Research Statement

Georgios Portokalidis

Throughout my academic career I have approached research from a systems perspective, driven by a desire to build working systems that explore the applicability of novel ideas. When I first begun working on network security, fast-propagating computer worms threatened the Internet infrastructure. For example, the Slammer worm infected 90% of vulnerable systems in less than 10 minutes. Such worms congested global networks and managed to infiltrate critical systems, like nuclear plants and banks. Concurrently, attackers started utilizing techniques like polymorphism and metamorphism to evade detection from intrusion detection systems (IDS). The increasing number of these attacks, their speed, and the inaccuracy of current detection methodologies was the primary motivation for my work on the Argos secure emulator [7].

Today, the broad range of applications targeted by attackers shows that it is not sufficient to focus on servers, or rely entirely on peripheral network defenses. It is necessary to protect a wider range of software and platforms, and deploy defensive mechanisms "near" software (*e.g.,* the host it resides in) to attain increased detection accuracy, and prevent compromise. In fact, increasing software resilience against various types of attacks has been a popular subject of research in systems security. While significant effort has been invested in techniques that improve software security early in the development cycle, like type-safe languages and various compiler extensions, frequently security is not the primary concern for many applications, due to usability or performance issues. In previous work [1–3,5], I investigated ways to retrofit protection measures on currently deployed software through the use of virtualization, and methodologies to improve their performance. In the future, it is necessary to invest more resources in such techniques, as solutions that rely on the availability of source code, and the recompilation and redeployment of software are adopted very slowly, and frequently cannot operate on systems were multiple binary components from different vendors coexist.

It is also important to consider emerging platforms and paradigms. For instance, mobile devices like smartphones are increasingly used to access the Internet and are often used for privacy-sensitive tasks, becoming highly valuable targets for attackers. They are also quite different from PCs, so that previous solutions are not always applicable, or do not offer comprehensive security. Outsourcing security in the Cloud can provide the necessary computation power for applying a variety of security techniques [4], but additional research is required to increase its efficacy.

Devising mechanisms that can mitigate the effects of attacks is also essential, as in practice, we may not be able to anticipate or protect against all possible attack vectors. For instance, many protection mechanisms resort to terminating execution to avoid compromise, while vendor patches that can correct the problem at the source are not immediately available. By employing techniques like software self-healing [6], we can extract information gathered when an application first terminates to allow it to gracefully recover from reoccurring faults. Accounting, or even undoing, the effects of attacks that go initially undetected is equally important. For instance, an attacker may corrupt or exfiltrate system data. Current approaches can only crudely determine if certain files have been corrupted, while accounting for exfiltrated data is frequently even harder. In both cases, labor-intensive analysis from experts is required. In the future, I plan to explore the combination of execution replication in the Cloud, as in [4], and data flow tracking [2] to enable data auditing and automatic recovery services.

Following, I discuss past work and future directions in some more detail.

Software Security

Software has been continuously increasing in size and complexity, and a considerable part of it is still written in languages like C and C++, which suffer from a variety of memory safety errors. Attacks targeting such

errors have been repeatedly used by attackers to gain control of remote systems, allowing them to gain access to sensitive data, like business documents and private keys, and install malicious software that is used to monitor the compromised system, send spam e-mail, or perform distributed denial of service (DDoS) attacks. Attacks exploiting previously unknown software faults are particularly challenging to tackle, since there is no information regarding the attack or the vulnerability being exploited. The large number of faults present in software and the increasing connectivity of software demand an accurate and automatic methodology for identifying and preventing compromise. A fact that frequently clashes with the demand for higher performance and usability.

These issues motivated me to develop the Argos secure emulator [7], which utilized dynamic taint analysis to provide a platform for running high-interaction honeypots (decoy systems), that can accurately detect various types of zero-day attacks, like program control data overwrites and code-injection. Argos was also able to automatically generate network intrusion signatures that could be deployed on the network using software like Snort. Due to its accuracy and its ability to run multiple operating systems (OSs), including various Windows versions, Argos sparked many follow-up projects, and several PhD works are based on it.

