SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
Trustworthy ICT

NETWORK OF EXCELLENCE

A European Network of Excellence in Managing Threats and Vulnerabilities in the Future Internet: *Europe for the World* [†]

Deliverable D5.5: Final Report on Malware and Fraud

Abstract: This deliverable presents the final report on what we have seen and worked on in regards to Malicious Software (Malware) and fraudulent activity on the Internet. It also presents the findings and outcome of discussions from four years of working group meetings. Together with the *SysSec* consortium and selected experts from both academia and the industry, it sheds a light on why so many computer systems are considered unsafe today.

Contractual Date of Delivery	September 2014
Actual Date of Delivery	November 2014
Deliverable Dissemination Level	Public
Editors	Christian Platzer, Martina Lindorfer
Contributors	All *SysSec* partners
Quality Assurance	POLIMI, IICT-BAS

The *SysSec* consortium consists of:

FORTH-ICS	Coordinator	Greece
Politecnico Di Milano	Principal Contractor	Italy
Vrije Universiteit Amsterdam	Principal Contractor	The Netherlands
Institut Eurécom	Principal Contractor	France
IICT-BAS	Principal Contractor	Bulgaria
Technical University of Vienna	Principal Contractor	Austria
Chalmers University	Principal Contractor	Sweden
TUBITAK-BILGEM	Principal Contractor	Turkey

[†] The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007.

Document Revisions & Quality Assurance

Internal Reviewers

1. Vladimir Dimitrov (IICT-BAS)
2. Michele Carminati (POLIMI)
3. Matthias Neugschwandtner (TUV)
4. Stefano Zanero (POLIMI)

Revisions

Ver.	Date	By	Overview
1.0.0	2014-11-12	*Editor*	Final review of deliverable consistency and elements.
0.3.1	2014-10-08	#3	Paper list included. Further proofread.
0.3.0	2014-09-23	#1	Quality check by IICT-BAS. Changes incorporated
0.2.0	2014-09-21	#4	Quality check by POLIMI. Changes incorporated
0.1.2	2014-09-20	#2	Remarks on opening and structure.
0.1.1	2014-08-20	*Editor*	Chapters harmonized and correlated to each other.
0.1.0	2014-08-13	*Editor*	Chapters added. Deliverable in first draft stage.
0.0.2	2014-06-06	*Editor*	Completed chapter structure and approximate content.
0.0.1	2014-04-04	*Editor*	Created document Stub.

Contents

List of Figures

1 Foreword

The document is the last deliverable for malware and fraud and therefore concludes the SysSec project for work package five.
In the past four years, malware and malware research did not receive a lot of dedicated attention. In the very first deliverable (D5.1), we gave a brief overview on what happened in recent years in the malware sector. The following documents were more directed towards social networks and fraud, which are undoubtedly an integral part of today's security landscape. There was the practical case study in D5.3, where actual malware contamination within the Turkish network was measured. This case study shed a light on the engineering effort necessary to even conduct such a study. Naturally, the used technologies for the case study were less sophisticated than recent efforts to find and analyze malware in the research community. The presented implementations usually require a good amount of engineering effort to finally result in a usable system. Still, some of these recent approaches will certainly reach maturity and finally see deployment in corporate environments. If that is the case depends on different factors like scalability, usability and general quality of an approach. This document was created to give an overview on advances and the underlying research in malware analysis, countermeasures and internet fraud. It covers the lifetime of the SysSec project and is organized in a semi-chronological manner to give an idea of how the topics generally evolved. As a conclusion of work package five, it refocuses on the core topic of malware again. Furthermore, we also address how malware acts as an enabler for further exploitation, fraud or data gathering. Undoubtedly, the topic covers a much broader area than can be discussed in the following pages. Therefore, we took care to select each chapter such that it paints a holistic picture of current research and approaches.

1.1 Malware and Fraud Related Works of the SysSec Consortium

In addition to the content in this deliverable, the SysSec Consortium published a number of papers in various Journals and Conference Proceedings. The following list iterates the most important research output of the SysSec Project in respect to Work Package 5.

- Farnaz Moradi, Tomas Olovsson, and Philippas Tsigas. **A Local Seed Selection Algorithm for Overlapping Community Detection.** In Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM14). August 2014, Beijing, China.
- Andrei Costin, Jonas Zaddach, Francillon Francillon, Aurlien, Davide Balzarotti. **A Large Scale Analysis of the Security of Embedded Firmwares.** In Proceedings of the 23rd USENIX Security Symposium (USENIX Security). August 2014, San Diego, CA, USA.
- Stefano Schiavoni, Federico Maggi, Lorenzo Cavallaro, and Stefano Zanero. **Phoenix: DGA-Based Botnet Tracking and Intelligence.** In Proceedings of the 11th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA). July 2014, Egham, UK.
- Martina Lindorfer, Stamatis Volanis, Alessandro Sisto, Matthias Neugschwandtner, Elias Athanasopoulos, Federico Maggi, Christian Platzer, Stefano Zanero, Sotiris Ioannidis. **AndRadar: Fast Discovery of Android Applications in Alternative Markets**. In Proceedings of the 11th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). July 2014. Egham, UK.
- Claudio Criscione, Fabio Bosatelli, Stefano Zanero, and Federico Maggi. **Zarathustra: Extracting WebInject Signatures from Banking Trojans.** In Proceedings of the 12th Annual International Conference on Privacy, Security and Trust (PST). July 2014, Toronto, Canada.
- Rafael A Rodrguez Gmez, Gabriel Maci Fernndez, Pedro Garca Teodoro, Moritz Steiner, and Davide Balzarotti. **Resource monitoring for the detection of parasite P2P botnets.** Computer Networks, Elsevier B.V., pages 302311, June 2014.
- Sebastian Neuner, Victor van der Veen, Martina Lindorfer, Markus Huber, Georg Merzdovnik, Martin Mulazzani, and Edgar Weippl. **Enter Sandbox: Android Sandbox Comparison.** In Proceedings of the IEEE Mobile Security Technologies Workshop (MoST). May 2014. San Jose, CA, USA.

- Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Michalis Polychronakis, Sotiris Ioannidis. **Rage Against the Virtual Machine: Hindering Dynamic Analysis of Mobile Malware.** In Proceedings of the 7th European Workshop on Systems Security (EuroSec). April 2014. Amsterdam, The Nederlands.
- Nick Nikiforakis, Federico Maggi, Gianluca Stringhini, M Zubair Rafique, Wouter Joosen, Christopher Kruegel, Frank Piessens, Giovanni Vigna, Stefano Zanero. **Stranger Danger: Exploring the Ecosystem of Ad-based URL Shortening Services.** In Proceedings of the 2014 International World Wide Web Conference (WWW). April 2014. Seoul, Korea.
- Zlatogor Minchev, Suzan Feimova. **Modern Social Networks Emerging Cyber Threats Identification: A Practical Methodological Framework with Examples.** In Proceedings of the 6th AFCEA Sixth Young Scientists Conference 'Future of ICT', at NATO C4ISR Industry Conference & TechNet International. March, 2014. Bucharest, Romania.
- Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. **Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares.** In Proceedings of the Network and Distributed System Security Symposium (NDSS). February 2014, San Diego, USA.
- Jonas Zaddach, Anil Kurmus, Davide Balzarotti, Erik Olivier Blass, Aurelien Francillon, Travis Goodspeed, Moitrayee Gupta, Ioannis Koltsidas. **Implementation and Implications of a Stealth Hard-Drive Backdoor.** In Proceedings of the 2013 Annual Computer Security Applications Conference (ACSAC). December 2013, New Orleans, LA, USA.
- Martina Lindorfer, Bernhard Miller, Matthias Neugschwandtner, Christian Platzer. **Take a Bite - Finding the Worm in the Apple.** In Proceedings of the 9th International Conference on Information, Communications and Signal Processing (ICICS). December 2013,Tainan, Taiwan.
- Martina Lindorfer, Matthias Neumayr, Juan Caballero, Christian Platzer. **POSTER: Cross-Platform Malware: Write Once, Infect Everywhere.** In Proceedings of the20th ACM Conference on Computer and Communications Security (CCS). November 2013, Berlin, Germany.
- Federico Maggi, Andrea Valdi, Stefano Zanero. **AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors.** In Proceedings of the 3rd Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM). November 2013, Berlin, Germany.

- Dennis Andriesse, Christian Rossow, Brett Stone-Gross, Daniel Plohmann, Herbert Bos. **Highly Resilient Peer-to-Peer Botnets Are Here: An Analysis of Gameover Zeus.** In Proceedings of the 8th IEEE International Conference on Malicious and Unwanted Software (MALWARE'13). October 2013, Fajardo, Puerto Rico, USA.

- Mariano Graziano, Andrea Lanzi, Davide Balzarotti. **Hypervisor Memory Forensics.** In Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID). October 2013, Saint Lucia.

- Istvan Haller, Asia Slowinska, Matthias Neugschwandtner, Herbert Bos. **Dowsing for overflows: A guided fuzzer to find buffer boundary violations.** In Proceedings of the 22nd USENIX Security Symposium (USENIX-SEC). August 2013, Washington, DC, USA.

- Matthias Neugschwandtner, Martina Lindorfer, Christian Platzer. **A view to a kill: Webview exploitation.** In Proceedings of the 6th USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET). August 2013, Washington, DC, USA.

- Zlatogor Minchev. **Security of Digital Society. Technological Perspectives & Challenges.** In Proceedings of the Jubilee International Scientific Conference "Ten Years Security Education in New Bulgarian University: Position and Perspectives for the Education in a Dynamic and Hardly Predicted Environment". June 2013, Sofia, Bulgaria.

- Luben Boyanov, Zlatogor Minchev and Kiril Boyanov. **Some Cyber Threats in Digital Society.** In International Journal "Automatics & Informatics". January 2013.

- Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, Christopher Kruegel. **DISCLOSURE: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow Analysis.** In Proceedings of the 2012 Annual Computer Security Applications Conference (ACSAC). December 2012, Orlando, FL, USA.

- Mario Graziano, Corrado Leita, Davide Balzarotti. **Towards Network Containment in Malware Analysis Systems.** In Proceedings of the 2012 Annual Computer Security Applications Conference (ACSAC). December 2012, Orlando, FL, USA.

- Martina Lindorfer, Alessandro Di Federico, Federico Maggi, Paolo Milani Comparetti, Stefano Zanero. **Lines of Malicious Code: Insights Into the Malicious Software Industry.** In Proceedings of the 2012 Annual Computer Security Applications Conference (ACSAC). December 2012, Orlando, FL, USA.

- Markus Kammerstetter, Christian Platzer, Gilbert Wondracek. **Vanity, Cracks and Malware: Insights into the Anti-Copy Protection Ecosystem.** In Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS). October 2012, Raleigh, NC, USA.

- Zlatogor Minchev. **Social Networks Security Aspects: A Technological and User Based Perspectives.** In Proceedings of the 20th National Jubilee Conference with International Participation (TELECOM2012). October 2012, Sofia, Bulgaria.

- Andrei Bacs, Remco Vermeulen, Asia Slowinska, Herbert Bos. **System-level Support for Intrusion Recovery.** In proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). July 2012, Heraklion, Greece.

- Christian Rossow, Christian Dietrich, Herbert Bos. **Large-Scale Analysis of Malware Downloaders.** In proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA). July 2012, Heraklion, Greece.

- Davide Canali, Andrea Lanzi, Davide Balzarotti, Mihai Christoderescu, Christopher Kruegel, Engin Kirda. **A Quantitative Study of Accuracy in System Call-Based Malware Detection.** In proceedings of the International Symposium on Software Testing and Analysis (ISSTA). July 2012, Minneapolis, MN, USA.

- Farnaz Moradi, Tomas Olovsson, Philippas Tsigas. **An Evaluation of Community Detection Algorithms on Large-Scale Email Traffic.** In proceedings of the 11th International Symposium on Experimental Algorithms (SEA). June 2011, Bordeaux, France.

- Zlatogor Minchev, Plamen Gatev. **Psychophysiological Evaluation of Emotions due to the Communication in Social Networks.** In Scripta Scientifica Medica, Volume 44, Issue 1, Supplement 1. April 2012.

- Federico Maggi, Andrea Bellini, Guido Salvaneschi, Stefano Zanero. **Finding Non-trivial Malware Naming Inconsistencies.** In proceedings of the 7th International Conference on Information Systems Security (ICISS). December 2011, Kolkata, India.

- Matthias Neugschwandtner, Paolo Milani Comparetti, and Christian Platzer. **Detecting Malwares Failover C&C Strategies with SQUEEZE.** In Proceedings of the2011 Annual Computer Security Applications Conference (ACSAC). December 2011, Orlando, FL, USA.

- Matthias Neugschwandtner, Paolo Milani Comparetti, Gregoire Jacob, Christopher Kruegel. **FORECAST Skimming off the Malware Cream.** In Proceedings of the2011 Annual Computer Security Applications Conference (ACSAC). December 2011, Orlando, FL, USA.

- Christian J. Dietrich, Christian Rossow, Felix C. Freiling, Herbert Bos, Maarten van Steen, Norbert Pohlmann. **On Botnets that use DNS for Command and Control.** In Proceedings of the 7th European Conference on Computer Network Defense (EC2ND). September 2011, Gteborg, Sweden.

- Danesh Irani, Marco Balduzzi, Davide Balzarotti, Engin Kirda, Carlton Pu. **Reverse Social Engineering Attacks in Online Social Networks.** In Proceedings of the8th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA). July 2011, Amsterdam, The Netherlands.

- Magnus Almgren, Zhang Fu, Erland Jonsson, Pierre Kleberger, Andreas Larsson, Farnaz Moradi, Tomas Olovsson, Marina Papatriantafilou, Laleh Pirzadeh, Philippas Tsigas. **Mapping Systems Security Research at Chalmers.** In proceedings of the 1st SysSec Workshop on Systems Security. July 2011, Amsterdam, The Netherlands.

- Farnaz Moradi, Magnus Almgren, Wolfgang John, Tomas Olovsson, Philippas Tsigas. **On Collection of Large-Scale Multi-Purpose Datasets on Internet Backbone Links.** In proceedings of the First Work- shop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS). April 2011, Salzburg, Austria.

- Leyla Bilge, Andrea Lanzi, Davide Balzarotti. **Thwarting Real-Time Dynamic Unpacking.** In proceedings of the 2011 European Workshop on System Security(EuroSec). April 2011, Salzburg, Austria.

- Zhang Fu, Marina Papatriantafilou, Philippas Tsigas. **CluB: A Cluster Based Proactive Method for Mitigating Distributed Denial of Service Attacks.** In proceedings of the 26th ACM Symposium on Applied Computing (SAC). March 2011, TaiChung, Taiwan.

- Leyla Bilge, Engin Kirda, Christopher Kruegel, Marco Balduzzi. **EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis.** In proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS). February 2011, San Diego, CA, USA.

- Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, Engin Kirda. **G-Free: Defeating Return-Oriented Programming through Gadget-less Binaries.** In Proceedings of the 26th Annual Computer

Security Applications Conference (ACSAC). December 2010, Austin, TX, USA.

- Zlatogor Minchev, Maria Petkova. **Information Processes and Threats in Social Networks: A Case Study.** In Conjoint Scientific Seminar "Modelling and Control of Information Processes". Organized by College of Telecommunications, Institute of ICT - Bulgarian Academy of Sciences and Institute of Mathematics and Informatics - Bulgarian Academy of Sciences. November 2010, Sofia, Bulgaria.

- Phuong Nguyen, Wil Kling, Giorgos Georgiadis, Marina Papatriantafilou, Anh Tuan Le, Lina Bertling. **Distributed Routing Algorithms to Manage Power Flow in Agent-Based Active Distribution Network.** Proceedings of 1st Conference on Innovative Smart Grid Technologies Europe. Gteborg, Sweden, October 2010.

2 Malware infection

Before diving into general details of malware, their different families and types, we would like to tackle the very basic problem of how malware reaches a target system in the first place. There still exist a good number of click-based infection strategies, where individuals are tricked into clicking a binary and install the malware themselves. These methods, however, are trivial from a technological perspective. Injection attacks, in contrast, are methods where vulnerabilities in a computer system are exploited without any user interaction. The research presented in this chapter is described in detail in [62]. This work mainly deals with code injection attacks which have become one of the primary methods of malware spreading. In a typical code injection attack, the attacker sends a malicious input that exploits a memory corruption vulnerability in a program running on the victim's computer. The injected code, known as *shellcode*, carries out the first stage of the attack, which usually involves the download and execution of a malware binary on the compromised host.

Once sophisticated tricks of the most skilled virus authors, advanced evasion techniques like code obfuscation and polymorphism are now the norm in most instances of malicious code [45]. The wide availability of ready-to-use shellcode construction and obfuscation toolkits and the discovery rate of new vulnerabilities have rendered exploit or vulnerability specific detection techniques ineffective [78]. A promising approach for the generic detection of code injection attacks is to focus on the identification of the shellcode that is indispensably part of the attack vector, a technique initially known as abstract payload execution [85]. Identifying the presence of the shellcode itself allows for the detection of previously unknown attacks without caring about the particular exploitation method used or the vulnerability being exploited.

Initial implementations of this approach attempt to identify the presence of shellcode in network inputs using static code analysis [85, 88, 87].

However, methods based on static analysis cannot effectively handle malicious code that employs advanced obfuscation tricks such as indirect jumps and self-modifications. Dynamic code analysis using emulation is not hindered by such obfuscations and can detect even extensively obfuscated shellcode. This kind of "actual" payload execution has proved quite effective in practice [61] and is being used in network-level and host-level systems for the zero-day detection of both server-side and client-side code injection attacks [63, 95, 18, 4, 22].

A limitation of the above techniques is that they are confined to the detection of a particular class of polymorphic shellcode that exhibits self-decrypting behavior. Although shellcode "packing" and encryption are commonly used for evading signature-based detectors, attackers can achieve the same or even higher level of evasiveness without the use of self-decrypting code, rendering above systems ineffective. Besides code encryption, polymorphism can instead be achieved by mutating the actual instructions of the shellcode before launching the attack—a technique known as *metamorphism* [82]. Metamorphism has been widely used by virus authors and thus can trivially be applied for shellcode mutation. Surprisingly, even *plain* shellcode, i.e., shellcode that does not change across different instances, is also not detected by existing payload execution methods. Technically, a plain shellcode is no different than any instance of metamorphic shellcode, since both do not carry a decryption routine nor exhibit any self-modifications or dynamic code generation. Consequently, an attack that uses a previously unknown static analysis-resistant plain shellcode will manage to evade existing detection systems.

In this chapter, we discuss a comprehensive shellcode detection technique based on payload execution. In contrast to previous approaches that use a single detection algorithm for a particular class of shellcode, this method relies on several runtime heuristics tailored to the identification of different shellcode types. This strategy also enables the detection of specialized polymorphic shellcode, which usually goes undetected by existing polymorphic shellcode detectors.

An implementation of the above technique was named *Gene*, a network-level detector that scans all client-initiated streams for code injection attacks against network services [62]. Gene is based on passive network monitoring, which offers the benefits of easy large-scale deployment and protection of multiple hosts using a single sensor, while it allows us to test the effectiveness of this technique in real-world environments. As the core engine can analyze arbitrary data, it allows the approach to be readily embedded in existing systems that employ emulation-based detection in other domains, e.g., for the detection of malicious websites [22] or in browser add-ons for the detection of drive-by download attacks [18].

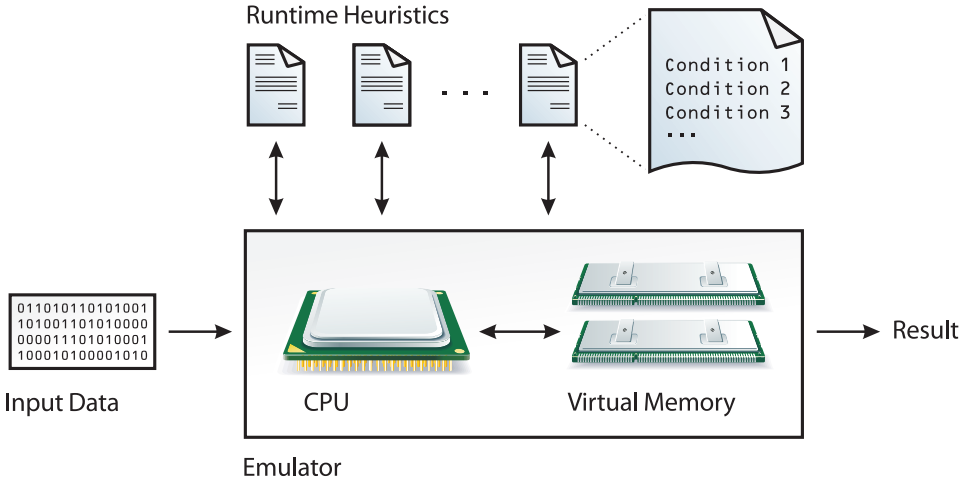


Figure 2.1: Overview of the shellcode detection architecture.

2.1 The Gene architecture

The *Gene* detection system is built around a CPU emulator that executes valid instruction sequences found in any inspected input. An overview of the approach is illustrated in Fig. 2.1. Each input is mapped to an arbitrary location in the virtual address space of a supposed process, and a new execution begins from each and every byte of the input, since the position of the first instruction of the shellcode is unknown and can be easily obfuscated. The detection engine is based on multiple heuristics that match runtime patterns inherent in different types of shellcode. During execution, the system checks several conditions that should all be satisfied in order for a heuristic to match some shellcode. Moreover, new heuristics can easily be added due to the extensible nature of the system.

Existing polymorphic shellcode detection methods focus on the identification of self-decrypting behavior, which can be simulated without any host-level information [63]. For example, accesses to addresses other than the memory area of the shellcode itself are ignored. However, shellcode is meant to be injected into a running process and it usually accesses certain parts of the process' address space, e.g., for retrieving and calling API functions. In contrast to previous approaches, the emulator used here is equipped with a fully blown virtual memory subsystem that handles all user-level memory accesses and enables the initialization of memory pages with arbitrary content. This allows to populate the virtual address space of the supposed process with an image of the mapped pages of a process taken from a real system.

