;login:

THE USENIX MAGAZINE

December 2003 • volume 28 • number 6

Focus Issue: Security

Guest Editor: Rik Farrow

inside:

USENIX

The Advanced Computing Systems Association

sebek

Covert Glass-Box Host Analysis

by Edward Balas

As a network security researcher at Indiana University's Advanced Network Management Lab and Honeynet Project team member, Edward Balas has focused on network infrastructure protection. Edward's professional interests include Network Monitoring and Traffic Analysis, as well as advanced honeynet data capture techniques.

ebalas@iu.edu

Introduction

To defeat your enemy you must know your enemy. For individuals who run networks or network services, anyone who attempts to gain or deny access in an illegitimate manner may be considered an enemy. One tool that allows us to learn about this enemy is the honeypot.

A honeypot is a host whose value lies in being compromised by an intruder (see *http://project.honeynet.org/papers* for more details). A "high-interaction honeypot" is nothing more than a regular host that is closely monitored; as an intruder breaks in, the researcher monitors the actions of the intruder on the honeypot.

The key to the honeypot concept is the capturing of intruder activities. For such data to be of use it is critical that an intruder not detect that his or her actions are being captured. If captured correctly this data allows one to identify the tools, techniques, and motives of the intruder.

Today, packet captures using libpcap are the most common data capture technique. Tools that use this technique include Snort, ethereal, p0f, and many more. However, the increased use of session encryption has made packet captures increasingly inadequate for observing attackers. In response to this trend, the Honeynet Project has developed a new tool called Sebek for the circumvention of such encryption. This paper will be an introduction to the Sebek data capture system and the broader impacts of this new type of forensic data.

The Goals of Data Capture

For any data capture technique, we want to determine information such as when an intruder broke in, how they did it, and what they did after gaining access. This information can, potentially, tell us who the intruder is, what their motivations are, and who they may be working with. Specifically there are two very important things we want to recover: the attacker's interactions with the honeypot, such as keystrokes, and any files copied to the honeypot.

Data Capture Techniques and Their Limits

When encryption is not used, it is possible to monitor the keystrokes of an intruder by capturing the network activity off of the wire and then using a tool like ethereal (which is excellent for this work) to reassemble the TCP flow and examine the contents of the session. This technique yields not only what the intruder typed but also what the user saw as output. Stream reassembly techniques provide a nearly ideal method to capture the actions of an intruder when the session is not encrypted. When the session is encrypted, stream reassembly yields the encrypted contents of the session. To be of use these must be decrypted. This route has proven quite difficult for many. Rather than trying to break the encryption of a session, we have looked for a way to circumvent encryption.

Information that is encrypted must at some point be decrypted for it to be of any use. The process of circumvention involves capturing the data post-decryption. The idea is to let the standard mechanisms do the decryption work, and then gain access to this unprotected data.

The first attempts to circumvent such encryption took the form of trojaned binaries. When an intruder broke into a honeypot, he or she would then log into the compromised host using encrypted facilities such as SSH. As they typed on the command line, a trojaned shell binary would record their actions.

To counter the threat posed by trojaned binaries, intruders started to install their own binaries. It became apparent that the most robust capture method involved accessing the data from within the operating system's kernel. When capturing data from within the kernel, the intruder can use any binary they wish, and we are still able to record their actions. Further, because user space and kernel space are divided, there is ample opportunity to improve the subtlety of the technique by hiding our actions from all users, including root.

The first versions of Sebek were designed to collect keystroke data from directly within the kernel. These early versions were the equivalent of a souped-up Adore rootkit that used a trojaned sys_read call to capture keystrokes. This system logged keystrokes to a hidden file and exported them over the network in a manner to make them look like other UDP traffic, such as NetBIOS. This system allowed users to monitor the keystrokes of an intruder, but it was complex, it was easy to detect through the use of a packet sniffers, and it had a limited throughput. The latter made it difficult to record data other than keystrokes.

The next and current iteration of Sebek, version 2, was designed not only to record keystrokes but all sys_read data. By collecting all data, we expanded the monitoring capability to all activity on the honeypot, including keystrokes and secure file transfers. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages.

