Programmable Data Gathering for Detecting Stegomalware

Alessandro Carrega
CNIT - S3ITI
alessandro.carrega@cnit.it

Luca Caviglione, Matteo Repetto, Marco Zuppelli
CNR - IMATI
{luca.caviglione, matteo.repetto, marco.zuppelli}@ge.imati.cnr.it

***Abstract*—The “arm race” against malware developers requires to collect a wide variety of performance measurements, for instance to face threats leveraging information hiding and steganography. Unfortunately, this process could be time-consuming, lack of scalability and cause performance degradations within computing and network nodes. Moreover, since the detection of steganographic threats is poorly generalizable, being able to collect attack-independent indicators is of prime importance. To this aim, the paper proposes to take advantage of the extended Berkeley Packet Filter to gather data for detecting stegomalware. To prove the effectiveness of the approach, it also reports some preliminary experimental results obtained as the joint outcome of two H2020 Projects, namely ASTRID and SIMARGL.**

***Index Terms*—eBPF, syscall tracing, stegomalware, covert channels, detection.**

I. Introduction

Modern business models are increasingly demanding for more agility in the creation, operation and management of ICT services. To this aim, new computing paradigms are being progressively introduced, which leverage virtualization and cloud, as well as service-oriented architectures to interconnect software, devices, and data over a pervasive and seamless computing continuum encompassing different technological and administrative domains, i.e., IoT, data centers, and telco infrastructures. Despite of the benefits in terms of time-to-market and dynamicity, novel paradigms outdate the legacy security perimeter model and lead to additional threats [1].

The difficulty to effectively deploy cyber-security appliances in virtualized and cyber-physical systems is making cloud and network services the preferred target for a wide-range of novel attacks. As a matter of fact, the growing adoption of micro-services and service mesh architectures to implement large digital value chains creates interdependencies among software processes in different domains and paves the way for multi-vector attacks that combine together social engineering, malware, steganography, software bugs, and network vulnerabilities. The effective detection of threats in such scenarios requires to collect and correlate even apparently independent events from multiple subsystems, which is not possible for legacy standalone security appliances.

According to recent reports from cyber-security vendors, attacks are becoming ever more complex and stealthy, in order to elude well-known detection techniques based on signatures and behavioral patterns. As a paradigmatic example, steganography can be used to hide the presence of malicious code in digital media and network traffic is used as the carrier to covertly exfiltrate data or to stealthily orchestrate nodes of a botnet. Therefore, many recent attacks are difficult to detect and such a trend is expected to continue to grow [2].

Under this evolutionary scenario, programmatic access to data and events is an important requirement to improve the likelihood of detecting stealthy software and communications. In this paper, we show how research outcomes from complementary projects can be joined to this purpose. We consider the challenging scenario of detecting stegomalware in virtualized services, which encompasses both cloud applications and network functions virtualization. The platform for lightweight and programmatic monitoring and inspection is taken from the ASTRID project[1], whereas SIMARGL[2] aims at developing scalable mechanisms to counteract novel malware endowed with steganographic techniques and crypto-lockers. While detection of network attacks has been largely discussed in the literature, information-hiding-capable threats pose new challenges, as they exploit bandwidth-scarce channels and their detection is a poorly generalizable process [3]–[5].

In more detail, we investigate the use of the programmable monitoring and inspection framework developed by ASTRID to collect data and measurements from virtualized environments, which are usually difficult to gather in an efficient way with existing tools. In addition, since the detection of steganographic threats is tightly coupled with the used carrier embedding the secret (e.g., the enumeration of sockets, the acquisition of a lock on a file or the manipulation of field within the header of a protocol), instrumenting virtual machines and network nodes without impacting their performances could be a hard and time-consuming task. To this aim, we leverage the extended Berkeley Packet Filter (eBPF), a framework integrated in the Linux kernel for the inspection of system calls, e.g., page faults and traffic stimuli [6]. Obtained measures can be then evaluated with toolkits envisaged in SIMARGL to reveal malware, hidden Command & Control (C&C) attempts or attacks targeting virtualized architectures.

[1]AddreSsing ThReats for virtualIzeD services (ASTRID). URL: https://www.astrid-project.eu.

[2]Secure Intelligent Methods for Advanced Recognition of Malware and Stegomalware (SIMARGL). URL: https://simargl.eu.

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

Summarizing, the contributions of this paper are: *i*) the review of works enforcing security through virtualization; *ii*) the design of an architectural blueprint to integrate ASTRID and SIMARGL with emphasis on the flows of information that can be exchanged to perform detection of emerging threats; *iii*) the identification of malware taking advantage of steganography and the various technological behaviors that should be considered to collect the data both to reveal and neutralize the attack; *iv*) a preliminary experimental campaign on the use of eBPF to support the detection of stegomalware exploiting a local covert channel between two malicious software components.

