Research Statement

Christopher Kruegel
Department of Computer Science,
University of California, Santa Barbara

The main focus of my research is systems security. I seek to create solutions that solve important security issues affecting a large number of users. The goal of my work is to build security systems, deploy them in real-world environments, and perform experiments to characterize and explain their behavior. I believe that creating working systems that address real-world problems not only provides a great incentive for my research, but also allows for the necessary sanity checks of the results. With these goals in mind, I have also contributed to open-source projects and developed software artifacts that have been publicly released.

I have always been fascinated by systems security research because it requires an intimate knowledge of many important computer science disciplines such as distributed systems, computer networks, operating systems, and also computer languages. This has allowed me to work on a variety of interesting topics. The following sections highlight some contributions in three areas in which I am active.

Malicious Code

Malicious code (or malware) is defined as code that fulfills the harmful intent of an attacker. Typical examples include viruses, worms, and spyware. Current systems to detect malicious code (most prominently, virus scanners) are mostly based on syntactic signatures, which specify byte sequences that are characteristic for a particular malware instance. This approach has two drawbacks. First, specifying precise, syntactic signatures makes it necessary to update the signature database whenever a previously unknown malware sample is found. Second, changes in the code layout (code obfuscation) are sufficient to evade syntactic signatures, even when the semantics of the code does not change. To address these problems, we proposed to characterize binary code using behavioral and structural properties. The key idea is to generate more abstract, semantically-rich descriptions of malware, and to characterize classes of malicious code instead of specific instances. When analyzing an unknown piece of code, the analysis engine attempts to identify sequences that implement the specified behavior or that exhibit the defined structure. Because the syntactic properties of code are (largely) ignored, these techniques are (mostly) resilient to evasion by code obfuscation. Also, in certain cases, they allow the detection of novel malware instances.

In [18], we introduced a behavioral characterization of rootkits that classifies a code sequence as malicious when it writes to certain areas of the kernel memory. Another behavioral specification, which characterizes spyware, was presented in [11]. Using this specification, we detect a Windows browser helper object as spyware when it first uses browser functions to spy out user activity and then invokes Windows system calls to transmit the collected data to a third party. Finally, we introduced more general specifications [6] that identify malicious behavior based on suspicious information access and propagation patterns. To this end, the system monitors the execution of a program for events where sensitive information that is unrelated to this program's execution is first read and then written to another file or a socket. This enables us to detect previously unknown malware programs that steal sensitive data or attempt to hide their presence on the host.

The aforementioned systems have in common that all behavioral specifications were provided manually. While this is effective for certain classes of malware, it requires a human analyst to develop new specifications whenever a new type of threat emerges. To address this limitation, we have proposed a technique to automatically derive specifications [7]. This technique mines patterns of malicious behavior by contrasting the execution traces of benign programs with those of malicious instances. By identifying those actions

that are unique to the malware, one can extract behaviors that are likely unwanted. This information can then be used to generate detection signatures or to inform a human analyst about previously unseen, malicious activity.

In addition to behavioral specifications, we introduced a structural characterization of binaries that can be used to identify metamorphic instances of a worm [16]. To this end, we scan network traffic for the excessive appearance of binary code that shares a similar structure and thus, likely belongs to instances of the same worm. The structure of binaries is captured by analyzing their control flow graphs. More precisely, the control flow graph of a program is divided into a number of subgraphs. Each subgraph is then converted into a fingerprint (hash) that uniquely represents this subgraph. To check whether two malware programs are related, we examine the number of fingerprints that these programs share.

To identify malicious behavior or to detect suspicious structure, it is necessary to analyze the code of unknown binary programs. To this end, static or dynamic code analysis approaches can be used. In [17], we discussed a static analysis technique based on symbolic execution of x86 machine instructions. The advantage of this static approach is that it takes into account the complete program and delivers precise analysis results. A significant problem for static analysis, however, is the fact that an unknown binary has to be disassembled first. While disassembly is relatively easy for regular binaries, the situation is different for malicious code. In fact, a number of techniques have been proposed that are effective in preventing a substantial fraction of a binary program from being disassembled correctly. To address this problem, we introduced an approach [19] based on control flow graph information and statistical methods that substantially improved the success of the disassembly process.

