BINDER: An Extrusion-based Break-In Detector for Personal Computers

Weidong Cui, Randy H. Katz, and Wai-tian Tan
University of California, Berkeley and Hewlett-Packard Laboratories
{*wdc,randy*}*@cs.berkeley.edu, dtan@hpl.hp.com*

Abstract

Compromised computers have been a menace to both personal and business computing. In this paper, we tackle the problem of automated detection of break-ins of new unknown threats such as worms, spyware and adware on personal computers. We propose Break-IN DEtectoR (BINDER), a host-based break-in detection system. Our key observation is that many break-ins make extrusions, stealthy malicious outgoing network connections. BINDER exploits a unique characteristic of personal computers, that most network activities are directly or indirectly triggered by user input. Since threats tend to run as background precesses and thus do not receive any user input, the intuition behind BINDER is that only threats generate connections without user input. By correlating outgoing network connections and processing information with user activities, BINDER can capture extrusions and thus break-ins.

1 Introduction

A variety of threats such as worms, spyware and adware, have affected both personal and business computing significantly. Remotely controlled bot networks of compromised systems are growing quickly [12] and represent a menace to today's computing infrastructure. Many research efforts [8, 10, 14] and commercial products [13, 18] have focused on preventing break-ins by filtering known exploits or unknown ones exploiting known vulnerabilities. To protect computer systems from new threats, these solutions have two requirements. First, some central entities must rapidly generate signatures of new threats after they are detected. Second, distributed computer systems must download and apply these signatures to their local databases in time. However, these requirements can leave computer systems with obsolete signature database unprotected from newly emerging threats. In particular, worms can propagate much more rapidly than humans can respond in terms of generation and distribution of signatures [11]. Instead of targeting at preventing infections, we focus on fast mechanisms for detecting break-ins after they occur that do not require a priori knowledge of exploit or vulnerability signatures. Such mechanisms can decrease the danger of information leak and protect computers and networks, and is complementary to existing prevention schemes.

Many threats send malicious outgoing network traffic that is usually happen unknown to users on the compromised personal computers. We refer to these *stealthy* malicious outgoing network activities as *extrusions*. The key feature of extrusions is that they are not triggered by user input. In contrast, most normal traffic in personal computers is initiated by users. Leveraging this anomaly of extrusions, we tackle the problem of automated detection of break-ins of new unknown threats such as worms, spyware and adware on personal computers in contrast to server computers. In this paper, we present Break-IN DEtectoR (BINDER), a host-based system that detects break-ins on personal computers by capturing extrusions. To capture extrusions, BINDER correlates outgoing network connections (initiated by the local computer) and process information with user activities (key strokes and mouse clicks). BINDER can detect certain kinds of break-ins after they occur without a priori knowledge of exploit or vulnerability signatures.

2 Design Overview

The main objectives we want to achieve for the BINDER design are: (1) *minimal false alarms*: this is the critical requirement for any intrusion detection system to be useful in practice; (2) *generality*: BINDER should work for a large class of threats without the need for signatures beforehand; (3) *small overhead*: BINDER must *not* use intrusive probing and affect the performance of the computers it sits on; (4) *security with open design*: we want

to design BINDER so threats cannot bypass it by knowing its scheme.

Patterns of network traffic and system calls have been used for intrusion detection [4, 6, 17]. To the best of our knowledge, BINDER is the first system to take advantage of a unique characteristic of personal computers: user-driven activities. By trusting the user input, BINDER simply detects extrusions of break-ins by determining they are unrelated to user actions. In Section 4, we discuss how BINDER can verify a user input is not faked or tricked by break-ins.

A natural approach for BINDER to take is to correlate network traffic with user input directly. However, a "smart" threat can bypass it by monitoring user input and sending traffic at appropriate times. To avoid this, BINDER also uses process information to limit the correlation. The intuition behind it is that only malicious processes of break-ins generate connections without user input. BINDER assumes that the boundary between processes is protected by the operating system. In other words, break-ins cannot inject their malicious code into other running processes and must run as independent processes. Although this may not be guaranteed until the next generation operating systems [7] are available, a large class of today's threats run as independent processes.

3 BINDER

BINDER detects break-ins by capturing extrusions. Instead of searching for conditions that can detect extrusions directly, BINDER looks for the cases where normal connections may be generated. This is in concert with our objectives of minimizing false alarms and detecting a large class of threats. By covering all normal cases, we can first control false alarms. Then, we can detect any threat that generates connections in a way that does not match any normal case.