The Sebek Design

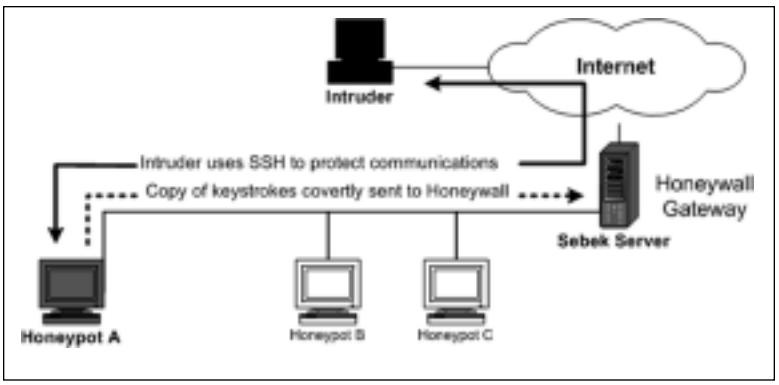

Figure 1

Sebek has two components, a client and server. The client captures data off of a honeypot and exports it to the network, where it is collected by the server (see Fig. 1). The server collects the data from one of two possible sources. The first is a live packet capture from the network; the second is a packet capture archive stored as a tcpdump formatted file. Once the data is collected, it is either uploaded into a relational database or the keystroke logs are immediately extracted.

Client Data Capture

Data capture is accomplished with a kernel module, which allows us access to the kernel space of the honeypot. Using this access, we then capture all read() activity. Sebek does this by replacing the stock read() function in the system call table with a new one. The new function simply calls the old function, copies the contents into a packet buffer, adds a header, and sends the packet to the server. The act of replacing the stock function involves changing one function pointer in the system call table.

When a process calls the standard read() function in user space, a system call is made. This call maps to an index offset in the system call table array. Because Sebek modified

the function pointer at the read index to point to its own implementation, the execution switches into the kernel context and begins executing the new Sebek read call. At this point Sebek has a complete view of all data accessed with this system call. This same technique could be used for any system call that we may wish to monitor.

Data that remains encrypted is of little use; to view the data or act on it in some way it must be decrypted. In the case of an SSH session, the keystrokes are decrypted and sent to the shell to have actions performed. This act typically involves a system call. By collecting data in kernel space, we can gain access to the data within the system call, after it has been decrypted but before it has been passed to the process that is about to use it. Thus we circumvent the encryption and capture the keystrokes, file transfers, Burneye passwords, etc.

To make the presence of the Sebek module less obvious, we borrow a few techniques used in modern LKM-based rootkits such as Adore. Because Sebek is entirely resident in kernel space, most of the rootkit techniques no longer apply; however, hiding the existence of the Sebek module is one example of direct technological benefit derived from its rootkit heritage. To hide the Sebek module we install a second module, the cleaner. This module manipulates the linked list of installed modules in such a way that Sebek is removed from the list. This is not a completely robust method of hiding, and techniques for detecting such hidden modules do exist.[1]

There are two side effects of this removal: Users can no longer see that Sebek is installed and, once it is installed, they are unable to remove the Sebek module without rebooting.

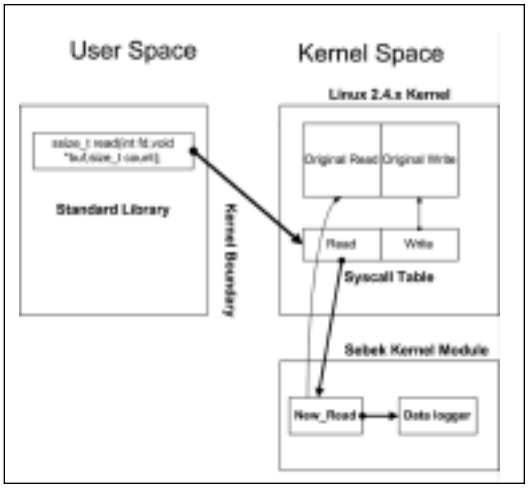

Figure 2

1. Phrack issue 61 has an article on detecting hidden kernel modules in its Linenoise section. The article describes a brute-force method for detecting hidden modules by looking for what appears to be the key module structure.

