Hayawardh Vijayakumar – Research Statement

My research interests are in computer systems security, with a core dissertation focus on achieving practical host security. During the course of my graduate studies, I have also performed work on transparent clouds monitoring and analyses of multi-layered security policies for consistency and mediation properties.

1 Local Vulnerabilities

With the emergence of targeted malware such as Stuxnet and the continued prevalence of spyware and other types of malicious software, host security has become a critical issue. Attackers break into systems through vulnerabilities in network daemons, malicious insiders or social engineering, and then attempt to *escalate privileges* to the administrator to gain complete control of the system by exploiting *local vulnerabilities*. Thus far, such local vulnerabilities have received little attention, and it has been taken for granted that any attacker break-in can easily be escalated to full control.

The core of my dissertation identifies and explores a fundamental reason that local vulnerabilities exist – a mismatch in programmer assumptions about the system deployment. Programs do not run in isolation, but rather, have to interact with the operating system (OS) to access resources, such as files. Such interaction is governed by the OS's access control policy. On the one hand, programs are written by programmers who have little idea about the variety of access control policies in OSes their program will be deployed on. On the other hand, access control policies are framed by OS distributors and system administrators, who in turn have little idea about programmer expectations. The current state of affairs is that all parties end up making assumptions about the program's *attack surface* – the particular threats the program faces in its deployment when it interacts with the OS.

However, there may be *mismatches* between a programmer's assumed attack surface and the real attack surface as configured by OS distributors and administrators. This causes a variety of program vulnerabilies that make up around 10-15% of the total vulnerabilities reported each year. Such vulnerabilities, including link following, untrusted search paths, and time-of-check-to-time-of-use race conditions have been known for decades, but still continue to plague today's programs because of the invalid assumptions discussed above. The definition, detection, and defense against such vulnerabilities is the focus of my dissertation.

Computing Attack Surfaces from System Deployment. My initial exploration began with the definition and automated evaluation of program attack surface [9]. Existing work was largely manual, and in addition, did not evaluate attack surface in the program's particular system deployment, which is crucial to understanding deployment-specific threats. The key technical contribution of this work was the definition and automated evaluation of a *per-program attack surface* relative to a system's mandatory access control (MAC) policy. Evaluation on the popular Ubuntu Linux distribution using a conservative attacker model indicated a manageable attack surface, where only around 10% of all program interactions were part of the attack surface. However, the attack surfaces that were located had previous vulnerabilities gradually discovered over the course of several years. In addition, we discovered a previously unknown vulnerability at an undefended attack surface. Such proactive identification of attack surfaces would thus help find vulnerabilities before attackers.

Finding Vulnerabilities During Name Resolution. My next study aimed to use the calculated attack surface to detect a class of previously unconnected vulnerabilities that I unified under the label of *name resolution vulnerabilities* [11, 10]. Name resolution is a process fundamental to computer science, and enables programmers to write portable code by identifying required resources using names instead of direct object identifiers, which vary across systems. However, attackers can subvert the name resolution process to divert the program to unintended resources, thereby compromising it. Programmers need to verify that they retrieved the intended resource, but they cannot do this effectively without knowing which name resolutions can be subverted by attackers, which is unique to each system deployment.

To detect such vulnerabilities in an automated manner, I developed an *in-vivo* tool that simultaneously tests

all programs running on a system for vulnerabilities. The key novelty of this work was the modeling of an attacker process that runs *in-vivo* inside the OS kernel. This process automatically performs malicious attacker actions to test victim programs whenever a risky name resolution is performed.

We found more than *25 previously-unknown vulnerabilities* across a variety of programs in the widely-used Fedora and Ubuntu Linux distributions. These included very mature programs such as the MySQL database server, the Apache webserver, and the Java Virtual Machine. Many of these vulnerabilities were reported and fixed. However, an interesting conundrum arose in some cases – programmers were unwilling to fix their code and claimed the system access control was broken, and OS distributors who framed the access control policy claimed the program was broken! These results confirmed that programs are not sufficiently customized for the OS they are deployed on. Our unified way of looking at several vulnerabilities as being due to weaknesses in name resolution was incorporated into the computer security course syllabus at Penn State University and North Carolina State University.