3.1 Normal Connection Rules

In the following discussion, we use three kinds of events: user events (user input), process events (process start and process finish), and network events (connection request, data arrival and domain name lookup). A normal outgoing network connection of a process may be triggered either by user inputs directly (e.g., download a news web page after a mouse click) or indirectly (e.g., download an image file embedded in the news web page or refresh a news web page periodically after it is loaded) or by schedule (e.g., a system daemon or a software update check). These cases can be covered by three rules: (1) *intra-process* rule: a connection of a process may be triggered by a user input, data arrival or connection request event of the same process; (2) *inter-process* rule: a connection of a process may be triggered by a user input or data arrival event of *another* process; (3) *whitelisting* rule: a connection of a process may be triggered according to a rule in the whitelist.

To verify if a connection is triggered by the *intra-process* rule, we just need to monitor all user and network activities of each single process. However, we need to monitor all possible communications among running processes to verify if a connection is triggered by the *inter-process* rule. This contradicts our objective of small overhead. Instead, we use the following two rules to approximate it: (1) *parent-process* rule: a connection of a process may be triggered by a user input or data arrival event received by its parent process before it is created; (2) *web-browser* rule: a connection of a web browser process may be triggered by a user input or data arrival event of other processes. The *web-browser* rule cannot be covered by the *parent-process* rule because, when a user clicks a hyperlink in a window of a process, the corresponding web page is loaded by an existing web browser process if there is any. The advantages of this approximation are that (1) the two rules do not require more information than the *intra-process* rule; (2) they cover a dominant fraction of connections triggered by the *inter-process* rule. The *whitelisting* rule can be classified into three categories: system daemons, software updates and network applications automatically started by the operating system.

3.2 Extrusion Detection Algorithm

The extrusion detection algorithm is based on the *intra-process*, *parent-process* and *web-browser* rule as well as whitelisting. It is actually about detecting normal connections correctly. If a connection is not triggered by any of the normal connection rules and thus is detected as anomalous, it is treated as an extrusion. The main idea of the extrusion detection algorithm is to limit the delay from a triggering event to a connection request event. There are three possible delays for a connection request though some of them may not exist. For a connection request made by process P, we define the three delays: (1) D_{new}: the delay since the last user input or data arrival event received by the parent process of P before P is created; (2) D_{old}: the delay since the last user input or data arrival event received by P (note that the triggering event can be from any process if P is a web browser process according to the *web browser* rule); (3) D_{prev}: the delay since the last connection request to the same host or IP address made by P. For normal connections, D_{old} and D_{new} are in the order of seconds while D_{prev} is in the order of minutes. Depending on how a normal connection is triggered, it must have at least one of the three

delays fall into a normal range.

The extrusion detection algorithm needs 3 parameters for the upper bound of the delays (defined as D_{new}^{upper}, D_{old}^{upper}, and D_{prev}^{upper}). We also assume BINDER can learn rules from previous false alarms. Each rule includes an application name (the image file name of a process) and a remote host name. The rule means any connection to the host made by a process of the application is always normal. Thus, given a connection request, it is a normal connection if it is in the rule set of previous false alarms, or is in the whitelist, or D_{prev} exists and is less than D_{prev}^{upper}, or D_{new} exists and is less than D_{new}^{upper}, or D_{old} exists and is less than D_{old}^{upper}. Otherwise, it is an extrusion.

3.3 Why BINDER Works

In this section, we discuss why connections made by a large class of threats can be detected as extrusions by the detection algorithm. When a threat breaks into a personal computer, the break-in can be split into two phases by the time of the first restart of the victim computer. The difference of these two phases is how malicious processes of a threat are started.

In the first phase, malicious processes are started either by an existing infected process or by a user. Connections made by those processes may not be initially detected as extrusions because, before they are started, their parent processes may have received user input (e.g., a user opens a virus attachment in an email client program) or data (e.g., a vulnerable process receives malicious traffic). However, BINDER can detect a break-in by observing even a single extrusion it makes. The chance of detecting a single extrusion is high unless an attacker crafts the exploit intentionally to evade BINDER in this phase.

In the second phase, malicious processes are started by the operating system without any user input or data arrival in history. Moreover, threats tend to run as background processes to avoid being detected or shutdown by computer users. A feature of background processes is that they do not receive any user input. BINDER can detect the break-in by capturing the first connection made by its malicious processes as an extrusion. This is because their parent processes do not receive any user input before they are created, and they do not receive any user input after it. So D_{old}, D_{new} and D_{prev} do not exist for such a connection. Therefore, BINDER can be guaranteed to detect break-ins of worms, spyware and adware after the victim computer is restarted.