The remainder of the paper is structured as follows. Section II reviews previous works on virtualization and security, while Section III provides the background. Section IV deals with the architecture combining the features of ASTRID and SIMARGL. Section V discusses technical aspects of data gathering and detection, and Section VI showcases a preliminary performance evaluation on the use of eBPF for detecting stegomalware. Section VII concludes the paper and hints at some possible future developments.

II. Related Works

The use of virtualization to support security-related duties and the mitigation of attacks targeting virtual services has been already partially investigated in the literature. For instance, the approaches proposed in [7], [8] take advantage of an orchestrator for controlling pervasive and lightweight security hooks embedded in the virtual layers of cloud applications. The work in [9] discusses a mechanism to enhance a network hypervisor with new functions for implementing a flexible monitoring service. The proposed approach can ease the engineering of monitoring and security-related services, especially when in the presence of complex architectures based on micro services or cloud technologies. For the case of cybersecurity applied to networking, Deep Packet Inspection (DPI) is an important technique as it allows to evaluate several aspects of a flow, including alterations in the header adopted by many covert channels [3], [10]. Indeed, network virtualization is often at the basis of large-scale scenarios where engines responsible of performing DPI can be deployed as software components on commodity hardware. To this aim, [11] proposes an approach for their dynamic placement to contain power consumptions and costs while delivering suitable degrees of scalability and performance. A more comprehensive framework is discussed in [12], where authors propose a virtualized security architecture to enforce integrity of virtual machines, isolate higher software layers, and provide adaptive network security appliances (e.g., intrusion detection systems and firewalls) encapsulated within virtual machines. Another important aspect concerns the definition and the implementation of effective orchestration policies. A possible idea exploits meta-functions to dynamically construct security services for satisfying various security requirements [13].

Concerning the detection of steganographic malware, at the best of our knowledge, there are not any works taking advantage of virtualization to gather data or neutralize attacks. In fact, literature abounds of works on the sanitization of carriers or the normalization of network traffic (see, e.g., references [5] and [3] and the references therein) but not any prior work investigates how a covert channel or a steganographic threat could be detected or prevented by means of virtualization. A variety of works address the security of virtualized architectures, but they mainly focus on the following classes of hazards [14]: access control of resources, DoS and DDoS attacks, virtualization to build the network infrastructure, and security management of virtualized assets. Even if the security risks caused by mobility, migration and hopping of virtual machines are well understood [15], steganographic threats are solely discussed without considering the overall architecture or the peculiarities of large-scale virtualized network scenario. Rather, they are addressed for very specific cases, e.g., colluding containers or virtual machines, but without gathering data via agents automatically deployed [10], [16].

III. Background

Despite a unique and precise definition is missing, the term *stegomalware* is used in SIMARGL to identify malware endowed with steganographic functionalities or able to exploit a covert channel [4]. In general, an attacker deploys steganography for the following use-cases.

Colluding Entities: they are those trying to create a covert channel to exchange data within the single host, mainly to bypass the security policies deployed in the underlying software and hardware layers, the guest OS or the hypervisor [5], [19]. Thus, colluding applications implement a sort of unauthorized inter-process communication service between several software entities, for instance, virtual machines and containerized applications, as well as regular applications and processes. To exchange data, the typical approach sets a local covert channel built by modulating the space available in the file-system, the CPU load or the state of TCP sockets. To detect such communications, two main techniques are available: monitor the syscalls handling the I/O or the access to the used carrier (e.g., a temporary file) or evaluate the sleep/wake patterns of processes as to identify those having overlapping behaviors. In general, this attack can be used to exfiltrate data from an entity to another, e.g., to allow a containerized application handling sensitive data without network privileges to push the stolen information to another container with less constraints [16]. Moreover, virtual machines could also cooperate for reconnaissance purposes, e.g., to allow the malware to determine if it is confined on a honeypot or to map the underlying physical infrastructure [22].

Network Covert Channels: they permit to stealthily transfer data by injecting the information within a network artifact acting as the carrier. To this aim, the sender can modulate the content of packets or alter some traffic features, e.g., he/she encodes information by manipulating the inter-packet time with proper delays. For the case of malware, network covert channels are typically used for: data exfiltration, implementation of a C&C infrastructure, development of cloaked transfer

TABLE I

MEASURE OF INTEREST TO BE PERFORMED VIA ASTRID TO FEED THE SIMARGL TOOLKIT TO COUNTERACT STEGOMALWARE AND CRYPTLOCKERS.

