Personal Anomaly Detection and Smart-Phone Security

Huijun Xiong Danfeng (Daphne) Yao
Department of Computer Science
Virginia Tech
Blacksburg VA 24060
huijun@cs.vt.edu, danfeng@cs.vt.edu

Lu Han Liviu Iftode
Department of Computer Science
Rutgers University
Piscataway NJ 08854
luhan@cs.rutgers.edu, iftode@cs.rutgers.edu

April 22, 2010

Abstract

Mobile devices increasingly become the computing platform for networked applications such as Web and email. This development requires strong guarantees on the system integrity and data security of mobile devices against malicious software (malware in short). This work introduces a new personalized anomaly detection approach that is able to achieve host security by modeling and enforcing the legitimate behavior characteristics of a human user. Specifically, we identify characteristic human-user behaviors (namely application-level user inputs via keyboard and mouse), developing protocols for fine-grained traffic-input analysis, and preventing forgeries and attacks by malware. Our solution contains a combination of cryptographic techniques, correlation analysis, and hardware-based integrity measures. Our evaluation is done in computers with real-world and synthetic malware. The uniqueness of this personalized anomaly detection technique is that it allows computer security to be realized without the need for continually monitoring ever-changing malware patterns.

Keywords: Network security, anomaly detection, smartphone, input, correlation.

1 Introduction

Most of existing malicious software (malware) detection solutions are effective in detecting *existing* malware and *known* attack patterns [11, 7, 8, 5, 6, 15, 12]. These approaches typically focus on analyzing the network traffic of potentially infected machines to identify suspicious network communication patterns. However, current defenses still leave much room for improvement. The reason is that malware constantly evolves to avoid detection – code used by Mariposa botnet changes every 48 hours, and thus their behaviors change accordingly. For example, although IRC is still the dominating botnet command and control protocol, recent studies have found that many botmasters are responding to detection systems by switching away from IRC to HTTP [10, 16]. Unlike IRC C&C traffic, HTTP traffic is usually allowed through firewalls and can be easily camouflaged to be used for covert channels. Sole reliance on following and leveraging bots' behaviors for detection requires continuous modifications in order to keep up with the newest development of botnets.

We view malicious bots or malware in general as ghost entities stealthily residing inside a human user's computer and interacting with the user's computing resources.

Key research challenges for host-based malware detection include: *how to select characteristic behavior features*, *how to prevent bot forgery*, and *how to make solution nonintrusive to users*. Certain features extracted from humans' computer interactions, such as click counts, email sizes, or chat message sizes, may not be characteristic enough and cannot be used to uniquely represent an individual. In addition, advanced malware may attempt to mimic human activities to spoof the legitimate user in the detection. Thus, the host used to collect behavior features should distinguish true events from fake events that are injected by malware in hope to circumvent the authentication system (i.e., fake events need to be identified). We propose to address both challenges by developing an advanced input-traffic correlation analysis framework.

In this poster, we describe our experience in designing host-based malware detection solutions utilizing user-input activities on personal computers, and illustrate how this approach can be applied to secure mobile personal devices such as smart phones. Instead of chasing the ever-changing malware behavior patterns, we propose a *personal anomay detection approach* that aims at enforcing the correct

Our key observation is that user-initiated outbound traffic all has corresponding human inputs, i.e., keystroke or mouse clicks [1]. Our goal is to block (malware) traffic that is not associated with legitimate user inputs. For both syntactic and semantic based analysis, we will extract and perform exact matching and more efficient approximated matching between the user inputs and contents in the outbound traffic, which is specific to an application, e.g., Web, Email, P2P file sharing.

2 Preliminaries

We propose to perform syntactic and semantic input-traffic correlation analysis for malware detection purpose in Section 3. In our preliminary study, we already implemented a TPM-based infrastructure, called TUBA integrity service, which cryptographically ensures the integrity of user input events and prevents malware injection of (fake) keystroke events [14]. We developed a *cryptographic provenance verification* technique for keystroke integrity verification and demonstrated it in a client-server architecture. *TUBA integrity service demonstrated the feasibility of 1) selecting keystrokes as characteristic of human behavior and 2) successfully preventing malware mimicking.* As it will become clear, in our proposed solution in Section 3, outbound traffic that cannot be mapped to legitimate user inputs is identified and flagged.

