European Commission
Directorate-General Home Affairs
Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme

HOME/2009/CIPS/AG/C2-050

i-Code: Real-time Malicious Code Identification

D0: Requirements Analysis

Workpackage:	WP0: Requirements Analysis
Contractual delivery date:	October 2010
Actual delivery date:	October 2010
Leading partner:	FORTH
Contributing partners:	Politecnico di Milano, Institute Eurécom, Technical University of Vienna, Vrije Universiteit
Editor:	Sotiris Ioannidis
Contributors:	Stefano Zanero, Davide Balzarotti, Paolo Milani, Herbert Bos
Internal Reviewers:	Andreas Moser, Andrea Lanzi, Kallia Marakomihelaki

Executive Summary:

In this deliverable we present our findings about the state of the art in high-speed pattern recognition, signature generation from attack vectors and malware analysis. All these are components necessary to perform real-time malicious code identification. Furthermore, we have interviewed experts in the field, from a variety of organizations, including ISPs, security companies, search providers, research institutions, etc. to provide guidelines about the context and direction that the i-Code project should follow.

With the support of the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme.
European Commission - Directorate-General Home Affairs

This project has been funded with the support of the Prevention, Preparedness and Consequence Management of Terrorism and other Security-related Risks Programme of the European Commission - Directorate-General Home Affairs. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

1 Introduction

As creation of malicious software continues to grow at alarming rates, we as security researchers are forced to develop new ways to counter this problem. Since we are now dealing with a threat that is growing in both scale and complexity, as well as in speed of infection new techniques are required. Real-time malicious code identification is one of the methods that we believe will play a significant role in countering this threat.

In this document we provide an overview of the state of the art real-time malicious code identification area. Specifically we discuss background work done in the area of high-speed pattern recognition, signature generation and malware analysis. These three areas form the backbone of real-time malware detection. We used this bibliographical survey as a guideline for the direction we should take towards developing a real-time malicious code identification framework.

To further guide our compilation of the set of requirements necessary, we conducted interviews with experts in the field of network security. We selected those experts from a large cross-section of industry and also academia. Specifically from, ISPs, NRENs, security companies, search providers, CERTs, research institutions, etc.

We conclude this document by laying out the axes along which we plan to carry out the work necessary to build a real-time malicious code identification framework.

2 Background

The goal of Internet attacks carried out by worms, viruses and other malware, is to subvert the protection of the target system and establish complete control. The way these attacks are usually carried out are by targeting a specific vulnerable application on the victim that enables the attacker to transfer the execution of the application to attacker-crafted code. This can be accomplished in a variety of ways, including, buffer and integer overflows, format string attacks, and others. These attacks aim to hijack the instruction pointer of the vulnerable program and then execute arbitrary code. Often the code executed is a command shell, and that is why the attack vector, that is the data sent to carry out the attack, is called a shellcode. Once a command shell, or other attacker-controlled code, starts executing, it does so with the permissions and privileges of the compromised application. Identifying malware at high network speed, that is, detecting the malware presence, fingerprinting it, classifying it and producing a notification is of paramount importance. We now look into the state-of-the-art in malware identification.

2.1 Signature Generation

To perform any kind of high-speed malware identification it is necessary to have some sort of signatures in order to identify the malware. Detecting zero-day polymorphic malware can be extremely difficult since no such signatures exist. The first attempts to detect such malware focused on detecting a sled component in the code of the attack vector [31, 32]. Unfortunately such sleds are not always present, as one can craft the attack to not require a sled, and most will know Internet worms did not employ one. Systems like Buttercup [33] try to identify polymorphic buffer overflows by detecting the ranges of possible return addresses for existing buffer overflow vulnerabilities. This approach is not particularly effective against more sophisticated attacks.

Other works try to automatically generate signature for unknown malware by looking for common byte sequences across different malware instances. These methods are capable of producing some sort of zero-day malware detection signatures by correlating data payloads from suspicious traffic flows [34, 35, 36]. Such approaches have a tendency to create a lot of false positives and cannot counter attack by using polymorphic malware, as they do not contain sufficiently long common byte sequences [37].