4 Countermeasures and Solutions

When BINDER's scheme is known to attackers, there are potential countermeasures that allow break-ins to send malicious traffic without being detected. Though we try to investigate all possible attacks against BINDER, we cannot argue that we have considered all possible vulnerabilities. To evade BINDER's detection, a break-in can (1) stop or remove BINDER from the compromised host; (2) fake user input by using system APIs or tricking a user to input on its window; (3) fake data arrivals by keeping receiving data from a collusive site on a connection triggered by a user; (4) fake whitelist applications by replacing their executables; (5) use covert channels to leak information. To eliminate these countermeasures, we can (1) run BINDER at kernel level and monitor its running status; (2) monitor related APIs and detect pop-up windows of processes that do not receive any user input; (3) add more constraints on how a normal connection may be triggered; (4) use the scheme proposed in [1]; (5) use techniques presented in [3].

5 Related Work

Many research efforts [8, 10, 14] and commercial products [13, 18] have focused on preventing break-ins by filtering known exploits or unknown ones exploiting known vulnerabilities. Anomaly-based intrusion detection [4, 6, 17] have been studied for detecting unknown exploits. The performance of anomaly-based approaches is very limited in practice due to its high false positive rate [2]. Computer worms and spyware [9] have been a menace to both personal computing and large networks. Fast worm detection and containment becomes critical since worms can propagate much more rapidly than human response [11]. Most research efforts [5, 15, 16] have focused on random scanning worms. Compared to previous work, BINDER has three features: (1) it detects break-ins without a priori knowledge of exploit or vulnerability signatures; (2) it leverages the unique characteristics of personal computers that most normal traffic is triggered by users to achieve minimal false positives; (3) it detects a large class of threats that infect personal computers.

6 Future Work

We are implementing a prototype of BINDER on Windows operating systems because they are the largest group attacked by the most threats [12]. We plan to evaluate BINDER in two environments. In a real world environment, we want to install BINDER on computers used for daily work and evaluate its performance on both false positives and false negatives. In a controlled testbed, we want to test BINDER with real world email worms to evaluate its performance on false negatives. We focus on email worms because it is harder for BINDER to detect

their break-ins due to the fact that they are usually triggered by user input. Our preliminary results show that BINDER can limit false alarms to once per week and detect break-ins of real world email worms and spyware successfully.

7 Conclusions

In this paper, we present the design of BINDER, a host-based system that detects break-ins of worms, spyware and adware on personal computers by capturing extrusions. BINDER takes advantage of a unique characteristic of personal computers: user-driven activities. By trusting the user input, BINDER simply detects break-in extrusions by determining they are unrelated to user actions. This implies a new direction for tackling the problem of intrusion detection on personal computers.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, Erich Nahum, for their valuable comments and suggestions. We are grateful to Minghua Chen, Yanmei Li, Mukund Seshadri, Rui Xu, Fang Yu, Haibo Zeng, Jianhui Zhang and Wei Zheng for their generous help of allowing us to install and evaluate BINDER on their computers. We are thankful to Dan Ellis, Jaeyeon Jung, Jon Kuroda and Zhenmin Li for sharing virus emails with us. We would like to thank Chris Karlof, Yaping Li and Zhi-Li Zhang for their insightful comments on a draft of this paper. Special thanks go to Vern Paxson, David Wagner, Nicholas Weaver and Li Yin for their helpful discussion and valuable feedback.

References

[1] A. Apvrille, D. Gordon, S. Hallyn, M. Pourzandi, and V. Roy. Digsig: Run-time authentication of binaries at kernel level. In *Proceedings of LISA*, November 2004.

[2] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of ACM CCS*, November 1999.

[3] K. Borders and A. Prakash. Web tap: Detecting covert web traffic. In *Proceedings of ACM CCS*, October 2004.

[4] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6(3):151–180, 1998.

[5] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *2004 IEEE Symposium on Security and Privacy*, May 2004.

[6] W. Lee and S. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.

[7] Microsoft, Next-Generation Secure Computing Base. http://www.microsoft.com/resources/ngscb/default.mspx.

[8] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[9] S. Saroiu, S. D. Gribble, and H. M. Levy. Measurement and analysis of spyware in a university environment. In *Proceedings of NSDI*, March 2004.

[10] Snort, The Open Source Network Intrusion Detection System. http://www.snort.org/.

[11] S. Staniford, V. Paxson, and N. Weaver. How to own the Internet in your spare time. In *Proceedings of the 11th Usenix Security Symposium*, August 2002.

[12] Symantec Internet Security Threat Report. http://enterprisesecurity.symantec.com/content.cfm?articleid=1539, September 2004.

[13] Symantec Norton Antivirus. http://www.symantec.com/.

[14] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of ACM SIGCOMM*, August 2004.

[15] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th Usenix Security Symposium*, August 2004.

[16] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. Technical Report Technical Report HPL-2002-172, HP Labs Bristol, 2002.

[17] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[18] ZoneAlarm. http://www.zonelabs.com/.