Network Security Through Data Analysis: Building Situational Awareness (2014)

Part II. Tools

This section is about a number of tools for use in data analysis. The primary focus of this section is on two pa rticular tools: SiLK and R. The System for Internet-Level Knowledge (SiLK) is a NetFlow analysis toolkit developed by the CERT at Carnegie Mellon University, which enables analysts to develop sophisticated flow analysis systems quickly and efficiently. R, a statistical analysis package developed at the University of Auckland, enables explorato ry data analysis and visualization.

At this time, there is no killer app for network analysis. Analysis requires using many tools, often in ways they weren't really designed for. The tools covered in this section form what I believe to be a basic functional toolkit for an analyst. Combining them with a light scripting language such as Python empowers analysts to explore data and develop operationally useful products.

The remainder of this section is divided into five chapters. Chapter 5 describes the SiLK suite, Chapter 6 descri bes R. Chapter 7 discusses IDS; while IDSes were briefly discussed in Part I, this chapter discusses the constructio n and maintenance of these tools—analysts will often produce ad hoc IDSes to identify or deal with attacks. Chapt er 8 discusses tools to identify the ways in which hosts are connected to the Internet, including reverse DNS looku ps, looking glasses, and tools such as *traceroute* and *ping*. Finally, Chapter 9 discusses additional tools that are us eful for particular analytic tasks.

Chapter 5. The SiLK Suite

SiLK, the System for Internet-Level Knowledge, is a toolkit originally developed by Carnegie Mellon's CERT to c onduct large-scale netflow analysis. SiLK is now used extensively by the Department of Defense, academic institut ions, and industry as a basic analytical toolkit.

This chapter focuses primarily on using SiLK as an analytical tool. The CERT Network Situational Awareness te am has published extensive references on using SiLK, installing collectors, and setting up the suite.

What Is SiLK and How Does It Work?

SiLK is a suite of tools for querying and analyzing NetFlow data. The SiLK suite enables an analyst to rapidly a nd efficiently query very large volumes of network traffic in order to identify complex aggregate phenomena or extr act individual events.

SiLK is effectively a database at the command line. Each tool performs a specific query, manipulation, or aggre gation of data, and commands are chained together to produce results. By chaining together multiple records alon g pipes, SiLK enables the analyst to create complex commands that field data along multiple channels simultaneo usly. For example, the following sequence of SiLK queries pull HTTP (port 80) traffic from flow data, producing a ti me series and a list of activity by busiest address. See Example 5-1 for the basics of SiLK operation: commands ar e passed through a series of pipes, which can be stdin, stdout, or fifos (named pipes).

Example 5-1. Some overly complicated rwfilter voodoo

```
$ mkfifo out2

$ rwfilter --proto=6 --aport=80 data.rwf --pass=stdout |

      rwfilter --input=stdin  --proto=6 --pass=stdout

      --all=out2 | rwstats --top --count=10 --fields=1 &




      rwcount out2 --bin-size=300
```

Data is maintained in an efficient binary representation up until the last moment, until commands that produce text (or some optional outputs) are called to produce output.

SiLK is very much an old-school Unix application suite: a family of tools tied together with pipes and using a lo t of optional arguments. By using this approach, it's possible to create powerful analytic scripts with SiLK, because the tools have well-defined interfaces that will efficiently handle binary data. Effectively using SiLK involves conne cting the appropriate tools together in order to process binary data and produce text only at the very end of the pro cess.

This chapter also uses some basic Unix shell commands such as ls, cat, and head. I don't require you to know the shell on an expert level.

Acquiring and Installing SiLK

The SiLK homepage is maintained at the CERT NetSA Security Suite web page. The SiLK package is available fr ee for download, and can be installed on most Unix systems without much difficulty. The CERT also provides a live CD image that can be used on its own.

The SiLK live CD comes with a training dataset called LBNL-05, anonymized header traces from Lawrence Berk eley National Labs in 2005. If you install the live CD, the data will be immediately accessible. If not, you can fetch t he data from The LBNL-05 reference data page.[5]

In addition to the live CD, SiLK is available in several package managers, including homebrew.

The Datafiles

The LBNL datafiles are stored in a file hierarchy; Example 5-2 shows the results of downloading and unarchivin g them.

Example 5-2. Downloading the SiLK archives

```
$ gunzip -c SiLK-LBNL-05-noscan.tar

$ gunzip -c SiLK-LBNL-05-scanners.tar

$ cd SiLK-LBNL-05

$ ls

README-S0.txt   in            out           silk.conf

README-S1.txt   inweb                        outweb

$ ls in/2005/01/07/*.01

in/2005/01/07/in-S0_20050107.01 in/2005/01/07/in-S1_20050107.01
```

When collecting data, SiLK partitions the data into subdirectories that divide traffic by the type of traffic and th e time the event occurred. This provides scalability and speeds up analysis. However, it's also generally a black bo x, and one we're breaking right now simply to have some files to work with. For the purposes of demonstration and education, we're going to work with four specific files:

§ *inweb/2005/01/06/iw-S0_20050106.20*

§ *inweb/2005/01/06/iw-S0_20050106.21*

§ *in/2005/01/07/in-S0_20050107.01*

§ *in/2005/01/07/in-S1_20050107.01*

These files are not special in any way. I chose them just to provide examples of scan and nonscan traffic. The f ollowing data discusses how to partition data and what the filenames mean.

