization approach that can aid in shortening this gap.

4. ARCHITECTURE

4.1 Approach

The first important design decision to be made was to choose the mechanism in order to detect the various types of binary payloads and the sensitive information that may appear within. While there is a vast variety of ways to do that, we chose to implement regular expression matching to identify binary payloads combined with limited emulation to seek and match sensitive information such as IP addresses, URLs, and so on.

The most important reasons that led to that decision are two. First, regular expression matching is fast and can be effectively used in deep packet inspection. It can also be made significantly faster, in case speed is an issue [25]. This provides the user with the option to anonymize attack traces on-the-fly, as they are produced by analysis and detection algorithms and tools. Second, regular expressions are expressive enough to cover both the case where sensitive information such as an IP address appears within the payload of an attack, as well as when that information is masked by an encoder or packer which has to be executed first, before the actual payload is executed.

In both cases, it is important to note that we are not aiming at providing a detection framework; several detection approaches have already been proposed. We assume that our input is a trace of the network activity which is the result of such a framework, when it detects suspicious behaviour. That means the user already knows the specifics of the attack inside the trace and can therefore produce a regular expression to match the encoder, shellcode and private information she wishes to anonymize.

Such regular expressions are easy to produce, share, and acquire. There are many examples of databases of such regular expressions. Most notable are the Snort rule database [11], the *nepenthes* project page [8], as well as several others related mostly to honeypots [2].

Table 1 mentions the names and short descriptions for each binary payload currently supported by `anontool`, as well as the number of attack traces each signature matched. We don't make mention of few other signatures which did not match any of the traces processed. We chose the *nepenthes* project information on shellcodes as a point of reference because of its detailed information related to the code provided. In Figure 4, you can see the regular expression that matches the code in Figure 1.

4.2 Implementation

The core of our implementation uses the PCRE library [9], to search for a given set of regular expressions characterizing different kinds of binary payloads within a packet trace. When found, we provide the user with the option of anonymizing the potentially sensitive information that may be contained in that piece of code. Instances of such information may be a hardcoded IP address inside the shellcode, which is typically the case. The external host information may as well be obtained at runtime, so further inspection or modification of the respective instructions might be needed.

This first pass can handle straightforward shellcode which implements a reverse shell technique. If the matched regular expression identifies a decoder, we need to emulate its behaviour, and then search for host information in the decrypted parts of the payload. The emulation process is carried out on a per-decoder basis. We do not use any emulation frameworks or external processes for this task, because most decoders are currently very simple in their operation. For the decoders in our prototype implementation, we simply emulate in the application level the operations carried out by the decoder. We do this in a very similar way to the nepenthes low interaction honeypot.

When the decryption process finishes, another scan is necessary to identify any possible information that may leak information; IP addresses, port numbers, URLs or anything else that may be used in order to fingerprint a host on the Internet (Figure 2).

The user is then given the opportunity to manipulate all of this information as she deems fit. One should also take into account that there's the possibility for sensitive information to be leaked even when a host name inside a shellcode is anonymized.

For instance, if the shellcode executed on an infected machine opens a connection to a given IP address, say $a.b.c.d$, it is possible that the flow between these two hosts is also captured. Should the sequence of packets that comprise the conversation between these two hosts is included in the attack trace, an attacker may infer that $a.b.c.d$ is a host that quite possibly plays some part in spreading malware or is part of a botnet, and so on. It's obvious that the IP address $a.b.c.d$ also needs to be anonymized. One needs to be aware of the semantics of the attack trace to apply anonymization policies efficiently.

Faster alternatives other than *libpcre*, which were already mentioned above, exist. We choose not to incorporate them into the prototype implementation for the following reason.

Name	Type	# of traces matched	Comment
Bielefeld	connectback shellcode	9552	None
Metasploit PexEnvSub	xor decoder	2608	None
rbot 256 byte	xor decoder	2575	None
adenau	xor decoder	1133	None
halle	xor decoder	987	None
schoenberg	xor decoder	129	None
langenfeld	xor decoder	21	None
Leimbach	xor decoder	16	contains TFTP download
kaltenborn	xor decoder	15	None
Wuerzburg	connectback file transfer	1	None

Table 1: List of regular expressions incorporated into `anontool` for encrypted shellcode anonymization.

