Ensuring Host Integrity With Cryptographic Provenance Verification[*]

Deian Stefan
Department of Electrical Engineering
The Cooper Union
New York, NY 10003
stefan@cooper.edu

Chehai Wu
AppFolio, Inc.
55 Castilian Dr.
Goleta, CA 93117
wuchehai@gmail.com

Danfeng (Daphne) Yao
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
danfeng@cs.rutgers.edu

Gang Xu
AT&T
200 Laurel Ave
Middletown, NJ 07748
gangxu@att.com

ABSTRACT

We propose a malware detection approach based on the characteristic behaviors of human users. We explore the human-malware differences and utilize them to aid the detection of infected hosts. There are two main research challenges in this study: one is how to select characteristic behavior features, and the other is how to prevent malware forgeries. We aim to address both questions in this poster.

Keywords: authentication, malware detection, cryptography, provenance, network

1. Introduction

Studies estimate that millions of computers worldwide are infected by malware and have become bots that are controlled by cyber criminals. The infected computers are coordinated and used by the attackers to launch malicious and illegal network activities, including perpetrating identity theft, sending spam (estimated 100 billion spam messages every day), launching distributed denial of service (DDoS) attacks, committing click fraud, as well as information warfare. A malicious bot is a specific type of malware, which is an umbrella term referring to malicious code including viruses, worms, rootkit, and spyware.

In this poster, we propose a different malware detection approach by focusing on the *characteristic behaviors of humans*. There are intrinsic differences between how a person and a bot uses and reacts to computer applications. For example, studies of online chatting behaviors of chat bots and humans have shown that bots and humans behave quite differently. These human-bot differences are furthermore utilized to distinguish humans from bots and aid the detection of infected hosts. Our ultimate goal is to design robust bot detection mechanisms that are difficult for future generations of botnets to circumvent. Several studies show that people have unique rhythms in many computer interactive activities. Research in keystroke dynamics has demonstrated that a person tends to type with characterizable patterns, which can be used as a biometric for authentication. Numerous research efforts have shown that people demonstrate unique patterns when they surf online. By investigating and utilizing unique user-behavior patterns, we may develop advanced bot detection solutions that are robust toward new generations of botnets.

There are, however, two main research challenges in this study: *how to select characteristic behavior features*, and *how to prevent malware forgery.* Certain features extracted from human-computer interactions, such as click counts or email 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 injected by malware in hope to circumvent the detection system (i.e., fake events need to be identified).

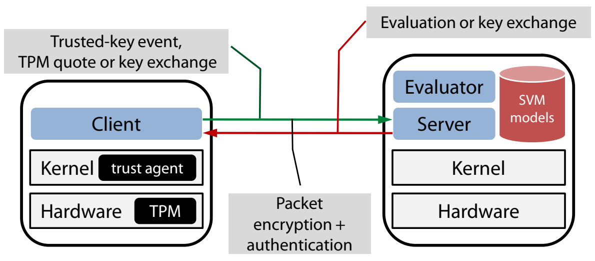
In this poster, we propose a cryptographic provenance verification approach, and demonstrate its applications in keystroke-based bot identification and rootkit traffic detection. Specifically, we present our design and implementation of a remote authentication framework called TUBA for monitoring a user's typing patterns and verifying their integrity. We then demonstrate our provenance verification approach in realizing a lightweight framework for blocking outbound rootkit-based malware traffic.

2. TUBA Integrity Service

In our preliminary study, we implement a TPM-based infrastructure, called *TUBA integrity service*, which cryptographically ensures the integrity of user input events and prevents malware injection or replay of (fake) keystroke

[*]This work has been supported in part by REU programs, NSF grant CCF-0728937, CNS-0831186, and the Rutgers University Computing Coordination Council Pervasive Computing Initiative Grant.

events [4]. We utilize our cryptographic provenance verification approach for keystroke integrity verification and realized it in a client-server architecture in Linux. We select keystrokes as characteristic of human behavior and show how to prevent malware mimicking of human inputs.



Trusted-key event, TPM quote or key exchange
Evaluation or key exchange
Client
Kernel
trust agent
Hardware
TPM
Evaluator
Server
SVM models
Kernel
Hardware
Packet encryption + authentication

Figure 1: Architecture of TUBA integrity service. Main operations include: trust agent and remote server key exchange; trust agent signs keystroke events; client relays signed events to the server; remote server also verifies kernel configuration.

