Securing Enterprise Networks Using Traffic Tainting

Anirudh Ramachandran, Yogesh Mundada, Mukarram Bin Tariq, and Nick Feamster
School of Computer Science, Georgia Institute of Technology
{avr,yhm,mtariq,feamster}@cc.gatech.edu

ABSTRACT

Enterprise networks are vulnerable to attacks ranging from data leaks to the spread of malware to insider threats. Previous defenses have largely focused on securing hosts; unfortunately, when hosts are compromised, these defenses become ineffective. Rather than attempting to harden the host against every possible attack (which is impractical) or constraining the software that can run on a host (which is inconvenient), we place a small amount of trusted code on the host to assist with tracking the provenance of network traffic, moving the rest of the trust and function to the network. We present *Pedigree*, a system that tracks information flow across processes and hosts within a network by annotating traffic with taints that reflect the process that generated the traffic and the inputs that process has taken (we call this function *traffic tainting*). A *tagger* on the host annotates network traffic with information about the "taints" that the sending process has acquired. Network devices act as *arbiters* to take appropriate actions (*e.g.*, blocking) based on the taints associated with the traffic and the enterprise network's security policy. We have implemented Pedigree's host-based tagger as a Linux kernel module and the arbiter using the OpenFlow platform. This demonstration presents a prototype deployment of Pedigree that identifies and prevents both sensitive data leaks and the spread of malware in a typical enterprise network setting. The demonstration will show that Pedigree can defend against these attacks without significant overhead at the host or the filtering device.

1. INTRODUCTION

Because enterprise networks often contain sensitive or valuable information, operators go to great lengths to secure them: these networks are often strongly partitioned from Internet traffic, and users and hosts within the network are subjected to strict security policies. Despite these measures, these networks remain vulnerable due to the inevitable bugs in applications, user carelessness, the heterogeneity of hosts, and insider threats. Enterprise networks continually fall victim to exfiltration (*i.e.*, leaks of confidential data from the network) and malware. Network administrators deploy a hodgepodge of solutions to mitigate these vulnerabilities, such as using watermarking techniques to prevent exfiltration, and deploying both antivirus tools on the host and deep-packet inspection devices in the network to control the spread of malware. These solutions are not effective in many cases. For example, watermarking can often be subverted, and antivirus tools, despite constant updates, often fail to detect polymorphic worms. This collection of point solutions begs the need for a general, low-overhead mechanism to mitigate these vulnerabilities without being specific to a few existing types of attacks.

To control how information flows within an enterprise and to protect the network against continually evolving threats, network administrators are faced with two approaches for defense: securing end hosts, and controlling network traffic. While securing end hosts provides some protection, it is not a panacea: First, applications can be susceptible to new ("zero-day") vulnerabilities, and malware may still find its way into enterprise hosts through compromised Web sites (*e.g.*, cross-site scripting attacks), email attachments, or by social engineering attacks. Second, enterprises are also vulnerable to insider threats, whereby an employee or other insider may steal or leak confidential information either intentionally or accidentally (*e.g.*, via a stolen laptop). Finally, end hosts are heterogeneous (*e.g.*, they may include hosts running different operating systems, PDAs, phones, etc.), so ensuring that every possible device on the network is patched is not practical. On the other hand, deploying firewalls and middleboxes after the fact requires operators to play a guessing game about how traffic should be controlled and filtered. We offer a different philosophy: End hosts should implement a simple OS-level function that allows the network to determine the *provenance* of the network traffic, and all security policy decisions should be enforced in the network.

In support of this philosophy, we demonstrate *Pedigree*, a distributed information flow tracking system that allows network devices within an enterprise to classify flows based on their features or "tags". A tag is a collection of identifiers (called "taints") that provide a history comprising the identifiers for the process that generated the information flow and other OS-level resources (processes or files) that may have affected the process. We refer to this history as the provenance of the traffic, and the process of annotating network traffic with these taints as *traffic tainting*. To prevent attacks with a general mechanism like *Pedigree*, we rely on an assumption that the user will not actively try to subvert his or her own system. This assumption is reasonable in most enterprise networks, where users or application programs are typically not allowed to modify or disable OS-security settings (*e.g.*, as an administrator or root user); thus, a small module added to the host OS can be reasonably assumed to be trusted. Section 2 describes how *Pedigree* uses a combination of host and network functions to enable this tracking. Section 3 concludes with a short description of the demonstration.

2. DESIGN AND IMPLEMENTATION

Traffic tainting presents significant scalability and implementation challenges. First, the number of taints that the system must track for each process can grow rapidly. Further, because there are a large number of ways that the information can flow among resources, *Pedigree* must perform gate-keeping in a way that allows it to track information flow in an efficient manner. Below, we briefly describe the design of the two main components in a *Pedigree*-enabled network: the *tagger* and the *arbiter*. Finally, we briefly overview the threat model for *Pedigree*.

2.1 Tagger

The tagger is a trusted component in the host operating system that is not under the user's control. Our prototype is implemented as a Linux kernel module using the Linux Security Modules framework [3]. The tagger monitors all events in the OS that involve information flow, and maintains and updates tags for all resources (processes, files, etc.) Tags of inactive resources (*e.g.*, unopened files) are stored on disk. Before processes send data to the network,

the tagger sends the latest tag associated with the sending process as a separate flow (called tagstream) to the remote machine. Similarly, before a process is allowed to accept a network connection, the tagger expects the remote host to send a tagstream that is incorporated into the listening process's local tag. The tagger's operation is thus transparent to applications.