Threat	Behavior of Interest	Sample Measures	Refs.
Colluding Applications	Activation statistic of processes or threads	stack trace of waker, total blocked time	[17], [18]
Colluding Apps. or Containers	Monitoring type and number of calls to VFS	type of VFS func, count	[16], [19]
Colluding Virtual Machines	Memory usage to infer anomalies	page cache hit/miss, read and write hit	[16], [20]
CC manipulating file-system	Processes that are performing disk I/O	type of op, disk, number of op	[3], [10], [17]
CC manipulating CPU	Time spent using the CPU	duration, count, load distribution	[3], [10], [17]
CC manipulating files	Reads and writes performed	reads, writes, r_kb, w_kb, type	[3], [10], [17]
CC enumerating sockets	TCP state change information	socket address, pid, cmd, state, duration	[3], [10], [17]
Miners, Cryptolockers and CC	Block disk I/O activity and performance	I/O latency, duration and count of operations	[4], [5]
Cryptolockers	Type of disk I/O operations and CPU usage	type of op, kbytes in R/W	[21]

services for retrieving additional software components, botnet orchestration, and elusion of firewall rules [3], [5], [10].

Cycle-stealing threats, ransomware and cryptolockers: despite their precise characterization is still an open problem, such threats share the aggressive usage of resources when attacking [23]. For instance, malicious code used for mining cryptocurrencies or orchestrating a botnet will impact on the average utilization of computing or network resources, while the encryption of content stored on the file-system accounts for a significant volume of syscalls handling I/O operations. Therefore, being able to filter and monitor specific events within the kernel is a prime requirement for building effective indicators. This is even more important since many malware increase their stealthiness by adopting "time bomb" mechanisms triggering the execution after a predefined period of time. This can reduce the chance of spotting the attack via a cause-effect observation (e.g., a lag in the graphical user interface due to excessive loads on the CPU) [24]. Similarly to the case of stegomalware, also cycle-stealing threats and ransomware require the access to different usage statistics, which are difficult to define *a priori*. A possible idea exploits abstract indicators, such as the used power [17]. In this case, being able to perform measurements "close" to the device driver or within the kernel is definitely important.

The SIMARGL project also aims at detecting **obfuscation**, which is an umbrella term for techniques enabling an application to look benign, even if endowed with malicious code. A prime class of methods embraces those using digital images as the carrier. For instance, some malware hides shell scripts by simply renaming them to mimic images (e.g., install.png) [25] or exploits steganography to embed malicious code within images, configuration data, and list of IP addresses to inspect or URL to contact for retrieving additional code in order to implement multi-stage loading mechanisms [4].

Table I resumes the most popular attacks discussed in the literature, the behavior of interest, as well as some possible measurements to perform detection. For instance, in order to detect colluding applications or containers, a possible approach could monitor the behavior of the Virtual File System (VFS).

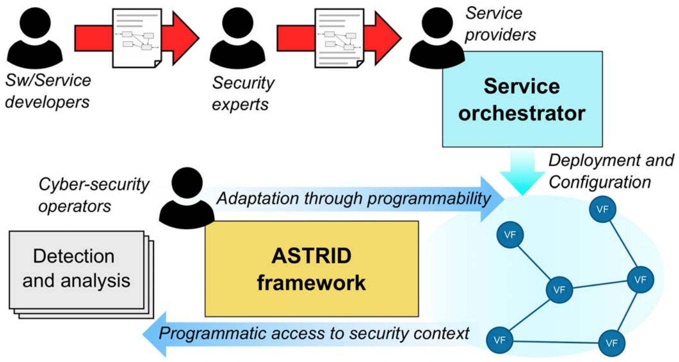


Fig. 1. The utilization flow implemented by the ASTRID framework to mediate between security appliances and the underlying context.

In this case, by looking for the type of VFS functions called and their distribution, it could possible to have a characterization of the workload and to spot anomalous bursts of operations. Other measures could be those related to the I/O activity, such as the latency or the distribution of the size of the block of data (i.e., the block disk I/O). These values can provide relevant statistical indicators to detect miners or cryptolockers.

IV. MONITORING VIRTUALIZED SERVICES

Model-driven engineering is being increasingly used both in the cloud and NFV domain to design high-level process models (defined as *service graphs*), which are then dynamically and automatically mapped into software functions and infrastructural resources based on the evolving context by orchestration tools. This approach brings unprecedented agility in life-cycle management of digital services, but has not been yet properly integrated with cyber-security paradigms.

The ASTRID architecture chases an effective solution to deliver multiple cyber-security services for virtualized applications. Its workflow starts with the revision of the service graph, made by cyber-security experts (see the chain at the top of Figure 1), before it is made available to service providers for automatic deployment through software orchestration tools.