Dynamic analysis assures that only actual program behavior is considered. This eliminates possible incorrect results due to overly conservative approximations that are often necessary when performing static analysis. To leverage the power of dynamic analysis, we developed Anubis, an execution environment for Microsoft Windows binaries [15] that is based on a x86 processor emulator. This environment allows the stealthy inspection of running malware programs without any modifications to the code under analysis or the operating system. Anubis was later extended to track information flows using taint analysis [8]. This provides a more complete view of a program's behavior and allows us to determine relationships between program inputs and outputs (for example, to detect that a program copies its binary code over the network). We have also made available a public interface to Anubis, which allows everyone on the Internet to submit unknown binaries for analysis. This service has become very popular over the last year and currently receives more than fifty thousand malware submissions every month.

In general, dynamic code analysis has the drawback that only a single execution trace is observed. To remedy this weakness, we proposed an approach in which multiple execution paths can be sequentially explored [9]. That is, the system recognizes program points where the continuation of the execution depends on previously read input. The aim is to force the execution along different paths and to cover as much of the execution space as possible. This allows us to create behavioral descriptions that are significantly more comprehensive. In addition, the analysis extracts the conditions under which the program follows a particular execution path. Using this information, we can determine the circumstances under which a damage routine or a propagation function is executed. This is important to identify trigger-based behaviors (such as time bombs) that are only executed on a certain day or when a certain user is logged in.

Web Security

The number of installed web applications and their importance to organizations and businesses have significantly increased over the last years. Most web applications are publicly available over the Internet, making them attractive targets for attackers and spreading malware. In addition, novel threats have emerged in which the adversaries are not only interested in taking control of remote machines, but also in stealing sensitive information from their users. A common theme among such client-side attacks, which include cross-site scripting attacks and phishing, is that sensitive information in possession of the victim (e.g., passwords or credit card numbers) is illicitly transmitted to the attacker. This information is then abused for online fraud and identity theft. Server-side and client-side attacks have turned the world-wide web into a hostile place. As part of my research, I am interested in solutions that improve the security for web users and restore their confidence in transactions over the Internet.

In an effort to improve the quality of server-side software, we have developed a number of tools to identify vulnerabilities in these applications. In [13], we presented a system that performs black-box testing to automatically find common errors such as cross-site scripting or SQL injection vulnerabilities. This allows an administrator to quickly scan her web site for the presence of these problems. In [14], we discussed Pixy, a static source code analysis tool for the detection of cross-site scripting vulnerabilities in scripting programs written in PHP. Pixy employs flow-sensitive, inter-procedural, and context-sensitive data flow analysis to discover vulnerable points in a program. The tool also features a precise, global alias analysis to enhance the quality of the vulnerability reports [12]. Pixy's analysis was further improved in [5] to take into account the sets of string values that variables can hold at certain program points. This allows us to identify incorrect input validation checks, for example, in case an already checked string still violates the security policy. Moreover, string analysis provides the means to check for SQL injection vulnerabilities and identify more precisely the interaction of an application with its environment (e.g., which files an application touches, which ports it opens). This helps to generate access control policies that limit the privileges of an application to the minimal set required to execute correctly. Using Pixy, we discovered hundreds of previously unknown vulnerabilities in popular open-source programs with a low false positive rate (about one false positive per true vulnerability). Many bug reports have lead to patches, making these applications more secure. We provide Pixy as an open-source vulnerability scanner. So far, it has been downloaded several thousand times.