The purpose of this functionality is twofold: First, it enables the construction of heuristics that check for memory accesses to process-specific

Abbreviation	Matching Shellcode Behavior
PEB	`kernel32.dll` base address resolution
BACKWD	`kernel32.dll` base address resolution
SEH	Memory scanning / SEH-based GetPC code
SYSCALL	Memory scanning

Table 2.1: Overview of the shellcode detection heuristics used in Gene.

data structures. Although the heuristics presented here target Windows shellcode, and thus the address space image used in conjunction with these heuristics is taken from a Windows process, some other heuristic can use a different memory image, e.g., taken from a Linux process. Second, this allows to some extent the correct execution of non-self-contained shellcode that may perform accesses to known memory locations for evasion purposes [7].

2.2 Heuristics

Each heuristic used in Gene is composed of a sequence of conditions that should *all* be satisfied *in order* during the execution of malicious code. Table 2.1 gives an overview of the four used heuristics. They focus on the identification of the first actions of different shellcode types, according to their functionality, regardless of any self-decrypting behavior.

2.2.1 Resolving kernel32.dll

The typical final goal of the shellcode is to give the attacker full control of the victim system. This usually involves just a few simple operations, such as downloading and executing a malware binary on the compromised host. These operations require interaction with the OS through the system call interface, or in case of Microsoft Windows, through the user-level Windows API.

A common fundamental operation is that the shellcode has to first locate the base address of `kernel32.dll`. Since this is an inherent operation that must be performed by any Windows shellcode that needs to call a Windows API function, it is a perfect candidate for the development of a generic shellcode detection heuristic. The following possibilities are feasible:

- **Process Environment Block:** Probably the most reliable and widely used technique for determining the base address of `kernel32.dll` takes advantage of the Process Environment Block (PEB), a user-level structure that holds extensive process-specific information and can be accurately identified.

- **Backwards Searching:** An alternative technique for locating `kernel32.dll` is to find a pointer that points somewhere into the memory area where the `kernel32.dll` has been loaded, and then search backwards until the beginning of the DLL is located [75]. A reliable way to obtain a pointer into the address space of `kernel32.dll` is to take advantage of the Structured Exception Handling (SEH) mechanism of Windows [59], which provides a unified way of handling hardware and software exceptions.

2.2.2 Process Memory Scanning

Some memory corruption vulnerabilities allow only a limited space for the injected code—usually not enough for a fully functional shellcode. In these exploits though, the attacker can inject a second, much larger payload which will land at a random, non-deterministic location, e.g., in a buffer allocated in the heap. The first-stage shellcode can then sweep the address space of the process and search for the second-stage shellcode (also known as the "egg"), which can be identified by a long-enough characteristic byte sequence. This type of first-stage payload is known as "egg-hunt" shellcode [76]. The following two scanning techniques and the corresponding detection heuristics can identify the execution of egg-hunt shellcode.

2.2.2.1 SEH

The first memory scanning technique takes advantage of the structured exception handling mechanism and relies on installing a custom exception handler that is invoked in case of a memory access violation.

Condition S1. The list of SEH frames is stored on the stack, and the current SEH frame is always accessible through `FS:[0]`. The first-stage shellcode can register a custom exception handler that has priority over all previous handlers in two ways: create a new SEH frame and adjust the current SEH frame pointer of the TIB to point to it [76], or directly modify the `Handler` pointer of the current SEH frame to point to the attacker's handler routine. In the first case, the shellcode must update the SEH list head pointer at `FS:[0]`, while in the second case, it has to access the current SEH frame in order to modify its `Handler` field, which is only possible by reading the pointer at `FS:[0]`. Thus, the first condition of the SEH-based memory scanning detection heuristic **(SEH)** is **(S1)**: *(i) the linear address of `FS:[0]` is read or written, and (ii) the current or any previous instruction involved the FS register*.

Condition S2. Another mandatory operation that will be encountered during execution is that the `Handler` field of the custom SEH frame (irrespectively if its a new frame or an existing one) should be modified to point to

the custom exception handler routine. This operation is reflected by the second condition **(S2)**: *the linear address of the `Handler` field in the custom SEH frame is or has been written*. Note that in case of a newly created SEH frame, the `Handler` pointer can be written before or after `FS:[0]` is modified.

Condition S3. Although the above conditions are quite constraining, a third condition can be applied by exploiting the fact that upon the registration of the custom SEH handler, the linked list of SEH frames should be valid. With the risk of stack corruption, the exception dispatcher routine performs thorough checks on the integrity of the SEH chain, e.g., ensuring that each SEH frame is dword-aligned within the stack and is located higher than the previous SEH frame [59]. Thus, the third condition requires that **(S3)**: *starting from `FS:[0]`, all SEH frames should reside on the stack, and the `Handler` field of the last frame should be set to 0xFFFFFFFF*. In essence, the above condition validates that the custom handler registration has been performed correctly.

2.2.2.2 System Call

The extensive abuse of the SEH mechanism in various memory corruption vulnerabilities led to the introduction of SafeSEH, a linker option that produces a table with all the legitimate exception handlers of the image. In case the exploitation of some SafeSEH-protected vulnerable application requires the use of egg-hunt shellcode, an alternative but less reliable method for safely scanning the process address space is to check whether a page is mapped—before actually accessing it—using a system call [76, 75]. As already discussed, although the use of system calls in Windows shellcode is not common, since they are prone to changes between OS versions and do not provide crucial functionality such as network access, they can prove useful for determining if a memory address is accessible.

2.2.3 SEH-based GetPC Code

Before decrypting itself, polymorphic shellcode needs to first find the absolute address at which it resides in the address space of the vulnerable process. The most widely used types of GetPC code for this purpose rely on some instruction from the `call` or `fstenv` instruction groups [63]. These instructions push on the stack the address of the following instruction, which can then be used to calculate the absolute address of the encrypted code. However, this type of GetPC code cannot be used in purely alphanumeric shellcode [45], because the opcodes of the required instructions fall outside the range of allowed ASCII bytes. In such cases, the attacker can follow a different approach and take advantage of the SEH mechanism to get a handle to the absolute memory address of the injected shellcode [77].

When an exception occurs, the system generates an exception record that contains the necessary information for handling the exception, includ-

ing a snapshot of the execution state of the thread, which contains the value of the program counter at the time the exception was triggered. This information is stored on the stack, so the shellcode can register a custom exception handler, trigger an exception, and then extract the absolute memory address of the faulting instruction. By writing the handler routine on the heap, this technique can work even in Windows XP SP3, bypassing any SEH protection mechanisms [77].

In essence, the SEH-based memory scanning detection heuristic described in Sec. 2.2.2.1 does not identify the scanning behavior per se, but the proper registration of a custom exception handler. Although this is an inherent operation of any SEH-based egg-hunt shellcode, any shellcode that installs a custom exception handler can be detected, including polymorphic shellcode that uses SEH-based GetPC code.

2.3 Evaluating Gene

An implementation of the heuristics above produced the results presented in the following section.

2.3.1 Detection Effectiveness

The first shellcodes under evaluation were those contained in the Metasploit Framework [48]. For Windows targets, Metasploit includes six basic payloads for downloading and executing a file, spawning a shell, adding a user account, and so on, as well as nine "stagers." In contrast to an egg-hunt shellcode, which searches for a second payload that has already been injected into the vulnerable process along with the egg-hunt shellcode, a stager establishes a channel between the attacking and the victim host for uploading other second-stage payloads. As shown in Fig. 2.2, both Gene and the GetPC-based heuristic detected the polymorphic versions of the shellcodes. However, the original (plain) versions do not exhibit any self-decrypting behavior and are thus detected only by Gene. For both plain and polymorphic versions, Gene identified the shellcode using the PEB heuristic. The use of the PEB-based method for locating `kernel32.dll` is probably preferred in Metasploit due to its reliability.

The evaluation was continued with 22 samples downloaded from the shellcode repository of the Nepenthes Project [53]. Two of the samples had a broken decryptor and could not be executed properly. By manually unpacking the two payloads and scanning them with Gene, in both cases the shellcode was identified by the PEB heuristic. From the rest 20 shellcodes, 16 were identified by the PEB heuristic, and one, named "Saalfeld," by the SEH heuristic. The Saalfeld shellcode is of particular interest due to the use of a custom SEH handler although it is not an egg-hunt shellcode.

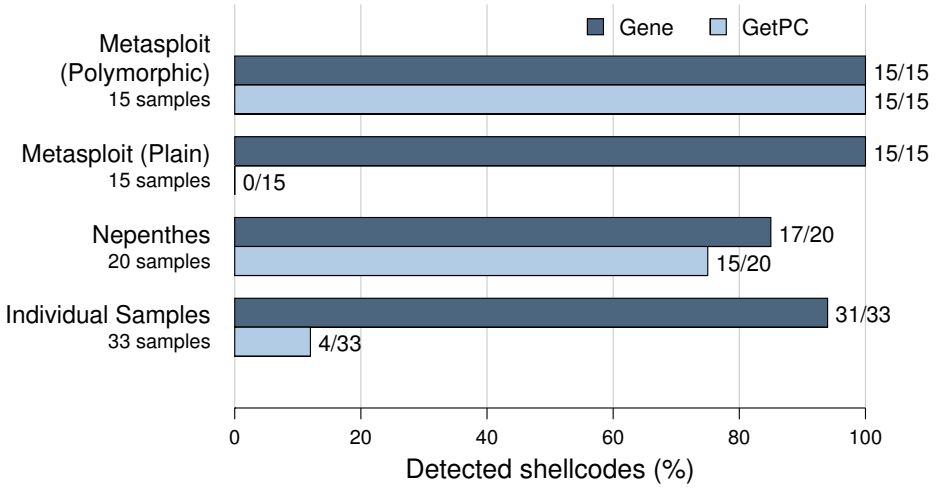


Figure 2.2: Number of shellcodes detected by Gene and the existing GetPC-based heuristic [63, 95, 4] for different shellcode sets. From a total of 83 different shellcode implementations, Gene detected 78 samples (94%), compared to 34 (41%) for the GetPC heuristic.

The SEH handler is registered for safely searching the address space of the vulnerable process starting from address 0x77E00000, with the aim to reliably detect the base address of `kernel32.dll`. The SEH heuristic identifies the proper registration of a custom SEH handler, so the shellcode was successfully identified.

Besides a few proof-of-concept implementations [75, 49] which are identified correctly by Gene, it was not possible to find any other shellcode samples that locate `kernel32.dll` using backwards searching, probably due to the simplicity of the alternative PEB-based technique. In addition to the Saalfeld shellcode, the SEH heuristic detected a proof-of-concept SEH-based egg-hunt implementation [76], as well as the "omelet" shellcode [90], an egg-hunt variation that locates and recombines multiple smaller eggs into the whole original payload. The SEH heuristic was also effective in detecting polymorphic shellcode that uses SEH-based GetPC code [77], which is currently missed by existing payload execution systems. The SYSCALL heuristic was tested with three different egg-hunt shellcode implementations [75, 76, 91], which were identified correctly. As shown in Fig. 2.2, the GetPC-based heuristic detected only four of the shellcodes that use simple XOR encryption, while Gene detected all but two of the samples, again due to the use of hard-coded addresses.

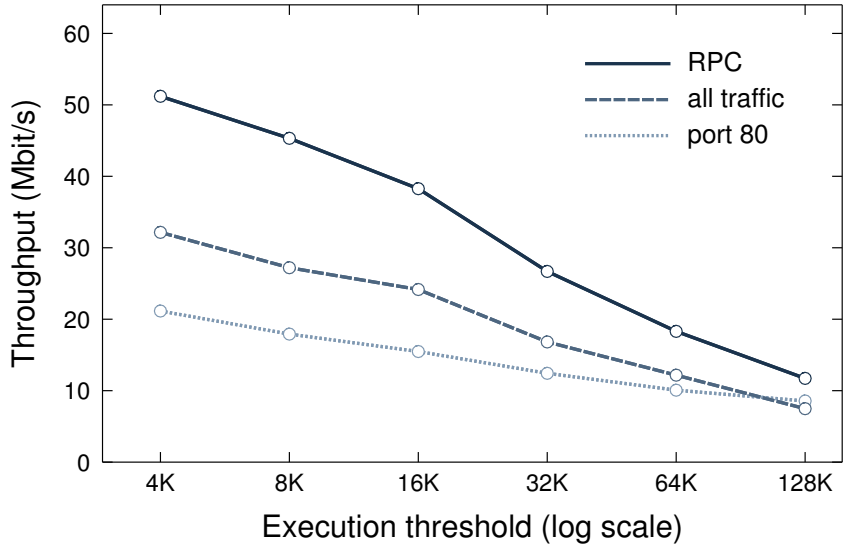


Figure 2.3: The raw processing throughput of Gene for different execution thresholds.

2.3.2 Runtime Performance

The processing throughput of Gene was evaluated using real network traffic traces. Gene was running on a system with a Xeon 1.86GHz processor and 2GB of RAM. Figure 2.3 shows the raw processing throughput of Gene for different execution thresholds. The throughput is mainly affected by the number of CPU cycles spent on each input. As the execution threshold increases, the achieved throughput decreases because more emulated instructions are executed per stream. A threshold in the order of 8–16K instructions is sufficient for the detection of plain as well as the most advanced polymorphic shellcodes [64]. For port 80 traffic, the random code due to ASCII data tends to form long instruction sequences that result to degraded performance compared to binary data.

The overall runtime throughput is slightly lower compared to existing emulation-based detectors [63, 64] due to the overhead added by the virtual memory subsystem, as well as because Gene does not use the zero-delimited chunk optimization used in these systems [63]. Previous approaches skip the execution of zero-byte delimited regions smaller than 50 bytes, with the rationale that most memory corruption vulnerabilities cannot be exploited if the attack vector contains null bytes. However, the detection heuristics of Gene can identify shellcode in other attack vectors that may contain null bytes, such as document files. Furthermore, this approach can be applied in other domains [22, 18], for example for the detection of client-side attacks, in which the shellcode is usually encrypted at a higher level using some

script language, and thus can be fully functional even if it contains null bytes.

In practice, Gene can monitor high speed links when scanning for server-side attacks because client-initiated traffic (requests) is usually a fraction of the server-initiated traffic (responses). In a preliminary deployments in production networks, Gene can scan traffic of up to 100 Mbit/s without dropping packets. Furthermore, Gene currently scans the whole input blindly, without any knowledge about the actual network protocol used. Augmenting the inspection engine with protocol parsing would significantly improve the scanning throughput by inspecting each protocol field separately.

2.4 Discussion

The increasing professionalism of cyber criminals and the vast number of malware variants and malicious websites make the need for effective code injection attack detection a critical challenge. To this end, shellcode detection using payload execution offers important advantages, including generic detection without exploit or vulnerability-specific signatures, practically zero false positives, while it is effective against targeted attacks.

Gene represents a comprehensive shellcode detection method based on code emulation. The approach expands the range of malicious code types that can be detected by enabling the parallel evaluation of multiple runtime heuristics that match inherent low-level operations during the execution of different shellcode types.

An experimental evaluation shows that the proposed approach can effectively detect a broad range of diverse shellcode types and implementations, increasing significantly the detection coverage compared to existing emulation-based detectors, while extensive testing with a large set of benign data did not produce any false positives. Gene detected 116,513 attacks against production systems in a period of almost five months without false positives.

Although Gene currently operates at the network level, the proposed detection heuristics can be readily implemented in emulation-based systems in other domains, including host-level or application-specific detectors.

3 Malware Evasion

The last chapter discussed how malware infections are achieved using shellcode exploits and how it is possible to counter these attacks. However, there is one case when a malware infection is actually wanted. Honeypots try to gather as many samples as possible to further analyze them in a protected environment.

Dynamic analysis of malicious code has increasingly become an essential component of defense against Internet threats. By executing malware samples in a controlled environment, security practitioners and researchers are able to observe its malicious behavior, obtain its unpacked code [32, 44], detect botnet command and control (C&C) servers [81] and generate signatures for C&C traffic [58] as well as remediation procedures for malware infections [54]. Large-scale dynamic malware analysis systems (DMAS) based on tools such as Anubis [9] and CWSandbox [92] are operated by security researchers[1] and companies[2][3]. These services are freely available to the public and are widely used by security practitioners around the world. In addition to these public-facing services, private malware analysis sandboxes are operated by a variety of security companies such as Anti-Virus vendors. Like most successful security technologies, malware analysis sandboxes have therefore attracted some attention from miscreants.

One way for malware to defeat dynamic analysis is to detect that it is running in an analysis sandbox rather than on a real user's system and refuse to perform its malicious function. For instance, code packers that include detection of virtual machines, such as Themida, will produce executables that exit immediately when run inside a virtual machine such as VMWare [38]. There are many characteristics of a sandbox environment that may be used to fingerprint it. In addition to using "red pills" that aim to detect widely de-

[1] Anubis: Analyzing Unknown Binaries (http://anubis.iseclab.org/)
[2] SunbeltLabs (http://www.sunbeltsecurity.com/sandbox/)
[3] ThreatExpert (http://www.threatexpert.com/)

ployed emulation or virtualization technology [74, 66, 55, 20, 21], malware authors can detect specific sandboxes by taking advantage of identifiers such as volume serial numbers or IP addresses. As we will discuss later, sandbox detection is not a theoretical problem; Table 3.1 holds a number of concrete examples of how malware samples have evaded analysis in the Anubis sandbox in the past.

One approach to defeating sandbox evasion is to try to build a *transparent* sandbox. That is, to construct an analysis environment that is indistinguishable from a real, commonly used production environment. This is the goal of systems such as Ether [16]. However, Garfinkel et al. [23] argue that it is fundamentally unfeasible to build a fully transparent virtual machine monitor, particularly if code running in the sandbox has access to the Internet and can therefore query a remote time source. In fact, Ether does not defend against timing attacks that use a remote time source, while Pek et al. [57] have introduced a tool called nEther that is able to detect Ether using local attacks. Even if transparent sandbox technology were available, a specific sandbox installation could be detectable based on the particular configuration of software that happens to be installed on the system, or based on identifiers such as the product IDs of installed software [8] or the universal identifiers of disk partitions.

Another approach relies on running a sample in multiple analysis sandboxes to detect deviations in behavior that may indicate evasion [14, 33, 6, 30]. This is the approach which is also used in [41]. For this, a malware sample is started in several sandboxes, obtaining a number of behavioral profiles that describe its behavior in each environment.

The implementation of this approach is a system called DISARM: Detect-Ing Sandbox-AwaRe Malware.

DISARM detects differences in behavior regardless of their cause, and is therefore completely agnostic to the way that malware may perform sandbox detection. Furthermore, it is also largely agnostic to the monitoring technologies used in the analysis sandboxes, since it does not require heavyweight, instruction-level instrumentation. Any monitoring technology that can detect persistent changes to system state at the operating system level can take advantage of these techniques.

Previous work on detecting and remediating analysis evasion has required fine-grained, instruction-level instrumentation [33, 30]. However, in a DMAS that processes tens of thousands of samples each day, large-scale deployment of instruction-level instrumentation is problematic. This is because it leads to an extremely slow emulated environment, to the point that some malware fail to perform network communication because of server-side timeouts. Furthermore, the produced log files are unmanageably large (up to half a Gigabyte for a single execution according to Kang et al. [33]). DISARM does not suffer from this limitation. This allows to apply the tech-

niques to a significant number of malware samples, revealing a variety of anti-analysis techniques.

Chen et al. [14] also performed a large-scale study of analysis-resistant malware. However, their work assumes that an executable is evading analysis whenever its executions differ by even a single persistent change. This assumption does not seem to hold on a dataset of modern malware: about one in four malware samples produces different persistent changes between multiple executions *in the same sandbox*. DISARM executes malware samples multiple times in each sandbox to establish a baseline for a sample's variation in behavior. Furthermore, it introduces behavior normalization and comparison techniques that allow to eliminate spurious differences that do not correspond to semantically different behavior.

DISARM does not, however, automatically identify the root cause of a divergence in behavior. Detect samples could therefore be further processed using previously proposed approaches to automatically determine how they evade analysis. For instance, the techniques proposed by Balzarotti et al. [6] can be used to automatically diagnose evasion techniques that are based on CPU emulation bugs. Differential slicing [30] is a more general technique that can likewise identify the root cause of a divergence, but it requires a human analyst to select a specific difference in behavior to be used as a starting point for analysis.

3.1 Evasion prevalence in malware

Dynamic malware analysis system (DMAS) like Anubis are mostly based on an instrumented Qemu [11] emulator. The main output of the analysis is a human-readable report that describes the operating system level behavior of the analyzed executable. Anubis has has been offering malware analysis as a free service since February 2007. This service has over 2,000 registered users, has received submissions from 200,000 distinct IP addresses, and has already analyzed over 50,000,000 malware samples.

Public-facing analysis sandboxes such as Anubis are particularly vulnerable to detection, because attackers can probe the sandbox by submitting malware samples specifically designed to perform reconnaissance. Such samples can read out characteristics of the analysis sandbox and then use the analysis report produced by the sandbox to leak the results to the attacker. These characteristics can later be tested by malware that wishes to evade analysis. However, because of sharing of malware samples between sandbox operators, private sandboxes may also be vulnerable to reconnaissance [94], so long as they allow executed samples to contact the Internet and leak out the detected characteristics.

Chen et al. [14] have proposed a taxonomy of approaches that can be used by malware for the detection of analysis sandboxes. These are not lim-

Table 3.1: Anubis evasion techniques according to taxonomy [14] (extended).

Abstraction	Artifact	Test
Hardware	unique id	disk serial number [8]
Environment	execution	MOD R/M emulation bug [55]
		AAM instruction emulation bug
Application	installation	C:\exec\exec.exe present
		username is "USER" [8]
		executable name is "sample.exe" [8]
	execution	popupkiller.exe process running
	unique id	windows product ID [8]
		computer name [8]
		volume serial number of system drive
		hardware GUID
Network	connectivity	get current time from Yahoo home page
		check Google SMTP server response string
	unique id	server-side IP address check [94, 35, 31]

ited to techniques that aim to detect virtualized [74] or emulated [66, 55] environments, but also include application-level detection of characteristic features of a sandbox, such as the presence of specific processes or executables in the system.