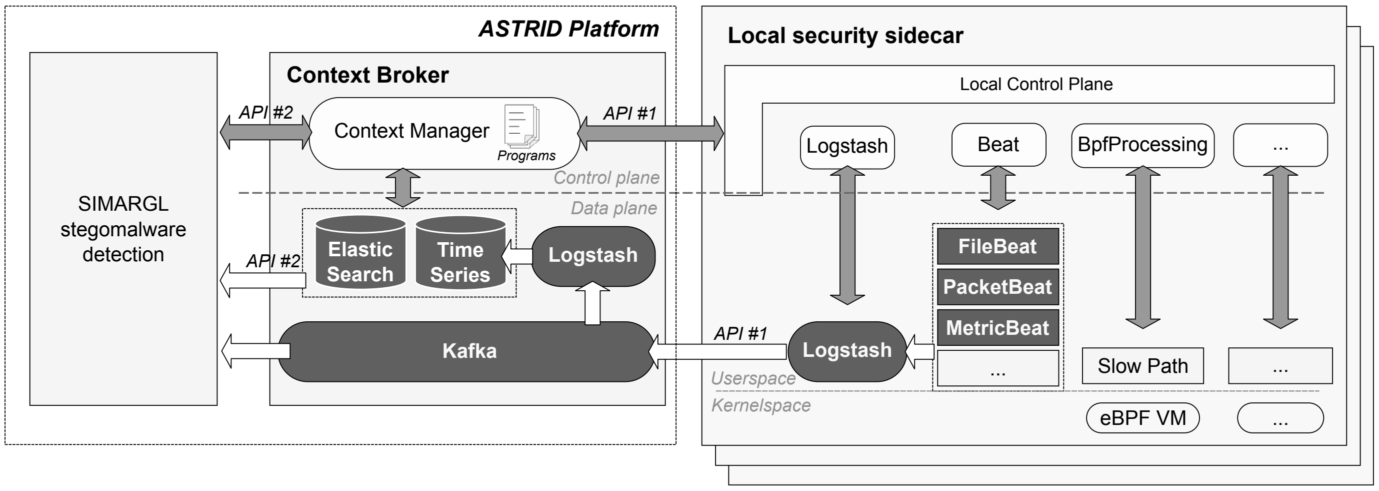


Fig. 2. Software architecture of the ASTRID platform.

The revision provides indications about security agents to be included in the deployment plans and the connection to the ASTRID platform. The definition of security agents as part of the design allows to deploy them whenever and wherever necessary, hence following the evolution of the service topology at run-time (e.g., scaling, replication, and replacement operations).

As depicted in Figure 1, the ASTRID platform is conceived as a mediation layer between the detection and analysis logic and the physical and virtualization environments where virtual applications run. This design allows to disaggregate detection and analysis algorithms from the necessary hooks in the monitored system, so many different cyber-security appliances can be loaded and run at run-time, depending on the specific needs of the service and the evolving threat landscape. The distinctive and innovative approach of ASTRID is the integration with software orchestration tools to automate the deployment and management of security agents. The ASTRID platform allows both to change the configuration and behavior of single agents and to perform management actions of the whole graph, through the service orchestrator. Possible actions include the deployment of additional inspection and monitoring processes and the replacement of compromised functions. The overall architecture of ASTRID includes a Context Broker, a Security Controller, a user Dashboard, complementary Security Services, and plugins to interact with different orchestration software [7]. In this paper, we only focus on the Context Broker, which is responsible to programmatically collect the security context from local security agents deployed within virtual functions and to feed the SIMARGL toolkit in an effective way.

Figure 2 shows the part of the ASTRID software architecture that is relevant for this work. It includes local agents deployed in each virtual function as security sidecars and a centralized Context Broker for collection and analysis of security-related data and measurements. We do not consider the external service orchestrator that deploys local agents and the other ASTRID components (Security Controller and user Dashboard), because they are not used in the rest of this work. The implementation of local agents and the Context Broker largely builds on and extends the proven Elastic Stack framework. It includes common agents (defined as *beats* in the Elastic jargon), for the collection of log files, system metrics, and packet statistics. Additional agents are being developed by ASTRID to run eBPF programs. The proposed framework can also implement a common Local Control Plane (LCP) for all agents in order to provide a unified interface for changing the configuration of Elastic beats, loading Logstash pipelines, and injecting eBPF programs into the local environment.

The Context Broker implements an abstraction of the service topology and current configuration of security agents allowing to know the type and structure of data that are delivered from them. The interaction between the Context Broker and local agents happens on two different planes: control and data. The control plane is used to access the LCP, whereas the data plane is the logical channel to push events, measurements, and data from local agents to the Context Broker, through a Kafka bus. The architecture of the Context Broker is based on the Elastic Stack as well. At the data plane, it stores data from local agents in Elasticsearch, and includes a Logstash instance for performing aggregation and data fusion. Moreover, a Time Series database is also provided for collection of big historical data. The Context Manager is the logical subcomponent that implements control and management actions. It discovers available security agents and their capabilities and it is used to change their configuration and to inject programs at run-time. The Context Manager implements a REST API interface for querying the internal databases and to ask configuration changes. Messages on the Kafka bus from local agents may be received directly from security appliances, by subscribing the corresponding topics.