Defense Against Attacks. Practical defense against these attacks faces several challenges. First, it is impractical to expect programmers to customize their program code to defend attacks in each unique system deployment. Second, such defensive code causes performance overheads. A classic example is found in the Apache webserver documentation, which advises that such defensive checks be turned off for optimal performance!

To solve these challenges, we took inspiration from network firewalls, which solve a similar problem – host protection by customizing network interactions according to the deployment. Our system, called the Process Firewall [12, 8], performs a similar function for the program's system call interface. Our key insight was that our system protected benign processes instead of sandboxing malicious processes (as intrusion detection systems do). This motivated our novel technique of *process introspection* to deduce the program's intent in making a particular resource request. Using rules, the Process Firewall was demonstrated to block a variety of attacks requiring changes in neither program code nor the access control policy. Perhaps most interestingly, we found that substituting a program's resource checks in code by equivalent Process firewall rules actually improved program performance, and demonstrated this by showing more than a 100% speedup for common defensive checks, and up to 8% increase in request handling rate for the widely-deployed Apache web server.

2 Cloud Security and Policy Analysis

Transparent Cloud Monitoring. Cloud computing, with its elasticity, availability, and ease of deployment, has been quickly adopted by companies and end-users alike. The security of these platforms is dependent on many components – the hypervisors, insiders, and several management services. However, cloud platforms, such as in Amazon EC2 and Microsoft Azure are *opaque* to its customers, which makes it difficult for customers to evaluate if these platforms satisfy an expected level of security. To remedy this, I implemented an initial prototype to enforce the customer's security requirements on the machine hosting the customer's cloud instance [5] This design, called an Integrity Verification Proxy, could guarantee to the customer that, for example, the cloud host was running a trusted operating system [7]. This was integrated into the OpenStack cloud framework [6, 1].

Policy Analyses for Mediation and Consistency. My dissertation showed that problems can arise when two components from distinct parties – programs and the OS – are put together. However, this problem occurs in many contexts, such as in networks and virtual machines. All of these have their own access control policies, and the problem becomes verifying that the composition of policies at various layers enforces a coherent security goal. We developed one of the first techniques to analyze multi-layered policies [4]. Subsequently, we noted that a variety of problems, including verifying security policies for mediation properties, can be modeled as graph cuts [3]. Using this, we developed a technique to evaluate whether a set of policies protects sensitive resources from adversaries – a property formally called complete mediation [2].

3 Current and Future Research

Large-Scale, Directed Testing. The general problem of security testing is how to find vulnerabilities before an attacker. My dissertation showed that to accurately find vulnerabilities, we need to test programs in their deployments. However, in-vivo testing faces challenges of scalability: *(i)* the system under test may not be powerful enough to compute results of test cases quickly, and *(ii)* testing entire programs becomes unscalable. To solve *(i)*, my insight is that test systems need only capture, but not evaluate, test cases. A cloned program environment from the test system could be shipped to other locations for testing. In particular, we could use the massive computational power of the cloud for large-scale testing of programs. To address *(ii)*, we use attack-surface directed testing to focus available computational resources. For example, current fuzz testing techniques could be made more scalable and directed by only testing those parts of the program reachable from attack surfaces. We are currently working on such a large-scale, attack-surface directed testing framework with collaborators at UC Berkeley.