Table 3.1 shows a number of Anubis evasion techniques that surfaced over the years, classified according to an extended version of this taxonomy. The unique identifier class is required because many of the detection techniques that have been used against Anubis are not targeted at detecting the monitoring technology used by Anubis, but a specific instance of that technology: The publicly accessible Anubis service. The connectivity class is required because the network configuration of a DMAS faces a trade-off between transparency and risk. It is typically necessary to allow malware samples some amount of network access to be able to observe interesting behavior. On the other hand, we need to prevent the samples from causing harm to the rest of the Internet. A malware sample, however, may detect that it is being provided only limited access to the Internet, and refuse to function. For instance, a DMAS needs to stop malware from sending SPAM. Rather than blocking the SMTP port altogether, it can redirect SMTP traffic to its own mail server. Some variants of the Cutwail SPAM engine detect this behavior by connecting to Gmail's SMTP servers and verifying that the server replies with a specific greeting message.

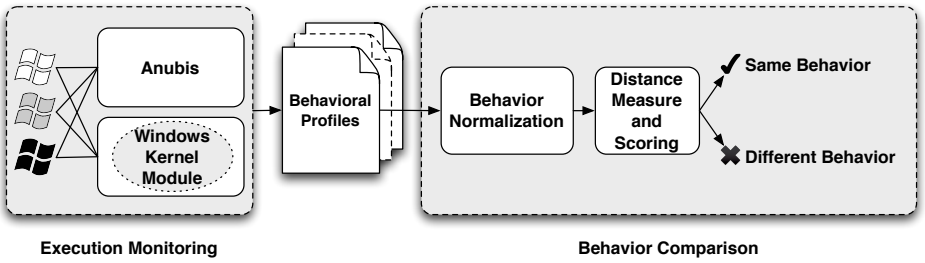


Figure 3.1: System Architecture of DISARM.

In the arms race between malware analysis systems and malware samples that evade analysis, its required to rely on more automation. For this, a scalable tools to screen large numbers of malware samples for evasive behavior is needed, regardless of the class of evasion techniques they employ. This is the role that DISARM aims to fill.

3.1.1 DISARM Architecture

DISARM works in two phases, illustrated in Fig. 3.1. In the execution monitoring phase, a malware sample is executed in a number of analysis sandboxes. The output of this execution monitoring provides the malware's behavior represented as a number of behavioral profiles (one for each execution). In the behavior comparison phase, the behavioral profiles are normalized to eliminate spurious differences. Then the distances between each pair of normalized behavioral profiles is computed. Finally, these distances are combined into an evasion score, that is compared against a threshold to determine whether the malware displayed different behavior in any of the sandboxes. Samples that are classified as showing signs of evasion can then be further analyzed in order to identify new evasion techniques and make sandboxes resilient against these attacks.

3.2 Behavior Comparison

When comparing behavioral profiles produced by different monitoring technologies, it is highly unlikely that they will contain the same amount of features. The reason is that each monitoring technology is likely to have significantly different runtime overheads, so a sample will not be able to execute the same number of actions on each system within a given amount of time. Nor is it possible to simply increase the timeout on the slower system to compensate for this, since monitoring overheads may vary depending on the type of load. Thus, given two sandboxes α and β and the behavioral profiles consisting of n_α and n_β features respectively, DISARM only takes into

account the first $min(n_\alpha, n_\beta)$ features from each profile, ordered by timestamp. In a few cases, however, this approach is not suitable. If the sample terminated on both sandboxes, or it terminated in sandbox α and $n_\alpha < n_\beta$, we have to compare all features. This is necessary to identify samples that detect the analysis sandbox and immediately exit. Samples that detect a sandbox may instead choose to wait for the analysis timeout without performing any actions. Therefore, all features are compared in cases where the sample exhibited "not much activity" in one of the sandboxes. For this, a threshold of 150 features is used, that covers the typical amount of activity performed during program startup. This is the threshold used by Bayer et al. [8], who in contrast observed 1,465 features in the average profile.

Not all features are of equal value for characterizing a malware's behavior. DISARM only takes into account features that correspond to persistent changes to the system state as well as features representing network activity. This includes writing to the file system, registry or network as well as starting and stopping processes and services. This is similar to the approach used in previous work [5, 14] and it leads to a more accurate detection of semantically different behavior.

3.2.1 Behavior Normalization

In order to meaningfully compare behavioral profiles from different executions of a malware sample, a number of normalization steps need to be performed, mainly for the following two reasons: The first reason is that significant differences in behavior occur even when running an executable multiple times within the same sandbox. Many analysis runs exhibit nondeterminism not only in malware behavior but also in behavior occurring inside Windows API functions, executables or services. The second reason is that behavioral profiles obtained from different Windows installations are compared. This is necessary to be able to identify samples that evade analysis by detecting a specific installation. Differences in the file system and registry, however, can result in numerous differences in the profiles. These spurious differences make it harder to detect semantically different behavior. Therefore, the following normalizations are performed on each profile.

- **Noise reduction.** Features generated by benign Windows programs (notepad.exe, calc.exe, winmine.exe, mspaint.exe) are consider as "noise". These features are filtered out of all behavioral profiles.

- **User Generalization.** Programs can write to the user's home directory in `C:\Documents and Settings\<username>` or the Registry without needing special privileges. Accesses to values at these locations are generalized.

- **Environment Generalization.** Other system specific values include hardware identifiers and cache paths which are also generalized.

- **Randomization Detection.** Malware samples often use random names when creating new files or registry keys. The generalized profiles are created by replacing the random names with a special token.

- **Repetition Detection.** Some types of malware perform the same actions on different resources over and over again. If any actions (such as a file write) are found that are performed on three or more such resources, a generalized resource is created in the queried directory and these actions are assigned to it.

- **File System and Registry Generalization.** For each sandbox, a snapshot of the Windows image's state at analysis start is created. This snapshot includes a list of all files, a dump of the registry, and information about the environment which is used to generalize the user and the environment.

3.2.2 Distance Measure and Scoring

The actions in the behavioral profiles are represented as a set of string features. Two behavioral profiles are compared using the Jaccard distance [29]:

$$J(a,b) = 1 - |a \cap b| / |a \cup b|. \tag{3.1}$$

Balzarotti et al. [6] observed that two executions of the same malware program can lead to different execution runs. About 25 % of samples execute at least one different persistent action between multiple executions in the same sandbox. Because of this, one cannot simply consider a high distance score as an indication of evasion. Instead, an evasion score is defined as:

$$E = \max_{1<i<n} \left\{ \max_{1<j<n, i \neq j} \left\{ \text{distance}(i,j) - \max\{\text{diameter}(i), \text{diameter}(j)\} \right\} \right\}. \tag{3.2}$$

Here, $\text{diameter}(i)$ is the full linkage (maximum) distance between executions in sandbox i, while $\text{distance}(i,j)$ is the full linkage (maximum) distance between all executions in sandboxes i and j. The evasion score is thus the difference between the maximum *inter-sandbox distance* and the maximum *intra-sandbox distance*. The evasion score is in the interval [0,1], with 0 representing the same behavior and 1 representing completely different behavior. If this score exceeds an evasion threshold, DISARM declares that the malware has performed semantically different behavior in one of the sandboxes.

Table 3.2: Sandboxes used for evaluation.

Sandbox	Monitoring Technology	Image Characteristics		
		Software	Username	Language
Anubis	Anubis	Windows XP Service Pack 3, Internet Explorer 6	Administrator	English
Admin	Driver	same Windows image as Anubis		
User	Driver	Windows XP Service Pack 3, Internet Explorer 7, .NET framework, Java Runtime Environment, Microsoft Office	User	English
German	Driver	Windows XP Service Pack 2, Internet Explorer 6, Java Runtime Environment	Administrator	German

3.3 Evaluation

To evaluate the proposed approach, the system was tested using two monitoring technologies and three different operating system images. Table 3.2 summarizes the most important characteristics of the four sandboxes.

Each sample is analyzed three times in each of the four different sandboxes, resulting in a total of 12 runs per sample.

3.3.1 Large Scale Test

DISARM was tested on a large dataset of 1,686 samples submitted to Anubis between December 2010 and March 2011. A maximum of five samples were selected per sample family as classified by Kaspersky to make sure the dataset is diverse. The evasion threshold of 0.4 was selected as in the previous section. Since there is no ground truth for this dataset, it is impossible to provide an accuracy score.

In this dataset, a total of 431 (25.56 %) samples resulted in an evasion score above the threshold. Table 3.3 breaks these results down for each pair of sandboxes. The sandboxes *Anubis* and *Admin* use the same Windows image. Conversely, different behavior for any combination of *Admin*, *User* and *German* indicates evasion or environment sensitivity related to differences in the Windows environment. The results for *Anubis-User* and *Anubis-German* are a combination of both factors and therefore produce the highest rates of evasion.

Table 3.3: Samples with evasion scores above the threshold of 0.4 for each pair of sandboxes.

	Anubis	**Admin**	**User**	**German**
Anubis	-	252 (14.95 %)	333 (19.75 %)	303 (17.97 %)
Admin	-	-	121 (7.18 %)	58 (3.44 %)
User	-	-	-	138 (8.19 %)
German	-	-	-	-

Table 3.4: Samples with at least one different persistent action for each pair of sandboxes.

	Anubis	**Admin**	**User**	**German**
Anubis	391 (23.19%)	729 (43.24%)	733 (43.48%)	755 (44.78%)
Admin	-	440 (26.10%)	727 (43.12%)	730 (43.30%)
User	-	-	443 (26.28%)	780 (46.26%)
German	-	-	-	435 (25.80%)

Table 3.4 demonstrates that simply assuming a malware sample is evading an analysis sandbox whenever two executions differ by as little as a single persistent action (as was done in [14]) leads to misleading results. Close to one quarter of samples display some difference in behavior even among executions in the same sandbox. The problem is further exacerbated when considering different sandboxes: Overall 990 samples (58.72 %) behave differently in at least one sandbox according to this criterion.[4]

3.3.2 Qualitative Results

To get a feeling for differences causing a high ranking in the behavior comparison, randomly selected samples were manually rated with an evasion score above the threshold. The following examples illustrate how DISARM succeeded in uncovering new methods used by malware to detect the presence of Anubis, as well as pointing out ways to improve the configuration of the Windows images in analysis sandboxes.

3.3.2.1 Targeted Evasion.

Several timing-sensitive samples were found, that evade analysis by detecting that the environment in which they are running is much slower than a real system. For this, they use the `GetTickCount` API call before and after executing several benign calls such as `GetCommandLine`. If the difference exceeds a threshold, these samples do not exhibit malicious behavior. These

[4]To allow for a fair comparison across different Windows images, the user and environment normalization was applied and network activity ignored.

samples can be detected because the in-the-box monitoring technology, despite running inside Qemu, is much faster than Anubis and does not exceed the specified thresholds. This further highlights the importance of evasion detection techniques that are compatible with lightweight instrumentation. Among these samples were representatives of several well known malware families such as Zeus, Virut, Agent and Rbot.

Win32.Packed.Krap.ag.5 and *Win32.TrojanSpy.Banker.9* evade Anubis by checking if the parent process is explorer.exe. In Anubis samples are started by an analysis daemon, which terminates right away. Thus, this condition is not met and these samples exit without performing any malicious activity.

To render known evasion techniques ineffective, Anubis already randomizes a number of identifiers at each execution, such as those listed in Table 3.1. This is implemented by intercepting the calls to known sources for this information and forging the results. Inspecting samples detected by DISARM, however, reveals that this feature needs to be improved. Several samples query a machine GUID stored in `HKLM\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\MACHINEGUID`, that is not yet covered by Anubis randomization.[5] In other cases, the randomization was insufficient to prevent evasion. *Trojan-Clicker.Win32.Wistler.d* detected Anubis by querying the computer name. The computer names have the format "pc" followed by a random number. Clearly, a stronger randomization of this identifier is needed. Finally, malware can also detect Anubis by checking the hard drive manufacturer information. The randomization feature of Anubis already intercepts the device control code `IOCTL_STORAGE_QUERY_PROPERTY` and the system call `NtQueryVolumeInformationFile` and forges the return information. Some samples, however, were able to bypass this randomization by instead using the device control code `DFP_RECEIVE_DRIVE_DATA` to retrieve the hard drive serial number and manufacturer.

3.3.2.2 Environment Sensitivity.

The results of the evaluation also exposed various configuration flaws in the image currently used in Anubis. In this image, third party extensions for Internet Explorer are disabled. *AdWare.Win32.InstantBuzz* queries this setting and terminates with a popup asking the user to enable browser extensions. Four samples, e.g. *Trojan.Win32.Powp.gen*, infect the system by replacing the Java Update Scheduler. Clearly, they can only show this behavior in the sandboxes in which the Java Runtime Environment is installed. Microsoft Office is only installed in one of the sandboxes and is targeted by *Worm.Win32.Mixor*. *P2P-Worm.Win32.Tibick.c* queries the registry for the presence of a file-sharing application and fails on images where the Kazaa

[5] Note that this is a different identifier than the hardware GUID listed in Table 3.1, which Anubis already randomizes.

file-sharing program is not installed. Using this insight it was possible to modify the image used in Anubis in order to observe a wider variety of malware behavior.

3.3.2.3 Driver Evasion.

Samples are prevented from loading drivers in order to maintain the integrity of the kernel module. Nonetheless, samples were found that not only detect the logging mechanism, but also actively tamper with SSDT hooks. At least 20 samples employ mechanisms to restore the hooks to their original addresses and therefore disable the logging in the driver. This can be done from user space by directly accessing `\device\physicalmemory` and restoring the values in the SSDT with the original values read from the ntoskrnl.exe disk image [83]. Another ten samples achieve the same effect by using the undocumented function `NtSystemDebugControl` to directly access kernel memory. These techniques are employed by several popular malware families such as Palevo/Butterfly, Bredolab, GameThief and Bifrose, probably as a countermeasure against Anti-Virus solutions. By disabling access to kernel memory and instrumenting additional system calls, it is possible to harden a driver against such techniques, so long as the kernel is not vulnerable to privilege-escalation vulnerabilities.

3.3.2.4 False Positives.

False positives were caused by samples from the Sality family. This virus creates registry keys and sets registry values whose name depends on the currently logged in user: `HKCU\SOFTWARE\AASPPAPMMXKVS\A1_0` for "Administrator" and `HKCU\SOFTWARE\APCR\U1_0` for "User". This behavior is not random and not directly related to the user name and therefore undetected by the normalization.

3.4 Discussion

Dynamic malware analysis systems are vulnerable to evasion from malicious programs that detect the analysis sandbox. In fact, the Anubis DMAS has been the target of a variety of evasion techniques over the years.

In this chapter, we introduced DISARM, a system for detecting environment sensitive malware. By comparing the behavior of malware across multiple analysis sandboxes, DISARM can detect malware that evades analysis by detecting a monitoring technology (e.g. emulation), as well as malware that relies on detecting characteristics of a specific Windows environment that is used for analysis. Furthermore, DISARM is compatible with essentially any in-the-box or out-of-the-box monitoring technology.

DISARM was evaluated against over 1,500 malware samples in four different analysis sandboxes using two different monitoring technologies. As a result, several new evasion techniques currently in use by malware were uncovered. These results, however, are not the end of the line. As already mentioned before, there is a never ending arms race between detection and evasion. How such new evasion techniques may look like is yet to be seen.

4 Malware experiments

In the previous chapter, we already mentioned how important it is to create a safe containment for analysis environments. Observing the host- or network-level behavior of malware as it executes constitutes an essential technique for researchers seeking to understand malicious code. Dynamic malware analysis systems like Anubis [10], CWSandbox [92] and others have proven invaluable in generating ground truth characterizations of malware behavior. The anti-malware community regularly applies these ground truths in scientific experiments, for example to evaluate malware detection technologies, to disseminate the results of large-scale malware experiments, to identify new groups of malware, or as training datasets for machine learning approaches. However, while analysis of malware execution clearly holds importance for the community, the data collection and subsequent analysis processes face numerous potential pitfalls.

In this chapter we explore issues relating to prudent experimental evaluation for projects that use malware-execution datasets. As with the previous chapters, detailed information is available in the original publication [73]. Our interest in the topic arose while analyzing malware and researching detection approaches, during which we discovered that well-working lab experiments could perform much worse in real-world evaluations. Investigating these difficulties led to identify and explore the pitfalls that caused them. For example, we observed that even a slight artifact in a malware dataset can inadvertently lead to unforeseen performance degradation in practice. Thus, we highlight that performing prudent experiments involving such malware analysis is harder than it seems. Related to this, we have found that the research community's efforts (including that of the SysSec consortium) frequently fall short of fully addressing existing pitfalls. Some of the shortcomings have to do with presentation of scientific work, i.e., authors remaining silent about information that they could likely add with

ease. Other problems, however, go more deeply, and bring into question the basic representativeness of experimental results.

As in any science, it is desirable for the community to ensure to undertake prudent experimental evaluations. We define experiments reported as prudent if they are are correct, realistic, transparent, and do not harm others. Such prudence provides a foundation for the reader to objectively judge an experiments results, and only well-framed experiments enable comparison with related work. However, experiments in our communitys publications could oftentimes be improved in terms of transparency, e.g., by adding and explaining simple but important aspects of the experiment setup. These additions render the papers more understandable, and enable others to reproduce results. Otherwise, the community finds itself at risk of failing to enable sound confirmation of previous results. In addition, we find that published work frequently lacks sufficient consideration of experimental design and empirical assessment to enable translation from proposed methodologies to viable, practical solutions. In the worst case, papers can validate techniques with experimental results that suggest the authors have solved a given problem, but the solution will prove inadequate in real use. In contrast, **well**-designed experiments significantly raise the quality of science. Consequently, it is important to have guidelines regarding both experimental design and presentation of research results. This chapter aims to frame a set of guidelines for describing and designing experiments that incorporate such prudence, hoping to provide touchstones not only for authors, but also for reviewers and readers of papers based on analysis of malware execution. To do so, we define goals that we regard as vital for prudent malware experimentation: transparency, realism, correctness, and safety. We then translate these goals to guidelines that researchers in our field can use. Most published papers could significantly benefited from considering these guidelines. This assessment is also backed up by a set of conceptually simple experiments performed using publicly available datasets. Following the proposed guidelines can be difficult in certain cases. Still, the proposed guidelines canwhen applicablehelp with working towards scientifically rigorous experiments when using malware datasets.

4.1 Designing experiments

In this section we discuss characteristics important for prudent experimentation with malware datasets. The inspiration to draw these criteria comes from extensive experience with malware analysis and malware detection, as well as from lessons learned when trying to assess papers in the field andin some casesreproducing their results. We emphasize that the goal is not to criticize malware execution studies in general. Instead, we highlight pitfalls when using malware datasets, and suggest guidelines how to devise prudent

experiments with such datasets. We group the pitfalls that arise when relying on data gathered from malware execution into four categories. Needless to say, compiling correct datasets forms a crucial part of any experiment. We further experienced how difficult it proves to ensure realism in malware execution experiments. In addition, transparency must be provided when detailing the experiments to render them both repeatable and comprehensible. Moreover, legal and ethical considerations mandate discussion of how to conduct such experiments safely, mitigating harm to others. For each of these four cornerstones of prudent experimentation, more specific aspects can be outlined and guidelines to ensure prudence can be described.

4.2 Correct Datasets

4.2.1 Check if goodware samples should be removed from datasets

Whereas goodware (legitimate software) has to be present for example in experiments to measure false alarms, it is typically not desirable to have goodware samples in datasets to estimate false negative rates. However, malware execution systems open to public sample submission lack control over whether specimens submitted to the system in fact consist of malware; the behavior of such samples remains initially unknown rather than malicious per se.It would be favorable to use sources of malware specimens gathered via means that avoid the possible presence of goodware; explicitly remove goodware samples from their datasets; or compile sample subsets based on malware family labels.

4.2.2 Balance datasets over malware families

In unbalanced datasets, aggressively polymorphic malware families will often unduly dominate datasets filtered by sample-uniqueness (e.g., MD5 hashes). Authors should discuss if such imbalances biased their experiments, and, if so, balance the datasets to the degree possible.

4.2.3 Check whether training and evaluation datasets should have distinct families

When splitting datasets based on sample-uniqueness, two distinct malware samples of one family can potentially appear in both the training and validation dataset. Appearing in both may prove desirable for experiments that derive generic detection models for malware families by training on sample subsets. In contrast, authors designing experiments to evaluate on previously unseen malware types should separate the sets based on families.

4.2.4 Perform analysis with higher privileges than the malwares

Malware with rootkit functionality can interfere with the OS data structures that kernel-based sensors modify. Such malware can readily influence monitoring components, thus authors ought to report on the extent to which malware samples and monitoring mechanisms collide. For example, kernel-based sensors could monitor whenever a malware gains equal privileges by observing if it is loading a kernel driver. Ideally, sensors are placed at a level where they cannot be modified, such as monitoring system calls with a system emulator or in a VMM.

4.2.5 Discuss and if necessary mitigate analysis artifacts and biases

Execution environment artifacts, such as the presence of specific strings (e.g., user names or OS serial keys) or the software configuration of an analysis environment, can manifest in the specifics of the behavior recorded for a given execution. Particularly when deriving models to detect malware, papers should explain the particular facets of the execution traces that a given model leverages. Similarly, biases arise if the malware behavior in an analysis environment differs from that manifest in an infected real system, for example due to containment policies.

4.2.6 Use caution when blending malware activity traces into benign background activity

The behavior exhibited by malware samples executing in dynamic analysis environments differs in a number of ways from that which would manifest in victim machines in the wild. Consequently, environment-specific performance aspects may poorly match those of the background activity with which experimenters combine them. The resulting idiosyncrasies may lead to seemingly excellent evaluation results, even though the system will perform worse in real-world settings. Authors should consider these issues, and discuss them explicitly if they decide to blend malicious traces with benign background activity.

