Know Your Enemy: Sebek

A kernel based data capture tool

The Honeynet Project
http://www.honeynet.org
Last Modified: 17 November 2003

Introduction:

Back at Berkley in the 1980's, Cliff Stoll quickly ran into a problem that still vexes honeypot researchers today: How do you observe an intruder without them noticing? In Cliff's situation he was able to directly monitor serial lines to capture the keystrokes of the intruder[1]. Much has changed since then, however our fundamental challenge has remained the same, how can we capture the activities of an intruder without them knowing? It is critical that a honeypot designed for the purposes of observing human intruders looks and behaves as any normal system, thus we must ensure that the intruder cannot detect that he or she is being observed. The traditional approach is to capture these activities by recording the associated network data. This is a desirable approach because it can be done in such a way as to be invisible to the intruder.

To nobody's surprise, Blackhats haven't stood still. It is increasingly common for even the most basic intruders to utilize encryption to protect their communications channels. If encryption services are not available on the victim host, it is not uncommon for them to install their own trusted services such as SSH, an encrypted GUI client, or SSL. Without the key, network based data capture tools are unable to view this channel.

To observe intruders using session encryption, researchers needed to find a way to break the session encryption. For many organizations this has proven extremely difficult. In an attempt to circumvent session encryption rather than break it, the Honeynet Project began experimenting with using kernel-based rootkits for the purpose of capturing the data of interest from within the honeypot's kernel. These experiments lead to the development of a tool called Sebek. This tool is a piece of code the lives entirely in kernel space and records either some or all data accessed by users on the system. It provides capabilities to: record keystrokes of a session that is using encryption, recover files copied with SCP, capture passwords used to log in to remote system, recover passwords used to enable Burneye protected binaries and accomplish many other forensics related tasks. This paper discusses Sebek version 2 which will be referred to as Sebek.

What follows is a detailed discussion of Sebek, how it works and its value. We will examine the architecture and key components. From there, we will drill down into the implementation issues and technical details of operation. Finally, we will show a usage example demonstrating the use of the Sebek including its new web interface.

The Sebek development cycle starts on a Linux system. Sebek is ported to other Operating Systems such as Win32, Solaris, and OpenBSD after it working on Linux. As a result, this paper, and its examples, will be based on the Linux version of Sebek, however many of the concepts discussed here apply to the other ports as well.

[1] The Cuckoo's Egg, Cliff Stoll. Pocket Books Nonfiction, 1990. New York, NY.

Sebek Overview:

Sebek is a data capture tool. As with all data capture tools, the goal is to capture data that will allow us to accurately recreate the events on a honeypot. 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. To determine what an intruder did after gaining access, we often want data that provides the intruder's keystrokes and the impact of the attack.

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 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. Steam 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, others 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 allows users to monitor the keystrokes of an intruder, but it was complex, easy to detect through the use of a packet sniffers and it had a limited throughput. This last issue 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, but not limited to, keystrokes. 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. A secondary goal was to make Sebek harder to detect, we focused on switching from obfuscating the logging traffic, to completely hiding it from a Blackhat. Now when a Blackhat runs a sniffer to detect suspicious traffic he or she is unable to detect any Sebek traffic.

Sebek is not just an alternative to TCP session reassemble, to be used only in the face of encryption. Sebek also provides the ability to monitor the internal workings of the honeypot in a glass-box manner, as compared to the previous black-box techniques. If an intruder where to install a piece of malware, then log out, we can now track the local actions of the malware even if it does not access the network.

The Architecture:

Sebek has two components: a client and server. The client captures data off of a honeypot and exports it to the network where is collected by the server (refer Figure 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. The communications used by Sebek are UDP based and as such are connectionless and unreliable.

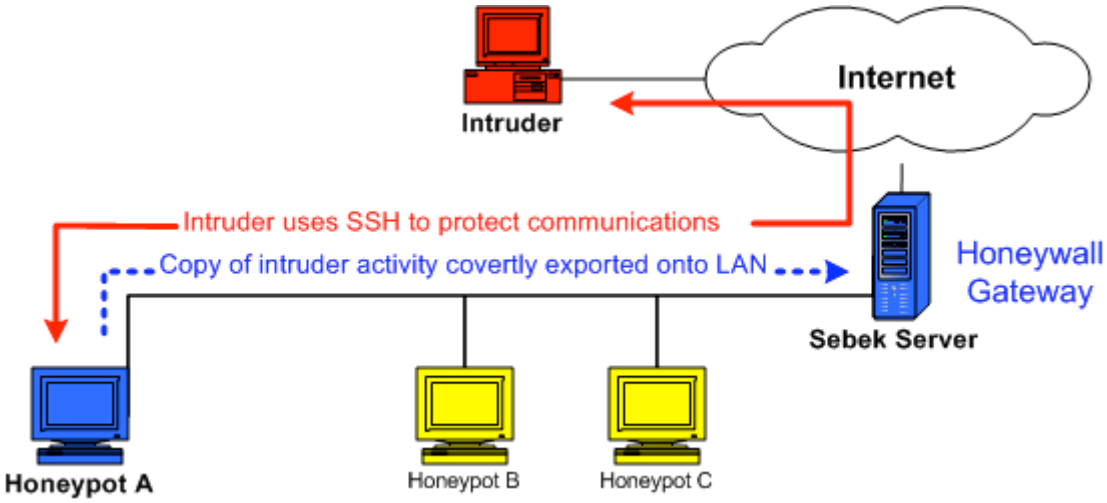

Figure 1
Typical Sebek deployment. The client module is installed on the honeypot (in blue). The attacker's activity captured by the honeypot is dumped to the wire (hidden to the attacker) and passively collected by the Honeywall Gateway.

The client resides entirely in kernel space on the honeypot and, in the case of the Linux version, is implemented as a Loadable kernel Module (LKM). The client can record all data that a user accessed via the read() system call. This data is then exported to the server over the net in a manner that is difficult to detect from the honeypot running Sebek. The server then gathers the data from all of the honeypots sending data. Because there is a standard platform independent log format the server can collect from any honeypot independent of Operating System type. Let us now take a closer look at how the client actually captures the data.

Client Data Capture:

Data capture is accomplished with the use of a kernel module. With this module we gain access to the kernel space of the honeypot. Using this access, we then capture all read() activity / data. 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.