Server-side solutions have the advantage that a security flaw fixed by the service provider is instantly propagated to all its clients. Unfortunately, not all administrators provide due diligence, and users remain open to abuse when a vulnerable web site is not willing or able to fix the security issue. To address this problem, we developed client-side solutions that prevent the unintended flow of sensitive information to third parties. For example, in [2], we presented a web browser extension that detects the reuse of passwords (and similar sensitive information) at sites that were not cleared to receive this data. In [10], we described an extension to the popular Firefox browser that uses a combination of static and dynamic analysis techniques to keep track of the use of sensitive information (such as cookies). Whenever a transmission of sensitive information to a third part is detected, the browser can block the connection. This allows us to prevent cross-site scripting attacks.

Intrusion Detection

Intrusion detection systems are used to detect evidence of malicious activities targeted against the network and its resources. The ability to identify malicious activity supports early warning and response in those cases where more traditional security mechanisms (e.g., firewalls and access control enforcement) are insufficient or cannot be applied. Intrusion detection systems implement either a misuse-based detection approach or an anomaly-based detection approach. Misuse-based intrusion detection techniques use a number of attack descriptions, or signatures, to analyze a stream of events looking for evidence that the modeled attacks are occurring. Anomaly-based systems, on the other hand, operate by comparing observed behavior to models of normal behavior. Attacks are detected when observed behavior diverges in some respect from the normal behavior captured by a model. The problem with misuse-based detection approaches is the inability to detect previously unknown attacks. Anomaly-based systems can identify novel attacks, but suffer from the problem that benign events that differ from normal behavior are misclassified as malicious, leading to an unacceptable high number of false alerts.

An important focus of my research has been the development of techniques to retain the capability of anomaly-based techniques to detect novel attacks while, at the same time, reduce the false positive rate. To reduce the false positive rate, two complementary approaches can be taken. On one hand, the quality of each individual sensor can be improved. On the other hand, the findings of multiple sensors can be combined, or correlated. This has the additional benefit that a distributed attack, which manifests itself as different events at different sensors, can be detected.

To improve the quality of individual sensors, we proposed a multi-model anomaly detection system that simultaneously evaluates a number of different features of each event [3]. This has the benefit that an anomalous but benign value of one feature does not cause the system to immediately raise an alert. Also, we developed expressive models that capture the structure of events instead of simple (first order)

statistical properties, which were commonly used by previous anomaly-based intrusion detection systems. These models were built to monitor system call arguments [1] and web requests [21]. The web anomaly detection system has been extensively evaluated on several, real-world traces (including traffic recorded at Google). The results demonstrate that anomaly-based intrusion detection can be used to detect a wide range of attacks while still raising an acceptable number of false positives. Finally, we developed a novel approach based on Bayesian networks to combine the detection results of individual models to reach a final decision on whether to classify an event as legitimate or malicious [20].

Alert correlation is a complex process that involves a number of steps to transform a set of low-level sensor alerts into a succinct and high-level view of occurring attacks. Most correlation approaches only concentrate on just a few components of the process. In [4], we presented a general correlation model that includes a comprehensive set of components and a framework based on this model. This approach proved to be more effective in achieving alert reduction and abstraction than previous solutions. We also addressed the issue of scalability in alert correlation. Previous approaches relied on a central node to collect sensor alerts. The drawback of this approach is that this central node quickly reaches its limits when having to deal with the data volume that is produced by sensors in large network installations. In [24], we presented a completely distributed correlation mechanism that enables nodes to collaborate in a point-to-point fashion to identify emerging hostile patterns.

In addition to anomaly-based detection techniques and alert correlation, I have been interested in increasing the performance of signature-based approaches. This is important to allow intrusion detection systems to monitor high-speed network links. In [22], we presented a novel, fast rule matching engine for Snort, a popular open-source intrusion detection system. In [23], we introduced an architecture that allows us to partition a network stream into multiple, smaller streams that can then be independently and in parallel analyzed by an array of sensors.

References

[1] Darren Mutz, Fredrik Valuer, Christopher Kruegel, and Giovanni Vigna. Anomalous System Call Detection. In *ACM Transactions on Information and System Security, Vol. 9, No. 1, ACM Press*, 2006.