4.3 Transparency

4.3.1 State family names of employed malware samples

Consistent malware naming remain a thorny issue, but labeling the employed malware families in some form helps the reader identify for which malware a methodology works. A large number of unique malware samples

does not imply family diversity, due to the potential presence of binary-level polymorphism. If page-size limitations do not allow for such verbose information, authors can outsource this information to websites and add references to their paper accordingly.

4.3.2 List which malware was analyzed when

To understand and repeat experiments the reader requires a summary, perhaps provided externally to the paper, that fully describes the malware samples in the datasets. Given the ephemeral nature of some malware, it helps to capture the dates on which a given sample executed to put the observed behavior in context, say of a botnets lifespan that went through a number of versions or ended via a take-down effort.

4.3.3 Explain the malware sample selection

Researchers oftentimes study only a subset of all malware specimens at their disposal. For instance, for statistically valid experiments, evaluating only a random selection of malware samples may prove necessary. Focusing on more recent analysis results and ignoring year-old data may increase relevance. In either case, authors should describe how they selected the malware subsets, and if not obvious, discuss any potential bias this induces. Note that random sample selections still may have imbalances that potentially need to be further addressed.

4.3.4 Mention the system used during execution

Malware may execute differently (if at all) across various systems, software configurations and versions. Explicit description of the particular system(s) used (e.g., Windows XP SP3 32bit without additional software installations) renders experiments more transparent, especially as presumptions about the standard OS change with time. When relevant, authors should also include version information of installed software.

4.3.5 Describe the network connectivity of the analysis environment

Malware families assign different roles of activity depending on a systems connectivity, which can significantly influence the recorded behavior. For example, in the Waledac botnet [79], PCs connected via NAT primarily sent spam, while systems with public IP addresses acted as fast-flux repeaters.

4.3.6 Analyze the reasons for false positives and false negatives

False classification rates alone provide little clarification regarding a systems performance. To reveal fully the limitations and potential of a given approach in other environments, we advocate thoughtful exploration of what led to the observed errors. Sommer and Paxson explored this particular issue in the context of anomaly detection systems [72].

4.3.7 Analyze the nature/diversity of true positives

Similarly, true positive rates alone often do not adequately reflect the potential of a methodology [72]. For example, a malware detector flagging hundreds of infected hosts may sound promising, but not if it detects only a single malware family or leverages an environmental artifact. Papers should evaluate the diversity manifest in correct detections to understand to what degree a system has general discriminative power.

4.4 Realism

4.4.1 Evaluate relevant malware families

Using significant numbers of popular malware families bolsters the impact of experiments. Given the ongoing evolution of malware, exclusively using older or sinkholed specimens can undermine relevance.

4.4.2 Perform real-world evaluations

A realworld experiment can be defined as an evaluation scenario that incorporates the behavior of a significant number of hosts in active use by people other than the authors. Realworld experiments play a vital role in evaluating the gap between a method and its application in practice.

4.4.3 Exercise caution generalizing from a single OS version, such as Windows XP

For example, by limiting analysis to a single OS version, experiments may fail with malware families that solely run or exhibit different behavior on disregarded OS versions. For studies that strive to develop results that generalize across OS versions, papers should consider to what degree we can generalize results based on one specific OS version.

4.4.4 Choose appropriate malware stimuli

Malware classes such as keyloggers require triggering by specific stimuli such as keypresses or user interaction in general. In addition, malware of-

ten expose additional behavior when allowed to execute for more than a short period [72]. Authors should therefore describe why the analysis duration they chose suffices for their experiments. Experiments focusing on the initialization behavior of malware presumably require shorter runtimes than experiments that aim to detect damage functionality such as DoS attacks.

4.4.5 Consider allowing Internet access to malware

Deferring legal and ethical considerations for a moment, we argue that experiments become significantly more realistic if the malware has Internet access. Malware often requires connectivity to communicate with command-and-control (C&C) servers and thus to expose its malicious behavior. In exceptional cases where experiments in simulated Internet environments are appropriate, authors need to describe the resulting limitations.

4.5 Safety

4.5.1 Deploy and describe containment policies

Well designed containment policies facilitate realistic experiments while mitigating the potential harm malware causes to others over time. Experiments should at a minimum employ basic containment policies such as redirecting spam and infection attempts, and identifying and suppressing DoS attacks. Authors should discuss the containment policies and their implications on the fidelity of the experiments. Ideally, authors also monitor and discuss security breaches in their containment.

4.6 Discussion

These guidelines are designed to aid with designing prudent malware-based experiments. In [72], the authors assessed these guidelines by surveying 36 original papers. The survey identified shortcomings in most papers from both top-tier and less prominent venues. By applying these guidelines, the prudence of most of the experiments could have significantly been improved. But what may be the reasons for such discouraging results? The shortcomings in experimental evaluation likely arise from several causes. Researchers may not have developed a methodical approach for presenting their experiments, or may not see the importance of detailing various aspects of the setup. Deadline pressures may lead to a focus on presenting novel technical content as opposed to the broader evaluation context. Similarly, detailed analyses of experimental results are often not given sufficient emphasis. In addition, page-length limits might hamper the introduction

of important aspects in final copies. Finally, researchers may simply overlook some of the presented hidden pitfalls of using malware datasets. Many of these issues can be addressed through devoting more effort to presentation. Improving the correctness and realism of experiments is harder than it seems, though. For instance, while real-world scenarios are vital for realistic experiments, conducting such experiments can prove time-consuming and may raise significant privacy concerns for system or network administrators. Furthermore, it is not always obvious that certain practices can lead to incorrect datasets or lead to unrealistic scenarios. For example, it requires great caution to carefully think of artifacts introduced by malware execution environments, and it is hard to understand that, for example, experiments on overlay datasets may be biased. The significance of imprudent experiments becomes even more important in those instances where current practices inspire others to perform similar experiments - a phenomenon we observed in our survey.

We hope that the guidelines framed in this deliverable improve this situation by helping to establish a common set of criteria that can ensure prudent future experimentation with malware datasets. While many of these guidelines are not new, this approach holds promise both for authors, by providing a methodical means to contemplate the prudence and transparent description of their malware experiments, and for readers/reviewers, by providing more information by which to understand and assess such experiments.

5 URL Shortening Services

In the previous chapters we focused on how malware infects a system and presented some approaches to detect and mitigate malware infections. What we did not yet cover is, how malware exploits a target system once it managed to infect it. One key enabler for these mostly fraudulent activities are URL shortening services. They provide an additional obstacle for anti-malware solutions since they introduce an additional layer of obfuscation. The target domain cannot easily be parsed anymore. Consequently, the information contained in the contacted domain is lost.

In a study characterizing the usage of short URLs [3] it clearly shows that these shortening services are on the rise. The growing popularity of such services is the result of their extensive usage in Online Social Networks (OSNs). Services, like Twitter, impose an upper limit on the length of posted messages, and thus URL shortening is typical for the propagation of content. Consequently, URL shortening services have become part of the web's critical infrastructure, posing challenging questions regarding its performance, scalability, and reliability.

5.1 Security Threats and Countermeasures

Users have grown accustomed to following a URL that looks like `http://bit.ly/lhBa6k`, even when the mapped URL may be `http://evil.com/attack?id=31337`. If it is usually difficult for a user to determine whether a URL is legitimate or not just by looking at it, this is even harder in case of short URLs. As a result, shortening services have been abused by miscreants for masquerading malicious target pages [68, 47, 25]. Large services such as Twitter, Facebook or YouTube have started running their own shortening service, upon which their social networks rely (e.g., `t.co`, `fb.me`, `youtu.be`). Unfortunately, when the hyperlinks of an entire social network rely upon

one, single URL "translator", speed and availability also become of concern (similarly to what happens with the DNS service).

In their paper [43], the authors perform a large-scale and global measurement study of the *threats to users* introduced by short URLs and the *countermeasures adopted* by shortening services. They first assess whether such countermeasures can substitute blacklist-based protections implemented in current browsers, so that users can actually trust URLs exposed by popular shortening services even when client-side defenses are not in place. According to the experiments, popular services react against attempts of shortening long URLs that expose malicious behavior at the time of submission—by either banning offending URLs or by displaying warning pages; however, preliminary experiments show that shortening services do not check existing short URLs periodically. Such checks are useful in case the aliased landing pages turn malicious (e.g., after a timeout expiration).

In addition, the *end users* and how they typically access pages containing short URLs are also considered. Instead of directly crawling short URLs found on web pages, the idea is to "crowdsource" the collection to a large pool of real web users. To this end, a publicly-available web service was developed and deployed, providing a much-needed feature, that is, a preview of a short URL's landing page. While browsing the Web, users submitted 24,953,881 distinct short URLs to the servers automatically, via browser add-ons. Although the users in the dataset rarely stumbled upon malicious short URLs, patterns that characterize malicious short URLs can still be identified. For a more detailed version of this study, please refer to Maggi et al. [43].

5.2 Current Countermeasures

The first goal is to understand what (if any) measures are taken by shortening services to prevent malicious URLs from being shortened and, if they are shortened, the amount of time it takes for such URLs to be flagged and

Service	Malware		Phishing		Spam	
	#	%	#	%	#	%
bit.ly	2,000	100.0	2,000	100.0	2,000	100.0
durl.me	1,999	99.9	1,987	99.4	1,976	98.8
goo.gl*	2000	99.9	994	99.4	1,000	100.0
is.gd	1,854	92.7	1,834	91.7	364	18.2
migre.me	1,738	86.9	1,266	63.3	1,634	81.7
tinyurl.com	1,959	99.5	1,935	96.8	587	29.4
Overall	9,550	95.5	9,022	90.2	6,561	65.6

Table 5.1: Number and percentage of malicious URLs that were accepted for shortening by the top services.

removed. To this end, three types of malicious URLs were submitted to the most popular short URL services that had a public API. More specifically, 10,000 URLs (2,000 for each of the five shortening services examined), picked randomly, among those that were recently submitted to Wepawet and that delivered drive-by-download exploits targeted at vulnerabilities in web browsers and browser plugins (e.g., Adobe Flash, Java) were submitted. In addition, 10,000 phishing URLs that were online and tracked by PhishTank, and 10,000 URLs that were recently observed in spam emails that we obtained from Spamhaus were submitted. The purpose of examining three types of URLs was to determine whether URL shortening services block one or more classes of threats. After submitting the URLs, it was recorded whether the shortening service allowed to shorten the URL in the first place. Then, if the service shortened the URL, it was tracked whether the corresponding short URL could be expanded on a daily basis for a four week period.

In addition to the URLs mentioned above, 10 URLs of each type were submitted, that were manually reviewed to ensure that they were actually still delivering live exploits at the time of submission, as it is common that a malicious URL, once discovered, is brought to the attention of a site administrator and removed. An overview of the results of these measurements is shown in Tab. 5.1. Interestingly, the most popular service, bit.ly, accepted all the malicious URLs that were submitted. Among the services that employs countermeasures, is.gd is particularly effective against spam, as it prevented the vast majority of spam URLs that we submitted from being shortened, while migre.me seems to perform some kind of phishing filtering on submitted URLs.

The situation changes significantly when looking at the warnings that are displayed when short URLs are accessed (expanded), as shown in Tab. 5.2. Overall 2,049 shortened malicious URLs were blacklisted after the submission by these services (about 21.45% of the 9,551 that passed the submission). Here, bit.ly covers a significant fraction of all malicious URLs: It indeed expands a short URL unless it believes the target is malicious. Overall, all services had quite effective spam URL detection systems. It was also

Service	Malware	Phishing	Spam
bit.ly	0.05	11.3	0.0
durl.me	0.0	0.0	0.0
goo.gl	66.4	96.9	78.7
is.gd	1.08	2.27	0.8
migre.me	0.86	14.0	0.0
tinyurl.com	0.66	0.7	2.04
Overall	21.45	26.39	31.38

Table 5.2: Shortened malicious URLs expanded without warnings when accessed.

rather surprising that goo.gl, was not as effective at blocking malware and phishing as (at least) Google's own blacklist.

In summary, bit.ly was the only one that flagged almost all malicious URLs that we shortened, although they were all accepted for shortening with no checks upon submission. On the other hand, is.gd prevented the majority of malicious URLs from being shortened when submitted—probably using lightweight, blacklist-based checks.

5.2.1 Deferred Malicious URLs

It was measured whether shortening services retroactively analyze malicious URLs, that is, it was determined if these services perform any repeated verification of the safety of the long URLs to which their existing short URLs point to. Thus, a web page was set up that served benign HTML content for a period of three days; this page's URL contained 32 random characters to ensure that no legitimate users accidentally stumbled upon the URL. After the third day, the page was modified to redirect to a web site serving non–legitimate content, i.e. malicious, spam or phishing content. All the shortening services that were tested did not detect the previously-benign page that was modified to redirect visitors to a malicious site. After setting up 3,000 distinct web pages hosted at random domains and URLs they were fed to each service, totaling 1,000 distinct short URLs per service for a total of 15,000 short URLs overall. After 72 hours each page was modified so it redirected to a malicious site. More precisely, 1,000 unique URLs were used serving drive-by exploits, 1,000 phishing pages, and 1,000 spam URLs. In other words, after 72 hours, the short URLs became active aliases of the set of 3,000 malicious URLs. The redirection chain of each short URL was monitored on a weekly basis to determine which shortening services displayed an alert or blocked the redirection—as a result of a security check performed *after* the shortening.

From the results in Tab. 5.3 it shows that only 20% of the malicious URLs were blocked by the shortening service when accessed after they became malicious—this 20% is actually due to the fact that durl.me *always* displays a preview for a short URL, regardless of whether the URL is benign or malicious, which is by not a very effective security mechanism. The

Threat	Shortened	Blocked	Not Blocked
Malware	5,000	20%	80%
Phishing	5,000	20%	80%
Spam	5,000	20%	80%
Overall	15,000	20%	80%

Table 5.3: Deferred malicious short URLs submitted and percentage of blocked versus not blocked ones.

other services, however, did not block any malicious short URL, neither at submission time nor after they were modified.

In summary, the most popular shortening services verify the URLs only upon submission, and an attacker can evade this check by shortening a benign URL that will begin to redirect to a malicious page a few moments later. Thus, URL shortening services should periodically sanitize past short URLs, so that benign pages turning malicious can be detected. Clearly, this is not an easy task as it presents the typical challenges of client-side threat analysis (e.g., cloaking, fingerprinting, evasion) [42].

These experiments were conducted in April 2011 and repeated in April 2012. The obtained results were statistically similar, showing that none of the shortening services changed their security measures against dangerous short URLs in the meantime.

5.3 Measurement Approach

Alarmed by the security assessment discussed in Section 5.2, it was important to analyze short URLs at a larger scale, to understand how they are used with malicious intents. Unlike previous work, the focus was directed towards the clients' perspective, so that usage habits can also be characterized: To this end, short URLs were collected while clients accessed web pages that contained short URLs. Moreover, the analysis is not limited to a selection of shortening services (e.g., the most popular ones) nor narrowed on short URLs published on a few, specific online social networks and news aggregators (e.g., Twitter). Instead, a wide variety of URL shortening services, up to 622, whose URLs appear in thousands distinct websites, are covered.

The collection system comprises a browser add-on (named "collector") and a centralized short URL resolver. The *collector* analyzes container pages while the user browses the Web. The *container page* is a web page that, when rendered on a browser, displays or contains at least one short URL. Each collector submits short URLs to a resolver, along with a timestamp, URL of the container page, and client IP address. The *resolver* finds the respective *landing page*.

The browser add-on works on Google Chrome, Mozilla Firefox, Opera, and any browser that support JavaScript-based add-ons. When the browser renders a page, the add-on searches its DOM for short URLs and submits them to the resolver along with the container page URL. The add-on instantiates contextual tooltips associated to each short URL submitted. These tooltips are revealed whenever the mouse cursor hovers on a resolved short URL. The tooltip displays details about the landing page (e.g., URL, size, title, and content type). Users can also contribute by reporting suspicious short

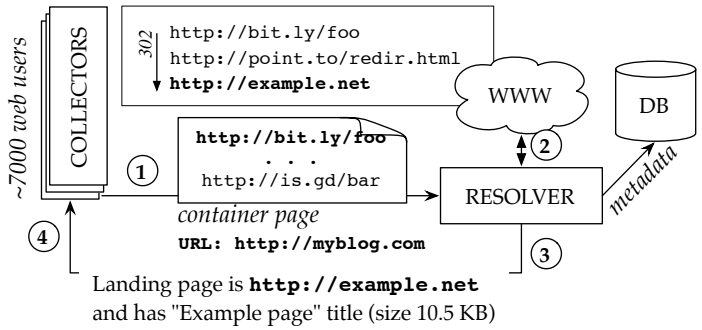


Figure 5.1: Overview of our collection approach.

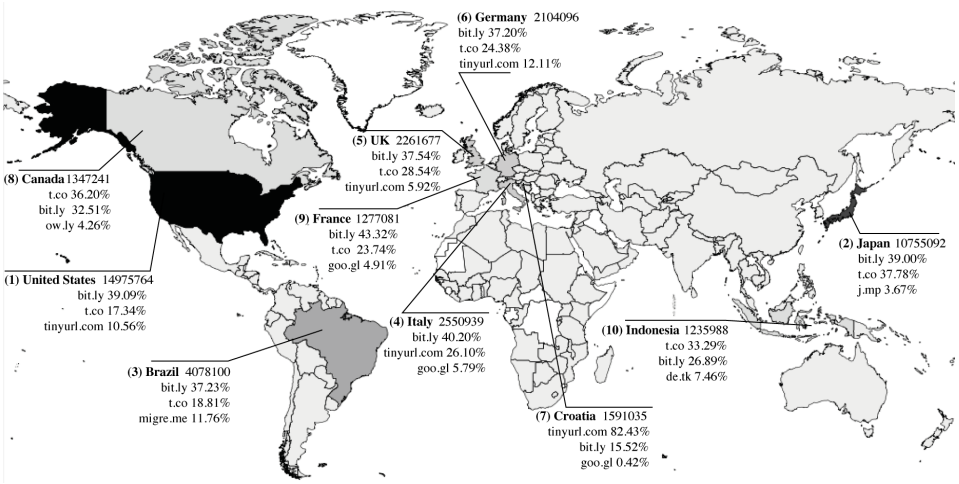


Figure 5.2: Contributors' geographical location.

URLs by clicking on a "flag as suspicious" link on the tooltip. This action is recorded in the database.

For each short URL received, the resolver obtains the landing page URL, title, size, and content type (e.g., HTML, XML, image), by visiting each short URL with a mechanized browser that follows and tracks redirections. When the resolver receives an HTTP 200 response, it assumes that the landing page has been reached and no further redirections follow. The resolver then extracts the relevant data from the landing page's source code and saves the redirection chain. In addition, the collectors' IP addresses are stored for aggregation purposes. The completion of the whole procedure may take up to a few seconds, depending on network conditions and responsiveness of the servers that host the landing and intermediate pages. For this reason, 100 load-balanced, parallel resolvers were deployed along with a caching layer (that stores results for 30 minutes) that ensures short response times. According to the measurements reported in [3], a short URL random suffix,

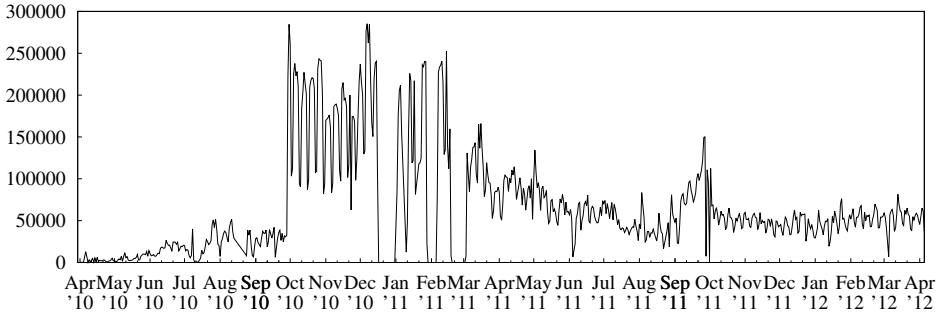


Figure 5.3: Log entries per day between late March 2010 and April 2012.

Distinct URLs		Log entries	
10,069,846	bit.ly	24,818,239	bit.ly
4,725,125	t.co	12,054,996	t.co
1,418,418	tinyurl.com	5,649,043	tinyurl.com
816,744	ow.ly	2,188,619	goo.gl
800,761	goo.gl	2,053,575	ow.ly
638,483	tumblr.com	1,214,705	j.mp
597,167	fb.me	1,159,536	fb.me
584,377	4sq.com	1,116,514	4sq.com
517,965	j.mp	1,066,325	tumblr.com
464,875	tl.gd	1,045,380	is.gd

Table 5.4: The 10 most popular services ranked by number of log entries in the database, and number of distinct short URLs collected. Highlighted rows indicate services at the same rank.

which is its identifier, takes much longer to expire and get recycled. When the cache expires, a snapshot of the log entry (i.e., the history of each short URL) is retained.

In summary, the service takes short URLs as input from clients and returns the aliased landing page. These "expansion" services have become useful for previewing the actual websites behind short URLs. The long time span of the measurement and the usefulness of the service—which is free of charge and publicly available through popular browser add-on marketplaces—allowed to collect a unique, large dataset.