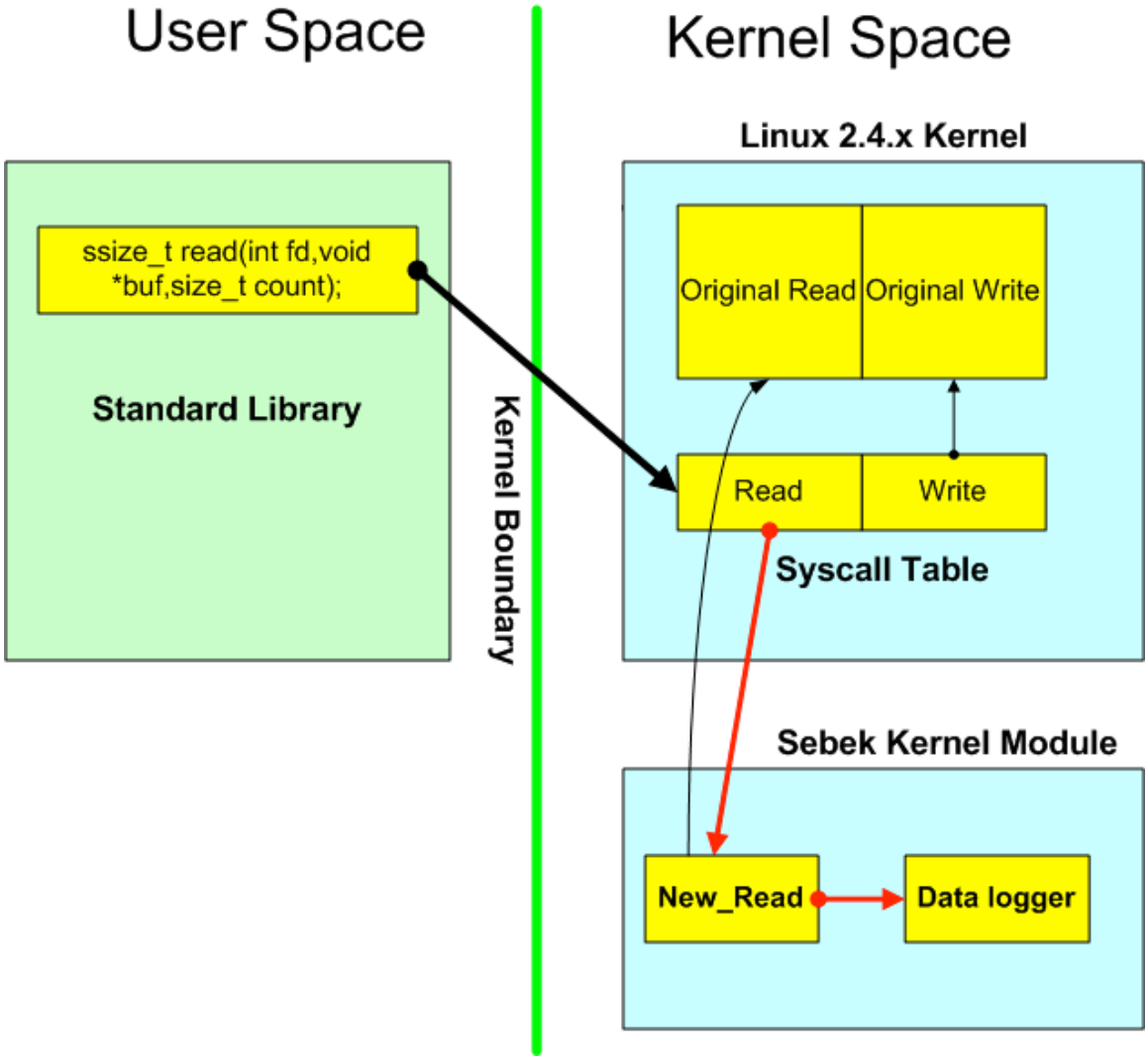

Figure 2
The conceptual representation of read system call redirection.

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 complete view 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 send 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.

Client Module Hiding:

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 now 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. 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.[2]

There are 2 side effects of this removal. First, users can no longer see that Sebek is installed. Second, once installed, users are unable to rmmod the Sebek module. This hiding ability can be disabled by setting the "Testing" variable to "1" in the sbk_install.sh install script.

Client Packet Export:

Once the Sebek client captures the data, it needs to send the data to the server without the intruder detecting that the host is sending this data. Although using the LAN network to send traffic to the server is not the most secure communication channel, it was decided that because of its ubiquity, it would be used to send data to the server. If Sebek were to simply send the data to the server over an UDP flow, an intruder could simply check for the presence of such traffic on the LAN to determine if Sebek was installed. Sebek does send data to the server using UDP, however, before it does this it modifies the kernel in a few ways to prevent users from seeing these packets. First it modifies the kernel such that system is unable to see 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. We will get into the details of packet hiding shortly.

If every honeypot on a LAN has Sebek installed, none of them can see any Sebek data, however the server has full access to this data. By deploying in this manner we have created a covert channel that allows the server gain access to the data captured by the client. For every read() request, Sebek generates one or more log packets. Each packet contains a bit of information about the context of the call made and the data that was accessed with the call. Each packet contains one Sebek record. The record contain fields that describe the Process that made the call, the time the call was made and the length of the data recorded as well as the data itself. These packets are generated entirely within Sebek without using the TCP/IP stack to generate or send the packets. Because of this, the system is unable to see or block the packets. After each packet is built, it is sent directly to the device driver, bypassing the raw socket code path as well as the packet filtering code path. Since packet sniffers are libpcap based and libpcap uses the raw socket interface to collect packets, sniffers installed on a host running Sebek are unable to see the Sebek packets generated.

[2] 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.

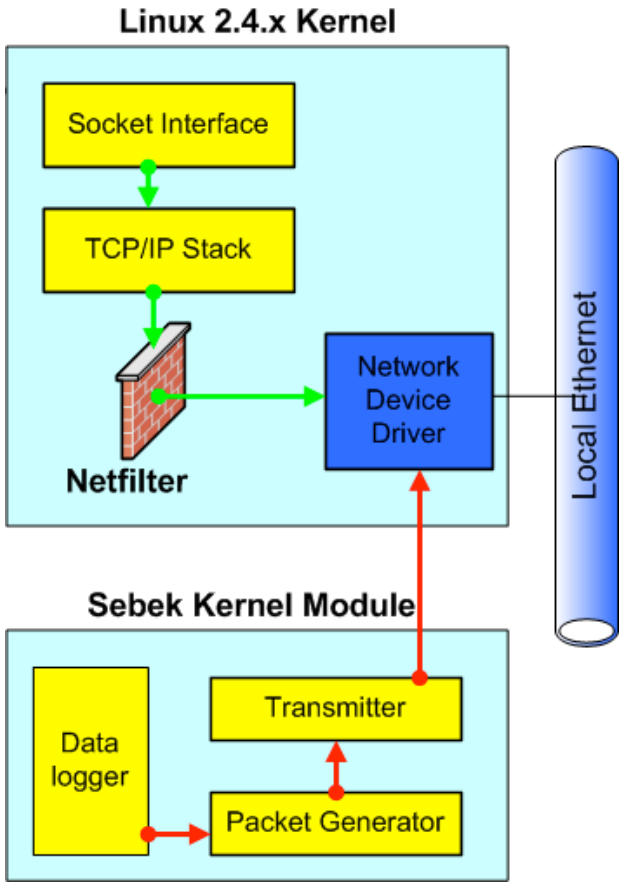

Figure 3
Conceptual representation of Sebek packet generation. Note how packets created by Sebek bypass the stack and go directly to the network device driver. This makes it far more difficult for attacker to detect Sebek activity.