A schematic drawing of our Telling hUman and Bot Apart (TUBA) integrity service architecture is shown in Figure 1. Our main design was having two channels attached to the remote TUBA server: plain keystroke events from a TUBA client and signed keystroke events from a module (i.e., trust agent) that was part of the kernel attested to using the TPM. The client-side trust agent and the remote trusted server shared a secret session key. The trust agent signs each keystroke event and the server verifies the signature. The server also monitors through TPM-based operations the client's integrity including its kernel and TUBA components. Our prototype realized the *trust agent* in kernel and a *trust client* in user-space as shown in Figure 1. The trust client is a program that forwards messages between the (kernel-level) trust agent and remote server. This integrity service can defend against advanced malware attacks including the replay of prior-captured user keystrokes, fake key-event injections, and tampering with TUBA client. Our prototype was implemented in a client-server architecture in Linux using the Intel Integrated TPM, following TPM Interface Specifications 1.2 [5, 6, 7] and a XTrap extension [1]. Details of the TPM-related operations including key establishment, bootup, and periodic verification of TPM quotes followed standard TPM specifications and are not described. The TUBA integrity service can also be applied to ensure the integrity of mouse events, e.g., mouse clicks. Preliminary performance results on the efficiency of the TPM-based integrity verification are omitted due to space limit, but can be found in [4].

3. Traffic Provenance Verification For Rootkit Detection

To further illustrate the generality of our host-based malware detection approach, we describe the design and implementation of a lightweight rootkit detection method. We detect stealthy outbound traffic of rootkits by enforcing a cryptographic provenance verification scheme on outgoing network packets. Rootkits that bypass normal user-mode network functions to send traffic are detected, as they are unable to provide their provenance proofs. We describe our experimental evaluation with real-world rootkits and throughput validation on upstream network traffic.

Rootkit is a type of malware that hides its presence from the hosting operating system, making it difficult to detect. Most malware constantly communicates with the outside world, with the intent of exporting sensitive data. Our detection is, recurrently, based on the observation that there are intrinsic differences between how a person and malware interacts with a computer. Legitimate outbound network traffic initiated by humans passes through the entire network stack in the host's operating system. In comparison, rootkit-based malware typically bypasses high-level inspection in the network stack by directly calling low-level network functions. Directly invoking data-link layer functions to send traffic is considerably hard in practice. These functions are also hardware-dependent. We explore the network stack and packet properties of outgoing traffic generated by humans and malware, and develop a robust cryptographic protocol for enforcing the proper *packet provenance on the network stack.*

Architecture of Traffic Provenance Verification: We provide an add-on to the host's network stack. It consists of a *Sign Module* and a *Verify Module*, as shown in Figure 2. The Sign Module is at the upper edge of the transport layer while the Verify Module is at the lower edge of the network layer. Thus, all legitimate network packets initially pass through the Sign Module and then the Verify Module. The Sign Module signs every packet and sends signatures as the packet provenance information to the Verify Module which verifies them. If a packet's signature cannot be verified, then it is labeled as suspicious, having bypassed the Sign Module, and likely generated by stealthy malware.

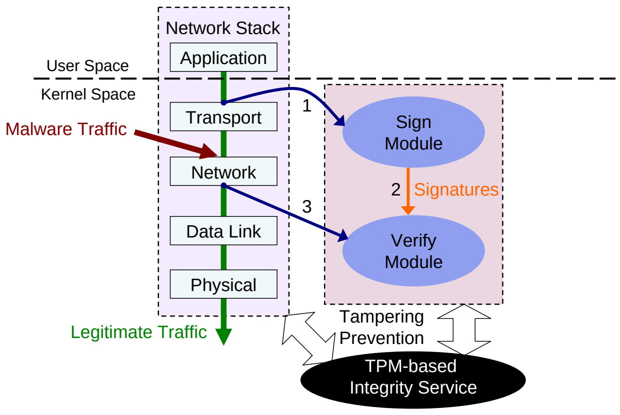


Figure 2: Schematic drawing of components in the framework and their interactions with the host's network stack. Legitimate traffic origins from application layer whereas rootkit traffic is injected into the lower layers.