Automated Deployment-Specific Security for Programs. The eventual (and critical) question to be answered is – given a program and its deployment, can we: *(i)* determine if the program will be secure in that deployment against attacks when accessing resources, and *(ii)* address any insecurities found in some automated way? While my dissertation answers parts of these questions, further investigation is necessary. Fundamentally, an attacker can trick a program into accessing: *(a)* a malicious resource when it expects a trusted one, or, *(b)* a trusted resource when it expects a malicious one. Thus, a comprehensive theory of when a program is secure in its deployment needs to be predicated on how an attacker can trick a program and what the intent of the program is. In general, evaluating these conditions requires knowledge of program internals, for which I envision developing deployment-specific versions of widely-used techniques such as symbolic program execution and static program analysis.

Systems research has great potential to contribute directly to society. However, for this to be realized, systems have to be implemented and evaluated in real-world settings. To this end, I have made the code for all my research projects available, and performed my evaluations on widely-used programs and operating systems. I have a strong motivation to continue doing this in my future work as well.

References

[1] Trent Jaeger, Divya Muthukumaran, Joshua Schiffman, Yuqiong Sun, Nirupama Talele, and Hayawardh Vijayakumar. Configuring cloud deployments for integrity. In *Proceedings of the Proceedings of the Computer And Security Applications Rendezvous: Cloud and Security (C&ESAR 2012)*, November 2012.

[2] Divya Muthukumaran, Sandra Rueda, Nirupama Talele, Hayawardh Vijayakumar, Trent Jaeger, Jason Teutsch, and Nigel Edwards. Transforming commodity security policies to enforce clark-wilson integrity. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*, December 2012.

[3] Divya Muthukumaran, Sandra Rueda, Hayawardh Vijayakumar, and Trent Jaeger. Cut me some security. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration (SAFECONFIG 2010)*, October 2010.

[4] Sandra Rueda, Hayawardh Vijayakumar, and Trent Jaeger. Analysis of virtual machine system policies. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT 2009)*, June 2009.

[5] Joshua Schiffman, Thomas Moyer, Hayawardh Vijayakumar, Trent Jaeger, and Patrick McDaniel. Seeding clouds with trust anchors. In *Proceedings of the 2010 ACM Workshop on Cloud Computing Security (CCSW 2010)*, October 2010.

[6] Joshua Schiffman, Yuqiong Sun, Hayawardh Vijayakumar, and Trent Jaeger. Cloud verifier: Verifiable auditing service for iaas clouds. In *Proceedings of the IEEE 1st International Workshop on Cloud Security Auditing (CSA 2013)*, June 2013.

[7] Joshua Schiffman, Hayawardh Vijayakumar, and Trent Jaeger. Verifying system integrity by proxy. In *Proceedings of the 5th International Conference on Trust and Trustworthy Computing (TRUST 2012)*, June 2012.

[8] Hayawardh Vijayakumar and Trent Jaeger. The right files at the right time. In *Proceedings of the 5th IEEE Symposium on Configuration Analytics and Automation (SAFECONFIG 2012)*, October 2012.

[9] Hayawardh Vijayakumar, Guruprasad Jakka, Sandra Rueda, Joshua Schiffman, and Trent Jaeger. Integrity walls: Finding attack surfaces from mandatory access control policies. In *Proceedings of the 7th ACM Symposium on Information, Computer, and Communications Security (ASIACCS 2012)*, May 2012.

[10] Hayawardh Vijayakumar, Joshua Schiffman, and Trent Jaeger. A rose by any other name or an insane root? adventures in name resolution. In *Proceedings of 7th European Conference on Computer Network Defense (EC2ND 2011)*, September 2011.

[11] Hayawardh Vijayakumar, Joshua Schiffman, and Trent Jaeger. Sting: Finding name resolution vulnerabilities in programs. In *Proceedings of the 21st USENIX Security Symposium (USENIX Security 2012)*, August 2012.

[12] Hayawardh Vijayakumar, Joshua Schiffman, and Trent Jaeger. Process firewalls: Protecting processes during resource access. In *Proceedings of the 8th ACM European Conference on Computer Systems (EUROSYS 2013)*, April 2013.