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. A secondary problem that must be addressed is the need to keep honeypot A from detecting Sebek packets from honeypot B. The use of Ethernet switching does not solve this problem. Sebek is naturally impervious to ARP spoofing because it does not use ARP to obtain the destination MAC address that corresponds to the destination IP address, however there are a couple of situations where A would see B's Packets[3]. In these situations, an intruder would be able to detect the presence of Sebek packets on the LAN by running a sniffer on honeypot A and would see honeypot B's Sebek packets.

To solve this problem, Sebek installs its own implementation of the Raw Socket interface. This new version is programmed to silently ignore Sebek packets. Sebek packets are defined as those that have both a predetermined destination UDP port and the proper magic number set in the Sebek header. If these two values matches what is expected, then we know this a packet to ignore. The implementation simply does nothing with Sebek packets; it drops them on the floor

[3] Ethernet switches will send ethernet frames to every switch port when they do not have an entry in their Switch Forwarding Table for the destination host. The most common situation we run into is where the MAC for the destination expires.

and moves on to the next packet in the queue. The end result is that an intruder is unable to capture the Sebek packets, even if he or she uses a sniffer compiled from known good sources.

Sebek Protocol Specification:

To ensure interoperability with all versions of Sebek, a common protocol has been defined. The communication channel between the client and the server is unidirectional, with packets originating from the client and destined to the server. This channel is UDP based and is connectionless and unreliable. Each packet contains one record. Records are variable length up to MTU, and have a fixed length header. Each record has a 48 byte header:

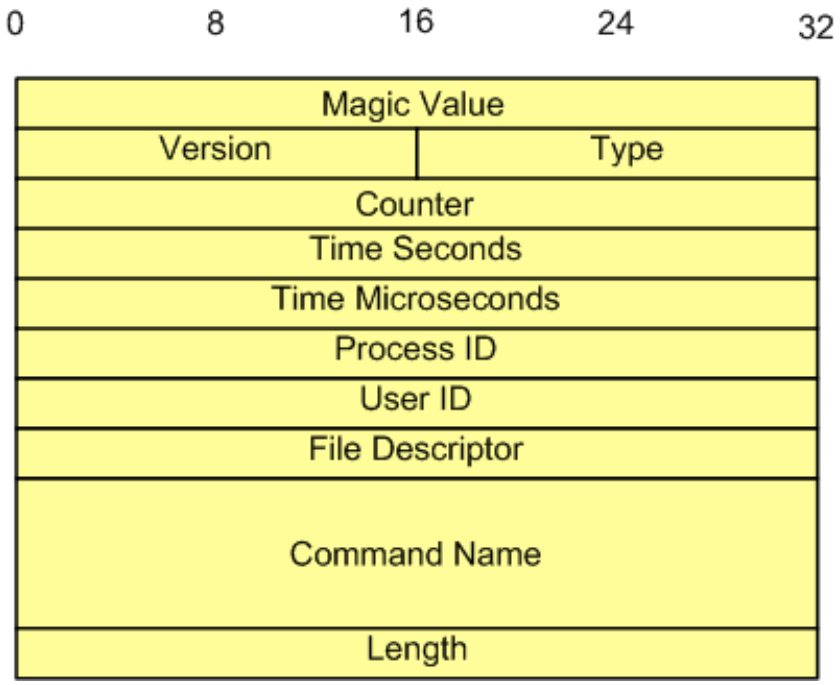

Figure 4
Sebek Record Header. This header comes after the IP/UDP header, and is followed by the data representing the activity on the honeypot (attacker keystrokes, files, passwords, etc).

Field Name	Data Type	Description
Magic	Unsigned 32 bit Int	Along with the DST Port, Magic is used by Sebek to identify packets which should be hidden
Version	Unsigned 16bit Int	Sebek Protocol Version, current version is “1”.
Type	Unsigned 16bit Int	Type of record represented. Read data is type 0, Write data is type 1. Currently only Read is implemented.
Counter	Unsigned 32bit Int	PDU counter, used to identify when packets are lost. This counter restarts at 0 when installed.
Time_sec	Unsigned 32bit Int	Seconds since UNIX epoch according to the honeypot
Time_usec	Unsigned 32bit Int	Residual Microseconds
PID	Unsigned 32bit Int	Process ID
UID	Unsigned 32bit Int	User ID
FD	Unsigned 32bit Int	File Descriptor
Com	12 Character Array	Records the first 12 characters of the command’s name.
Length	Unsigned 32bit Int	Length in octets of the PDU’s body.

Figure 5
Structure of a Sebek packet sent by the client.

When Sebek intercepts the read() system call, it not only records the contents of the read, but it also takes note of the context. The PID, UID, FD and Com fields of the record header are derived from the information the kernel keeps about the process making the request. The Timestamp fields are based on the system time, and thus are susceptible to tampering.

The Length field is based on the return length from the read request. If the length of data returned from the read() call is longer than the LAN MTU, then Sebek chops the read data into multiple fragments that all fit in onto the LAN. Each one of these fragments is a full Sebek record with its own header.

Within Sebek clients, the decision to hide a given packet from user space is based on the following criteria: the packet must be UDP, the UDP destination port must match the defined value, and the Sebek header Magic field must match the defined value. The UDP destination port is not used on its own because this would make it trivially easy to identify packet hiding by brute force. The Magic value is added so that in order for a brute force detection attempt to work, one must guess the correct magic value and destination, this gives you 281 billion possible combinations. If you had an application that could check 500,000 combinations per second it could take you up to 6.5 days to test.

Sebek’s Limits:

For Sebek to be of use in a honeypot it must not be detected by an intruder. It is possible to detect Sebek, or any covert tool of this type using techniques common to rootkit detection. As a result, the decision to use this system must consider the increased potential for detection by an intruder.

In the linux client, this increased risk is caused by the presence of kernel module support as well as the /dev/kmem feature. These facilities are very powerful and make the linux kernel quite

flexible. Just as we used them to install Sebek, intruders can use them to detect and disable sebek[4]. Fortunately, by the time Sebek has been disabled, the code associated with the technique and a record of the disabling action has been sent to the collection server. Thus in the future one avenue to limit the risk of detection will be to have the server detect when Sebek has been attacked and then disable the client.

Client Misuse Precautions:

As with any tool, Sebek has the potential to be used in an unintended or malicious manner. As it is a good tool for the monitoring of intruders, it would be a useful tool for the intruders to use on compromised systems to gather passwords or spy on legitimate users. To reduce the potential danger of Sebek, a few design decisions have been made.

In the early version of Sebek, export data was encrypted and the packets headers were spoofed to obfuscate the source of the data, we abandoned this approach in favor of current solution to make it easier to detect inappropriate usage. Because the currently method of hiding only works on honeypots with Sebek installed, any other host on the LAN can plainly see the Sebek packets. This means if an attacker attempts to use Sebek for their own compromised systems, organizations will easily be able to detect this as they will see the Sebek data exported to the wire. To further reduce chances of misuse we removed the standard rootkit features from Sebek. Thus an intruder would not have a single LKM enabling password sniffing and rootkit features.