A. Towards the Integration with SIMARGL

To integrate the aforementioned software layers with SIMARGL, the toolkit has to subscribe for specific measurements/events to the Context Manager, which configures local agents via the BpfProcessing agent[3]. Once configured and activated, it starts sending events on the Kafka bus. Data is caught both by the central Logstash instance and the SIMARGL toolkit: the former stores data in Elasticsearch also to make information available for off-line processing. The SIMARGL toolkit listens on the Kafka bus to retrieve data as soon as they are published. Alternatively, it could periodically query Elasticsearch. We point out that algorithms implemented for detecting stegomalware can define and enumerate at run-time the needed measurements, and they can even load into ASTRID new eBPF programs for tailored data and statistics. This represents a major advance with respect to existing technology as recent cybersecurity appliances and tools are quite flexible in the definition of the context to be collected (i.e., logs from specific applications, packet filters, and event notifications), but are rather rigid in the management. In fact, if a monitoring tool (e.g., a log collector or a packet classifier) is not available a manual intervention from humans is required. In a similar way, reaction is often limited to traffic diversion, packet filtering, and changes in access rules.

V. Data Gathering for Stegomalware

The broad range of potential covert channels and carriers to be used for steganographic purposes makes the detection of stegomalware a challenging task. As a matter of fact, stegomalware may leverage network packets, file system properties, memories and caches, CPU performance, and so on. Common detection mechanisms based on the analysis of log files and network statistics could be largely ineffective in this respect.

The detection of stegomalware requires temporal and spacial correlation of fine-grained system properties and actions. DPI can be used to spot anomalous usage of some header fields, while system tracing is required to analyze the behavior of the operating system. For example, by only narrowing the discussion to the most popular attacks targeting the single host/node, we can report the following considerations:

- Colluding Entities: typical attacks modulate the space available in the file-system, the CPU load or the state of TCP sockets. To detect them, two possible techniques can be used: *i*) monitor the syscalls handling the I/O or the access to the used carrier (e.g., a temporary file) or *ii*) evaluate the sleep/wake patterns of processes as to identify those having overlapping behaviors, thus possibly colluding.
- Cryptolockers (or ransomware): in general, such threats produce a huge load on the filesystem to encrypt its content. Thus, monitoring operations on the I/O could be an effective indicator.

[3]Configuration includes the injection of an eBPF program from an internal library.

Several data sources are already present in the kernel to follow the execution of system calls and internal functions: *kprobes*[4], *uprobes*[5], *tracepoints*[6], *dtrace-probes* [26], and *lttng-ust*. Multiple tools are available to collect information from these tracing hooks (*ftrace*, *perf*, *sysdig*, *SystemTap*, *LTTng*), but eBPF is probably the most powerful framework for gathering data in the perspective of detecting information-hiding-capable threats. Specifically, there are no additional modules to install in the kernel, custom programs can be defined and attached to tracing hooks to do any kind of aggregation and lightweight processing. As an example of the broad range of applicability of the eBPF framework, Table II lists existing programs from the BPF Compiler Collection[7] (BCC) useful to detect some well-known covert channels.

Put briefly, BCC is a toolkit for creating efficient kernel tracing and manipulation programs, and already includes several tools for tracing the most common features. Besides, it allows writing eBPF programs in C language and compiling them with LLVM. It also includes front-ends in Python and Lua. ASTRID improves this framework by feeding Elastic Stack with the output from eBPF programs.

This framework can be used to implement an effective suite of probes to gather the data needed by the SIMARGL toolkit for detecting stegomalware. Through ASTRID, investigation should start from eBPF programs to detect high-level indicators, while additional programs are then injected as soon as the scope is narrowed. Since there is no dependency on a specific set of programs, new programs can be implemented at run-time to cope with new or unknown covert channels and threats using information hiding.

VI. Experimental Results

To prove the effectiveness of eBPF-based tracing, we selected a colluding applications threat where two endpoints exchange data through the chmod-stego technique. The chmod-stego is a covert channel that injects secrets in Unix file privilege numbers. The application is made of two peers, a sender and a receiver. The sender encodes data in the privilege of a set of files within a given directory, while the receiver listens for the specified file permissions. Then, according to changes, it decodes the secret message.

Since the chmod-stego technique is based on the manipulation of the file-system, the most straightforward way to design a detection strategy is by tracing the `__x64_sys_chmod` kernel function, which provides better indications than more generic I/O activity (e.g., read/write operations through `__x64_sys_read` and `__x64_sys_write`).

To validate such an idea, we investigated if and under what conditions a steganographic transmission could be detectable during system activity. We created an experimental setup composed of a Virtual Machine with Debian GNU/Linux 10

[4]https://lwn.net/Articles/132196/.