5.3.1 Measurement

The data collection infrastructure was deployed in March 2010 and, as of April 2012, the database contained 24,953,881 unique short URLs. More than 7,000 web users downloaded and installed the browser add-on and submitted short URLs; some users also requested support for additional shortening services. Around 100 out of the 622 that are currently supported by the system were suggested by users. The collection infrastructure re-

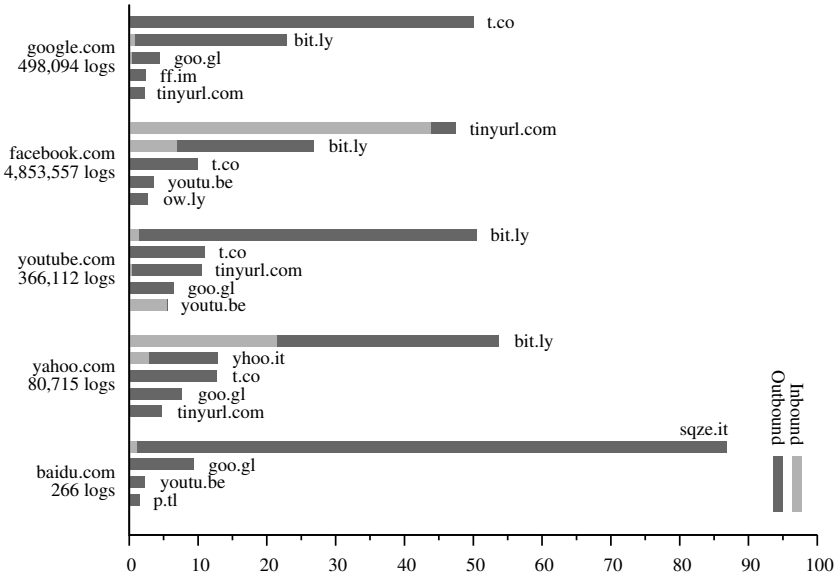


Figure 5.4: Top 5 services by percentage of log entries of outbound short URLs of Alexa top 5 domains.

ceives data from 500 to 1000 active users every day. A record (*Log entry*) is stored in the database for each short URL submitted. Each log entry contains the source (geo-localized) IP address, the container page and landing page URLs, and the timestamp. Thus, each log entry corresponds to one short URL found in a given container page, and represents the fact that a user viewed the container page at a given time. Identifying information possibly related to the specific user who submitted a log entry is never retained.

Overall Dataset Statistics Fig. 5.3 shows the daily number of log entries whereas Fig. 5.2 shows the contributors' geographical location. Albeit the system was deployed in March 2010, the vast majority of users became active contributors starting from Oct 2010. However, at its beginnings the system received 20,000 to 50,000 log entries per day. At steady usage rates, an average of 90,000 log entries was stored per day. Each of the 1,649,071 distinct IPs contributed around 37 requests on average, with a standard deviation of 7,157. Distinct IPs may not correspond to distinct users, either because multiple users could share the same sets of IPs (e.g., via NATs) or because of dynamic IP-assignment policies employed by ISPs. The system experienced three outages due to database failures throughout one year: in late December 2010, in late January 2011, and between late February and March 2011. Nevertheless, a large amount of short URLs were collected which are useful to conduct this study.

Before analyzing security aspects of short URLs, it is necessary to describe the dataset through four aggregated statistics: (1) distinct short URLs, (2) log entries in the database, (3) log entries of *inbound* short URLs (dis-

tinct short URLs pointing to the sites' pages), and (4) *outbound* short URLs (short URLs that are found in their container pages and that point to both external and internal pages). Shortening services with many distinct short URLs are more popular (i.e., they have become the "shortener of choice" for several users), whereas those characterized by many log entries have their short URLs posted on many popular container pages. As shown in Tab. 5.3 the top most popular services in the dataset are bit.ly, t.co and tinyurl.com, respectively. As expected, popular shortening services hold a steadily large number of short URLs, whereas site-specific shortening services exhibit a behavior that is typical of content shared through social networks. Fig. 5.4 shows the ranking of the top websites with respect to inbound and outbound short URLs.

5.4 Results

The objective is to assess if malicious short URLs have distinctive features (§5.4.1) and typical usage patterns (§5.4.2) that criminals may leverage to target their campaigns.

5.4.1 Malicious Short URLs

First, it is necessary to understand how frequently the users in the database encounter malicious short URLs in a page. For this, four datasets are leveraged: the Spamhaus DBL, a list of DNS domains that are known to host spam pages, Wepawet, a service able to detect drive-by-download exploit pages, Google Safe Browsing, a list of domains known for hosting malware or phishing sites, and PhishTank, a blacklist of URLs that are involved in phishing operations. For Spamhaus, the domain was checked against the database. For the other three blacklists, the full URL was checked. Landing URLs are categorized into three classes: spam, phishing, and malware, according to the dataset they were flagged in: URLs detected by Wepawet are flagged as malware, domains found in Spamhaus are marked as spam, and URLs from PhishTank as phishing. Google Safe Browsing classifies both phishing and malware sites. Tab. 5.4.1 summarizes the breakdown of malicious short URLs.

In total, 44,932 unique short URLs were observed, pointing to 19,216 malicious landing pages. By looking at the referrer, these URLs were hosted on 1,213 different domains. A more detailed analysis on the container pages of malicious URLs is provided in the next section. In total, the malicious URLs in the dataset have been rendered by 1,747 users in their container pages via the browser add-ons: 378 users (about 21.6%) were located in South Korea, 282 (about 16.1%) in the United States, and 98 (about 5.6%) in Germany.

Category	Short URLs	Long URLs	Ratio
Phishing	3,806	920	4.1
Malware	27,203	8,462	3.2
Spam	13,184	10,306	1.2

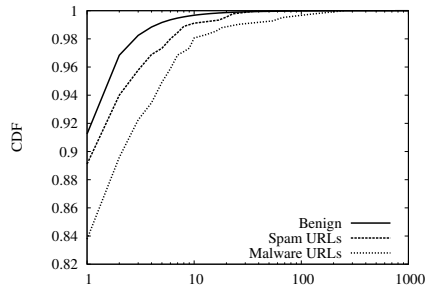
Blacklist	Phishing	Malware	Spam
Spamhaus	-	-	10,306
Phishtank	7	-	-
Wepawet	-	6,057	-
Safe Browsing	913	2,405	-

Table 5.5: Number of short and long URLs, respectively, classified as Phishing, Malware, and Spam. The dash '-' indicates that the blacklist in question provides no data about that threat.

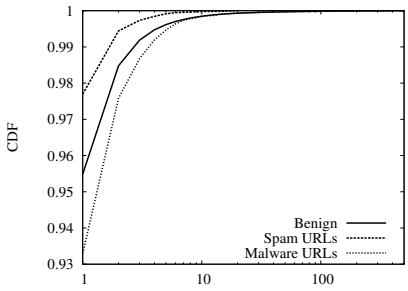
Unsurprisingly, bit.ly is the top most common service, serving 10,392 malicious short URLs, followed by tinyurl.com with 1,389, and ow.ly with 1,327. As a side result, it was also measured whether users perceive and report malicious short URLs. Only 2,577 distinct short URLs have been signaled as malicious through the browser add-ons. Only 2 of these URLs were actually malicious according to at least one of the aforementioned blacklists.

Dissemination of Malicious Short URLs Then it was investigated how malicious pages are aliased through short URLs, and whether this trend changed over time. During the first year of analysis, multiple short URLs were sometimes used to point to the same malicious page, although the average ratio was low. About 2.01 distinct short URLs were used to alias a malicious landing page, whereas an average of 1.3 distinct short URLs were observed per distinct *benign* landing page. Looking at a 2-year span period, however, those average numbers became very similar: about 1.17 unique short URLs per malicious page versus 1.14 unique short URLs per benign page. This comparison is better explained in Fig. 5.5(a) and Fig. 5.5(b), which show the empirical cumulative distribution function for the ratio of short URLs per landing page URL of legitimate vs. malicious pages of the two periods. The pattern here is that benign URLs used to have, in general, less short URLs pointing to them when compared to malicious URLs. Interestingly, in the second year of measurement, the situation changed slightly. In particular, as shown in Fig. 5.5(b), the practice of using multiple short URLs pointing to the same spamming long URL is less used than in the past (Fig. 5.5(a)), where the aliasing of spam URLs was more evident.

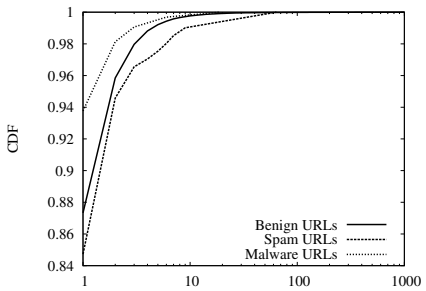
Then the frequent container pages abused to publish malicious short URLs were analyzed through the HTTP requests' `referer` issued while expanding the URLs (9,056 of these had a referrer specified).



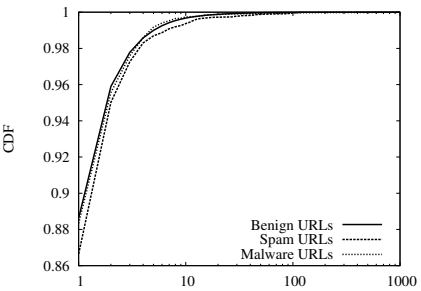
(a) Distinct short URLs per distinct malicious or benign landing URL from April 2010 to April 2011.



(b) Distinct short URLs per distinct malicious or benign landing URL from April 2010 to April 2012.



(c) Distinct containers per distinct malicious or benign short URL from April 2010 to April 2011.



(d) Distinct containers per distinct malicious or benign short URL from April 2010 to April 2012.

Figure 5.5: Comparison of the number of distinct short URLs per unique landing page (a, c) and distinct container page per unique short URL (b, d) after 1 year (a, b) and after 2 years (c, d).

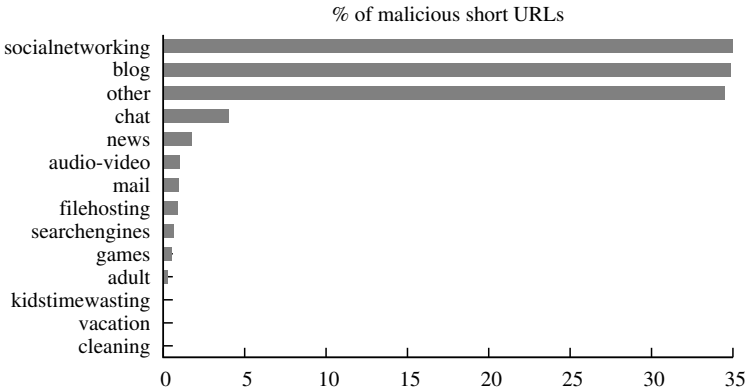


Figure 5.6: Malicious short URLs: Categories of container page ranked by the amount of short URLs they held.

As summarized in Fig. 5.6, the majority of those URLs were found on social networks. More precisely Twitter accounted for 5,881 URLs, 30% of the total. In second position there is Facebook—228 requests, accounting for 1.18% of the total. The third most common referrer is a Belgian news site with 137 requests, accounting for 0.7% of the total. It is plausible that this website was victim of massive comment spam. It is also interesting to look at which sites, among those containing malicious short URLs, attracted the most number of users. Twitter is in first position, with 104 potential victims, followed by Facebook with 31, and by a hacking forum with 27 distinct IP addresses visiting it. This forum is probably another example of comment spam. However, these container pages, which are the most targeted ones, do not contain many short URLs, as detailed in Fig. 5.8: One can argue that the cyber criminals are not considering the "density" of short URLs per container page, but rather its popularity.

Lifespan of Malicious Short URLs In the previous section it was analyzed whether the dissemination of short URLs exhibits different characteristics between malicious and benign content, whereas in this section a comparison between them by means of timing patterns is presented. The maximum lifespan of each collected URL based on historical access logs to their container pages is derived. Then, the *maximum lifespan* (or simply lifespan) is calculated as the delta time between the first and last occurrence of each short URL in the database. More specifically, the definition of lifespan accounts for the fact that short URLs may disappear from some container pages and reappear after a while on the same or other container pages. Fig. 5.7 shows the empirical cumulative distribution frequency of the lifespan of malicious versus benign short URLs. About 95% of the benign short URLs have a lifetime around 20 days, whereas 95% of the malicious short URLs lasted about 4 months. For example, a spam campaign spanning between April 1st and June 30th 2010 was observed that involved 1,806 malicious short URLs redirecting to junk landing pages; this campaign lasted about three months until removed by tinyurl.com administrators. The MessageLabs Intelligence Annual Security Report [2] for that year corroborates these findings: The Storm botnet, which made a significant reappearance in April 2010, seems to be the culprit of this massive spam campaign that contains several shortened URLs.

For the sake of clarity, short URLs involved in such spam campaigns were removed from the second dashed curve in Fig. 5.7; nevertheless, it shows that malicious short URLs last longer than benign URLs, in general. Recall that each short URL may have different container pages at the same point in time, and these can vary over time. Also recall that the longevity of short URLs on each container pages is quite low, as observed in [3] by Antoniades et al. A short URL can make its first appearance on a certain page, disappear to make room for new pages, and reappear a few moments later (even) on

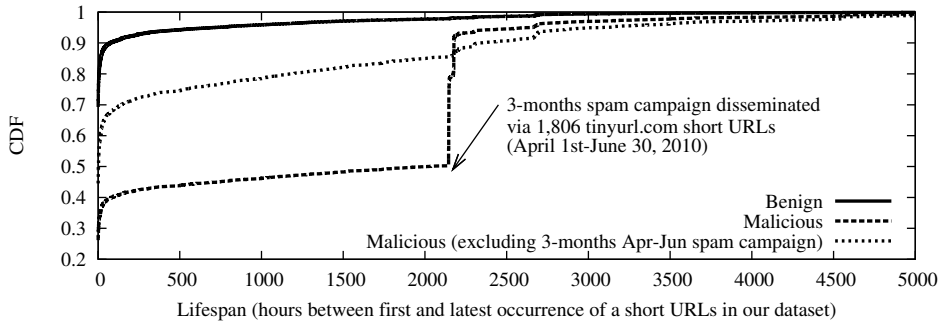


Figure 5.7: Delta time between first and latest occurrence of malicious versus benign short URLs.

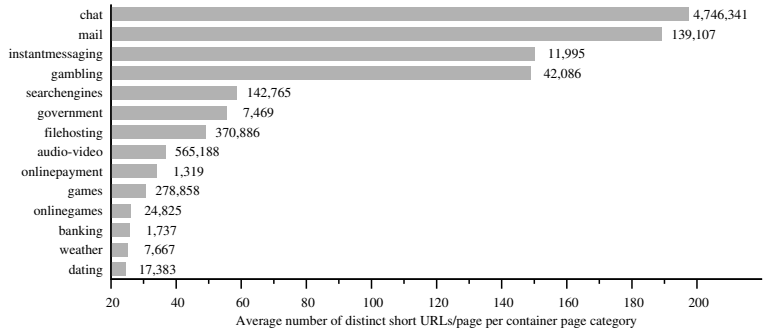


Figure 5.8: Categories of container page ranked by the average number of short URLs/page they held.

different container pages. From this observation, one can argue that, from the miscreants' point of view, the lifespan as we calculate it—across different container pages—seems to be of more importance than the lifespan on a single container page. In fact, short URLs that have a longer lifespan—regardless if they migrate intermittently across several pages—have higher chances of receiving visits from a large audience while remaining stealthy even months after publication. However, those URLs that survive on very popular pages only for a few hours may have their one day of fame before disappearing or being deleted by the container page administrators.

Although we clicks are not tracked on short URLs, the collection method ensures that short URLs are tracked as soon as they appear the web pages visited by the users in our large pool. This ensures us good visibility over their evolution. This is corroborated by the statistics about the abuse of short URLs found in latest three APWG reports [68, 69, 70]: After a growing trend, at the beginning of our measurement (2010), the subsequent reports highlight a stable (2011) and decreasing (2012) trend.

5.4.2 The Short URLs Ecosystem

As part of their research, Antoniades and colleagues in [3] have analyzed the category of the pages to which bit.ly and ow.ly short URLs typically point to, along with the category of the container page, that they had available for bit.ly URLs only. They assigned categories to a selection of URLs. Here, a similar yet more comprehensive analysis was performed by characterizing all the short URLs that were collected by means of the categories described in the following.

Categorizing an arbitrary-large number of websites automatically is a problem that has no solution. However, the goal was to obtain a coarse-grained categorization. To this end, community-maintained directories and blacklists were utilized. More precisely, the container pages (about 25,000,000 distinct URLs) and landing pages (about 22,000,000 distinct URLs) were classified using the DMOZ Open Directory Project (http://www.dmoz.org) and URLBlacklist.com. The former is organized in a tree structure and includes 3,883,992 URLs: URLs are associated to nodes, each with localized, regional mirrors. These nodes are expanded by recursively merging the URLs found in these mirrors. The latter complements the DMOZ database with about 1,607,998 URLs and domains metadata. URLBlacklist.com is used by web-filtering tools such as SquidGuard (http://www.squidguard.org) and contains URLs belonging to clean categories (e.g., gardening, news), possibly undesired subjects (e.g., adult sites), and also malicious pages (i.e., 22.6% of the sites categorized as "antispyware", 18.15% of those categorized as "hacking", 8.29% of pages falling within "searchengine" domains, and 5.7% of the sites classified as "onlinepayment" are in this order, the most rogue categories according to an analysis that was run through McAfee

SiteAdvisor [1]). Overall, it resulted in 74 categories. For clearer visualization, the 48 most frequent categories were selected. These include, for example, "socialnetworking," "adult," "abortion," "contraception," "chat," etc. The word "other" was reserved for URLs belonging to the less meaningful categories that we removed, or for URLs that remained unclassified. Note that each URL can belong to multiple categories.

Frequent and Infrequent Categories Tab. 5.4.2 details the most and least frequent categories of the landing pages pointed to by short URLs of the top services.It shows that the five most popular services are used to refer to various categories including news, audio-video content, blog, and online social networks. However, the majority of short URLs collected come from user-authored content (e.g., online social networks, blog posts), mainly because these sites are very popular (e.g., Facebook). A different viewpoint is provided by plotting the number of short URLs *per page* (Fig. 5.8).

Short URLs are seldom used as aliases of reference, science, and health-related pages. A possible explanation could be that users may have somehow perceived that, as short URLs shall expire sooner or later, they are not reliable for spreading really important content (e.g., health). Secondly, and more importantly, users post short URLs in email and chat messages, weather sites, search-engine pages (including all `*.google.com` pages), do-it-yourself sites, and news pages. In summary, the majority of short URLs point to content that expires quickly. Therefore, from a security viewpoint, real-time countermeasures against malicious short URLs such as WARNINGBIRD [39] are of paramount importance and much more effective than blacklists. As detailed in Fig. 5.6, however, the categories that were most targeted by malicious short URLs in 2010–2012 are social networks and blogs.

Content-category Change To understand how web pages are interconnected through short URLs, it was analyzed whether clicking on a short URL brings the user to a landing page of a category that differs from the category of the container page (e.g., from a news website to a file-hosting website).

In Fig. 5.9, the top 50 shortening services are ranked by the median frequency of category change (plotted as a dot). More precisely, for each service and for each category, the fraction of short URLs that result in a "change of content category" were calculated—such fraction is then normalized by the total number of unique short URLs. Then, the 25- and 75-percent quantiles were derived to define a confidence interval around the median; this is useful to visually highlight how frequencies are distributed. Values close to 100% are not plotted for the sake of clarity. Services with 0–30% change frequency typically deal with a small set of categories and have short URLs often posted on websites of similar subjects. For example, flic.kr is used exclusively within the Flickr ecosystem; therefore, it covers very few cat-

[1] http://siteadvisor.com/sites/

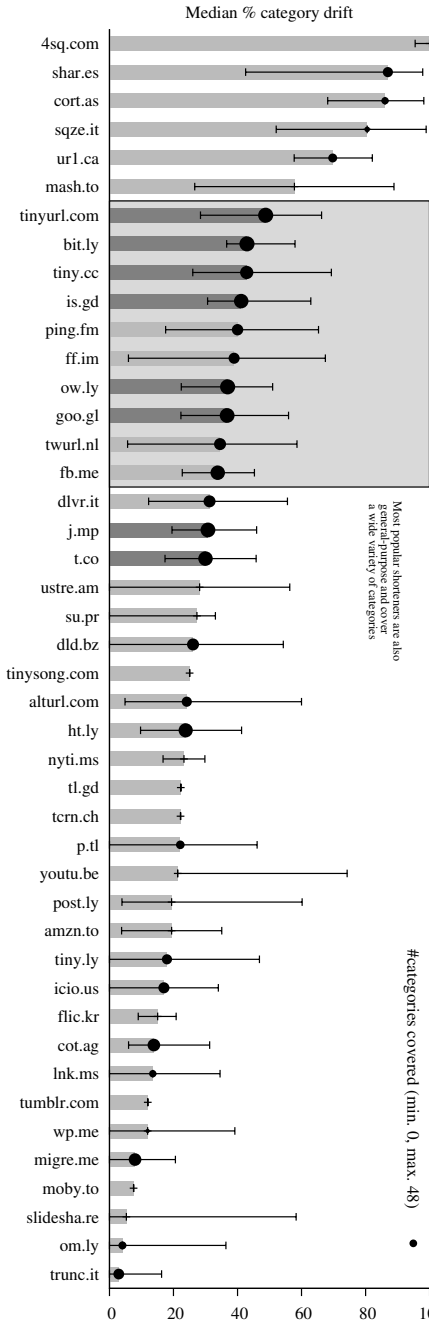


Figure 5.9: Frequency of change of category (median with 25- and 75-percent quantiles) and number of categories covered (size of black dot) of the top 50 services.

Service	Most freq.	%	Least frequent	%
bit.ly	News	23.56	Naturism	$8.3 \cdot 10^{-4}$
	Audio-video	10.62	Contraception	$7.7 \cdot 10^{-4}$
	Socialnet	9	Astrology	$1.6 \cdot 10^{-4}$
t.co	Audio-video	29.42	Naturism	$1.07 \cdot 10^{-3}$
	File-hosting	27.43	Anti-spyware	$8.89 \cdot 10^{-4}$
	News	17.48	Contraception	$1.78 \cdot 10^{-4}$
tinyurl	News	24.08	Contraception	$4.5 \cdot 10^{-3}$
	Audio-video	10.61	Naturism	$6.29 \cdot 10^{-4}$
	File-hosting	9.36	Childcare	$2.51 \cdot 10^{-4}$
goo.gl	News	19.10	Gardening	$3.34 \cdot 10^{-3}$
	Audio-video	12.23	Weapons	$1.69 \cdot 10^{-3}$
	Socialnet	11.65	Naturism	$1.69 \cdot 10^{-3}$
ow.ly	News	23.38	Contraception	$2.5 \cdot 10^{-3}$
	Socialnet	12.84	Childcare	$1.32 \cdot 10^{-3}$
	Audio-video	10.03	Naturism	$1.32 \cdot 10^{-3}$

Table 5.6: Most- and least-popular landing page categories for the top 5 shortening services.

egories and exhibits a very low change frequency, meaning that its short URLs are posted on websites that regard the same subjects, or even on Flickr directly. Another popular example is nyti.ms. On the opposite side, services with values above 50% also cover a small set of categories. However, differently from the first tier (i.e., 0–30%), the categories of the containers of these short URLs rarely match the categories of their landing pages. This is the case, for example, of 4sq.com (about 100%), whose short URLs always bring from online social-networking sites to pages categorized as "other". The most popular shortening services (e.g., bit.ly, goo.gl, ow.ly) fall into the second tier (i.e., 32–48%), together with those services that cover a wide variety of categories, and typically interconnect pages of different categories. The most general-purpose services are those that are more abused to create aliases of malicious URLs: Here is indeed where the vast majority of malicious short URLs can be found. Unfortunately, general-purpose shortening services rely on ineffective countermeasures.