Client Installation:

Now that we have an understanding of the operation of the Sebek client, let us address how to install the client. Installation of the client involves building the client for the version of the kernel that will be on the honeypot.[5] Once the client is successfully built, a binary tarball will be available for copying onto the honeypot. This tarball will include the file sbk_install.sh. The sbk_install.sh contains both the commands to install Sebek and the configuration variables. To install, simply edit the options you want, then run the install script on the honeypot. Upon execution sbk_install.sh installs the kernel module with the specified configuration. There are eight values to configure in the script, often these values will be the same for all honeypots on the LAN.

Interface:
This specifies which interface to log on. It will default to eth0. The interface does not need to have a configured IP address.

Destination IP: This field defines the Destination IP address used in all the Sebek packets generated. Because the Sebek server does not look at the Destination IP address when it collects packets, it is not required to set this to IP of the Sebek server. Further it is not recommended, if by chance the intruder were able to see the packets they would be able to then identify the host that was most likely performing all of the data control on the Honeynet.

Destination MAC: This field specifies Destination Ethernet MAC address to use for all packets. As Sebek doesn't use ARP, this is a mandatory option. If the logging server is more than one hop away from the

[4] An article by Phrack on Local Honeypot Identification outlined a technique that simply discovered the original sys_read call pointer and replaced it in the system call table, http://wwww.phrack.nl/phrack62/p62-0x07.txt.

[5] Client build details are contained in the source distribution, however recall that kernel modules must be build against the kernel source that corresponds to the running kernel version on the Host. If the honeypot is running version 2.4.18 you must compile Sebek using the 2.4.18 kernel source.

honeypot, then the destination MAC should be set to the default gateway. Lastly, setting this to FF:FF:FF:FF:FF:FF will cause all packets to be broadcast to every host on the LAN, a helpful trick for testing.

Magic Value: This value, along with the destination port, is used to identify which packets are hidden from the honeypots. It is highly recommended that all instances of Sebek on a given LAN be configured with the same magic value and destination port.

Destination UDP Port: This field defines the UDP destination port to be configured in the Sebek packets. This value is used by the Server to identify packets of interest and is also used by the clients in conjunction with the magic value to identify packets that need to be hidden.

Source UPD Port: This field defines the UDP source port to be configured in the Sebek packets.

Keystrokes only: This flag accepts two values: 1 or 0. It this flag is set, Sebek will only collect keystrokes. Otherwise it will collect all read data. This flag **must not** be set if you wish to recover SCP file transfers.

Testing: The testing option is a binary flag. If this flag is set two things happen, first, the kernel module is not hidden, and second, additional debugging is activated in the module. Beware however that this additional debugging may betray the presence of Sebek by sending debugging data to syslog.

When configuring all of the honeypots on the same LAN it is important that they all use the same magic value and destination port values. This keeps one honeypot from seeing another's packets. When configuring the IP and MAC address values, keep in mind that the MAC value is the most important. If you incorrectly set the DST IP address, but configure the MAC correctly, the Sebek records will still make it to the server, presuming that the UDP port is set correctly. Further, when the server is running on a Honeywall Gateway, the interface doesn't have an IP address configured, so the only way to ensure that the packets are recorded by the server would be to configure the Destination MAC to that of the default gateway or to the Honeywall MAC address.

If you are logging to a remote host(one that is not on the LAN), then the Destination IP must be set to the IP of that host, and the MAC must be set to the MAC address of the Default Gateway for the LAN. After configuration, execution of *sbk_install.sh* will install the Sebek client, and it will then begin capturing and exporting data.

Server Installation:

We will now discuss how to recover the Sebek data from the client, and once recovered, how to analyze that data. If you recall from Figure 1, it is common to install the server on the Honeywall Gateway.

The server has two possible sources of recovering Sebek client data from the network. The first option is to capture all the network data from a standard, tcpdump log file, then extract just the Sebek data from that file. The second option is to capture traffic live and directly from the

network interface. Extracting Sebek data out of a tcpdump file provides the ability to examine archived data or data for which no direct access is possible (such as from a fellow researcher who sends you data files).

The server is made up of three components. The first component is called *sbk_extract.* It either captures the data directly off of an interface, operating as a sniffer or it gathers it from a tcpdump file. Either way you will have to use this tool to recover the Sebek data. Once sbk_extract extracts the data, you can do one of two things with it. The first option is to send it through a tool called *sbk_ks_log.pl*, which is a Perl script that takes the attacker's keystrokes and sends them to STDOUT. The second option is called *sbk_upload.pl*, which is a Perl script that loads the Sebek data into a mysql database. To share or archive data, it is recommended that one treat binary tcpdump packet captures as the canonical source of data. As it is common to keep a record of all traffic in and out of a honeypot, this archiving is already in place, the addition of Sebek data archival comes with no additional effort.

sbk_extract is the application that extracts Sebek Records from its input data. Independent of the type of analysis, *sbk_extract* is the first application in the pipeline. This application takes Sebek Packets from either network or a packet capture file then sends the record extracted from the packet to the specified utility. *sbk_extract* has three options:

- **- f** This specifies a pcap file to extract data from.
- **- i** Defines the interface to listen on.
- **- p** Specifies the UDP destination port number of interest

Loading Sebek records into a database:	Run *sbk_extract* and pipe the output to *sbk_upload.pl* ex. **`sbk_extract	sbk_upload.pl`**
Command Line monitoring of keystrokes:	Run *sbk_extract* and pipe to *sbk_ks_log.pl* ex: **`sbk_extract	sbk_ks_ks_log.pl`**
Custom Analysis:	Run *sbk_extract* and pipe to you custom analysis application.

Monitoring keystroke Activity from the command line:

sbk_ks_log.pl allows you view keystroke activity on the host running the Sebek client from a command line prompt on the server. It has no options and takes input from *sbk_extract* via STDIN.

Example:

```
sbk_extract –i eth0 –p 1101 | sbk_ks_log.pl
```

In this example, *sbk_extract* is sniffing from interface eth0 and it is expecting those records on UPD port 1101. *sbk_extract* then sends those records to *sbk_ks_log.pl* for extraction of keystroke activity. The output of *sbk_ks_log.pl* can be seen bellow.

```
[2003-07-23 20:03:45  10.0.0.13 6673 bash 500]whoami
[2003-07-23 20:03:48  10.0.0.13 6673 bash 500]who
[2003-07-23 20:03:50  10.0.0.13 6673 bash 500]su
[2003-07-23 20:03:57  10.0.0.13 6886 bash 0]cd /var/log
[2003-07-23 20:03:57  10.0.0.13 6886 bash 0]ls
[2003-07-23 20:04:01  10.0.0.13 6886 bash 0]mkdir ...
[2003-07-23 20:04:20  10.0.0.13 6886 bash 0]tcsh
[2003-07-23 20:04:20  10.0.0.13 6921 tcsh 0]0
[2003-07-23 20:04:20  10.0.0.13 6920 tcsh 0]vt
[2003-07-23 20:04:20  10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:20  10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:27  10.0.0.13 6920 tcsh 0]cd /tmp
[2003-07-23 20:04:28  10.0.0.13 6920 tcsh 0]ls
[2003-07-23 20:04:42  10.0.0.13 6920 tcsh 0]cd /usr/lib
[2003-07-23 20:04:42  10.0.0.13 6920 tcsh 0]ls
```

The output from *sbk__ks_log.pl* is similar to what users see when on a terminal. However, we only see commands entered and not the output of those commands. Control characters are escaped when present. For example, when a Backspace is present, it is replaced with [BS]. Each line displayed by the *sbk_ks_log.pl* tool has the following format:

[TIMESTAMP IP_ADDR PID COMMAND UID] Text

- TIMESTAMP shows the first time keystrokes where entered for that context
- IP_ADDR is the address of the honeypot.
- PID is the process ID.
- COMMAND is the first 10 characters of the command name.
- UID is the User ID for the owner of the process.