We propose to examine every single outgoing network segment in the correlation analysis, which means that no packet should circumvent our detection. In our preliminary study, we already implemented a host-based framework, called CompareView [17], which cryptographically ensures that all outbound traffic must flow through a single transport-layer checkpoint. We extended our cryptographic provenance verification technique to the host's network stack and implemented two kernel modules, *Sign* and *Verify* modules, that coordinate to ensure the network flow. TPM-based integrity service similar to TUBA's ensures the integrity of CompareView and prevents malware tampering on the framework itself. *CompareView demonstrated the feasibility of installing a single checkpoint on network stack for all (outbound) packets, e.g., at the Sign module, without worrying about being bypassed.* Our extensive experimental evaluation showed that overhead CompareView imposes to outbound traffic stream is minimal when transport-layer segment size is large (e.g., 64KB). In what follows, our correlation analysis may be performed at the Sign module of CompareView.

In next section, we describe an input-traffic correlation approach that naturally leverages and systematically expands on our above infrastructures to build up the foundation and develop advanced techniques for human-behavior driven malware detection.

3 Syntactic and Semantic Input-Traffic Correlation Analysis

Recently, researchers have attempted to perform simple timing-based correlation analysis between user inputs and outbound traffic [2], which, however, is vulnerable to sev-

[1] In what follows, user inputs refer to data that a human *physically* enters into a computer through keystroke events or mouse events, as opposed to the (formatted) input fields to a program (e.g., web application). The latter concept is typically used in existing program-analysis based security literature such as tainted inference for detecting SQL injection or cross-site scripting attacks [13].

eral attacks such as piggybacking attacks [2]. In another study Not-A-Bot [9], the authors were unable to perform application-specific input analysis, due to the use of virtual machine monitor (VMM) and the resulting technical difficulties in accessing and reconstructing dynamic application-level data.

In what follows, we describe our ongoing work on designing advanced and robust input-traffic correlation framework. We will syntactically and semantically examine outgoing contents and identify abnormal traffic by collecting two data streams, one for inputs and one for outbound network packets, and designing a correlation engine, which is a stand-alone program independent to the application. The analysis will be performed at CompareView's checkpoint on the transport layer.

3.1 Syntactic correlation analysis

We will first support syntactic matching between user inputs (e.g., URL) and outbound traffic (e.g., HTTP Request packet), e.g., exact or approximate matching between strings. For each outgoing transport layer segment, we will examine the application-layer payload [3] and extract the content, e.g., URL of a web object. We plan to adapt and compare several existing exact and approximate matching algorithms, such as edit distance, to our settings.

3.2 Semantic interpretation of user inputs

Meanings of user inputs need interpretation due to their transformation. For example, mouse clicks on a web page need to be translated to the actual hyperlinks assisted by the browser. Computation on user inputs is another type of transformation. For example, in P2P file sharing, a search keyword (e.g., Britney Spears) is entered by a user, the hash of which (e.g., $H(BritneySpears)$), is used to form a P2P search request for corresponding file. Also, browser may generate automatic HTTP requests for embedded objects without any user inputs. Thus, our approach is to perform application-specific semantic interpretation on user inputs and enforce outbound traffic with interpreted inputs. There are two main operations: *semantic interpretation* and *matching*. Matching operation between two strings (i.e., interpreted inputs and application-layer payload) will be similar to that of the above syntactic analysis.

To better analyze the impact of user inputs on network traffic, we adopt a graph approach for formalizing the *causal relationships* among inputs and subsequent *outbound* requests for plug-in objects that are direct or indirect results of inputs. The causal relationship is expected to form a tree-like structure where the root is the user input, which we call an *emission graph* [4]. The task of malware detection is to identify and block any network traffic that cannot be attributed to an emission graph rooted by certain legitimate user inputs. Our detection method need be valid even if the attacker hosts his own server.

In the Web environment, the construction of emission graphs requires the content analysis on web pages [5]. We will study graph properties of emission graphs, e.g., tree depth, degrees of nodes, and node types (leaf node vs. internal node), which represent normal or abnormal traffic structures. We will examine the impact of dynamic contents (e.g., randomized advertisement rotation) on the accuracy of tree construction and traffic classification. We will begin our investigation with static webpages and gradually move on to process more dynamic AJAX-types of contents that are prevalent today. Existing tools, such as Gecko [4] for manipulating web content such as JavaScript, will also be utilized for handling dynamic link creation in the construction of emission graphs. We note that our method does not require language-based static or dynamic analysis on JavaScript, which is more complex due to JavaScript's flexibility. Visualization tools for examining emission graphs dynamically in web environments can also be produced by utilizing existing graph drawing programs [1, 3]. We expect the above work to produce a powerful and general method that will enable the precise control over network activities of a host through leveraging human inputs.