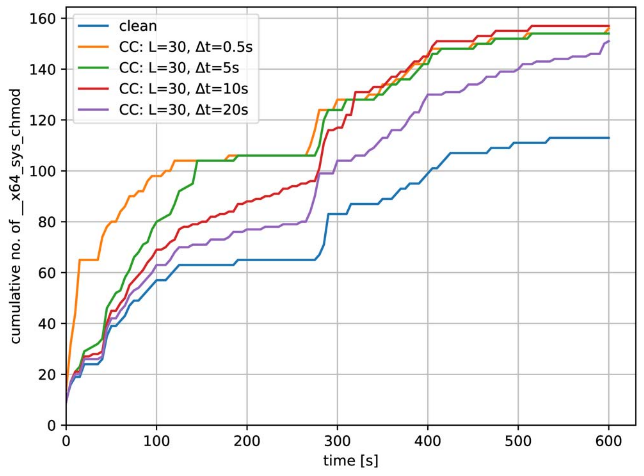
[5]http://www.brendangregg.com/blog/2015-06-28/linux-ftrace-uprobe.html.

[6]https://lwn.net/Articles/379903/.

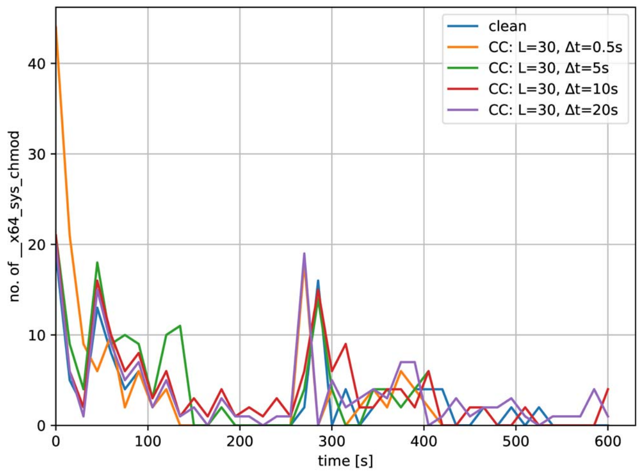
[7]https://github.com/iovisor/bcc.

TABLE II
EXAMPLES OF BCC TOOLS THAT CAN BE USED TO DETECT COVERT CHANNELS.

Attack	**BCC tools**
Colluding Applications	execsnoop, offwaketime, wakeuptime
Colluding Apps. or Containers	vfscount, vfsstat, biotop, ext4/xfs/zfsdist, ext4/xfs/xsflower, vfsreadlatency
Colluding Virtual Machines	cachestat, cachetop, biosnoop, bitesize, dcsnoop, dcstat, llcstat
CC manipulating file-system	biotop, biosnoop, bitesize, disksnoop, filteop
CC manipulating CPU	pidperscc, cachestat, cachetop, cpudist
CC manipulating files	filelife, filetop, filesnoop
CC enumerating sockets	tcpv4connect, opensnoop, solisten, tcpconnect, tcpstates, tcplife, tcpaccept, sofdsnoop
Miners, Cryptolockers and CC	biolatency, vfscount, biotop, biosnoop, funccount, funclatency, profile
Cryptolockers	biotop, biosnoop, bitesize



(a) Cumulative time evolution



(b) Instantaneous time evolution

Fig. 3. Detected invocation of the `__x64_sys_chmod` kernel function with $L = 30$ and $\Delta t = 0.5, 5, 10, 20$ s.

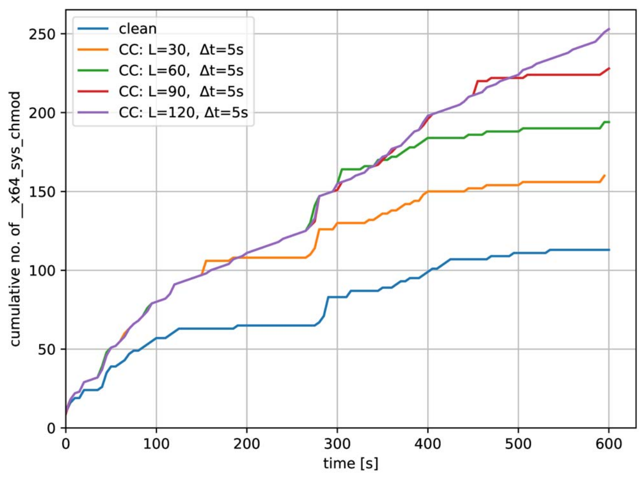
(buster) running the Linux kernel 4.20.9 and the aforementioned chmod-stego[8] steganographic method. A kernel compilation was run as the main system activity, which entails many I/O system calls and can be easily replicated for comparison. In essence, such a scenario allowed to investigate the presence of a malware which exfiltrates data or bypasses some local security policies, e.g., the sandboxes.