Systems such as [38, 39, 40, 41] create signatures that can match polymorphic malware. Detecting common invariants that exist in different malware, like return addresses, protocol framing, etc. does this. The generated signatures can be expressed as regular expressions or statistical distributions, and be used for high-speed detection by systems as discussed in the previous subsection. However, more aggressively polymorphic malware render such systems largely ineffective. The problem is that for such systems to work they require a lot of malware instances before classifying something as an attack. This means that targeted attacks may pass unnoticed [47].

2.2 High-speed Pattern Recognition

Pattern matching is the most critical operation that affects the performance of network intrusion detection systems. These systems can be used to identify malware as they propagate through the network. Pattern matching algorithms can be classified into single- and multi-pattern algorithms.

In single pattern matching algorithms, each pattern is searched for in a given text individually. This means that if we have k patterns to be searched, the algorithm must be repeated k times. Knuth-Morris-Pratt [18] and Boyer-Moore [6] are some of the most widely used single pattern matching algorithms. Knuth- Morris-Pratt is able to skip characters when a mismatch occurs in the comparison phase using a partial-match table for each pattern. Each table is built by preprocessing every pattern separately. Boyer-Moore is the most widely used single-pattern algorithm. Its execution time can be sub-linear if the suffix of the string to be searched for appears infrequently in the input stream, due to the skipping heuristics that it uses.

Multi-pattern string matching algorithms, search for a set of patterns in a body of text simultaneously. This is achieved by preprocessing the set of patterns and building an automaton that will be used in the matching phase to scan the text. The automaton can be thought of as a state machine that is represented as a trie, a table or a combination of the two. Each character of the text will be searched only once. Multi-pattern matching scales much better than algorithms that search for each pattern individually. Multi-pattern string matching algorithms include Aho-Corasick [1], Wu-Manber [2] and Commentz-Walter [3].

Most Network Intrusion Detection Systems (NIDS) use finite automata and regular expressions [4,5,6] to match patterns. Coit et al. [7] improved the performance of Snort by combining the Aho-Corasick keyword trie with the skipping feature of the Boyer-Moore algorithm. Fisk and Vaghese enhance the Boyer-Moore-Horspool algorithm to simultaneously match a set of rules. The new algorithm, called Set-wise Boyer-Moore-Horspool [8], was shown to be faster than both Aho-Corasick and Boyer-Moore for sets with less than 100 patterns. Tuck et al. [9] optimized the Aho-Corasick algorithm by applying bitmap node and path compression.

Snort from version 2.6 and onwards uses only flavors of the Aho-Corasick for exact-match pattern detection. Specifically, it contains a variety of implementations that are differentiated by the type of the finite automaton they use (NFA or DFA), and the storage format they use to keep it in memory (full, sparse, banded, trie, etc.). It should be mentioned, however, that the best performance is achieved with the full version that uses a deterministic finite automaton (DFA) at the cost of high memory utilization [10].

To speed-up the inspection process, many IDS implementations are based on specialized hardware. By using content addressable memory (CAM), which is suitable to perform parallel comparison for its contents against the input value, they are very well suited for use in intrusion detection systems [11, 12]. However, they have a high cost per bit.

Many reconfigurable architectures have been implemented for intrusion detection. Most approaches involve building an automaton for a string to be searched, generating a specialized hardware circuit using gates and flip-flops for the automaton, and then instantiating multiple such automata in the reconfigurable chip to search the streaming data in parallel. However, the circuit implemented on the FPGA to perform the string matching is designed based on the underlying hardware architecture to adjust to a given specific rule set. To adjust to a new rule set, one must program a new circuit (usually in a hardware description language), which is then

compiled down through the use of CAD tools. Any changes in the rule set require the recompilation, regeneration of the automaton, re-synthesis, replacement and re-routing of the circuits, which is a time consuming and difficult procedure.

Sidhu and Prasanna implemented a regular expression matching architecture for FPGAs [13]. Baker et al. also investigated efficient pattern matching as a signature based method [14]. In [15], the authors used hardware bloom filters to match multiple patterns against network packets at constant time. Attig et al. proposed a framework for packet header processing in combination with payload content scanning on FPGAs [16].

Several approaches attempt to reduce the amount of memory needed to economically fit in on-chip memory [17,18,19]. However, the on-chip hardware resource consumption grows linearly with the number of characters to be searched. In [20], the authors convert a string set into many tiny state machines, each of which searches for a portion of the strings and a portion of the bits of each string.