The integrity to correlation analysis needs to be realized

[2] A piggybacking attack is where malware traffic is only sent out whenever a user has input activities. This attack is possible because of the ease of doing keylogging.

[3] Typical application layer payload can fit in one transport layer segment whose maximum segment size (MSS) is usually 64KB. Thus, defragmentation is not needed.

[4] An emission graph will be different from a DOM tree in browser as we focus on hierarchical relationships in the (materialized) outbound network connections, not the document objects

[5] Web page contents can be retrieved without the browser, e.g., wget, which we call out-of-band retrieval.

to detect and deter the tampering of the analysis framework itself. One approach is to extend our existing TPM-based attestation prototype [14].

A detailed description of our HTTP-anomaly-detection prototype implementation and evaluation in personal computers can be found in [18].

The above approaches have never been systematically investigated before. The main technical enabler in our study is our (established) ability to ensure the integrity of user input events and outgoing network traffic (described in Section 2).

4 Securing Android Phones with Input-Traffic Monitoring

We are currently working on applying our input-traffic technique to smart phones with Android operating system. We describe our work in progress in this section. Android is a software stack for mobile devices that includes an operating system, middleware and build-in applications. The Android is developed by Google and Open Handset Alliance (OHA) as an open source, lightweight and full featured platform, which allows developers to write managed code using JAVA programming language and control the smart device via necessary JAVA libraries and APIs. Android does not use established Java standards (i.e. JAVA SE or ME) so that it prevents compatibility.

4.1 Android Architecture and Functionalities

Figure 1 shows the five major components (in different colors) of Android platform: build-in applications, application framework, libraries, Android runtime, and open source Linux kernel.

Build-in Application The Android platform comes with a set of build-in applications include all necessary smart phone functionalities: SMS program, contact book, maps, email client, browser and others. The build-in applications are developed using Java as the user application, and can be replaced with user applications.

Application Framework The application framework is a software framework that implements the standard structure of an application in Android. It utilizes managers, content provider and other service modules to manage application life cycle.

Libraries The Android native libraries are written in C/C++ and used by various components of the Android system. The libraries can be called through the Android software framework. For example, the WebKit library is the open source web browser engine; the SQLite library is used to support structured data storage, etc.

Android Runtime There are two components in the Android Runtime module: core libraries of the Java programming language, and Dalvik Virtual Machine. Instead of using Java Virtual Machine (JVM), android utilizes Dalvik VM. Dalvik VM is a register-based VM which is designed to allow multiple VM instances to run efficiently, and optimized for low memory requirements.

Linux Kernel Linux kernel in Android acts as a hardware abstraction layer (HAL) between hardware and the rest of the software stack. Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model.

Nexus phone supports full smartphone functionalities such as GSM telephony, Bluetooth, EDGE, 3G, WiFi, Camera, GPS, compass and accelerometer. Android provides rich development environment for developers including a device emulator, debugging tools and a plug-in for Eclipse IDE for application development. Compared with Symbian based smartphones and IPhone, Android is supported by large number of associations (OHA) and provides full open source development environment for developers.

4.2 Intercept Network Traffic and User Inputs on Android

Android relies on Linux kernel for network stack management. Usually, in a Linux system, we utilize `tcpdump` (`libpcap` library) to capture the packets being transmitted or received over network. The libpcap is included in the Android fully build tree. This allows us to utilize libpcap/tcpdump to intercept the network traffic.