To gather data, a simple eBPF filter was injected to trace invocations of the `__x64_sys_chmod` kernel function and to report its relevant parameters, i.e., file and permissions, the Process ID (PID), and the Thread ID (TID).

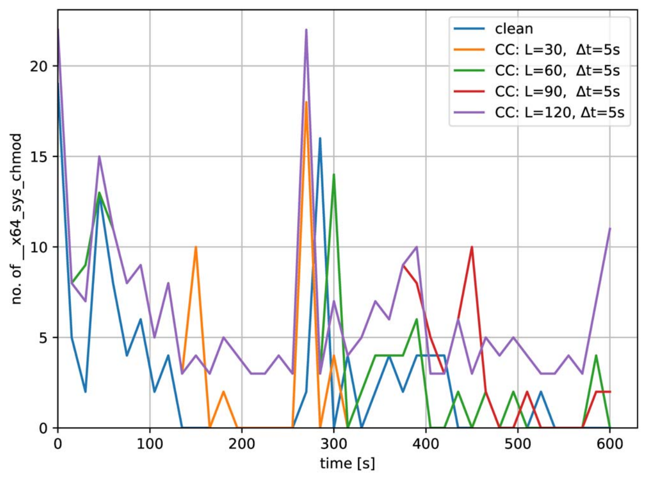
We performed two different tests. The first aimed at evaluating the tradeoff between the steganographic bandwidth of the covert channel and its detectability. To this aim, we fixed the length L of the secret message to be transmitted and we varied the time between the transmission of two consecutive characters, denoted as Δt. Specifically, we conducted trials with $L = 30$ and $\Delta t = 0.5, 5, 10, 20$ s. In the second round of tests, we investigated the influence of the size of the data exchanged between the two colluding applications. Hence, we set $\Delta t = 5$ s and we performed trials with $L = 30, 60, 90, 120$ characters, which may be representative of the exfiltration of a PIN, a cryptographic key or the information of a credit card. In both tests, the "clean" configuration has been considered the one characterized by the load of traced kernel functions due to the compilation of the Linux kernel 5.5.5. All the trials lasted 10 minutes. We point out that such parameters allowed to consider a wide range of threats (e.g., slow and long communications characterizing advanced persistent threats or malicious applications wanting to exfiltrate as quick as possible sensitive information) while guaranteeing the adequate statistical relevance.

[8]https://github.com/operatorequals/chmod-stego.

Figure 3 depicts the results of the first round of tests. As shown, the presence of an exchange of information through a covert channel (denoted as CC in the figure) affects both the number and the distribution of the `__x64_sys_chmod` kernel functions. Specifically, the higher the steganographic bandwidth (i.e., Δt decreases) the higher the load of



(a) Cumulative time evolution



(b) Instantaneous time evolution

Fig. 4. Detected invocation of the `__x64_sys_chmod` kernel function with $\Delta t = 5$ s and $L = 30, 60, 90, 120$.

`__x64_sys_chmod` kernel functions at the begin. In fact, higher transmission rates reduce the time needed to transmit the secret message. This can be viewed in Figure 3(b), where the instantaneous time evolution is shown. Similar results have been observed for the second set of experiments, which are showcased in Figure 4. In this case, the steganographic bandwidth is fixed and the length of the message is the unique factor that makes the transmission more or less detectable.

For what concerns detection, in general, channels with a higher steganographic bandwidth and longer messages are easier to detect. Indeed, they imply either sudden peaks or larger volumes of `__x64_sys_chmod` kernel functions. Clearly, on-line detection is not straightforward, because of the difficulty to find an effective decision rule able to discriminate between legitimate usage and the presence of hidden transmissions for different use cases. Luckily, for the case of chmod-stego technique, a possible signature is given by a quick change in the volume of `__x64_sys_chmod` kernel functions at the end of the trials. This is due to the sender that restores the original file permissions, as to avoid the detection by common file system monitoring tools. Moreover, taking into account additional parameters available from tracing (e.g., the file names) can be used to further improve the likelihood of the detection. In this perspective, the SIMARGL toolkit will take into account multiple complementary indicators, possibly independent of the specific threat.

VII. Conclusions and Future Works

In this paper, we showcased how the framework developed within ASTRID can be used by the SIMARGL toolkit for the detection of novel threats, such as stegomalware and cryptlockers. To prove the effectiveness of this vision, we presented a preliminary performance evaluation on the use of the eBPF to collect variable of interests. The provided data can then be used to feed detection models or create datasets to train machine-learning-capable techniques.

Future works aim at refining the approach. In particular, the main objective is the definition of a more programmatic process to progressively narrow down the scope from generic indicators to fine-grained tracing of execution patterns for specific covert channels or information-hiding-capable threats. In this respect, ongoing research deals with the development of threat-independent signatures such as energy consumption, RAM usage patterns, and the time statistics of running processes.