[2] Engin Kirda and Christopher Kruegel. Protecting Users Against Phishing Attacks with AntiPhish. In *The Computer Journal, Volume 49, Number 5, Oxford University Press*, 2006.

[3] Christopher Kruegel, William Robertson, and Giovanni Vigna. A Multi-model Approach to the Detection of Web-based Attacks. In *Computer Networks Journal, Vol. 48, No. 5, Elsevier*, 2005.

[4] Fredrik Valeur, Giovanni Vigna, Christopher Kruegel, and Richard Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. In *IEEE Transactions on Dependable and Secure Computing. Vol. 1, No. 3, IEEE Computer Society Press*, 2004.

[5] Marco Cova, Vika Felmetsger, Davide Balzarotti, Nenad Jovanovic, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, USA, May 2008.

[6] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *14th ACM Conference on Computer and Communications Security (CCS), ACM Press*, USA, October 2007.

[7] Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. Mining Specifications of Malicious Behavior. In *6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM Press*, Croatia, September 2007.

[8] Manuel Egele, Christopher Kruegel, Engin Kirda, Heng Yin, and Dawn Song. Dynamic Spyware Analysis. In *Usenix Annual Technical Conference*, USA, June 2007.

[9] Andreas Moser, Christopher Kruegel, and Engin Kirda. Exploring Multiple Execution Paths for Malware Analysis. In *IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, USA, May 2007.

[10] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In *Network and Distributed System Security Symposium (NDSS), Internet Society*, USA, February 2007.

[11] Engin Kirda, Christopher Kruegel, Greg Banks, Giovanni Vigna, and Richard Kemmerer. Behavior-based Spyware Detection. In *15th Usenix Security Symposium*, Canada, August 2006.

[12] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Precise Alias Analysis for Static Detection of Web Application Vulnerabilities. In *ACM Workshop on Programming Languages and Analysis for Security (PLAS), ACM Press*, Canada, June 2006.

[13] Stefan Kals, Engin Kirda, and Christopher Kruegel. SecuBat: A Web Vulnerability Scanner. In *15th International World Wide Web Conference*, United Kingdom, May 2006.

[14] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper). In *IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, USA, May 2006.

[15] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, Germany, May 2006.

[16] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic Worm Detection Using Structural Information of Executables. In *8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, USA, September 2005.

[17] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Automating Mimicry Attacks Using Static Binary Analysis. In *14th Usenix Security Symposium*, USA, August 2005.

[18] Christopher Kruegel, William Robertson, and Giovanni Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *20th Annual Computer Security Applications Conference (ACSAC), IEEE Computer Society Press*, USA, December 2004.

[19] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna. Static Analysis of Obfuscated Binaries. In *13th Usenix Security Symposium*, USA, August 2004.

[20] Christopher Kruegel, Darren Mutz, William Robertson and Fredrik Valeur. Bayesian Event Classification for Intrusion Detection. In *19th Annual Computer Security Applications Conference (ACSAC), IEEE Computer Society Press*, USA, December 2003.

[21] Christopher Kruegel and Giovanni Vigna. Anomaly Detection of Web-based Attacks. In *10th ACM Conference on Computer and Communications Security (CCS), ACM Press*, USA, October 2003.

[22] Christopher Kruegel and Thomas Toth. Using Decision Trees to Improve Signature-based Intrusion Detection. In *6th Symposium on Recent Advances in Intrusion Detection (RAID), Lecture Notes in Computer Science, Springer Verlag*, USA, September 2003.

[23] Christopher Kruegel, Fredrik Valeur, Giovanni Vigna and Richard Kemmerer. Stateful Intrusion Detection for High-Speed Networks. In *IEEE Symposium on Security and Privacy, IEEE Computer Society Press*, USA, May 2002.

[24] Christopher Kruegel and Thomas Toth. Distributed Pattern Detection for Intrusion Detection. In *Network and Distributed System Security Symposium (NDSS), Internet Society*, USA, February 2002.