Non-obvious Uses of Short URLs It was also investigated how short URLs interconnect together pages of different categories, to understand whether some categories have a majority of container or landing pages. To this end, the average frequency of category change from the perspective of the container page and landing page was calculated. With this data a weighted digraph with 48 nodes was created, each corresponding to a category. The weights are the frequencies of change, calculated between each pair of categories—and normalized over all the short URLs and pages within each category. Then, the average weight of incoming, $\mathrm{In}(cat)$, and outgoing, $\mathrm{Out}(cat)$, edges for each category cat were computed, and finally the ratio $\rho(cat) = \frac{\mathrm{In}(cat)}{\mathrm{In}(cat)+\mathrm{Out}(cat)}$ was derived. When $\rho \to 0$, the category has a ma-

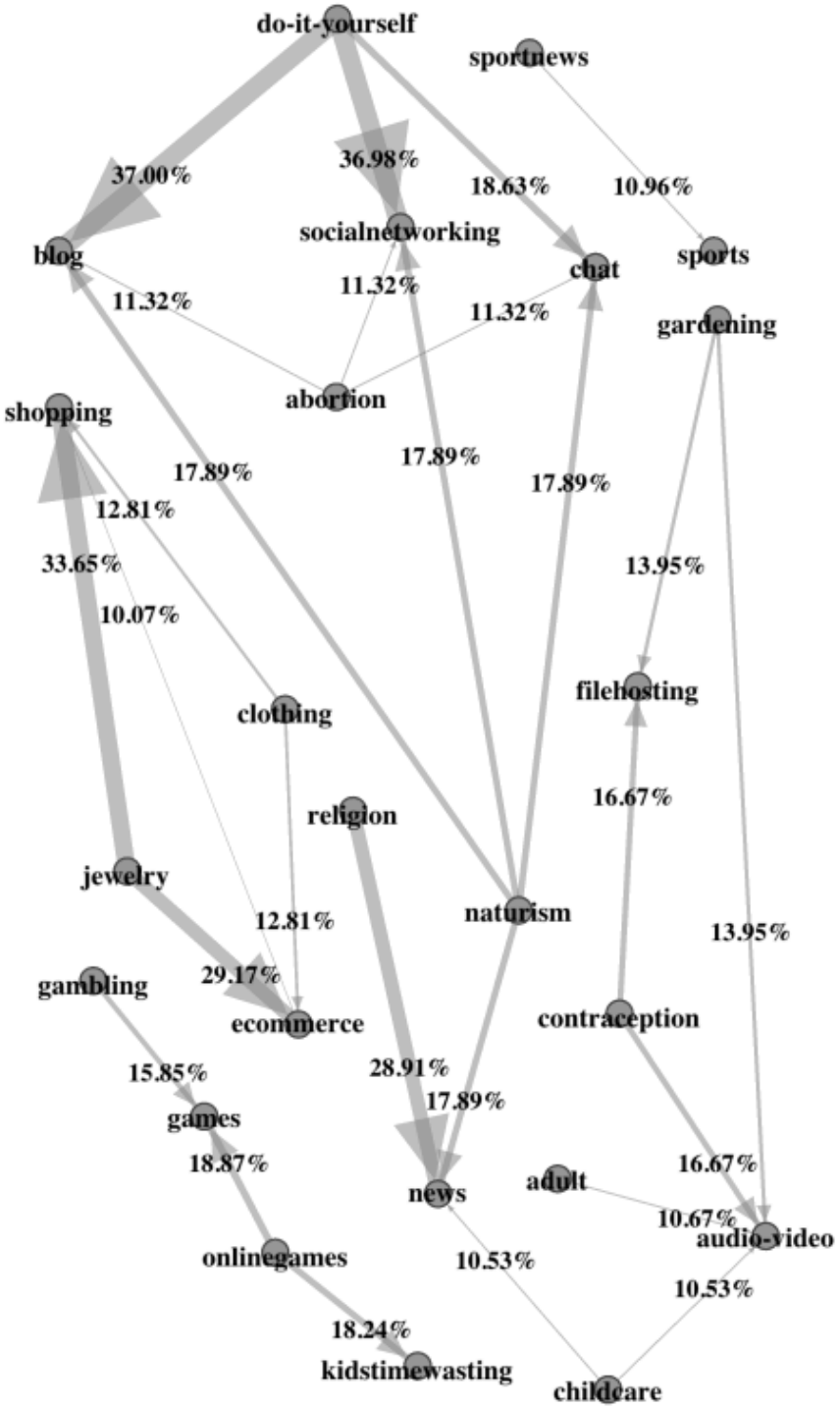


Figure 5.10: Digraph showing the connections between container- and landing-page categories.

ρ	Category		
0.00	abortion	0.07	religion
0.00	antispyware	0.08	personalfinance
0.00	cellphones	0.10	gambling
0.00	childcare	0.14	government
0.00	contraception	0.16	medical
0.00	do-it-yourself	0.16	vacation
0.00	naturism	0.18	onlinegames
0.01	gardening	0.22	onlinepayment
0.01	hacking	0.22	sportnews
0.01	instantmessaging	0.30	searchengines
0.01	jobsearch	0.33	dating
0.01	pets	0.47	kidstimewasting
0.01	weapons	0.55	sports
0.02	artnudes	0.59	adult
0.02	drugs	0.60	games
0.02	jewelry	0.73	ecommerce
0.02	onlineauctions	0.78	shopping
0.02	weather	0.82	blog
0.03	mail	0.82	socialnetworking
0.04	banking	0.83	chat
0.04	cleaning	0.88	news
0.04	clothing	0.90	filehosting
0.06	drinks	0.92	audio-video
0.07	culinary	1.00	astrology

Table 5.7: Ranking of categories by the ratio of incoming and outgoing connections via short URLs.

jority of outgoing short URLs (i.e., many container pages of such category), whereas $\rho \to 1$ indicates that the category has a majority of incoming short URLs (i.e., many landing pages of such categories). The digraph is shown on Fig. 5.10.

As summarized in Tab. 5.7, there are clearly categories that exhibit a container-like usage, that is, they typically contain more outgoing short URLs than incoming short URLs. Besides a few extreme cases, which are mostly due to the scarcity of short URLs, container-like categories include, for instance, "onlineauctions," "mail" (web based emails contain outgoing short URLs more often than being referred to by short URLs), and "hacking."

In summary, categories that would be considered as aggregators (i.e., containers) of short URLs are actually more often used as landing pages. The most notable example is "socialnetworking" ($\rho = 0.82$), which could be expected to have many outgoing links as people share lots of resources through them. Instead, it turns out that, from a global viewpoint, this is no longer true. As expected, landing pages of a category with a high ρ (e.g., "socialnetworking", "blog", "audio-video") are the most obvious target of attacks: Many short URLs that point to malicious resources have their landing page within these categories.

5.5 Discussion

On a global scale, users are seldom exposed, while browsing, to threats spread via short URLs, or at least no more than they are exposed to the same threats spread via long URLs. Although a relatively small number of malicious short URLs can be seen in the wild, it is possible to evade the security measures currently adopted by the top shortening services to filter dangerous URLs, with a simple time-of-check to time-of-use attack. However, shortening services are not—and should not be—a definitive protection layer between users and malicious resources. In-the-browser defense tools such as blacklists can alert users before visiting malicious URLs, regardless of whether they are short or long URLs. Since it is very inefficient for shortening providers to monitor all their aliases periodically, we believe that this is not necessary when modern browsers are already prepared for counteracting known malicious URLs.

On the other hand, the previous chapter explains, why URL shortening services are not necessarily the method of choice to veil malicious URL in click fraud attacks or spam links. While it is true that the exact location of the contained domain can be covered, the underlying security mechanism could very well influence the success rate of a phishing attack. This leads directly to the next chapter, a topic which is even deemed dead by some security researches: Spam.

6 Spam Mitigation

The last chapters dealt with technological background of malware, how it can be analyzed and how to conduct the necessary elements in the first place. For a comprehensive picture of the malware landscape, we want to show what malicious software is used for and how it can be monetized. Malware is the underlying platform for online crime: From spam to identity theft, denial of service attacks or the emergence of fake-antivirus scams in recent years [80], malware covertly installed on a victim's computer plays an essential role in criminals' online operations. Thus, investigating the fraudulent schemes performed by criminals in order to monetize malware installations is also an important part of malware research.

In this chapter we look at one monetization vector of modern malware in particular: The sending of unsolicited bulk emails, or spam. First, we look at the detection of bots engaged in sending spam. We show that by modeling legitimate ("ham") and unsolicited ("spam") email traffic using social network properties (see Section 6.1) we can identify abusive bots in the network.

Furthermore, as we discuss in Section 6.2, there is also an often overlooked gray area between legitimate (ham) and unsolicited (spam) emails. Messages in this area are often hard to classify with existing antispam solutions, resulting in serious consequences for the end user: As our experiments show, users often find it difficult to distinguish between spam and ham messages, putting them at risk of falling victim to botnet-generated spam campaigns.

6.1 Spam Bot Identification

Eliminating the excessive amount of unsolicited *spam* which is consuming network and mail server resources is quite challenging. These email communications mostly originate from botnets of compromised machines [67, 34]

that are also likely the source of other malicious activities on the Internet. Although current antispam tools are efficient in hiding spam from users' mailboxes, there is a clear need for moving the defense against spam as close to its source as possible. Therefore, it is necessary to understand the network-level behavior of spam and how it differs from legitimate traffic in order to design antispam mechanisms that can identify spamming bots on the network. We study the network-level behavior of email by examining real email traffic captured on an Internet backbone link. From the collected traffic, we have generated *email networks* in which the nodes represent email addresses and the edges represent email communications. To the best of our knowledge, this is the largest email traffic dataset used to study the structure of email networks which contain both legitimate (*ham*) and unsolicited email traffic.

We show that the legitimate email traffic exhibits the same structural properties that other social and interaction networks (e.g., online social networks, the Internet topology, the Web, and phone call graphs) typically exhibit, therefore, it can be modeled as a *scale-free*, *small-world* network. We also show that the email traffic containing spam cannot be modeled similarly, and because the unsocial behavior of spam is not hidden behind the social behavior of legitimate traffic, the structure of email networks containing both ham and spam differ from other social networks. Moreover, we show that the temporal variations in the social network properties of email traffic can reveal more distinct properties of the behavior of spam.

In this study our goal is to identify the differences in the social network properties of spam and ham traffic, and leverage these differences to spot the abusive nodes in the network.

For a more detailed version of this study, please refer to Moradi et al. [51].

6.1.1 Data Collection and Pre-processing

We have used two distinct email datasets to generate email networks. The datasets were created from passively captured SMTP packets on a 10 Gbps link of the core-backbone of the SUNET[1]. Each dataset was collected during 14 consecutive days with a year time span between the collections. In the following we refer to the larger dataset as *dataset A*, and the smaller dataset as *dataset B*.

We pruned the unusable email flows, including those with no payload or missing packets and encrypted communications from the datasets. We classified the remaining emails as being either *accepted* (delivered by the receiving mail server) or *rejected* (unfinished SMTP command exchange phase and consequently not containing any email headers and body). Rejection is generally the result of spam pre-filtering strategies deployed by mail servers

[1] Swedish University Network (http://www.sunet.se/)

(e.g., blacklisting, greylisting, DNS lookups). Then, we classified all accepted email communications to be either *spam* or *ham* to establish a ground truth for our study. Similar to related work [24, 86], we used a well-trained filtering tool[2] for this classification. Finally, we anonymized all email addresses and discarded the email contents in order to preserve privacy.

After data collection and pre-processing, we generated a number of email networks from the datasets. In an email network the email addresses, which are extracted from the SMTP commands ("MAIL FROM" and "RCPT TO"), represent the nodes, and the exchanged emails represent the edges. In order to study and compare the characteristics of different categories of email, from each dataset we have generated a *ham network*, a *spam network*, and a *rejected network*, in addition to the complete *email network*.

6.1.2 Structural and Temporal Properties of Email Networks

In this section we briefly introduce the most significant structural and temporal properties of social networks.

Degree Distribution. The degree distribution of a network is the probability that a randomly selected node has k edges. In a *power law distribution*, the fraction of nodes with degree k is $n(k) \propto k^{-\gamma}$, where γ is a constant exponent. Networks characterized by such degree distribution are called *scale-free* networks. Many real networks such as the Internet topology [19], the Web [13], phone call graphs [52], and online social networks [50] are scale free.

Average Path Length. In *small-world* networks any two nodes in the network are likely to be connected through a short sequence of intermediate nodes, and the network diameter shrinks as the network grows [40].

Clustering Coefficient. In addition to a short average path length, *small-world* networks have high clustering coefficient values [89]. The clustering coefficient of a node v is defined as $C_v = 2E_v/(k_v(k_v - 1))$, where, k_v denotes the number of neighbors of v, $k_v(k_v - 1)/2$ the maximum number of edges that can exist between the neighbors, and E_v the number of the edges that actually exist. The average C_v of a social network shows to what extent friends of a person are also friends with each other and its value is independent of the network size [71].

Connected Components. A connected component (*CC*) is a subset of nodes of the network where a path exists between any pair of them. As social networks grow a giant CC (GCC), which contains the vast majority of the nodes in the network, emerges in the graph and its size increases over time [71]. Moreover, the distribution of CC size for some social networks follows a power law pattern [13, 52].

[2] SpamAssassin (http://spamassassin.apache.org)

6.1.2.1 Measurement Results

In the following we present the observed structural and temporal properties of our email networks. These properties can be used in order to model the behavior of legitimate traffic and to find the distinguishing properties of the unsocial behavior of spam. Although the duration of our data collections is not long enough to study the evolution of email networks, it is still possible to track the changes in the structure of email networks in order to better understand the distinct characteristics of ham and spam traffic.

Degree Distribution. Figures 6.1(a) and 6.1(e) show that none of the email networks generated from datasets *A* and *B* exhibit a power law degree distribution in all points. However, the ham networks generated from each of the datasets are scale free as their degree distribution closely follow the distribution $n(k) \propto k^{-\gamma}$ with $\gamma_A = 2.7$ and $\gamma_A = 2.3$, respectively [3]. The in-degree (out-degree) distribution for ham networks, which are shown in Figures 6.1(b) and 6.1(f), also follows a power-low distribution with $\gamma_{A_{in}} = 3.2$ ($\gamma_{A_{out}} = 2.3$) and $\gamma_{B_{in}} = 3.2$ ($\gamma_{B_{out}} = 2.1$), respectively. Moreover, in contrast to previous studies [24, 12], neither the spam, nor the rejected networks are completely scale free (Figures 6.1(c), 6.1(g), 6.1(d), and 6.1(h)).

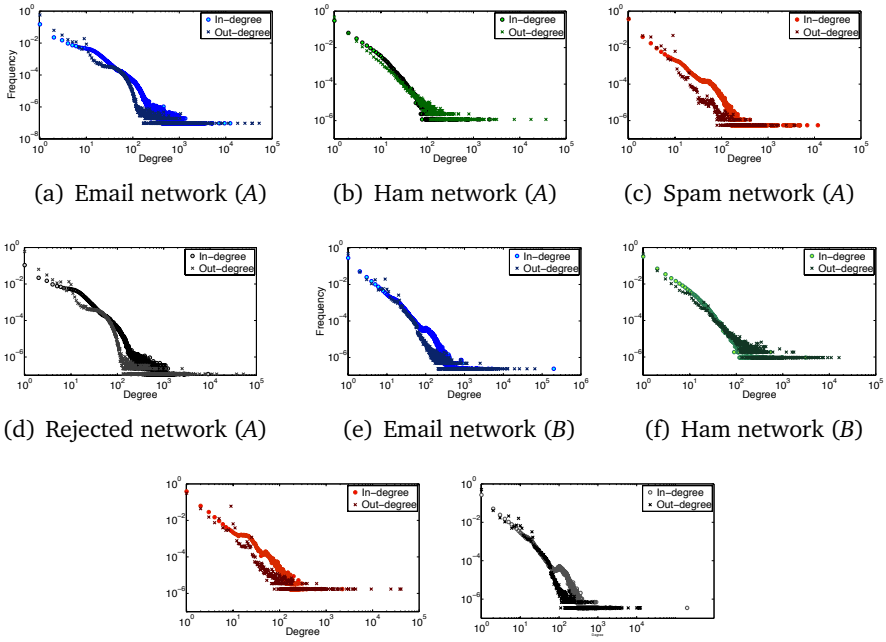
Figure 6.2(a) and 6.2(e) show that the shape of the degree distributions of the complete email networks may change over time as the networks grow. The shape of the degree distribution of spam and rejected networks can also change over time (Figures 6.2(c), 6.2(g), 6.2(d), and 6.2(h)). However, the ham networks always follow a power law distribution with an almost constant exponent (Figures 6.2(b) and 6.2(f)).

Clustering Coefficient. The observed average clustering coefficients for our ham (spam) networks generated from both dataset are quite similar: $C_{A_{ham}} = 9.92 \times 10^{-3}$ ($C_{A_{spam}} = 1.59 \times 10^{-3}$) and $C_{B_{ham}} = 9.80 \times 10^{-3}$ ($C_{B_{spam}} = 1.54 \times 10^{-3}$). These values, similar to small-world networks, are significantly greater than that of random networks with the same number of nodes and average number of edges per node, and as Figures 6.3(b) and 6.3(f) show they remain relatively constant as the networks grow.

Average Path Length. The ham and spam networks generated from both datasets have short average path lengths, $\langle l \rangle$, as expected in small-world networks: $\langle l_{ham_A} \rangle = 7.0$, $\langle l_{spam_A} \rangle = 8.5$, $\langle l_{ham_B} \rangle = 6.7$, and $\langle l_{spam_B} \rangle = 7.8$. Figures 6.3(a) and 6.3(e) show that $\langle l \rangle$ decreases for all networks as they grow, confirming the shrinking diameter phenomenon observed in [40] for other social networks.

Connected Components. Figure 6.1.2.2 shows the distribution of the size of the CCs for ham and spam networks. It can be seen that the GCCs of the networks are orders of magnitude larger than other CCs. The distribution

[3] The power law fits were calculated using the Maximum Likelihood estimator for power law and Kolmogorov-Smirnov (KS) goodness-of-fit as presented in [15].



(a) Email network (*A*) (b) Ham network (*A*) (c) Spam network (*A*)

(d) Rejected network (*A*) (e) Email network (*B*) (f) Ham network (*B*)

(g) Spam network (*B*) (h) Rejected network (*B*)

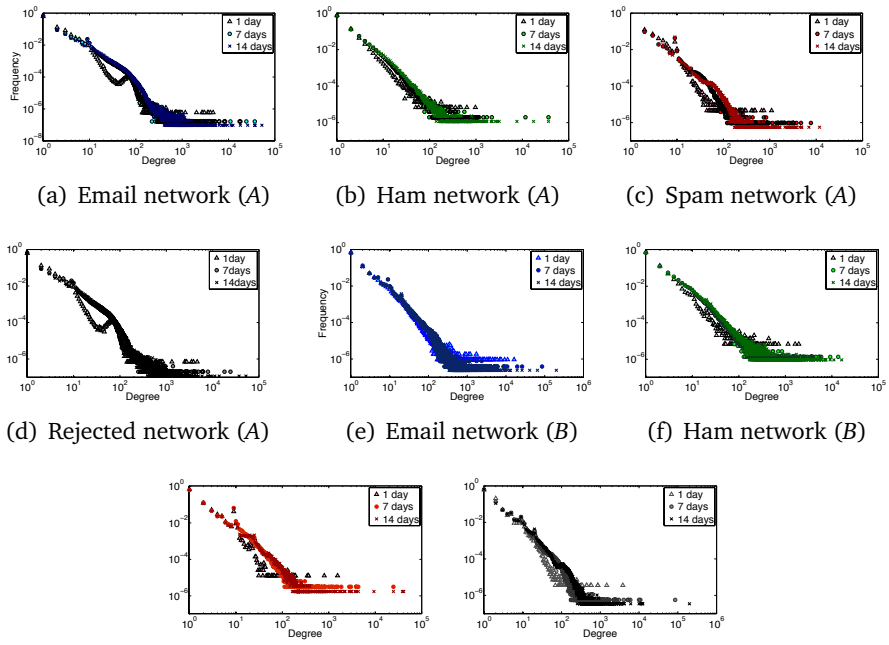
Figure 6.1: Only the ham network is scale free as the other networks have outliers in their degree distribution.

of the CC size for the ham network, similar to Web [13] and phone call graphs [52], follows a power law pattern, but the spam network can have outliers in their distribution. Figures 6.3(d) and 6.3(h) show that the number of CCs in all of the ham and the spam networks increases over time, but this increase is much faster for spam. Moreover, as shown in Figure 6.3(c), the respective size of the GCC of the networks generated from dataset *A* increases for the ham but does not change much for the spam network. However, although the ham network generated from dataset *B* shows exactly the same behavior (Figure 6.3(g)), the spam network shows an increase in the percentage of nodes in its GCC over time.

6.1.2.2 Observations

In the following paragraphs we briefly discuss our observations regarding the structure of email networks and discuss to what extent our dataset is representative for the structural and temporal analysis of email networks.