Other approaches involve the cooperation with network processors in order to pipeline the processing stages assigned to each hardware resource [21], as well as the entire implementation of an IDS on a network processor [22, 23]. Computer clusters have also been proposed to offload the workload of a single computer [24, 25, 26, 27]. The cost however remains high, since it requires multiple processors, a distribution network, and a clustered management system.

On the contrary, modern GPUs have low design cost while their increased programmability makes them more flexible than ASICs. Most graphic cards manufacturers provide a high-level APIs that offer high programming capabilities and further ensure forward compatibility for future releases, in contrast with most FPGA implementations that are based on the underlying hardware architecture and need to be reconfigured whenever a change occurs in the rule set. Furthermore, their low design cost, the highly parallel computation and the potential that they are usually underutilized, especially in hosts used for intrusion detection purposes, makes them suitable for use as an extra low-cost coprocessor for time-consuming problems, like pattern matching.

PixelSnort [28] is a port of the Snort IDS that offloads packet matching to an NVIDIA 6800GT. The GPU programming was complicated, since the 6800GT did not support a general-purpose programming model for GPUs (as the G80 used in our work). The system encodes Snort rules and packets to textures and performs the string searching using the KMP algorithm on the 16 fragment shaders in parallel. However, PixelSnort does not achieve any speed-up under normal-load conditions. Furthermore, PixelSnort does not have any multi-pattern matching algorithms ported to GPU. This is a serious limitation since multi-pattern matching algorithms are the default for Snort. In a more recent work, Marziale et al. [29] evaluated the effectiveness of offloading the processing of a file-carving tool to the GPU. The system was implemented on the G80 architecture and the results showed that GPU support can substantially increase the performance of digital forensics software that relies on binary string search. Gnort [30] is a high-performance GPU based implementation of the Snort IDS. Performance results showed that Gnort can scale to the gigabits per second when performing intrusion detection.

2.3 Malware Analysis

When a malware is discovered it must be analyzed to determine the kind of threat it poses to the network. Since malware generation by the attackers if often automatic,

such samples grow at very high rates and spread on the Internet in high numbers [38, 45, 46]. To counter this analysis must be automatic. Traditional approaches, statically analyze the code of the malware. More advanced approaches take a different route. Behavioral analysis executes malware samples for preset time duration inside an emulated environment. During that time, the malware is being monitored closely. The observation assists in automatically generating a report on the behavior of the malware, which can be used in further analysis. The difference with the traditional methods is that it is a black-box process in which one does not look inside the malware, but instead it looks at the interactions it has with the environment. This simplifies things greatly, since as we have mentioned before most of the malware today is polymorphic and metamorphic. Simply executing it, deals with the process of decryption, obfuscation and packing, as these steps happen on their own. On the down side, each execution can only monitor one execution path of the malware, so reports may be incomplete. To address this limitation Moser et al. [42] proposed a method for multiple execution path analysis.

3 Interviews with the Experts

In order to have a more complete and clear view of the issues involved in real-time malicious code identification, we conducted a series of interviews with experts in the field of network security. To get a well-rounded view, we selected those experts from a large cross-section of industry and also academia. Specifically from, ISPs, NRENs, security companies, search providers, CERTs, research institutions, etc.

To conduct these interviews we compiled, but were not limited to, a set of discussion points, which we used as guidelines. Here we summarize the core list of discussion points:

1. Do you do protection on the net or on the host?
2. What sort of tools or protection do you use? Specifically, do you use any of the following: IDS, IPS, honeypots, firewalls, botnet detectors, extrusion detection tools, DDoS protection/detection tools, and/or other forensics tools?)
3. What sort of resources do you protect and with what tools (see above)?
4. What sort of links do you protect and with what tools? (Link type, capacity, etc.)
5. Who are you trying to protect and from whom?
6. Do you look at the entire traffic? Do you do sampling? Do you look at specific flows? Do you do header-only?
7. What sort of network speeds would real-time detection make sense today?
8. What about in the next 3 years?
9. Do you do deep packet inspection?
10. What percentage of traffic is encrypted?
11. What types of things should we be looking for? Shellcode, botnet communications or any other malware? Any suggestions?
12. What sort of information would be useful to present to the "user" of such a system? Statistics? Attacks as they happen? Something else?