Loading Sebek Data into a Database:

sbk_upload.pl is the perl script that uploads records into a mysql database. There are several options for this script:

-u	Database user
-s	Database Server, defaults to localhost.
-**d**	Database Name
-p	Password
-P	Port Number

Example:

sbk_extract –i eth0 –p 1101 | sbk_upload.pl -u Sebek –p secret – d Sebek

As with the previous example, *sbk_extract* is looking for Sebek packets on UDP port 1101 and it is sniffing on interface eth0. The extracted records are sent to *sbk_upload.pl*, which is loading them into a database on the localhost using username “Sebek” password “secret” and the inserts

are going into the "Sebek" database. The *sbk_upload.pl* script then inserts each record into a mysql database. The schema is defined in Appendix A. Once records have been inserted into the mysql database, the Sebek data can be viewed using the Web Interface or accessed directly using SQL queries.

The Web Interface:

Sebek now comes with a web based analysis interface. This interface provides users with the ability to monitor keystroke activity, search for specific activity, recover SCPed files and in general provides an improved data browsing capability. This interface is implemented with PHP and only examines the data contained in the database; it does not use data from other sources such as packet captures or syslogs. It is designed to support the workflow of forensic investigation, however it does require a fair degree of technical skill to understand. The intention was to make this a tool for Sebek data like ethereal is for packet captures. The interface has 3 primary options: viewing keystrokes, searching, and browsing.

- The Keystroke Summary view provides a summary of all keystroke activity.
- The Search view allows users the ability to query for certain information.
- The Browse view, or table view, provides a summary of all activity, including non keystroke activity.

Rather than going into the design of the interface, we will follow a usage scenario and how the interface was used in that scenario. In Figures 5 through 10 bellow, a user has logged into a honeypot with the IP address of 10.0.1.13. After logging in, the user downloads a file named malware. This file is a Burneye protected executable binary. Malware is a tool used to gain unauthorized root access[6]. The user executes the "malware" binary and gains root access to the honeypot. The intruder gained access to only one honeypot.

Sebek was configured on the client to record all read data and to send it to the MAC address of the Honeywall Gateway. On the Honeywall, *sbk_extract* was run piping its output to the *sbk_upload.pl* script. All other honeypots on the LAN also had Sebek installed. To keep the Sebek traffic from being detected by an intruder, all honeypots used the same Magic value. In the example we will demonstrate the interfaces ability to:

1. Recover a file copied to the honeypot using SCP.
2. Identify the password used to enable the Burneye binary
3. Identity the exact point where user gained root access.

[6] The tool gains unauthorized privilege escalation by leveraging the ptrace vulnerability CAN-2003-0127.

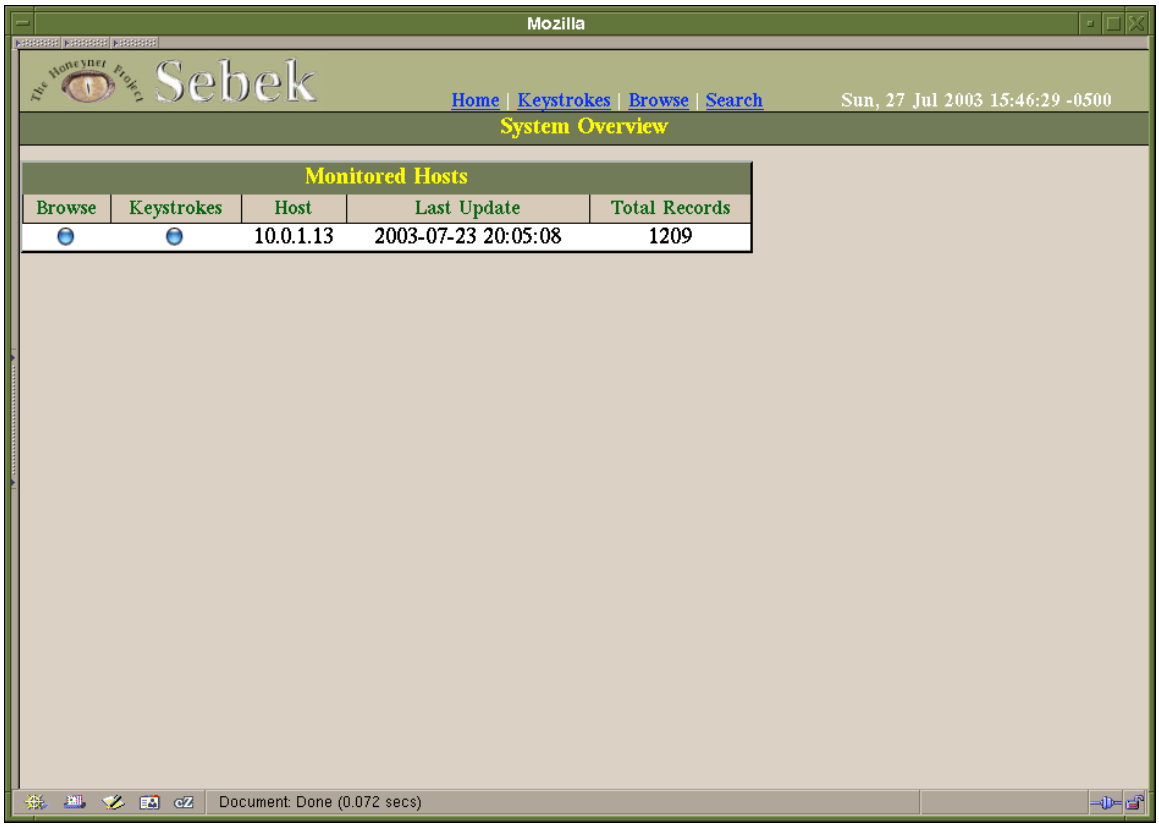

Figure 5
System Overview page.

We start out at the System Overview page in Figure 5. It provides a list of all the hosts we are monitoring and the last time we saw any activity from the host. By clicking on the Keystrokes button we can get a summary of what most likely is humanly derived activity.