We implement our rootkit detection tool in Windows XP. The Sign Module is realized as a TDI filter device at the upper edge of the transport layer in the Windows TCP/IP stack. All legitimate network packets from the Winsock

API is captured and signed by the Sign Module. The Verify Module is an NDIS intermediate miniport driver at the lower edge of the network layer. It intercepts and verifies all packets just before they are sent to network interface card drivers. In this prototype, the signature algorithm is UMAC (message authentication code using universal hashing) which is fast and lightweight.

Key Management and System Integrity: When the system starts up, the Sign Module and the Verify Module generate their public/private key pairs and notify each other of their respective public keys. Taking advantage of public key cryptography, the two modules securely exchange two symmetric keys; one is for signature generation and verification, while the other is used to encrypt signatures from the Sign Module to the Verify Module.

To ensure that the integrity of the detection framework and signing key secrecy, we utilize the on-chip TPM to generate the signing keys and to attest kernel and module integrity at boot. The approach is similar to that of TUBA, where the attestation of kernel and module integrity requires a remote trusted server. Enlisting a remote server for integrity purpose was also previously used in [2]. Although more complex, under certain assumptions of the secure storage and evaluation of attestation values, it is also possible to realize the integrity service on the same host (in a standalone architecture), details are omitted due to space limit. In comparison to the virtualization-based traffic detection approach by Srivastava and Giffin [3], our solution provides an effective cryptographic alternative that leverages the available trusted computing infrastructure.

Experimental Evaluation: We test our tool against a piece of proof-of-concept malware that can bypass the transport layer to send outgoing packets. Our experiments show that the Verify Module detects such an attack. However, the malware can disable URL filtering functionality of Trend Micro OfficeScan Client. An extended version of our detection implementation is able to identify real-world rootkits (weaker than our proof-of-concept malware), including `Fu_Rootkit`, `hxdef`, and `AFXRootkit`, all of which hide process information and opening ports. Figure 3 shows the network throughput with and without using our rootkit detection mechanism. With provenance verification on each packet, the throughout decreases in general. However, as the packet size grows (e.g., 64KB), the costs of signing and verification are amortized and the throughput approaches the ideal value. The observed performance degradation is minimal and acceptable in practice, since most PCs have low upstream traffic even with peer-to-peer applications running.

4. Future Work

We proposed a cryptographic provenance verification technique for host-based malware detection, and illustrated how it can be used to leverage human-malware differences in keystroke-based bot identification and rootkit traffic detection.

For future work, we plan to extend our cryptographic provenance verification approach to develop advanced input-traffic correlation and tracking analysis. In almost all client-server or pull architectures (e.g., web applications), users initiate the requests, which typically involve keyboard or mouse events. Few exceptions such as Web server refresh operations can be labeled using whitelists. We will investigate how to characterize and enforce normal traffic and input correlations in applications such as Web browsing and P2P file-sharing in the face of sophisticated malware exploits.

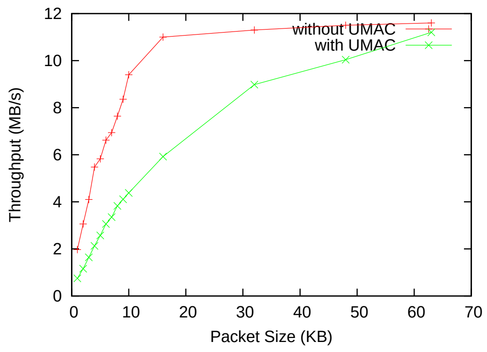


Figure 3: Performance comparison with or without the provenance-verification based traffic detection.

5. References

[1] D. Annicchiarico, R. Chesler, and A. Jamison. Xtrap architecture. *Digital Equipment Corporation, July*, 1991.

[2] A. Baliga, V. Ganapathy, and L. Iftode. Automatic inference and enforcement of kernel data structure invariants. In *24th Annual Computer Security Applications Conference (ACSAC)*, 2008.

[3] A. Srivastava and J. Giffin. Tamper-resistant, application-aware blocking of malicious network connections. In *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*, pages 39–58, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[5] October 2003. Trusted Computing Group. Security in mobile phones whitepaper.

[6] October 2003. Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 62.

[7] TCG PC Client Specific TPM Interface Specification (TIS), Version 1.2. Trusted Computing Group. `http://www.trustedcomputinggroup.org/groups/pc_client/`.