Client Data Export

Once the Sebek client captures the data, it needs to send the data to the server without being detected. If Sebek were simply to send the data to the server over a UDP connection, an intruder could simply check for the presence of such traffic on the LAN to determine whether Sebek was installed. Sebek does send data to the server using UDP, but first it modifies the kernel to prevent users from seeing Sebek packets, not just the packets generated by the local host, but any appropriately configured Sebek packet. Next, when Sebek transmits data onto the network, it ensures that the system cannot block the transmission or even count the packets transmitted.

Because Sebek generates its own packets and sends them directly to the device driver, there is no ability for a user to block the packets with iptables or monitor them with a network sniffer. This prevents an intruder on a honeypot from detecting the presence of Sebek by examining the LAN traffic.

The Broader Impact

Not too long after development, it became clear that not only was Sebek allowing us to circumvent encryption, but it was providing a previously unavailable source of data. Sebek was allowing us to look at the honeypot as a glass box rather than a black box. It was easy to monitor the keystrokes of an intruder, but we could also observe the actions of applications that never send data over the network. We initially tried to filter such data, but eventually we realized that such data could help researchers understand the intention and functioning of an unknown and potentially hostile binary installed on a system.

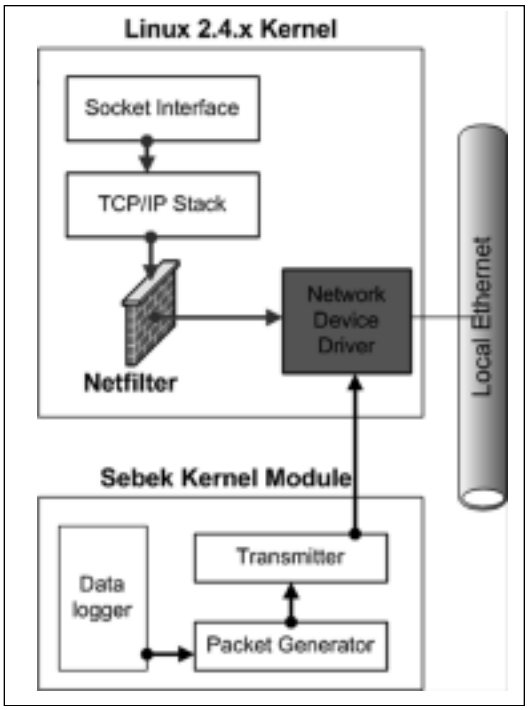

Figure 3

Once these techniques trickle down to the least skilled form of attacker, the script kiddie, it is anticipated that checking for Sebek on a compromised host will be common practice.

We haven't been the only ones to take notice of the potential power of this monitoring technique. Recently, a paper was published on a site purporting to be affiliated with the computer security group Phrack. This paper not only covered the risks associated with running honeynets, but also provided techniques used to detect and disable Sebek. (The honeynet site has a mirrored copy of the article at *http://www.honeynet.org/papers/honeynet/anti-honeypots.txt.*) Once these techniques trickle down to the least skilled form of attacker, the script kiddie, it is anticipated that checking for Sebek on a compromised host will be common practice. The most common technique involves installing a kernel module that attempts to reset the system call table.

The Future

In the near future, the primary goal is to ensure that Sebek is stable and can identify or perhaps even withstand attempts by attackers to disable it. The second priority will be Sebek data analysis. Within the Sebek data we see repeating patterns that are indications of illegitimate privilege escalation. Just as network-based intrusion detection systems examine libpcap data for patterns that represent known bad events, it is anticipated that an IDS based on Sebek data could be developed to detect bad events at the host level. Further, for a few situations it may be that such rules can be contained with the Sebek client, and when the client detects such a situation it would cause the kernel to take remedial action. This would be equivalent to a host-based intrusion prevention system.

Summary

Sebek is a kernel-based data capture tool that was originally designed to covertly monitor activity on a honeypot. Sebek circumvents encryption by capturing the activity in kernel space, where it is in an unencrypted form. Because of this we can capture keystrokes, recover passwords, and monitor any communication including IRC chats, email, and SSH/SCP activity.

Sebek allows an excellent view into the internal activities on a honeypot. Its methodology has not only provided a way to circumvent session encryption but also captures an entirely new type of data. This new data type may lead to the development of new technologies that will help make general-purpose systems more secure.

More information on the Sebek data capture system can be found at *http://www.honeynet.org/tools/sebek/*.