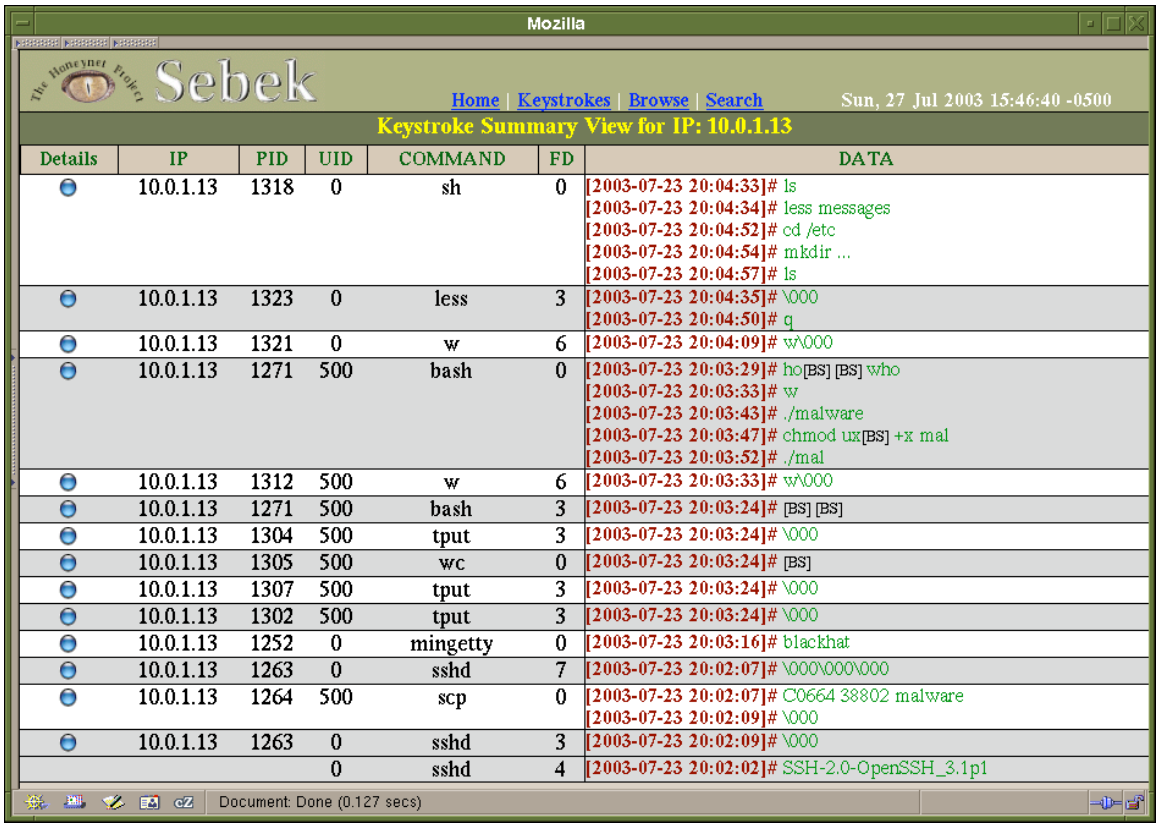

Figure 6
Keystroke summary page.

The Keystroke Summary view in Figure 6 provides the last 5 or so lines of text from all sessions on a host. Sessions are nothing more than activity on a specific File Descriptor for a given process. The sessions are sorted by time of last activity, with most recent at the top. Within each summary the keystroke logs are sorted in the order they occur. As with the command line tool *sbk_ks_log.pl*, control characters are escaped. There are two interesting events we see when we examining this summary. The first involves PID 1271; it shows what appears to be the first user access on the system. After checking who was logged in, the user ran a command called "malware". The second area of interest is PID 1264. This is an indication of a file being SCP transferred to the honeypot. Also note that the name of the file transferred is also "malware". When one is strictly interested in the keystrokes entered by the intruder, the best way to reduce the amount of uninteresting data is to use the search interface and search for activity for the name of the shell being used. Next we will drill down on PID 1264 to see if we can recover a copy of "malware".

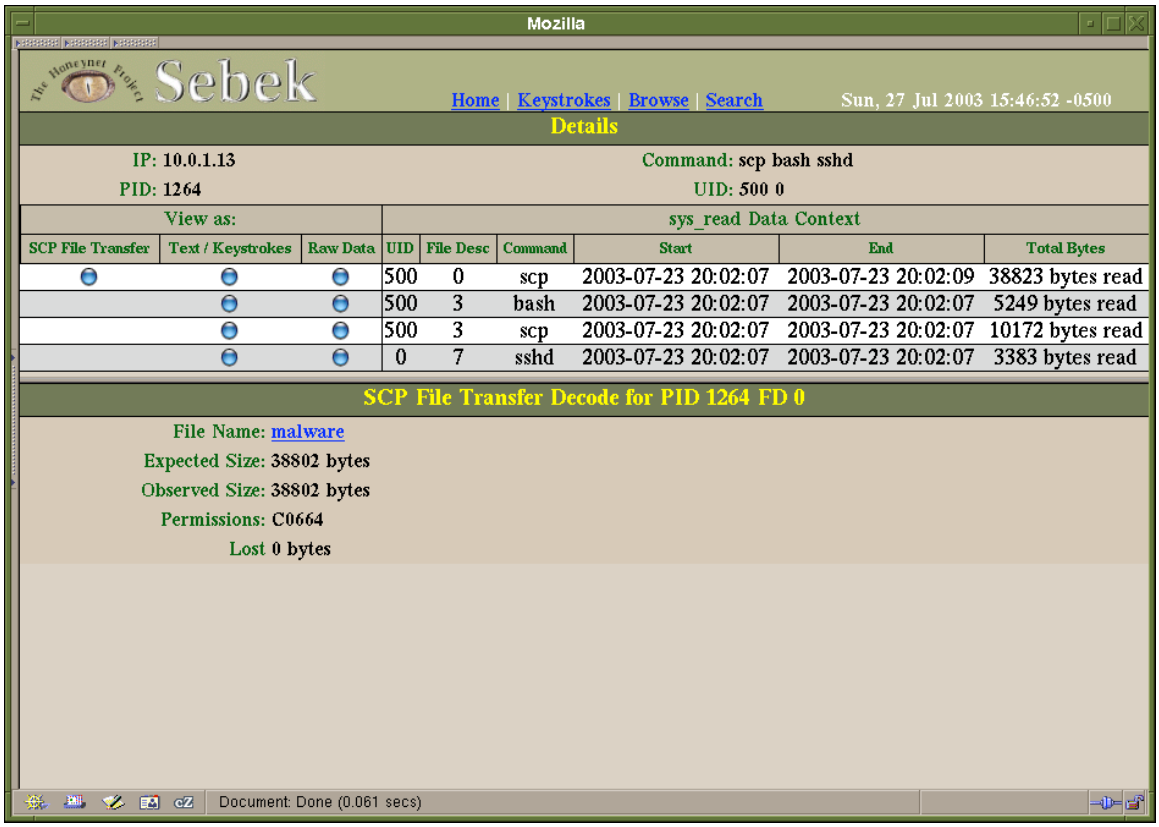

Figure 7
Details page, for PID 1264.