As one can see from the above discussion points, what we tried to accomplish is first determine the importance of the network versus the end host. Then we drilled down on specific types of tools and resources that need protection. The next set of points try to determine the rates that any real-time malware identification system must cope with. Finally, we focus on types of attacks and delivery methods as well as what must be presented to the “end user” of such a system.

3.1 Network and Host

Almost all interviewees reported that protection is taking place at both the network and the host. There were however a couple of exceptions. Those were from the experts that worked for network providers. This was expected, as the business model of network providers is to provide connectivity. One comment was "*... as an ISP doesn't have any firewall functionality that is offered as a service. We do have an anti-DDoS service installed at the core of our network which can mitigate some attacks.*"

3.2 Tools and Resources

With respect to the tools used, the experts reported that they use a variety of solutions. These solutions cover the entirety of security tool-types, both commercial and public, open source. It is clear that the development of a real-time malicious code identification framework would fit well with the tools already used out there in practice. Security tools such as IDS, IPS and antivirus being the most prominent examples.

One expert noted: "*However, we also exploit these tools to collect information about compromised hosts to support the incident handling service of our CERT.*" The work proposed here would assist them, and others like them, in real-time reporting and handling of incidents.

All the experts noted that they use those security tools to protect their vulnerable resources on the network and on the end hosts.

3.3 Rate Requirements

In our interviews, we wanted to get a feel of what data rates were typical used in the experts' organizations. The respondents placed traffic in the order of a few to ten gigabits per second. The responses were split, however when asked about in how much detail they examined the traffic. The more security conscious responded that they perform deep packet inspection when possible, but others, just relied at examining headers and collecting statistics ("*We do header-only, sampled netflow. Main reason for this is the total amount of traffic*"). They did agree however that: "*Deep packet inspection is the preferred method*".

There was no clear consensus on the amount of traffic that is encrypted; we did get an estimate of 10%. This leaves plenty of opportunity to perform deep packet inspection to try to detect malware inside the network. Even if all traffic were encrypted, we could still perform it on the end hosts, when packets get decrypted.

Experts estimate that in a few years time we will be talking about network speeds in the ten to one hundred gigabits per second.

3.4 Malware Types, Delivery and Presentation

With respect to attacks we should be looking for, there were no clear-cut responses, instead there is a variety of opinions. Botnet traffic was brought up along with the associated malware. Malicious code delivered inside office and PDF documents.

Browser attacks are believed to continue. Also attacks on different kinds of devices were brought up, for example, mobile devices being a likely target in the future.

As far as reporting goes, there was feeling that this is a very important matter. An expert noted: "*Our experience has shown, that detailed information about the attack and especially how to recover post-attack is very important. Otherwise, the user will likely miss some backdoors or other malware and the attack will go on.*"

3.5 Additional Remarks

Some of the experts also expressed interest in working with us and recommended conference calls to further discuss new ideas. They also expressed interests in the end product on this work: "*... sure we would be willing to test your solutions*". This is something we plan to pursue.

4 Requirements

Given the prior work done in the field, along with the input we have come across from a diverse set of experts in the field of network security and malware, we will focus our work (i) on detecting malicious code on the network, and (ii) on classifying this malware and providing information concerning it. We believe that the project addresses a very important issue for the following reasons:

First, the proposed system will be able to identify malicious code before this reaches the victim computer. Second, by deploying our systems in appropriate network points one will be able to see both incoming and outgoing attacks and identify not only external attackers, but also "internal" computers which have been compromised.

Designing, implementing and deploying the proposed system presents several challenges. We classify them in the subsections that follow and go into detail about the specific requirements.

4.1 Speed

As network bandwidth doubles every few years, it is getting increasingly difficult to perform sophisticated malware detection and identification at line speeds. Currently it is possible to perform such tasks at 100Mbits to 1Gbit per second relatively easy. We can also do it at speeds up to 10Gbit per second with some extra effort and additional or specialized resources.

From the input we had from the interviews with the experts, it is safe to assume that this will not suffice in a few years time. We are moving to networks of speeds that will reach 40 to 100Gbits per second. Any techniques, protocols and tools we develop must cope with the aforementioned traffic rates.