As attackers targeted client applications with increasing frequency (*e.g.,* web browsers, email clients, PDF readers, *etc.*), the need for broader defenses was made apparent. I developed Eudaemon [3] to investigate the on-demand application of DTA on binaries. The on-demand activation of certain defenses is a particularly interesting direction, exactly because it enables us to offer increased security to more software, with less overhead, albeit only for part of its execution. Another path that I wish to explore is applying different security techniques on different parts of the same software based on the requirements of each part. For instance, segments of an application may face exposure to diverse threats, or be exposed to dissimilar levels of risk, based on the user being served, the data being accessed, or other innate properties of the software. Through the combination of static and dynamic analysis, these segments can be identified and different security monitors installed.

In Columbia, I have been collaborating with multiple PhD students to develop a faster and more flexible platform for dynamically applying data flow tracking (DFT), including DTA. One of the tools we developed, namely libdft [2], is a reusable framework that can be used to develop various DFT-powered tools, and it is significantly faster that previous generic frameworks. I have also explored the use of offline static analysis to harvest information that can be utilized to reduce runtime overhead. This resulted in our recent NDSS publication on accelerating software-based dynamic DFT [1]. In the future, I would also like to explore the use of static analysis to enable the efficient parallelization of DFT on modern multicore CPUs.

I have also looked into using more lightweight techniques that can protect against a smaller set of attacks like code-injection. In particular, I created a lightweight version of software-based instruction-set randomization (ISR) for binary-only software [5]. Future directions in this area involve extending ISR to prevent returned-oriented and jump-oriented shellcode from executing, while keeping execution overhead in a manageable range.

Finally, I am exploring other methodologies for detecting and preventing memory corruption-based attacks at the exact point where the error occurs. Researchers have previously demonstrated that this is possible by recompiling software, but this is frequently not possible for many of the components used in a system. I am looking into ways to extend existing methodologies to achieve memory access integrity for binary-only components. My goals are to improve accuracy over previous work, extend its applicability to more software, and safe guard both against program compromise and data leakage. The latter type of attacks is many times used to break other defense mechanisms, like address layout randomization (ASLR).

Mobile Device Security

Recently, researchers have been focusing in the area of mobile device security and especially smartphones. These devices are becoming increasingly powerful and popular, to the point, that they have begun to replace desktops and notebooks as the primary mean to access the Internet. As a result, they will become a prime target for attackers as both opportunity and motive are materializing. Additionally, they are highly mobile, many times moving between different networks with a variety of security policies and disproportionate exposure to threats.

However, these devices hold significantly less processing power than PCs and are far more energy constrained. In my paper, *Paranoid Android* [4], I explored the decoupling of execution from security analysis for smartphones. I would like to further explore this model of offering security services to mobile devices through the Cloud, which would allow us to exploit the great disparity in processing power to offer extensive security-oriented services. With *Paranoid Android*, I explored the possibility of using DTA to protect core services on the phone, but other issues can also be tackled this way. For instance, identifying malicious or untrustworthy applications downloaded from various App stores, preventing sensitive data leaks, and offering auditing services (*e.g.,* an execution black box for smartphones).

Mitigating the Effects of Attacks

The introduction of various defenses has increased software resilience against attacks, but on almost all cases (*e.g.,* consider the well-known stack-smashing protection and ASLR), these defenses resort to terminating the vulnerable program just before it is compromised. As a result, even though the attack is thwarted, the availability of the protected application is affected. Other times an attack is detected with some delay, which may leave the system in an corrupted state. Consider the recent attack against kernel.org, the host of the Linux kernel. After its maintainers realized that their servers had been compromised, files were manually examined to ensure that no files or code (*e.g.,* a backdoor) were planted within the kernel tree. As a result, kernel.org was offline for months. Mitigating the effects of attacks is an important research direction. My goal is to create a platform that would enable software to self-heal, and resume operation automatically or with little intervention.