The tagger creates a new identifier (called "taint") whenever a file is executed, based on a hash sum of the file's contents). All processes and files possess tags; when a process reads another resource (*i.e.*, other processes and files), the tagger updates the reading process's tags with the taints in the tag of the resource being read. Such information flow tracking is also possible for two machines communicating over the enterprise network: before a process sends data to the network, the tagger sends the process's current tag to the remote host as a separate flow (called a "tagstream"). The remote host's tagger incorporates taints in the incoming tagstream into the reading process's tag. Taggers on communicating hosts ensure that the latest taint set of each process is communicated to the other end; if one process's taint set changes during a connection, the tagger sends an update to the initial taint set to the remote host.

2.2 Arbiter

Tagstreams allow network elements (called *arbiters*) to monitor the potential provenance of the information contained in flows generated by machines within the enterprise. Thus all information flow within a host, as well as between hosts across the enterprise network, is accompanied by tags carrying the provenance of the information, which can be used for a number of useful applications, such as filtering information flows that are undesirable (*e.g.*, attacks or scans initiated by malware, leakage of secret information).

The arbiter is a network device that can process tagstreams and apply filter based on the taints in tagstreams. In our deployment, the arbiter is an OpenFlow-enabled switch that communicates with a controller machine running Linux. Tagstreams (which precede every connection) are sent to the controller who checks the tag for certain taints corresponding to malware or to secret data; if the tag contains such a taint, the controller instructs the switch to install a rule to block the flow to which the tagstream corresponds.

2.3 Threat Model

The threat model for *Pedigree* assumes that users do not actively try to subvert the system either by bypassing the tagger-enabled OS or by mounting sophisticated attacks that a normal user-space program cannot (*e.g.*, using a PCI card in master mode to directly access physical memory using DMA). Users, however, may unwittingly cause their computers to be infected by malware or may otherwise cause data leakage. Because users (and any programs they run) often have superuser access to their operating systems, the tagger must be protected from subversion even from superuser processes. *Pedigree* separates user-space administrative privileges from kernel administrative privileges using the Linux capabilities mechanism; in addition, *Pedigree* requires user-space processes—including administrative processes—to present a credential (typically a passphrase) to modify kernel parameters.

3. DEMONSTRATION

We present exfiltration prevention and preventing malware spread as two case studies of how *Pedigree* can filter traffic based on non-trivial information tracking. To prevent exfiltration, network administrators inject one or more "secret" taints into the tags of confidential files prior to distributing them within the enterprise. Because these taints are preserved irrespective of any operation performed by a process that reads the file (*i.e.*, re-formatting, encryption, etc.), they are not vulnerable to attacks that defeat watermarking schemes. Arbiter elements at the exit points of the enterprise network are equipped to drop all flows that carry the secret taints in their tags. *Pedigree* can also help stem the spread of malware: even if a malware executable infected one host *before* its details (such as the MD5 hash value of the executable) were known, this information will exist in all flows generated by the malware process; these taints will be preserved even if the malware is polymorphic. Operators can configure the arbiter to raise alarms for all flows that have one or more malware taints in their tagstream.

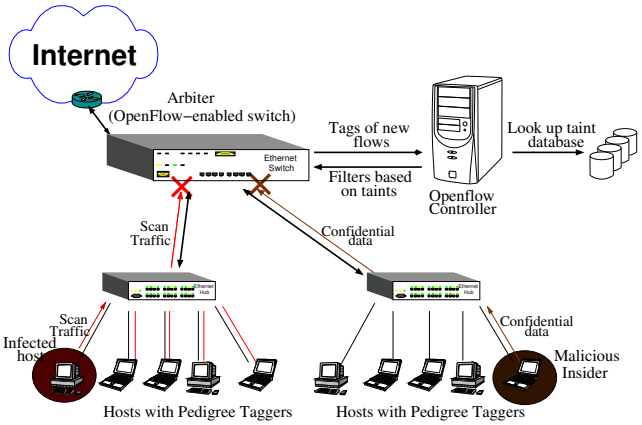

Figure 1: A typical Pedigree deployment in an enterprise network, where a malicious host is trying to scan its local network.

Figure 1 shows the setup of our demo. We first set up many *Pedigree*-enabled end-hosts on a switched network. The switch supports the OpenFlow [1] standard and is capable of performing filtering decisions at high speed; this switch and its controller (on a different end-host) comprise *Pedigree*'s arbiter. Through this demo, we wish to demonstrate the following goals.

1. *Pedigree* defends against a large class of attacks in enterprise networks such as malware spread and exfiltration without requiring updates to end-host software.
2. *Pedigree* performs filtering at line rate and imposes little overhead on host and network resources. In particular, the overhead of deploying the tagger at end-hosts is comparable to running an antivirus software.
3. *Pedigree* is resistant to a variety of evasion attacks including privilege escalation attacks and polymorphic worms.

Our technical report describes an earlier version of *Pedigree* [2].

REFERENCES

[1] N. McKeown et al. OpenFlow: Enabling Innovation in College Networks. `http://www.openflowswitch.org/documents/openflow-wp-latest.pdf`.

[2] A. Ramachandran, K. Bhandankar, M. B. Tariq, and N. Feamster. Packets With Provenance. Technical report, Georgia Tech, May 2008. Georgia Tech CSS Technical Report, available at `http://www.cc.gatech.edu/~avr/pedigree.pdf`.

[3] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proc. 11th USENIX Security Symposium*, San Francisco, CA, Aug. 2002.