4.2 Coverage

Since up to a few years ago, attacks had been large-scale high, impact, and to some extent, short lived. This trend has been reversed in the last few years. Cyber-attackers continuously develop even stealthier methods making it increasingly difficult to detect them. Taking this further, these attacks may take the form of targeted attacks. That is, attacks that may be customized to strike a certain target in a particular and unique way.

Developing countermeasures for this type of malicious activity can prove very hard to counter. However, we should strive to offer as much coverage as possible in any mechanisms we develop. Good coverage may translate in capturing a large percentage of attacks, even if some, very stealthy and limited scope attacks get through.

4.3 False Positives

Malware detection systems often suffer from false positives: detection of behaviour that seems anomalous, but is not malicious. One may think that this is not a significant problem as it is better to be "safe than sorry". On second thought, however it becomes clear that things are not that simple. False positives create extra workload for the defender, that is the more time the defender spends on examining ultimately benign traffic, the less time they have to deal with actual threats.

This has been identified as one of the main problems when dealing with attacks. Our malware analysis system must be able to quickly and accurately decide what is malicious and what is not. This will offload the detection engine making it more capable when dealing with the high rates of traffic we expect in the next few years.

4.4 Device Heterogeneity

End user devices come in all shapes and forms. Detection systems may need to take this heterogeneity into account and provide solutions for the different types of devices. The variety of devices, often translate to different operating systems, libraries and applications. Any malware analysis and detection system that will be developed has to take this into account.

In the context of the current project we do not expect to implement systems that provide protection for all possible combinations of devices and software stacks??(do you mean attacks?. That would be infeasible given the limited time provided for this project. What we do plan however is to implement solutions for the more popular hardware devices and software stacks. We also plan to create solutions that can be applied to a broad set of hardware devices and software stacks.

4.5 Network Heterogeneity

We are evolving from DSL and leased lines, to fibre to the customer, while on the wireless front we are witnessing an explosion of different technologies for cellular, broadband and personal networks, and on the logical side, intranets, extranet and overlays. All these types of networks create new opportunities for the attackers. For example, now the attackers have a lot of capacity in terms of the rate of attack, should they try to be aggressive. They also have multiple avenues in terms of reaching the victim, by selecting which network to use in order to carry out an attack.

The malware identification system we develop must take these new topological realities in mind. The placement of detectors must be such that they can counter attacks following variable attack paths.

4.6 Protocol Diversity

The richness of the application space in today's Internet generates data traffic at rates and richness that seemed unimaginable a decade ago. This data traffic on modern networks is transferred over a plethora of network protocols. Protocols are being developed to address specific application needs such as, HTTP, SOAP, a plethora of P2P protocols, FTP, VoIP etc.: Any malware detection and forensics system must be

able to identify attacks propagating on any of the protocols being used today. This is necessary as attackers launch their attack vectors against any, and every, potential target application.

This requirement forces us to closely look at data packets instead of simply relying on a quick inspection. The algorithms and systems we develop must be capable of performing deep packet inspection on traffic flows when trying to identify covert attacks.

5 Preliminary Design

We will spend the next few paragraphs going over a preliminary design for our real-time malicious code identification framework. Our design is broken into multiple steps and will move along the requirements outlined in the previous section.

5.1 Network-level Emulation and Communication Pattern Detectors

To accommodate our requirements of coverage and diversity, we plan to start by implementing malware detection method that operates at the network level. This eliminates a lot of the problems of stand-alone, end-host detectors. We will use NEMU as a starting point [43]. NEMU uses a heuristic detection method that scans network traffic streams for the presence of previously unknown polymorphic shellcode. This approach relies on an embedded CPU emulator that executes every potential instruction sequence in the inspected traffic, aiming to identify the execution behavior of polymorphic shellcode. We plant to extend NEMU with capabilities to detect metamorphic malware as well. We also want to adjust this work to detect malware living inside PDF, and similar files.

Malware is often used to take over hosts to create botnets, and those same botnets are then used to spread more malware (among other things). Communication patterns between bots are often encrypted making them harder to detect. We plan to develop methods for detecting the command and control infrastructures of botnets using entropy in the communication patterns. We believe this method will work in the presence of both diverse and encrypted protocols.

5.2 Malware Extraction and Analysis

We are currently working on developing better shellcode-extraction methods. While experimenting with these shellcodes, we plan to improve our classification and clustering methods. Our new approach will focus less on flow-control graphs, as previous work has done, and more on novel methods. What we are interested in is to understand what a shellcode downloads, after it is activated, and from where.