While in Columbia, I developed REASSURE [6], a self-contained tool for allowing software to self-heal using rescue points. Rescue points are functions that can be used to return a valid error code to the application and restore its state, when an error contained within itself, or a called function, occurs. That is, they enable us to use existing error handling code to treat unknown errors. In the future, I would like to expand this work to automatically identify and decide which rescue point to deploy, without the need for application source-code, by using static analysis and the return values of known program elements, like system calls in Linux and API calls in Windows. This line of research can be also applied on program crashes due to other program errors, such as null pointer dereferences.

Another direction I would like to explore is combining execution replaying and data tracking to identify data corrupted during an attack, and automatically restore it, if possible, to a pristine state. By recording the execution of an application running on a production host, execution can be later replayed and DFT dynamically applied to detect tampered content, while concurrently preserving legitimate updates performed after the attack. Even if the process cannot be completely automated, it can assist users to accelerate the data recovery process and provide insights on the attackers' motives.

Data Flow Control and Auditing

With the new trend of migrating data and services to the Cloud, businesses and individuals have started to worry about the loss of control over their data. These worries have been fortified by various information leakage incidents that have occurred due to user or programmer error. As a result, businesses are often reluctant to move their infrastructure on the Cloud and to collaborate with third parties to enrich their services. Mechanisms that offer data flow control, or enable business owners to keep track of *who looks at which data*, will be in great demand in the future. One of the greater challenges for their adoption will be reducing their impact on performance to acceptable levels for production systems.

I have worked with students in Columbia to develop Taint-exchange [8], a system that performs fine-grained data tracking (*i.e.,* propagates data tags) across processes and hosts, thus allowing one to track data as it propagates within a distributed system like the Cloud. We built this system to investigate the impact of distributed data tracking on communications within the Cloud, as well as between Cloud and user. Some of its possible uses include restricting the flow of sensitive data or reporting the location and movement of data. In the future, I would like to explore the combination of various types of data tracking to create a low-overhead, yet powerful, data tracking platform. For instance, by combining fine- and coarse-grained data tracking, and applying it at different levels (*e.g.,* VMs, interpreters, or the OS).

References

[1] K. Jee, G. Portokalidis, V. P. Kemerlis, S. Ghosh, D. I. August, and A. D. Keromytis. A general approach for efficiently accelerating software-based dynamic data flow tracking on commodity hardware. In *Proceedings of the* 19^{th} *Annual Network and Distributed System Security Symposium (NDSS)*, February 2012.

[2] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. In *Proceedings of the* 8^{th} *International Conference on Virtual Execution Environments (VEE)*, March 2012.

[3] G. Portokalidis and H. Bos. Eudaemon: Involuntary and on-demand emulation against zero- day exploits. In *Proceedings of ACM SIGOPS EUROSYS'08*, pages 287–299, Glasgow, Scotland, UK, April 2008. ACM SIGOPS.

[4] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile protection for smartphones. In *Proceedings of the 2010 Annual Computer Security Applications Conference (ACSAC)*, December 2010.

[5] G. Portokalidis and A. D. Keromytis. Fast and practical instruction-set randomization for commodity systems. In *Proceedings of the 2010 Annual Computer Security Applications Conference (ACSAC)*, December 2010.

[6] G. Portokalidis and A. D. Keromytis. REASSURE: A self-contained mechanism for healing software using rescue points. In *Proceedings of the* 6^{th} *International Workshop on Security (IWSEC2011)*, November 2011.

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

[8] A. Zavou, G. Portokalidis, and A. D. Keromytis. Taint-Exchange: a generic system for cross-process and cross-host taint tracking. In *Proceedings of the* 6^{th} *International Workshop on Security (IWSEC2011)*, November 2011.