Although the studied datasets differ in size and collection time, our observations reveal that legitimate email always exhibit the structural properties that are similar to other social and interaction networks. Previous studies on the structure of legitimate email networks have also shown that these networks can be modeled as scale free, small-world networks [17, 40,



(a) Email network (*A*) (b) Ham network (*A*) (c) Spam network (*A*)

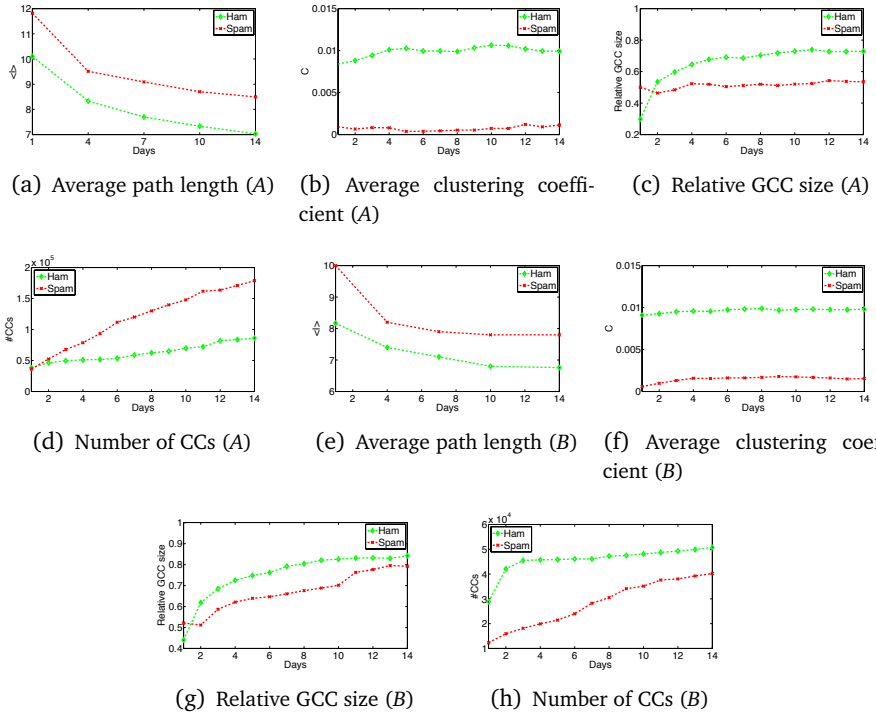
(d) Rejected network (*A*) (e) Email network (*B*) (f) Ham network (*B*)

(g) Spam network (*B*) (h) Rejected network (*B*)

Figure 6.2: Temporal variation in the degree distribution of email networks.

36, 12, 24]. In contrast, a vast majority of spam are automatically sent, typically from botnets, and it is expected that they show unsocial behavior. We have shown that the structural and temporal properties of spam networks can reveal their anomalous nature. Although spam networks show some properties that are similar to ham (i.e., small-world network properties), they can still be distinguished from ham networks as they have significantly smaller average clustering coefficient and larger average path length, regardless of the size of the networks. Overall, we have shown that although the behavior of spam might change over time, its unsocial behavior is not hidden in the mixture of email traffic, even when the amount of spam is less than ham (dataset *B*).

The datasets used in this study to analyze the characteristics of spam do not contain the email communications that do not pass the measurement location. Due to asymmetric routing and load-balancing policies deployed by the network routers, not all traffic travels the link, and less traffic is seen in the outgoing than the incoming direction of the link. However, our goal is to perform a comparative analysis of the distinguishing behavior of spam and ham traffic that are observed over the link. Therefore, it is not required to generate a complete email network of all exchanged emails to be able to study the differences in the social network properties of legitimate and spam traffic.



(a) Average path length (*A*) (b) Average clustering coefficient (*A*) (c) Relative GCC size (*A*)

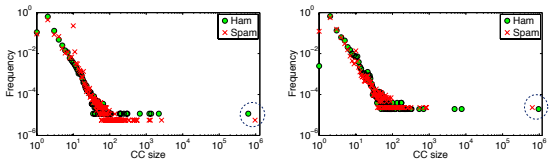
(d) Number of CCs (*A*) (e) Average path length (*B*) (f) Average clustering coefficient (*B*)

(g) Relative GCC size (*B*) (h) Number of CCs (*B*)

Figure 6.3: Both networks are small-world networks (a,b,e,f), however, ham has a higher average clustering coefficient. The ham networks become more connected over time (c,g), and the number of CCs increases faster for the spam networks (d,h).

In addition, the "missing past" problem, which is not limited to our dataset, exists since it is not possible to gather data reaching all the way back to a network's birth. Leskovec et al. [40] showed that the effect of missing past is minor as we move away from the beginning of the data observation. We investigated the effect of missing past by constructing an email network which lacked the first week of data from dataset *A* and comparing it with the network containing both weeks. We have observed that the structural properties of the email networks was relatively similar for both of the networks particularly for the legitimate email.

Earlier studies [17, 36, 24, 12, 86, 37] have also used incomplete email networks to study the structure of email networks or to deploy a social network-based approach to mitigate spam. Even though our measurement duration was shorter than previous studies [17, 40, 24, 36], we have generated the largest and most general datasets used for this type of analysis. The 14 days of data collection might not be large enough to study the evolution of email networks, but our analysis of the temporal variation in the structure of email networks provides us with evidence on how their structure might change with longer periods of measurements.



(a) CC distribution (*A*) (b) CC distribution (*B*)

Figure 6.4: The distribution of size of CCs. The GCCs of the networks are orders of magnitude larger than other CCs.

Overall, this work has provided us with very large datasets of real traffic traversing a high speed Internet backbone link. These datasets allow us to model the behavior of email traffic as observed from the vantage point of a network device on the link and reveal the differences in the network-level behavior of ham and spam traffic.

6.1.3 Anomalies in Email Network Structure

The structural properties of real networks that deviate from the expected properties for social networks, suggest anomalous behavior in the network [1]. In this section, we show that the anomalies caused by the unsocial behavior of spam can be detected in the email networks by using an outlier detection mechanism.

We have shown in Section 6.1.2 that the ham networks exhibit power law out-degree distributions with $\gamma_{A_{out}=2.3}$ and $\gamma_{B_{out}=2.1}$ for dataset *A* and *B* respectively. The outliers in the out-degree distribution of the email networks are of particular importance, as we are interested in finding the nodes that send spam.

We detect outliers from the out-degree distribution in the following steps: First we calculate the ratio of the out-degree distribution of the email network, containing both ham and spam, and our model. Then we deploy the Median Absolute Deviation (MAD) method to calculate the median of the absolute differences of the obtained ratios from their median. Finally, we mark the nodes in the network that have an out-degree that deviates a lot (based on a threshold value) from the median as outliers.

Table 6.1.3 shows the percentage of spam that were sent in different networks and the percentage of spam sent by the identified outlier nodes. The nodes in the email networks generated from dataset *A* (*B*) have sent in average around 70% (40%) spam and the identified outlier nodes have sent just slightly more spam than the average node. The reason is that the outlier detection method tends to mark both nodes that have sent only one email and those that have sent a large number of email as outliers. However, we have observed that the nodes which have sent only one email had sent ham and spam with the same probability, and the nodes with high out-degree have

Table 6.1: Percentage of total spam, spam sent by all the identified outlier nodes, and those with degree between one and 100, in email networks containing both ham and spam.

Dataset	Network	Total spam	Spam sent by outliers	Spam sent by outliers with $1 < k < 100$
A	1 day	68%	69.9%	95.5%
	7 days	70%	74.0%	96.8%
	14 days	70%	74.8%	96.9%
B	1 day	40%	43.6%	82.7%
	7 days	35%	42.8 %	81.3%
	14 days	39 %	46.7%	87.3%

mostly sent legitimate email (e.g., mailing lists). By excluding the nodes that have a high out-degree (100 in our experiments) from the outliers as well as the nodes that have only sent one email during the collection period, we can see that more than 95% (81%) of the email sent by the identified outliers in dataset *A* (*B*) have actually been spam. Moreover, these nodes have actually sent around 25% (35%) of the total spam in the network.

The outliers in the out-degree distribution of the complete email network which in addition to ham and spam contains rejected email can be identified similarly. As an example, the nodes in the complete email network generated from one day of email traffic in dataset *A* have sent in average 94.8% spam and rejected email. The emails sent by the outlier nodes detected by our method have been 99.3% spam or rejected.

6.1.4 Discussion

In this study we have investigated the social network properties of email networks to study the characteristics of legitimate and unsolicited emails. The email networks were generated from real email traffic which was captured on an Internet backbone link. We have analyzed the structural and temporal properties of the email networks and have shown that legitimate email traffic generates a small-world, scale-free network that can be modeled similar to many other social networks. Moreover, the unsocial behavior of spam, which might change over time, is not hidden in the mixture of email traffic. Therefore, email networks that contain spam do not exhibit all properties commonly present in social networks.

Moreover, we have shown that by identifying the anomalies in the structural properties of email networks, it is possible to reveal a number of abusive nodes in the network. More specifically, we have shown that the outliers in the out-degree distribution of email networks to a large extent represent the spamming nodes in the network. Therefore, the social network properties of email networks can potentially be used to detect malicious hosts on the network.

6.2 Emails in the Gray Area

Nowadays, many antispam filters provide a good level of protection against large-scale unsolicited email campaigns. However, as spammers have improved their techniques to increase the chances of reaching their targets, also antispam solutions have become more aggressive in flagging suspicious emails. On one side, this arms race has lead to a steady increase in the detection rate. On the other, it also contributed to the increase of the false positives, with serious consequences for the users whenever an important message is erroneously flagged as spam. It is a well known fact that most of the users regularly check their spam folder to verify that no important messages have been misclassified by the antispam filter.

Unfortunately, this process is very time-consuming and error-prone. Antispam solutions are not very helpful in this direction, and do not usually provide any additional information to help users in quickly identifying marketing emails, newsletters, or "borderline" cases that may be interesting for the users. While most of the existing research deals with the problem of efficiently and accurately distinguishing spam from ham, we focus on the thin line that separates the two categories. In particular, we limit our study to the often overlooked area of gray emails [93], i.e., those ambiguous messages that cannot be clearly categorized one way or the other by automated spam filters.

For a more detailed version of this study, please refer to Isacenkova et al. [27].

6.2.1 Approach

We start our study by analyzing a real deployment of a challenge-response antispam solution to measure the extent of this gray area. We use system's quarantined emails that already exclude the majority of the ham and spam messages as an approximation of the gray emails category. We analyze the messages further in order to improve our understanding about them and the reasons that make them difficult to categorize. In particular, we adopt a three-phase approach based on message clustering, classification, and graph-based refinement. Extracted email features are applied in a context of email campaigns instead of individual emails. In particular, we start by clustering emails based on the email headers. We then extract a set of features based on a number of campaign attributes and we use them to train a classifier in order to predict the campaign class. Finally, we employ a graph-based refinement technique to further increase the coverage and precision of our classification.

Data Collection. The amount and diversity of the available data is crucial in order to successfully identify email campaigns. Messages should be

collected from multiple feeds, cover numerous recipients, several organizations, and for a long period of time [56, 60]. Our email dataset fulfills these requirements as it was collected from a commercial Challenge-Response (CR) spam system deployed in tens of different organizations. A CR filter is a software that automatically replies with a challenge (in our case a CAPTCHA) to any previously-unknown sender of incoming emails. If the sender solves the challenge, the message is delivered to the recipient and the sender is added to a whitelist; if not, it remains in a quarantined folder, where its recipient can manually view and whitelist/blacklist it.

The monitoring period covered 6 months, from August 2011 to January 2012. During this period around 11 million messages were delivered to the monitored mail servers. 29.4% of them belonged to the class of gray messages. To protect the privacy of both the users and the companies involved in the study, the data we used in our experiments did not include the email bodies, and the headers were sanitized and analyzed in an aggregated form.

We also instrumented the CR-system to collect additional information: opened emails by the users, and whitelisted messages (thus showing that the user manually classified them as legitimate). This provides insights on the users ability to distinguish harmless from harmful messages. Finally, our sensor collected the delivery status information, e.g. sent, bounced, and delivered, for each challenge email sent back by the CR system.

Email Clustering. Previous results were very successful in identifying email campaigns, but, unfortunately, often relied on the content of the email body. Our dataset is limited to the email headers, thus forcing us to use a different approach based only on the email subjects. The main limitation of this technique is that the email subjects have to be long enough to minimize the chances of matching different messages by coincidence.

We decided to use a simple approach for grouping similar subjects based on "almost exact" text matching, extended to include subjects with a variable part. The latter could be a varying phrase in the subject, including random words, identifiers, or user names.

Feature-based Classification. To be able to differentiate and classify the identified clusters, we extract a set of eleven features grouped in three categories (see Table 6.2).

Before performing our classification, we build a training set by randomly selecting 2,000 campaigns and performing a manual labeling of them. We labeled 1,581 (79%) as legitimate and 419 (21%) as spam campaigns. We take a conservative approach, and flag as spam only campaigns with potentially illegal content that may involve malicious, fraudulent or illegal online activities. This includes different "business models": illegal product sellers, malware spreading emails, personal data and credential thieves, or advanced fee fraud specialists. Finally, we consider any email belonging to a commercial marketing campaign as legitimate (in the sense that general an-

Table 6.2: Cluster features.

Group A	
Sender IPs	Distribution of network prefixes (/24)
Sender names	Distribution of email sender names
Sender add.domain	Distribution of email domain names
Sender add.prefix	Distribution of email prefixes
Group B	
Rejections	Percentage of rejected emails at MTA
White emails	Percentage of whitelisted emails
Challenges bounced	Percentage of bounced challenges
CAPTCHAs solved	Percentage of solved challenges
Unsubscribe header	Percentage of Unsubscribe headers
Group C	
Number of recipients per email	Normalized number of unique recipients per email
Recipient's header	Location of recipient's email: To/Cc/Bcc/Mixed
Countries	Distribution of countries based on originating IPs

Table 6.3: Campaign classification results.

Campaign Type	**Manual Sampling**	**%**	**Unlabeled**	**%**
Legitimate	1,581	79%	8,398	81.9%
Spam	419	21%	1,852	18.1%
Total	2,000		10,250	

tispam filters should not block them, unless they are specifically instructed to do so by the user).

Using the eleven listed presented above, we trained a binary classifier. Finally, we applied the model extracted from our training set to predict the classification of the remaining unlabeled campaigns. Results are presented in Table 6.3.

Graph-based Refinement. Although we achieved a relatively high accuracy using our classifier, we still found that for some campaigns our algorithm gave uncertain results.

With the use of several graph-based techniques, detailed in Isacenkova et al. [27], we were able to reduce the false positives from 0.9% to 0.2% and split the entire dataset into three groups: legitimate (80%), spam (17%) and gray (2.9%) messages.

6.2.2 Attribute Analysis

In this section we analyze the characteristics of spam and legitimate campaigns. Our classifier provides some information about the relevance of each feature. Interestingly, the least important attributes are the ones in Group B,

and in particular the percentage of already whitelisted emails in the cluster. The most important ones are the distributions of countries and IP addresses, followed by the average number of recipients, and the sender email address similarity. The latter proved to be useful because spammers often change sender emails, while legitimate campaigns use a single or several repetitive patterns. In particular, we found the number of originating countries to be the most indicative parameter.

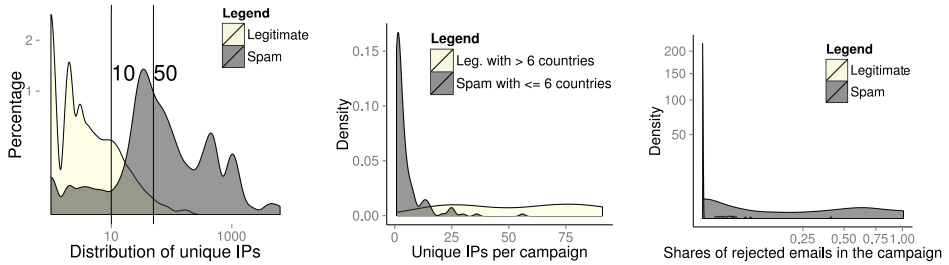
The Role of IPs and Geolocation. IP address variation is often regarded as a strong indicator of botnet activity and often used as a reliable metric to detect spam. However, it is unclear what should be adopted as a threshold for this metric, how many different IPs should alert us of a distributed malicious activity, or how accurately we can classify email campaigns simply by looking at their IP address distribution.

In a previous study of spam campaigns, Qian et al. [65] used a threshold of 10 IPs per campaign to separate spam campaigns from legitimate ones. To evaluate this threshold, we applied it on our gray dataset as shown in Figure 6.5 (a). The graph plots the distribution of unique IP prefixes for both spam and legitimate campaigns. Around 90% of the legitimate campaigns are indeed below the 10 IP threshold, while 90% of the spam is above - resulting in a global error rate of 9.2%. In comparison, this error is 5 times higher than the one of our classifier.

By looking at Figure 6.5 (a), we notice that above 50 IP prefixes there are few legitimate campaigns left and 99.8% of legitimate campaigns are below this threshold. However, half of the spam campaigns are located above the threshold and another half in between the two thresholds (10-50). This suggests that there is not a single value that separates the two classes with an acceptable error rate.

When we look at IP country distribution, the results improve considerably as some legitimate campaigns have many IP prefixes, but originate from few countries. This could be explained by one commercial campaign being distributed by several marketing companies in different locations. In contrast, the vast majority of spam campaigns originate from multiple IP prefixes *and* multiple countries. In fact, by using a six-countries threshold (the one chosen by our classifier) we misclassify only 0.4% of legitimate and 12% of spam campaigns - resulting in a total error rate of 2.8%. Figure 6.5 (b) shows the classification error.

Finally, we investigate closer this group of spam campaigns with few origins. Interestingly, the classifier for most of them gave a weak score. At a closer manual inspection, these cases mainly corresponded to phishing and Nigerian scams. Several of these campaigns are sent in low volume and for short periods of time using webmail accounts, thus hiding under benign IP addresses.



(a) Logarithmic plot of unique IP prefix distribution

(b) Campaigns with a threshold of 6 countries

(c) Emails rejections per campaign

Figure 6.5: Attribute distributions in campaigns.

Recipient-Oriented Attributes. The email recipient can be specified in three different headers: *To*, *Cc*, and *Bcc*. Interestingly, we found no campaigns using the *Cc* header, and some campaigns that seem to randomly change the location of the recipient over time (we categorize them as *Mixed*). We also looked at the number of recipients per incoming email and at the number of non-existing email accounts (rejected at MTA-in because of non-existent user) in multiple recipient emails.

Around 75% of the legitimate campaigns use the *To* header, whereas spammers often mix different headers in the same campaign. The *Bcc* header is adopted by both campaign types, although less frequently. However, it is very common among gray campaigns: in fact, half of them use exclusively this header to specify the recipient. Again, this is very common between the previously mentioned scam campaigns.

Since the campaigns located in the gray zone often use the *Bcc* field, they have shorter recipient lists including on average only 1.2 recipients per email. In contrast, 94% of legitimate campaigns have a single recipient, while spammers tend to include an average of at least three recipients per email.

However, these features alone cannot be used to reliably separate spam from legitimate messages. For example, 36% of spam campaigns used only one recipient per email, and in 30% of the cases specified in the *To* header. Interestingly, by combining these two criteria with the fact that these campaigns also have high IP prefix distribution, we can deduct that they originate from infected machines or botnets.

When some of the messages in a campaign are rejected, it is an indicator that the sender's recipient list was unverified or not up-to-date. Although sometimes users make typos while providing their email addresses, a higher rejection ratio, as shown in Figure 6.5 (c), along with multiple recipients is a good indicator of spammer activity. In fact, only 1% of spam campaigns sent with two recipients per email have a rejection ratio lower than 0.1. Thus,

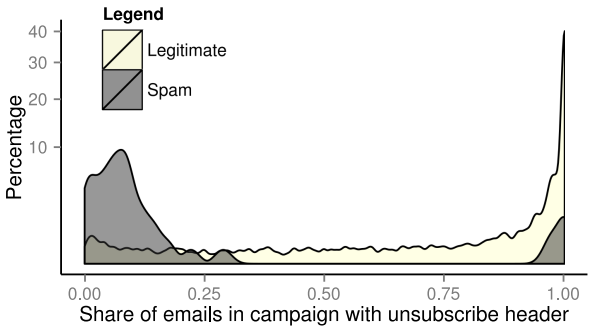


Figure 6.6: Newsletter subscription header distribution.

the combination of these two characteristics performs well for campaign classification.

Newsletter Subscription Header. One of our features counts the presence of the *List-Unsubscribe* header in the emails. This header is intended specifically to indicate bulk email senders in order to treat such emails separately, and normally points to a URL or email address that can be used to unsubscribe from a mailing list. This header is recommended to be used by regular bulk senders.

Figure 6.6 shows the percentage of each campaign type that uses the unsubscribe header. Only 10% of the spam campaigns adopt the header, counting only for a total of 0.6% of the spam messages. While legitimate campaigns tend to use the header in most of their emails, around half of the campaigns do not use it at all. This is due to several different email marketing companies advertising the same campaign, where some include the header, and some do not. In total, around half of the legitimate campaigns include the header, and 27% of all legitimate campaigns have the header present in all messages.

In conclusion, we find it uncommon for spammers to use the *Unsubscribe* header, but at the same time legitimate campaigns use it in only half of their emails. While this attribute seems to be a good feature to identify marketing campaigns, spoofing the *Unsubscribe* header is extremely easy and could be done with minimal additional costs for spammers.

6.2.3 Email Campaigns

In this section we present four categories of email campaigns that we identify in the gray area. We already separated spam from legitimate campaigns. We further divide the spam in two categories: the one generated by distributed and dynamic infrastructures (likely sent by botnet or infected machines) from the smaller campaigns sent by few IPs.

We also split the legitimate campaigns into two groups. The first sent by private marketing companies as a service to distributes legitimate bulk advertisements, i.e., commercial campaigns. The second including newsletters that are sent to the users subscribed to a web services or mailing lists, and the automatically generated notifications (e.g. for online registrations). Again, the first ones are delivered by large infrastructures, while the second ones are often sent from a limited and constant set of IP addresses.