Our ultimate goal is to improve end-host-level detection of malware. This can be accomplished by (i) evaluating different behavior-based techniques against a large dataset of malicious/benign behaviors, (ii) extracting high-quality behavioural signatures, and (iii) designing and building the infrastructure to match them.

5.3 High-speed Detection

Our final step is to work on every component operating in real-time. To do this we plan to build a high-speed infrastructure. To achieve this we initially intend to take a two pronged approach. The first one is to parallelize NEMU. The second method is to use graphics processors to accelerate the basic operations needed. That is string matching for fast, signature detection [30].

5.4 Integration

All of the above components will be brought together at the integration phase. For this we will build a forensics console. The forensics console will collect information from the various components and present them in a comprehensive way to the user of the real-time malware identification system.

6 Bibliography

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. Communications of the ACM, 18(6):333–340, June 1975.
[2] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical Report TR-94-17, 1994.
[3] B. Commentz-Walter. A string-matching algorithm fast on the average. In Proceedings of the 6th International Colloquium on Automata, Languages and Programming, pages 118–131.
[4] M. Roesch. Snort: Lightweight intrusion detection for networks. In Proceedings of the 1999 USENIX LISA Systems Administration Conference, November 1999.
[5] V. Paxson. Bro: A system for detecting network intruders in real-time. In Proceedings of the 7th conference on USENIX Security Symposium (SSYM '98), pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.
[6] C. IOS. IPS deployment guide. http://www.cisco.com.
[7] C. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of Snort. In Proceedings of DARPA Information Survivability Conference & Exposition II (DISCEX '01), June 2001.
[8] M. Fisk and G. Varghese. Applying fast string matching to intrusion detection. Technical Report In preparation, successor to UCSD TR CS2001-0670, University of California, San Diego, 2002.
[9] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In Proceedings of the IEEE Infocom Conference, pages 333–340, 2004.
[10] TheSnortProject.Snortusersmanual2.8.0.http://www.snort.org/docs/snort-manual/2.8.0/snort manual.pdf.
[11] F. Yu, R. H. Katz, and T. V. Lakshman. Gigabit Rate Packet Pattern-Matching Using TCAM. In Proceedings of the 12th IEEE International Conference on Network Protocols (ICNP '04), pages 174–183, Washington, DC, USA, October 2004. IEEE Computer Society.
[12] S. Yusuf and W. Luk. Bitwise optimised CAM for network intrusion detection systems. In Proceedings of International Conference on Field Programmable Logic and Applications, pages 444–449, 2005.
[13] R. Sidhu and V. Prasanna. Fast regular expression matching using FPGAs. In IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM01), 2001.
[14] Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on FPGAs. In Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04), pages 223–232, New York, NY, USA, 2004. ACM.
[15] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep packet inspection using parallel bloom filters. IEEE Micro, 24(1):52–61, 2004.
[16] M. Attig and J. Lockwood. A framework for rule processing in reconfigurable network systems. In Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '05), pages 225–234, Washington, DC, USA, 2005. IEEE Computer Society.
[17] Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on

FPGAs. In Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04), pages 223–232, New York, NY, USA, 2004. ACM.

[18] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In Proceedings of the IEEEInfocom Conference, pages 333–340, 2004.

[19] S. Dharmapurikar and J. Lockwood. Fast and scalable pattern matching for content filtering. In Proceedings of the 2005 ACM symposium on Architecture for networking and communications systems (ANCS '05), pages 183–192, New York, NY, USA, 2005. ACM.

[20] L. Tan, B. Brotherton, and T. Sherwood. Bit-split string-matching engines for intrusion detection and prevention. ACM Transactions on Architecture and Code Optimization, 3(1):3–34, 2006.

[21] C. Clark, W. Lee, D. Schimmel, D. Contis, M. Kone, and A. Thomas. A hardware platform for network intrusion detection and prevention. In Proceedings of the 3rd Workshop on Network Processors and Applications (NP3), 2004.

[22] H. Bos and K. Huang. Towards software-based signature detection for intrusion prevention on the network card. In Proceedings of 8th International Symposium on Recent Advances in Intrusion Detection (RAID), Seattle, WA, September 2005.