Acknowledgment

This work has been supported by the EU Project ASTRID, Grant Agreement No 786922, and by the EU Project SIMARGL, Grant Agreement No 833042.

References

[1] R. Rapuzzi and M. Repetto, “Building situational awareness for network threats in fog/edge computing: Emerging paradigms beyond the security perimeter model,” *Future Generation Computer Systems*, vol. 85, pp. 235–249, August 2018.

[2] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander, “The new threats of information hiding: The road ahead,” *IT Professional*, vol. 20, no. 3, pp. 31–39, 2018.

[3] S. Zander, G. Armitage, and P. Branch, “A survey of covert channels and countermeasures in computer network protocols,” *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[4] W. Mazurczyk and L. Caviglione, “Information hiding as a challenge for malware detection,” *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89–93, 2015.

[5] ——, “Steganography in modern smartphones and mitigation techniques,” *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 334–357, 2014.

[6] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, “Creating complex network services with ebpf: Experience and lessons learned,” in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*. IEEE, 2018, pp. 1–8.

[7] M. Repetto, A. Carrega, and G. Lamanna, “An architecture to manage security services for cloud applications,” in *4th IEEE International Conference on Computing, Communication & Security (ICCCS-2019)*, Rome, Italy, Oct., 10th–12th, 2019.

[8] S. Covaci, R. Rapuzzi, M. Repetto, and F. Risso, "A new paradigm to address threats for virtualized services," in *IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, Jul. 23rd-27th, 2018, pp. 689–694.

[9] Y. Sun, S. Nanda, and T. Jaeger, "Security-as-a-service for microservices-based cloud applications," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 50–57.

[10] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Computing Surveys (CSUR)*, vol. 47, no. 3, pp. 1–26, 2015.

[11] M. Bouet, J. Leguay, and V. Conan, "Cost-based placement of virtualized deep packet inspection functions in sdn," in *MILCOM 2013-2013 IEEE Military Communications Conference*. IEEE, 2013, pp. 992–997.

[12] J. Li, B. Li, T. Wo, C. Hu, J. Huai, L. Liu, and K. Lam, "Cyberguarder: A virtualization security assurance architecture for green cloud computing," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 379–390, 2012.

[13] Z. Lin, D. Tao, and Z. Wang, "Dynamic construction scheme for virtualization security service in software-defined networks," *Sensors*, vol. 17, no. 4, p. 920, 2017.

[14] X. Luo, L. Yang, L. Ma, S. Chu, and H. Dai, "Virtualization security risks and solutions of cloud computing via divide-conquer strategy," in *2011 Third International Conference on Multimedia Information Networking and Security*. IEEE, 2011, pp. 637–641.

[15] H.-Y. Tsai, M. Siebenhaar, A. Miede, Y. Huang, and R. Steinmetz, "Threat as a service?: Virtualization's impact on cloud security," *IT professional*, vol. 14, no. 1, pp. 32–37, 2011.

[16] X. Gao, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "Containerleaks: Emerging security threats of information leakages in container clouds," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017, pp. 237–248.

[17] L. Caviglione, M. Gaggero, J.-F. Lalande, W. Mazurczyk, and M. Urbański, "Seeing the unseen: revealing mobile malware hidden communications via energy consumption and artificial intelligence," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 4, pp. 799–810, 2015.

[18] M. Urbanski, W. Mazurczyk, J.-F. Lalande, and L. Caviglione, "Detecting local covert channels using process activity correlation on android smartphones," *International Journal of Computer Systems Science and Engineering*, vol. 32, no. 2, pp. 71–80, 2017.

[19] C. Marforio, H. Ritzdorf, A. Francillon, and S. Capkun, "Analysis of the communication between colluding applications on modern smartphones," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 51–60.

[20] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 305–316.

[21] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, "Cryptolock (and drop it): stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2016, pp. 303–312.

[22] P. Wang, L. Wu, R. Cunningham, and C. C. Zou, "Honeypot detection in advanced botnet attacks," *International Journal of Information and Computer Security*, vol. 4, no. 1, pp. 30–51, 2010.

[23] K. Liao, Z. Zhao, A. Doupé, and G.-J. Ahn, "Behind closed doors: measurement and analysis of cryptolocker ransoms in bitcoin," in *2016 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 2016, pp. 1–13.

[24] R. Andriatsimandefitra and V. V. T. Tong, "Detection and identification of android malware based on information flow monitoring," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. IEEE, 2015, pp. 200–203.

[25] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and accurate classification of obfuscated android malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 309–320.

[26] B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in *Proceedings of the annual conference on USENIX Annual Technical Conference (ATEC '04)*, June 2004.