We are currently working on collecting user keyboard inputs and store the logged information on the mobile device. We note that media inputs such as voice and pic-

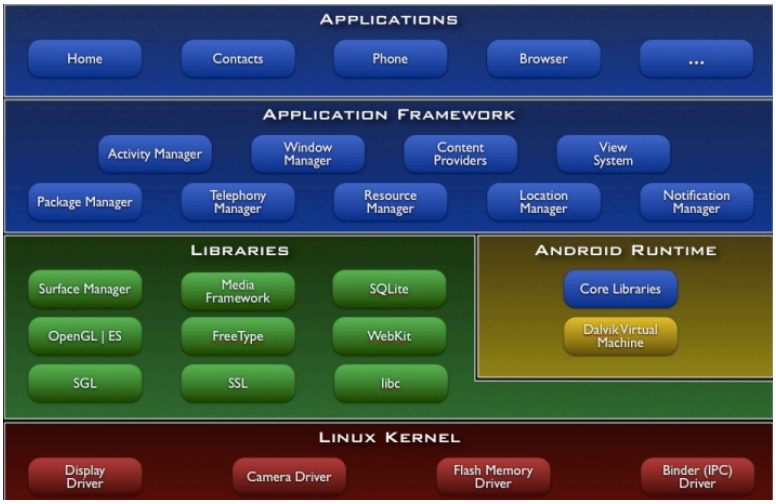

Figure 1: Components of Android software stack.

tures will not be collected by us. Android offers both the hardware keyboard and software keyboard (virtual keyboard) for users. The hardware keyboard is a `QWERTY` keyboard, the virtual keyboard usually supports three keyboard layouts: the QWERTY, compact QWERTY and T9. Therefore, we need to collect users' key strike, scroll and key touch from both physical and the virtual keyboard. Android provides rich UI components for human computer interactions. Various View classes are for users to compose the layout of programs. Callback methods are called by Android framework to handle UI events. Therefore, to capture user input is to capture the events from the specific View object. We can utilized these components to intercept user input by having the onKeyPressed or onTouchEvent events set, and override the event listener callback methods onKeyDown or onKeyUp to handle the events.

5 Summary

In this poster, we described a human-behavior driven malware detection approach, and gave feasible techniques for performing advanced input-traffic correlation analysis in both PCs and smart phones. Our proposed methods are to build on our existing TUBA infrastructure that ensures the provenance of user input events and prevents fake event injection, and CompareView infrastructure that enforces outbound traffic flow. We are currently implementing our detection system in Android developer phones. We plan to finish the implementation of our solution and perform extensive experiments on its accuracy, efficiency, and usability with both real-world and synthetic malware samples.

References

[1] N. S. Barghouti, J. Mocenigo, and W. Lee. Grappa: A Java graph package. In *Fifth International Symposium on Graph Drawing*, pages 336–343. Springer-Verlag, 1997.

[2] W. Cui, R. H. Katz, and W. tian Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *ACSAC*, pages 361–370. IEEE Computer Society, 2005.

[3] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz and dynagraph - static and dynamic graph drawing tools. *Graph Drawing Software*, 2003.

[4] Gecko. `https://developer.mozilla.org/en/Gecko`.

[5] J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In *Proceedings of First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[6] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer botnets: Overview and case study. In *Proceedings of First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[7] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[8] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[9] R. Gummadi, H. Balakrishnan, P. Maniatis, and S. Ratnasamy. Not-a-Bot: Improving service availability in the face of botnet attacks. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NDSI)*, 2009.

[10] N. Ianelli and A. Hackworth. Botnets as a vehicle for online crime, 2005. `http://www.cert.org/archive/pdf/Botnets.pdf`.

[11] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.

[12] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. My botnet is bigger than yours (maybe, better than yours). In *Proceedings of the First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[13] R. Sekar. An efficient black-box technique for defeating web application attacks. In *ISOC Network and Distributed Systems Symposium (NDSS)*, February 2009.

[14] D. Stefan and D. Yao. Keystroke dynamics authentication and human-behavior driven bot detection. Technical report, Rutgers University, 2009.

[15] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of First USENIX Workshop on Hot Topics in Understanding Botnets*, April 2007.

[16] S. Webb, J. Caverlee, and C. Pu. Predicting web spam with HTTP session information. In *Proceedings of the Seventeenth Conference on Information and Knowledge Management (CIKM 2008)*, October 2008.

[17] C. Wu and D. Yao. Compareview - a provenance verification framework for detecting rootkit-based malware. IEEE Symposium on Security and Privacy. Oakland, CA. May 2009. Extended Abstract.

[18] H. Xiong, P. Malhotra, D. Stefan, C. Wu, and D. Yao. User-assisted host-based detection of outbound malware traffic. In *Proceedings of International Conference on Information and Communications Security (ICICS)*, December 2009.