The Details view, Figure 7, shows all activity for a given PID. In this case we are looking at PID 1264. What we are seeing is an inbound SCP file transfer to the honeypot. From the examination we can see that the interface detected that File Descriptor 0 corresponds to the contents of the File Transfer, and the interface represents this by providing a SCP button. By clicking on the SCP File Transfer view button, the lower half of the view renders File Descriptor 0 as a file. Because Sebek uses UDP to export log data, there is a chance that some data is lost; this is especially true for intra-LAN transfers. To account for this potential difference, the interface displays the Expected and Observed Sizes of the file. Clicking on the file name will download the file. If you were to download this file and run the UNIX utility “strings” on the file you would see that this is Burneye protected binary. The utility “strings” is useful for extracting plain text information from binaries. In the case of Burneye protected ELF binaries, one of the first strings in the binary allows us to positively identify it as a Burneye. Now that we have recovered the file and have identified it as a Burneye binary, we need to see if we can recover the password used to activate the binary.

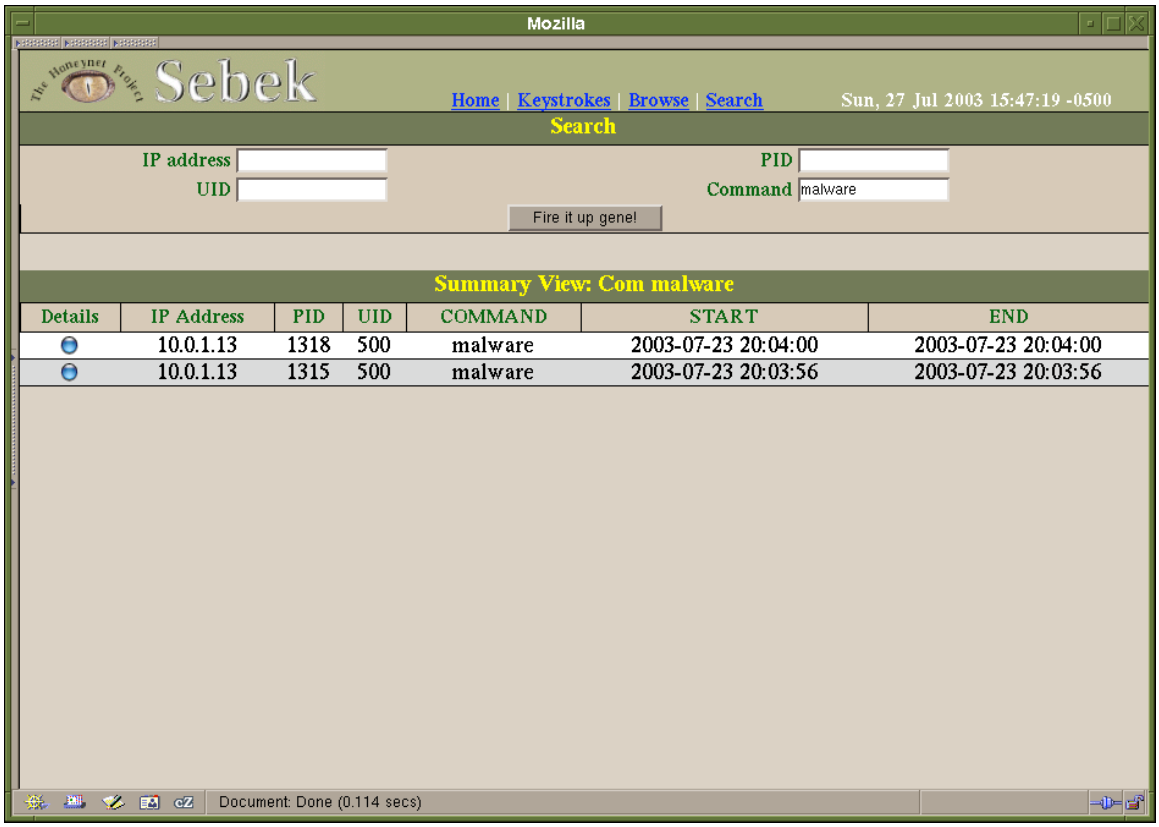

Figure 8
Search results for search on Command "malware".

The easiest way to look for all executable binary activity is to use the Search interface, shown in Figure 8. In this example we are looking for the executable called "malware". By entering the name of the executable into the Command field, we can search for all instances of a process running with a matching command name. It should be noted that when a fork occurs we will see multiple instances of the command executing, even if it were called from the command line only once. This is what happens in the case above. We see two processes, PID 135 and 1518, running with the Command name "malware". The next step will be for us to examine PID 1315 in detail as this Process terminated first. To examine PID 1315-, users simply click on the corresponding Details button.

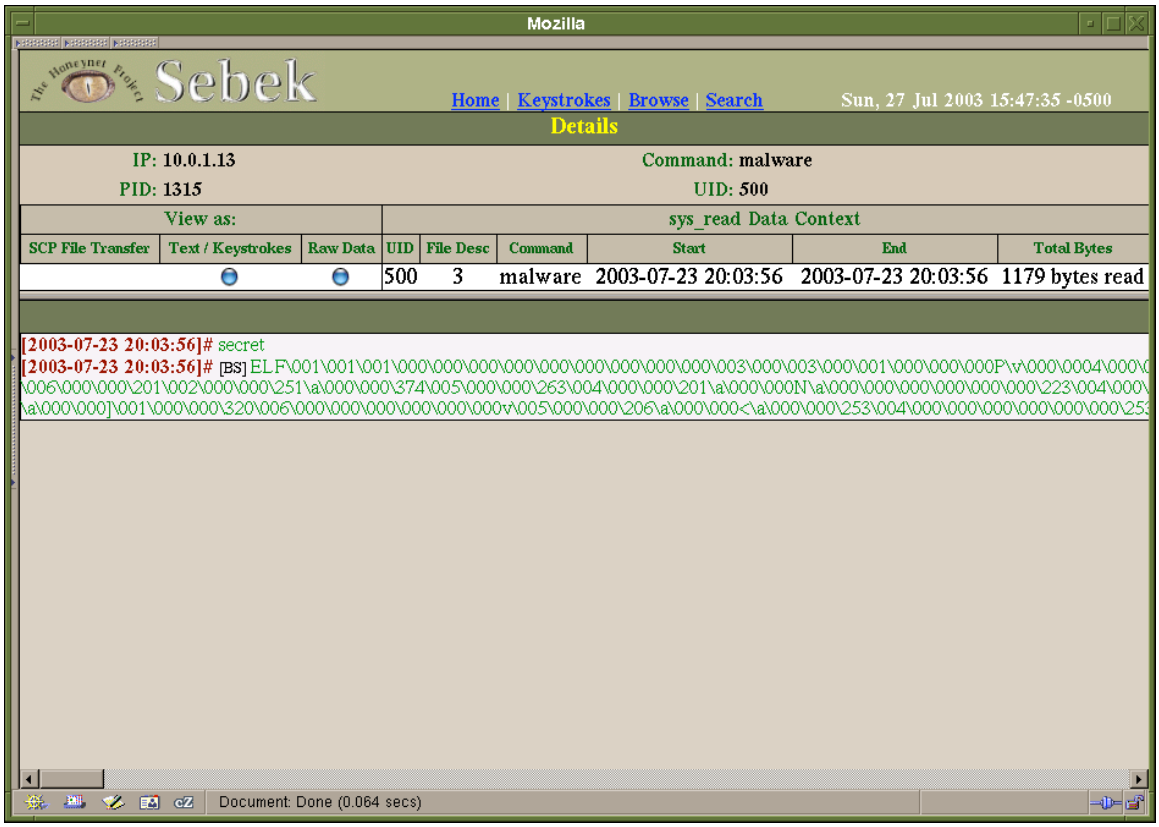

Figure 9
Details for PID 1315.