Packet trace anonymization is at the moment an off-line process. Indeed, when it comes to anonymizing traffic as it appears on a network interface, speed is critical. However, when it comes to binary payloads, there are two processes which precede anonymization: the first is detection of the executable content, malicious or not, and the second is the analysis needed to reverse-engineer, classify, and produce a signature for it. Although all of them could be considered time-critical components of computer security, analysis and classification is a lengthy process, and is usually done manually and takes even few days for newly observed payloads (not variants of existing ones).

5. EVALUATION

In this Section, we present the evaluation of our implementation of binary payload anonymization. We tested our implementation against a set of packet traces containing malicious payloads obtained using the Network-level Emulation attack detection method formally described in [12,20], which identifies the presence of self-modifying polymorphic shellcode in network streams. The alerts generated contain full payload traces in libpcap format. Each trace corresponds to a single attack attempt and contains all packets of the network flow (quintuple) of the particular attack instance, including the initial TCP 3-way handshake.

In total, the number of attack traces generated by NEMU was 21726, spanning a time period from January 11, 2007 to April 6, 2008. `Anontool` detected and anonymized sensitive data within 17036 of those traces, a 78.4% of the total number of alert-generating traces.

For verification purposes, we manually checked and inspected some of the attack traces, to determine two things: first, whether we anonymized the bytes at the correct offset(s), and second to determine whether exploits that were not anonymized did not actually contain any sensitive data or we just happened to lack a regular expression for the corresponding decoder.

For the first case, we chose a random trace for which `anontool` reported that an IP contained in it was identified, and tried to determine whether this IP was anonymized correctly. Figure 5 shows the anonymized output for the example explained in Section 2, taken from the anonymized trace produced by `Anontool`. Our tool managed to find and anonymize the sensitive information contained in the encrypted payload. The outlined bytes show the offset at which the IP address was identified and subsequently masked. We therefore were able to verify the anonymization process was correct.

We then proceeded to examine in detail a few attack traces which `anontool` did not anonymize. Most of these traces contained a remote root exploit for the Knox Arkiea Server [6]. This exploit does not connect back to any host, and thus its payload does not contain any sensitive information. We do not include any regular expressions for this kind of shellcode in our tool, since by definition it lacks any sensitive information that could expose an innocent host to attackers, and thus can be shared without any risk.

The user needs to be aware that the regular expressions contained in the tool are not a panacea, and other types of shellcode possibly exist, that are not currently handled by our tool. On the other hand, given the fact that we designed and implemented our tool with modularity and extensibility in mind, adding support for a new kind of shellcode or binary payload is easy and intuitive.

We expect that, in order for our tool to be widely deployed and used, this is a necessary quality, especially since we cannot predict future advances in the area of polymorphic and metamorphic shellcode construction. Experts in that field, however, should be given the convenience of easily implementing a code module for our tool to anonymize their traces of choice.

6. CONCLUSION

We have presented a novel method for the anonymization of sensitive information contained in the shellcode found in network packet traces of code injection attacks. To the best of our knowledge, this is a novel way of performing anonymization of sensitive data that may appear in the payload of a packet, which are not exposed on the wire. This is therefore a first step in making payloads available when distributing and sharing network logs, and enable sharing of traces containing malicious activity.

There are significant benefits for all concerned parties, such as security analysts and researchers, whether it involves sharing traces inside a group of few trusted security-related organizations, or in the form of a centralized repository of attack code, to aid in analysis and defense against malicious internet activity. To this end, we have already started making attack traces available through [7], and we will continue to do so in the future.