Commercial Campaigns. This is the largest category in our dataset covering 42% of the identified campaigns, with an average of 148 emails each. By looking manually at these clusters, we confirm that these messages are mainly generated by professional email marketers sending. We were able to identify some of the main players (both national and international), and often confirmed that they actually run a legal business. On their websites, they repeatedly underline the fact that "they are not spammers", and that they just provide to other companies a way to send marketing emails within the boundaries of the current legislation. In fact, they also offer an online procedure for users to opt-out and be removed from future communications. These companies also use wide IP address ranges to run the campaigns, probably to avoid being blacklisted. Moreover, we find quite interesting that some of these companies also provide a pre-compiled list of emails (already categorized by user interests) that can be used to acquire new clients.

On average, this class of campaigns lasts for 26 days, but some also continue for several months. Different email marketing companies are often involved in sending a single campaign, where each company is only active during a certain time frame. Also, each marketing service provider has its own dedicated range of IP addresses, which explains sometimes high IP address variance and high geographical distribution of campaigns in this group.

To conclude, commercial campaigns can be highly distributed, but, at the same time, they often adopt consistent email patterns with similar sender names and email addresses.

Newsletter Campaigns. The newsletter senders rely mostly on static and small mailing infrastructure. The sender is often the actual company distributing the emails, with typically a small and fixed IP address range. This category contains half of the emails of the previous one (probably because most of the legitimate mailing lists do not get into the quarantined area as they are already whitelisted by their customers) and covers around 30% of the total campaigns with an average size of 90 emails each.

A manual inspection seems to confirm that these campaigns consist mainly of notifications and newsletters sent by online services to which users have subscribed in the past. The senders are geographically localized (we encountered only one exception of a distributed newsletter campaign) and have extremely consistent sending patterns. Since we cluster campaigns

based on their subjects, newsletters tend to last for very short periods of time. In addition, they normally use valid email recipient lists, and exhibit the lowest IP address, country, and sender email address variations. The consistent patterns in the email headers of this category indicate that the senders are making an effort to build a reputation and successfully deliver their correspondence. Not surprisingly, this is also the category that is whitelisted most often by the users.

Botnet-Generated Campaigns. Unsurprisingly, botnet-generated campaigns have highly dynamic attribute values, making them the easiest category to identify automatically. This category contains clusters that accounts for only 17% of all campaigns (also because most of the spam emails were already excluded from the gray emails by other antispam filters). Botnet campaigns have the highest geographical distribution as they are sent by infected computers from all over the world: 172 unique /24 networks per campaign, spread on average over 28 countries. Another prevalent characteristic is the use of multiple recipient emails sent using unverified email lists. Consequently, this leads to the highest email rejection rates (24%), and highest bounced CAPTCHA requests. The *Unsubscribe* header is rarely used, and sender email addresses have low similarities.

On average, botnet campaigns are the ones lasting the longest, with one drug-related campaign sent slowly over the entire six-months period of our experiments.

Despite the easily recognizable characteristics of these campaigns, users show a surprisingly high interest in these emails. This category has the highest number of email views per campaign, suggesting that users are often curious about products promoted and sold on the black market [46].

Scam and Phishing Campaigns. These campaigns contain phishing and Nigerian scam emails. Fraudsters trick their victims using threatening messages or by trying to seduce them with huge monetary gains. The characteristics of this category largely resemble those of commercial campaigns, thus making it difficult to automatically separate these campaigns without analyzing the email body. In fact, most of these campaigns belong to the gray area of our classifier. This is the reason why we needed to verify this set manually. These kind of threats are more likely to be identified by content-based detection techniques, e.g., by looking at email addresses and phone numbers [28], or URL [56, 84] included in the body.

We found only 12,601 of such emails, with an average campaign size of 84 emails. Phishing campaigns often spoofed the email addresses using well known company names (e.g. banks, eBay, Paypal), whereas Nigerian scammers relied mostly on webmail accounts [28]. In this case, many senders solved the CAPTCHA challenge – confirming that there is usually a real person behind these kinds of scams. The IP addresses from where the CAPTCHAs were solved are mostly located in West-African countries, like

Nigeria or Ivory Coast. None of the messages in this category include an *Unsubscribe* header.

Unfortunately, users seemed to often fall victims to this type of attack, as they opened and even whitelisted messages in these campaigns.

6.2.4 User Behavior

Our dataset also provides information about which actions were performed by the users on the quarantined emails. In particular, we collected information regarding the messages that were read, added to a user whitelist or blacklist, and the CAPTCHA that was later solved by the sender. These data can give us some useful insights on the ability of average users to identify suspicious emails.

Table 6.4 presents three user action statistics. As expected, user activity involves mainly legitimate and gray campaigns. In fact, the main reason for users to go through the emails in this folder is to spot missed notifications or undelivered benign messages. However, a large fraction of users also opened spam messages, maybe attracted by some deceiving subjects. The highest campaign viewing rates are produced by botnet-generated campaigns, overpassing even newsletters. Over 3,888 spam emails were viewed by users during our six-month experiments, resulting in the fact that *one out of five users* has viewed at least one spam message, and, on average, *opened 5 of them*.

After a manual inspection of botnet-generated campaigns where the emails were read and whitelisted, we confirmed that those campaigns were promoting illegal products, e.g. drugs and pirated software. This may suggest two things: either users have problems in distinguishing legitimate emails from harmful, or some users are genuinely interested in the products promoted by spammers. It is difficult to draw conclusions as both hypotheses might be true for different users, but, clearly, most of them are unaware of the security threats involved in opening malicious emails.

Meanwhile, we should compare the reported statistics of viewed emails with the number of emails that actually got whitelisted – an action that could be interpreted as the equivalent of clicking the “Not Spam” button provided by several webmail services. The number of whitelisted emails per botnet-generated campaign (1.26 emails) is the lowest among all the categories, suggesting that most users successfully differentiate them. However, we notice that scam/phishing campaigns have almost the same number of emails being whitelisted per campaign as commercial campaigns (2.25 vs 2.9). This suggests that users might have difficulties in differentiating these categories. It is important to remember that this category was manually sampled by domain experts, which is not the case for the typical users as most of them are untrained and are more likely to fall for these kind of fraud.

Table 6.4: User actions performed on campaigns.

	Viewed	**Whitelisted**	**CAPTCHA solved**
Legitimate	42%	12%	3.5%
Spam	25%	6%	0.2%
Gray	40%	17%	10%

To further measure how significant this phenomenon is, we compute that there is a 0.36% probability that a certain user whitelists a legitimate email and 0.0005% that she whitelists a spam message. These numbers may seem low, but they rapidly increase when multiplied by the number of users and the number of messages received. In total, an average of 3.9 emails get whitelisted per legitimate campaign compared to 1.1 emails per spam campaign.

To conclude, user-generated actions on gray emails are erroneous and thus are inaccurate to use for prediction. They often open even potentially dangerous emails, ignoring security risks.

6.2.5 Discussion

We presented a system to identify and classify campaigns of gray emails. As an approximation of this set, we chose to use the quarantined folder of a challenge-response antispam filter, since it is already clean from obvious spam and ham messages.

Our analysis unveiled the most and the least predictive email campaign class attributes. We also demonstrated that previous techniques used for email campaign classification [65] did not provide acceptable results in our settings, confirming that the gray area contains the hardest messages to classify. Additionally, we confirmed and extended some of the findings of previous studies regarding botnet campaigns [56].

Our system could be used in different ways. First of all, it can help understanding how large commercial campaigns work, how they originate, and how they differ from other unsolicited emails. It could also serve as an input to automatically place marketing campaigns and newsletters in a separate folder, so that users can clearly differentiate these messages from other forms of spam. In fact, the users in our study often opened botnet-generated emails and were especially prone to errors when dealing with scam and phishing messages; we believe that a separate folder dedicated to legitimate bulk emails would create an extra layer between the users and the malicious messages, thus allowing users to focus on the bulk folder when looking for missing and misclassified emails. Interestingly, after we completed our study, a similar solution was deployed by GMail [26], to place user newsletters, notifications, and other commercial email into distinctive categories.

7 Conclusions

With this final deliverable we provided an overview of the most interesting research on malware and fraud topics. Both areas are quite hard to separate since one often depends on the other. Fraudulent activity, for instance, almost always relies on some kind of malware which needs to be installed at the victim's host. Still, the areas we covered are far from complete. There are a lot of interesting other topics ranging from advanced malware analysis over detailed botnet takedowns to online banking fraud. Covering them in a single deliverable, however, is not feasible. Furthermore, we covered only technological aspects of the two worlds. There are just as many sociological and human aspects connected to fraud alone. In order not to drop them completely, they are discussed in the final roadmap, which is deliverable D4.4.

Here we also cover another important aspect of work package 5: The expert meetings. All the input that led to this deliverable originally stems from various publications created by members or collaborations of the consortium. The working group meetings were held with the goal to get input beyond the consortium. And it worked surprisingly well. These discussions resulted in ideas about future research directions but also brought existing work to attention. In the case of working group 5 (working group on malware and fraud), for example, we were provided with insight about banking fraud but from the moment when an account is already compromised. These insights inspired new ideas on where to disrupt the scamming scheme and not necessarily on the technological side. Again, a more detailed discussion is provided in the final roadmap.

These insights were exactly what we were targeting with the SysSec project. Building a network of qualified and knowledgeable people from various fields, to give ideas and discuss experiences in their respective topic.

Bibliography

[1] L. Akoglu, M. McGlohon, and C. Faloutsos. OddBall: Spotting Anomalies in Weighted Graphs. In *PAKDD*, 2010.

[2] P. W. e. al. MessageLabs Intelligence: 2010 Annual Security Report. Technical report, Symantec, 2010.

[3] D. Antoniades, E. Athanasopoulos, I. Polakis, S. Ioannidis, T. Karagiannis, G. Kontaxis, and E. P. Markatos. we.b: The web of short URLs. In *WWW '11*, 2011.

[4] P. Baecher and M. Koetter. libemu, 2009. http://libemu.carnivore.it/.

[5] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated Classification and Analysis of Internet Malware. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2007.

[6] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna. Efficient Detection of Split Personalities in Malware. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS)*, 2010.

[7] P. Bania. Evading network-level emulation, 2009. http://piotrbania.com/all/articles/pbania-evading- nemu2009.pdf.

[8] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel. A View on Current Malware Behaviors. In *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[9] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR) Annual Conference*, 2006.

[10] U. Bayer and F. Nentwich. Anubis: Analyzing Unknown Binaries, 2009. http://anubis.iseclab.org/.

[11] F. Bellard. QEMU, a Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference*, 2005.

[12] P. O. Boykin and V. P. Roychowdhury. Leveraging social networks to fight spam. *Computer*, 38(4), 2005.

[13] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer Networks*, 33(1-6), 2000.

[14] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In *Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks (DSN)*, 2008.

[15] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Reviews*, June 2007.

[16] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2008.

[17] H. Ebel, L. Mielsch, and S. Bornholdt. Scale-free topology of e-mail networks. *Physical Review E*, 66, 2002.

[18] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In *Proceedings of the 6th international conference on Detection of Intrusions and Malware, & Vulnerability Assessment (DIMVA)*, 2009.

[19] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29, 1999.

[20] P. Ferrie. Attacks on Virtual Machine Emulators. Technical report, Symantec Research White Paper, 2006.

[21] P. Ferrie. Attacks on More Virtual Machines, 2007.

[22] S. Ford, M. Cova, C. Kruegel, and G. Vigna. Wepawet, 2009. http://wepawet.cs.ucsb.edu/.

[23] T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is Not Transparency: VMM Detection Myths and Realities. In *Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI)*, 2007.

[24] L. H. Gomes, R. B. Almeida, L. M. A. Bettencourt, V. Almeida, and J. M. Almeida. Comparative graph theoretical characterization of networks of spam and legitimate email. In *Proc. of the Conference on Email and Anti-Spam*, 2005.

[25] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *CCS '10*, pages 27–37, New York, NY, USA, 2010. ACM.

[26] G. inc. Inbox tabs and category labels. http://gmailblog.blogspot.fr/2013/05/a-new-inbox-that-puts-you-back-in.html.

[27] J. Isacenkova and D. Balzarotti. Shades of gray: A closer look at emails in the gray area. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2014.

[28] J. Isacenkova, O. Thonnard, A. Costin, D. Balzarotti, and A. Francillion. Inside the scam jungle: A closer look at 419 scam email operations. *IWCC*, 2013.

[29] P. Jaccard. The Distribution of Flora in the Alpine Zone. *The New Phytologist*, 11(2), 1912.

[30] N. M. Johnson, J. Caballero, K. Z. Chen, S. McCamant, P. Poosankam, D. Reynaud, and D. Song. Differential Slicing: Identifying Causal Execution Differences for Security Applications. In *IEEE Symposium on Security and Privacy (2011)*, 2011.

[31] V. Kamluk. A black hat loses control. http://www.securelist.com/en/weblog?weblogid=208187881, 2009.

[32] M. G. Kang, P. Poosankam, and H. Yin. Renovo: A Hidden Code Extractor for Packed Executables. In *ACM Workshop on Recurring Malcode (WORM)*, 2007.

[33] M. G. Kang, H. Yin, S. Hanna, S. McCamant, and D. Song. Emulating Emulation-Resistant Malware. In *Proceedings of the 2nd Workshop on Virtual Machine Security (VMSec)*, 2009.

[34] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: an empirical analysis of spam marketing conversion. *Proc. of the ACM conf. on computer and communications security*, 52(9), 2009.

[35] P. Kleissner. Antivirus Tracker. http://avtracker.info/, 2009.

[36] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757), 2006.

[37] H. Lam and D. Yeung. A learning approach to spam detection based on social networks. In *Proceedings of the Conference on Email and Anti-Spam*, 2007.

[38] B. Lau and V. Svajcer. Measuring virtual machine detection in malware using DSD tracer. *Journal in Computer Virology*, 6(3), 2010.

[39] S. Lee and J. Kim. WarningBird: Detecting Suspicious URLs in Twitter Stream. In *NDSS '12*, 2012.

[40] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery Data*, 1(1), 2007.

[41] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti. Detecting environment-sensitive malware. In *Recent Advances in Intrusion Detection*, pages 338–357. Springer, 2011.

[42] B. Livshits. Finding malware on a web scale. *Computer Network Security*, 2012.

[43] F. Maggi, A. Frossi, S. Zanero, G. Stringhini, B. Stone-Gross, C. Kruegel, and G. Vigna. Two years of short URLs internet measurement: Security threats and countermeasures. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, 2013.

[44] L. Martignoni, M. Christodorescu, and S. Jha. OmniUnpack: Fast, Generic, and Safe Unpacking of Malware. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2007.

[45] J. Mason, S. Small, F. Monrose, and G. MacManus. English shellcode. In *Proceedings of the 16th ACM conference on Computer and communications security (CCS)*, 2009.

[46] D. McCoy, A. Pitsillidis, G. Jordan, N. Weaver, C. Kreibich, B. Krebs, G. M. Voelker, S. Savage, and K. Levchenko. Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs. *USENIX*, 2012.

[47] D. K. McGrath and M. Gupta. Behind phishing: an examination of phisher modi operandi. In *LEET '08*, pages 4:1–4:8, Berkeley, CA, USA, 2008. USENIX Association.

[48] Metasploit. The metasploit project. http://www.metasploit.com/.

[49] Microsoft. Win32 assembly components, Dec. 2002. http://lsd-pl.net.

[50] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[51] F. Moradi, T. Olovsson, and P. Tsigas. Towards modeling legitimate and unsolicited email traffic using social network properties. In *Proceedings of the Fifth Workshop on Social Network Systems (SNS)*, 2012.

[52] A. A. Nanavati, R. Singh, D. Chakraborty, K. Dasgupta, S. Mukherjea, G. Das, S. Gurumurthy, and A. Joshi. Analyzing the structure and evolution of massive telecom graphs. *IEEE Trans. on Knowledge & Data Engineering*, 20(5), 2008.

[53] Nepenthes. Common shellcode naming initiative, 2009. http://nepenthes.carnivore.it/csni.

[54] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. Giffin, and S. Jha. Automatic Generation of Remediation Procedures for Malware Infections. In *Proceedings of the 19th USENIX Conference on Security*, 2010.

[55] R. Paleari, L. Martignoni, G. F. Roglia, and D. Bruschi. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX Workshop on Offensive Technologies (WOOT)*, 2009.

[56] A. Pathak, F. Qian, Y. C. Hu, Z. M. Mao, and S. Ranjan. Botnet spam campaigns can be long lasting: Evidence, implications, and analysis. *SIGMETRICS*, 2009.

[57] B. B. Pek, G. and and B. L. nEther: In-guest Detection of Out-of-the-guest Malware Analyzers. In *ACM European Workshop on System Security (EUROSEC)*, 2011.

[58] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010.

[59] M. Pietrek. A crash course on the depths of Win32™structured exception handling, 1997. http://www.microsoft.com/msj/0197/exception/exception.aspx.

[60] A. Pitsillidis, C. Kanich, G. M. Voelker, K. Levchenko, and S. Savage. Taster's choice: a comparative analysis of spam feeds. *IMC*, 2012.

[61] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. An empirical study of real-world polymorphic code injection attacks. In *Proceedings of the 2nd USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, April 2009.

[62] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. Comprehensive shellcode detection using runtime heuristics. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 287–296. ACM, 2010.

[63] M. Polychronakis, E. P. Markatos, and K. G. Anagnostakis. Network-level polymorphic shellcode detection using emulation. In *Proceedings of the Third Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2006.

[64] M. Polychronakis, E. P. Markatos, and K. G. Anagnostakis. Emulation-based detection of non-self-contained polymorphic shellcode. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2007.

[65] F. Qian, A. Pathak, Y. C. Hu, Z. M. Mao, and Y. Xie. A case for unsupervised-learning-based spam filtering. *SIGMETRICS*, 2010.

[66] T. Raffetseder, C. Kruegel, and E. Kirda. Detecting System Emulators. In *Information Security Conference (ISC)*, 2007.

[67] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. *SIGCOMM Comput. Commun. Rev.*, 36, 2006.

[68] R. Rasmussen and G. Aaron. Global Phishing Survey: Trends and Domain Name Use in 1H2010. Technical report, APWG, Oct. 2010.

[69] R. Rasmussen and G. Aaron. Global Phishing Survey: Trends and Domain Name Use in 1H2011. Technical report, APWG, Nov. 2011.

[70] R. Rasmussen and G. Aaron. Global Phishing Survey: Trends and Domain Name Use in 1H2012. Technical report, APWG, Oct. 2012.

[71] A. Reka and Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, June 2002.

[72] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. Van Steen, F. C. Freiling, and N. Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, pages 78–88. ACM, 2011.

[73] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen. Prudent practices for designing malware experiments: Status quo and outlook. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 65–79. IEEE, 2012.

[74] J. Rutkowska. Red Pill... or how to detect VMM using (almost) one CPU instruction. http://invisiblethings.org/papers/redpill.html, 2004.

[75] Skape. Understanding windows shellcode, 2003. http://www.hick.org/code/skape/papers/win32-shellcode.pdf.

[76] Skape. Safely searching process virtual address space, 2004. http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf.

[77] SkyLined. SEH GetPC (XP SP3), July 2009. http://skypher.com/wiki/index.php/Hacking/Shellcode/Alphanumeric/ALPHA3/x86/ASCII/Mixedcase/SEH_GetPC_(XP_sp3).

[78] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo. On the infeasibility of modeling polymorphic shellcode. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*, 2007.

[79] B. Stock, J. Gobel, M. Engelberth, F. C. Freiling, and T. Holz. Walowdac-analysis of a peer-to-peer botnet. In *Computer Network Defense (EC2ND), 2009 European Conference on*, pages 13–20. IEEE, 2009.

[80] B. Stone-Gross, R. Abman, R. Kemmerer, C. Kruegel, D. Steigerwald, and G. Vigna. The Underground Economy of Fake Antivirus Software. In *Proceedings of the Workshop on Economics of Information Security (WEIS)*, 2011.

[81] B. Stone-Gross, A. Moser, C. Kruegel, K. Almaroth, and E. Kirda. FIRE: FInding Rogue nEtworks. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2009.

[82] P. Ször. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, February 2005.

[83] C. K. Tan. Defeating Kernel Native API Hookers by Direct Service Dispatch Table Restoration. Technical report, SIG2 G-TEC Lab, 2004.

[84] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. *IEEE Symposium on Security and Privacy*, 2011.

[85] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of the 5th Symposium on Recent Advances in Intrusion Detection (RAID)*, Oct. 2002.

[86] C. Tseng and M. Chen. Incremental SVM model for spam detection on dynamic email social networks. In *Conf. on Computational Science and Engineering*, 2009.

[87] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu. Still: Exploit code detection via static taint and initialization analyses. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, 2008.

[88] X. Wang, C.-C. Pan, P. Liu, and S. Zhu. Sigfree: A signature-free buffer overflow attack blocker. In *Proceedings of the USENIX Security Symposium*, Aug. 2006.

[89] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684), 1998.

[90] B.-J. Wever. SEH Omelet Shellcode, 2009. http://code.google.com/p/w32-seh-omelet-shellcode/.

[91] G. Wicherski. Win32 egg search shellcode, 33 bytes, Feb. 2009. http://blog.oxff.net/2009/02/win32-egg-search- shellcode-33-bytes.html.

[92] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using CWSandbox. *IEEE Security and Privacy*, 5(2):32–39, 2007.

[93] W.-t. Yih, R. McCann, and A. Kolcz. Improving spam filtering by detecting gray mail. *CEAS*, 2007.

[94] K. Yoshioka, Y. Hosobuchi, T. Orii, and T. Matsumoto. Your Sandbox is Blinded: Impact of Decoy Injection to Public Malware Analysis Systems. *Journal of Information Processing*, 19, 2011.

[95] Q. Zhang, D. S. Reeves, P. Ning, and S. P. Lyer. Analyzing network traffic to detect self-decrypting exploit code. In *Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2007.