When we drill down on PID 1315(Figure 9), we see only one File Descriptor active. By clicking on Text / Keystroke, we can view the activity on File Descriptor 3. The first line appears to be plain text; the second appears to be an ELF binary. This first line of plain text is of interest. It is the first bit of data read by any of the instances of “malware” and from prior analysis we know that “malware” is a Burneye binary. In order for a Burneye to run the first thing that happens in the binary must be activated by a password. Although the interface does not flag this explicitly, this first line is the password used to activate the binary. At this point we have demonstrated how to recover an SCPed file, and how to recover the password of a Burneye protected binary. The last task will be to identify when the User gained root access. To do this we need to examine the other instance of “malware”.

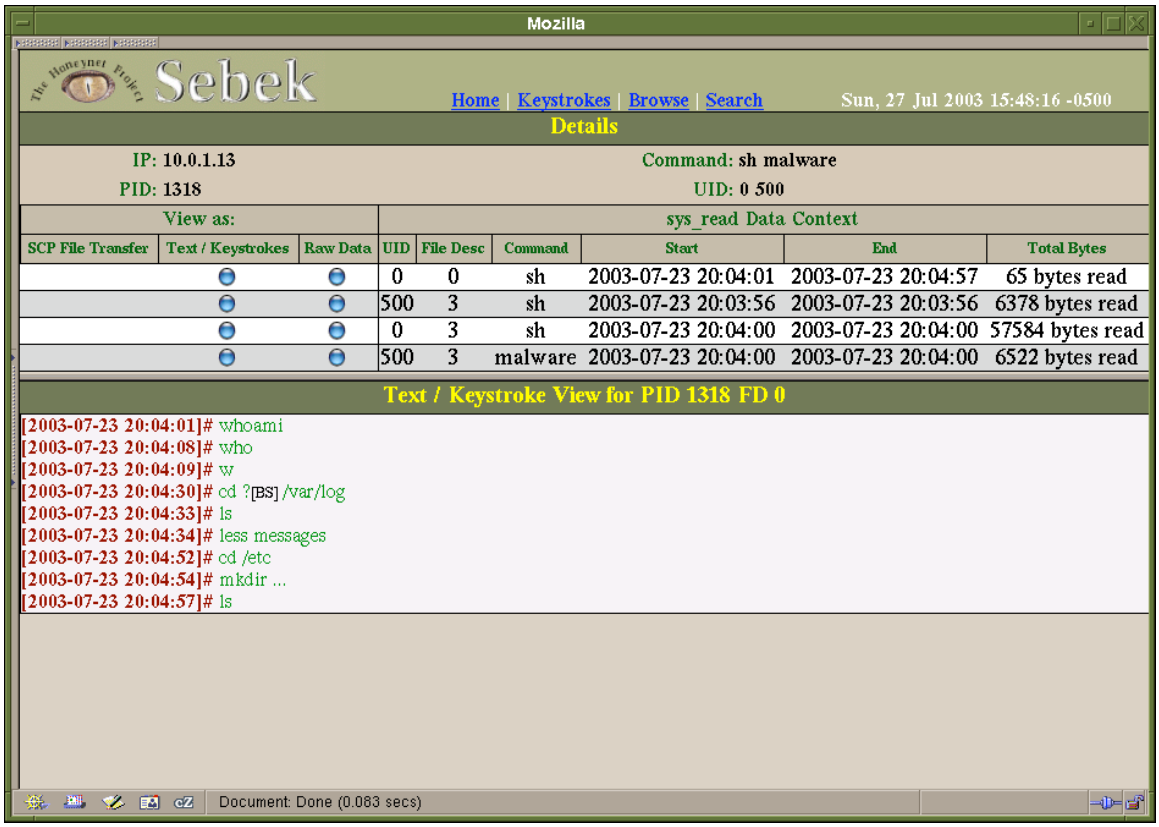

Figure 10
Details for PID 1318.

PID 1318 shows the other instance of “malware” and it provides us with a wealth of information. The process starts its execution with the command name of “malware” and terminates with the name of “sh”. More critically, it starts its execution as UID 500 and terminates as UID 0, or root. Thus we know that using this process ID the user gained root access. The user gained root access at 2003-07-23 20:04:01 GMT.

Summary:

Sebek is a kernel based data capture tool. It is designed to capture all activity on the honeypot in a covert manner. We circumvent 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. In general, Sebek provides an excellent view into the internal activities on a honeypot.

The current version of Sebek has its limits. For a skilled intruder who has intimate knowledge of the Operation System there are ways of detecting the presence of Sebek[7]. However it is unclear if this is something we can work around. One goal of Sebek is to make it sufficiently subtle as to not raise the suspicions of the intruder. It is anticipated that intruders will develop scripts that will automate the detection of Sebek on a host. As a result it is anticipated that further work will be required to reduce the chances of detection.

[7] Tools such as kstat and chrootkit are designed to find these sorts of systems.

The future goals for Sebek not only include increasing the difficulty of detection but also expanding the types of data we collect and the analysis we can perform. We will continue the transformation from keystroke logger to honeypot glass box analysis tool. For more information check out the Sebek homepage at:

http://www.honeynet.org/tools/sebek/

Questions, comments, bug reports, or suggestions for the development of Sebek should be referred to "Edward Balas" ebalas@iu.edu.

Appendix A:

The Sebek Database Schema is a simple single table schema. Most of the fields map directly with the Sebek Record format. The exception is the insert_time field, which is used as a secondary timestamp that references when the record was inserted into the database.

```
CREATE TABLE read_data (
        id              INT         UNSIGNED AUTO_INCREMENT,
        ip_addr         INT         UNSIGNED NOT NULL,
        insert_time     TIMESTAMP   default 'now()',
        time            DATETIME    NOT NULL,
        command         CHAR(20)    NOT NULL,
        counter         INT         UNSIGNED NOT NULL,
        filed           INT         UNSIGNED NOT NULL,
        pid             INT         UNSIGNED NOT NULL,
        uid             INT         UNSIGNED NOT NULL,
        length          INT         UNSIGNED NOT NULL,
        data            BLOB,

        PRIMARY KEY   ( id ),
        INDEX time_idx( time ),
        INDEX ip_idx  ( ip_addr),
        INDEX ip_time_idx(ip_addr, time),
        INDEX ip_pid_idx ( ip_addr, pid)
);
```