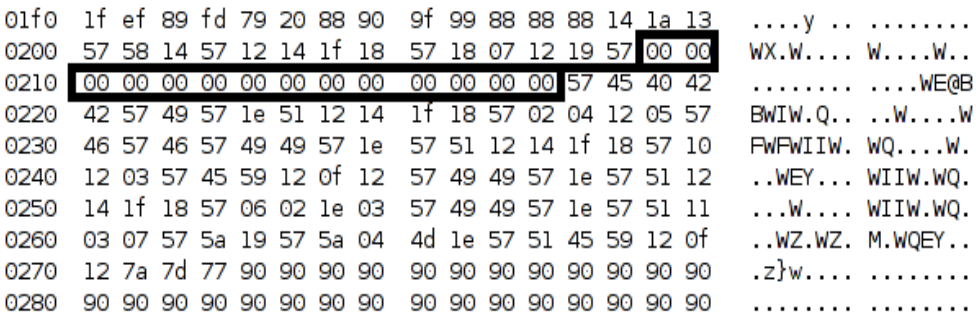

Figure 5: The anonymized part of the Wuerzburg shellcode as produced by `anontool` as its final output. The IP address is marked for clarity.

7. AVAILABILITY

Anontool can be downloaded from `http://dcs.ics.forth.gr/Activities/Projects/anontool.html`. The application has been installed and tested on Redhat, Ubuntu and Debian Linux distributions. All the signatures referenced inside this paper are also available with the source code.

Acknowledgments

This work was supported in part by the IST project LOBSTER funded by the European Union under contract number 004336. Michaels Foukarakis, Demetres Antoniades, and Michalis Polychronakis are also with the University of Crete.

8. REFERENCES

[1] Amun honeypot: breakdown of the wuerzburg shellcode. "`http://zero.ram.rwth-aachen.de/amun/shellcode.php?id=1`".

[2] Amun: Python honeypot. "`http://zero.ram.rwth-aachen.de/amun/shellcodes.php`".

[3] European network of affined honeypots. "`http://www.fp6-noah.org/`".

[4] The honeynet project. "`http://www.honeynet.org/`".

[5] K2, admmutate. "`http://www.ktwo.ca/c/ADMmutate-0.8.4.tar.gz.`".

[6] Knox arkiea server backup local/root remote exploit. "`http://archives.neohapsis.com/archives/fulldisclosure/2005-02/att-0397/arksink2.c`".

[7] Network traces of attacks captured at various lobster passive monitoring sensors. "`http://lobster.ics.forth.gr/traces/`".

[8] Official nepenthes website. `http://nepenthes.mwcollect.org/contact`".

[9] Perl compatible regular expressions library. "`http://www.pcre.org/`".

[10] Sans top 20 security risks for 2007. "`http://www.sans.org/top20/`".

[11] Sourcefire vrt certified rules. "`http://www.snort.org/pub-bin/downloads.cgi`".

[12] M. P. et al. Emulation-based detection of non-self-contained polymorphic shellcode. In *Proceedings of RAID'07*, 2007.

[13] W. Y. et al. Scrub-tcpdump: A multi-level packet anonymizer demonstrating privacy/analysis tradeoffs. In *Proceedings of SECOVAL '07*, 2007.

[14] Greg Minshall. Tcpdpriv. "`http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`".

[15] D. Koukis, S. Antonatos, D. Antoniades, P. Trimintzios, and E. Markatos. A generic anonymization framework for network traffic. In *Proceedings of the IEEE International Conference on Communications (ICC 2006)*, June 2006.

[16] C. Kreibich. Netdude: Network dump data displayer and editor. "`http://netdude.sourceforge.net`".

[17] J. Mogul. Trace anonymization misses the point. presentation on www 2002 panel on web measurements.

[18] J. C. Mogul and M. Arlitt. Sc2d: an alternative to trace anonymization. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 323–328, 2006.

[19] R. Pang and V. Paxson. A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In *Proceedings of the ACM SIGCOMM Conference*, August 2003.

[20] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. Network-level polymorphic shellcode detection using emulation. In *Proceedings of DIMVA'06*, 2006.

[21] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.

[22] P.Szor. Hunting for metamorphic. In Proceedings of the Virus Bulletin Conference, 2001.

[23] M. Roesch. Snort: Lightweight intrusion detection for networks. November 1999. (available from *http://www.snort.org/*).

[24] Y. T.Detristan, T.Ulenspiegel and M.V.Underduk. Polymorphic shellcode engine using spectrum analysis. Phrack magazine, "`http://www.phrack.org/issues.html?issue=61\&id=9#article`".

[25] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz. Fast and memory-efficient regular expression matching for deep packet inspection. In *ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, pages 93–102, New York, NY, USA, 2006. ACM.