[23] NVIDIA. NVIDIA CUDA Compute Unified Device Architecture Programming Guide, version 1.1. http://developer.download.nvidia.com/compute/cuda/1 1/ NVIDIA CUDA Programming Guide 1.1.pdf.

[24] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In Proceedings of the IEEE Symposium on Security and Privacy, pages 285– 294, May 2002.

[25] K. Watanabe, N. Tsuruoka, and R. Himeno. Performance of network intrusion detection cluster system. In Proceedings of The 5th International Symposium on High Performance Computing (ISHPC-V), 2003.

[26] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID), pages 107–126, 2007.

[27] L.Schaelicke,K.Wheeler,and C.Freeland.SPANIDS: a scalable network intrusion detection load balancer. In CF '05: Proceedings of the 2nd conference on Computing frontiers, pages 315–322, New York, NY, USA, 2005. ACM.

[28] N. Jacob and C. Brodley. Offloading IDS computation to the GPU. In Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference (ACSAC '06), pages 371–380, Washington, DC, USA, 2006. IEEE Computer Society.

[29] G. G. R. I. Lodovico Marziale and V. Roussev. Massive threading: Using GPUs to increase the performance of digital forensics tools. Digital Investigation, 1:73–81, September 2007.

[30] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In Proceedings of 11th International Symposium on Recent Advances in Intrusion Detection (RAID), pages 116–134, 2008.

[31] Aycock, J., deGraaf, R., Jacobson, M.: Anti-disassembly using cryptographic hash functions. Department of Computer Science, University of Calgary, Technical Report, pp. 793–824, (2005).

[32] Venable, M., Chouchane, M.R., Karim, M.E., Lakhotia, A.: Analyzing memory accesses in obfuscated x86 executables. In: Proceedings of the conference on

detection of intru- sions and malware and vulnerability assessment (DIMVA), (2005).
[33] Pasupulati, A., Coit, J., Levitt, K., Wu, S., Li, S., Kuo, J., Fan, K.: Buttercup: on network-based detection of polymorphic buffer overflow vulnerabilities. In: Proceedings of the Network Operations and Management Symposium (NOMS), pp. 235–248, (2004).
[34] Kim,H.-A.,Karp,B.:Autograph: towards automated, distributed worm signature detection. In: Proceedings of the 13th USENIX Security Symposium, pp. 271–286, (2004)
[35] Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI), (2004).
[36] Kreibich, C., Crowcroft, J.: Honeycomb–creating intrusion detection signatures using honeypots. In: Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II), (2003).
[37] Kolesnikov, O., Dagon, D., Lee, W.: Advanced polymorphic worms: evading IDS by blending in with normal traffic. In: College of Computing, Georgia Institute of Technology, Atlanta, GA 30332, http://www.cc.ga- tech.edu/ ok/w/ok_pw.pdf, (2004).
[38] Newsome, J., Karp, B., Song, D.: Polygraph: automatically Generating signatures for polymorphic worms. In: Proceedings of the IEEE Security & Privacy Symposium, pp. 226–241, (2005)
[39] Tang, Y., Chen, S.: Defending against Internet worms: a signature-based approach. In: Proceedings of the 24th Annual Joint Conference of IEEE Computer and Communication societies (INFOCOM), (2005)
[40] Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: Proceedings of the 7th International Symposium on Recent Advanced in Intrusion Detection (RAID), pp. 201–222, (2004)
[41] Li, Z., Sanghi, M., Chen, Y., Kao, M.Y., Chavez, B.: Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp. 32–47, 2006.
[42] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring Multiple Execution Paths for Malware Analysis. In Proceedings of 2007 IEEE Symposium on Security and Privacy, 2007.
[43] Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. Network-level Polymorphic Shellcode Detection using Emulation. In Journal in Computer Virology, vol. 2, no. 4, pp. 257-274, February 2007.
[44] Bos, Herbert, de Bruijn, Willem, Cristea, Mihai ,Nguyen, Trung and Portokalidis, Georgios, FFPF: fairly fast packet filters. In Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, pp. 24-24, San Francisco, CA, 2004.
[45] H.-A. Kim and B. Karp. Autograph: toward automated, dis- tributed worm signature detection. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
[46] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2004.
[47] R. Perdisci*, D. Dagon, W. Lee, P. Fogla, M. Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection. IEEE Symposium on Security and Privacy, 2006