Choosing and Formatting Output Field Manipulation: rwcut

SiLK records are stored in a compact binary format. They can't be read directly, and are instead accessed usin g the rwcut tool (see Example 5-3). In the following example, and any other examples with an output greater than 8 0 characters, the lines are manually broken for clarity.

Example 5-3. Simple file access with rwcut

```
$ rwcut inweb/2005/01/06/iw-S0_20050106.20 | more

            sIP|           dIP|sPort|dPort|pro|   packets|     bytes|\

   flags|                 sTime|   dur|                 eTime|sen|

 148.19.251.179|   128.3.148.48| 2497|   80|  6|        16|      2631|\

FS PA   |2005/01/06T20:01:54.119| 0.246|2005/01/06T20:01:54.365|  ?|

 148.19.251.179|   128.3.148.48| 2498|   80|  6|        14|      2159|\

 S PA   |2005/01/06T20:01:54.160| 0.260|2005/01/06T20:01:54.420|  ?|

...
```

In its default invocation, rwcut outputs 12 fields: source and destination IP addresses and ports, protocol, num ber of packets, number of bytes, TCP flags, start time, duration, end time, and sensor of a flow. These values have been discussed previously in Chapter 2, except for the sensorfield. SiLK can be configured to identify individual se nsors, which is useful when you're trying to figure out where traffic came from or where it's going. The sensor field is whatever ID is assigned during configuration. In the default data there are no sensors, so the value is set to a qu estion mark (?).

All SiLK commands have built-in documentation. Typing **rwcut –help** brings up an enormous help page. We will cover the basic options. A fuller description of options can be found in the SiLK documentation for rwcut.

The most commonly used rwcut commands select the fields displayed during invocation. rwcut can actually pr int 29 different fields, in arbitrary order. A list of these fields is in Table 5-1.

rwcut fields are specified using the --fields= option, which takes the numeric values in Table 5-1 or the string va lues, and prints the requested fields in the order specified, as in Example 5-4.

Table 5-1. rwcut fields

Field	**Numeric ID**	**Description**
sIP	1	Source IP address
dIP	2	Destination IP address
sPort	3	Source port
dPort	4	Destination Port: if ICMP, the ICMP type and code is encoded here also
protocol	5	Layer 3 protocol
packets	6	Packets in the flow
bytes	7	Bytes in the flow
flags	8	OR of TCP flags
sTime	9	Start time in seconds
eTime	10	End time in seconds
dur	11	Duration (eTime−sTime)

Field	Numeric ID	Description
sensor	12	Sensor ID
in	13	SNMP ID of the incoming interface on the router
out	14	SNMP ID of the outgoing interface on the router
nhIP	15	Next hop address
sType	16	Classification of the source address (internal, external)
dType	17	Classification of the destination address (internal, external)
scc	18	Country code of the source IP
dcc	19	Country code of the destination IP
class	20	Class of the flow
type	21	Type of the flow
sTime +msec	22	sTime in milliseconds
eTime +msec	23	eTime in milliseconds
dur +msec	24	duration msecs
icmpTypeCode	25	ICMP type and code
initialFlags	26	Flags in the first TCP packet
sessionFlags	27	Flags in all packets *except* the first
attributes	28	Attributes of the flow observed by the generator

Field	**Numeric ID**	**Description**
application	29	Guess as to the application in the flow

Example 5-4. Some examples of field ordering

```
$# Show a limited set of fields

$ rwcut --field=1-5 inweb/2005/01/06/iw-S0_20050106.20 | head -2

           sIP|           dIP|sPort|dPort|pro|

 148.19.251.179|   128.3.148.48| 2497|   80|  6|

$#Note the -, now explicitly enumerate

$ rwcut --field=1,2,3,4,5 inweb/2005/01/06/iw-S0_20050106.20 | head -2

           sIP|           dIP|sPort|dPort|pro|

 148.19.251.179|   128.3.148.48| 2497|   80|  6|

$#Field order is based on what you enter in --field

$ rwcut --field=5,1,2,3,4 inweb/2005/01/06/iw-S0_20050106.20 | head -2

pro|           sIP|           dIP|sPort|dPort|

  6| 148.19.251.179|   128.3.148.48| 2497|   80|

$#We can use text instead of numbers

$ rwcut --field=sIP,dIP,proto inweb/2005/01/06/iw-S0_20050106.20 |head -2

           sIP|           dIP|pro|

 148.19.251.179|   128.3.148.48|  6|
```

rwcut supports a number of other output formatting and manipulation tools. Some particularly useful ones, wh ich let you control the lines that appear in the output, include:

--no-title

Commonly used with SiLK commands that produce tabular output. Drops the title from the output table.

--num-recs

Outputs a specific number of records, eliminating the need for the head pipe in the previous example. The defa ult value is zero, which makes rwcut dump the entire contents of whatever file it’s reading.

--start-rec-num ***and*** --end-rec-num

Can be used to fetch a range of records in the file.

Example 5-5 shows a few ways to manipulate record numbers and headers.

Example 5-5. Manipulating record numbers and headers

```
$# Drop the title
$ rwcut --field=1-9 --no-title inweb/2005/01/06/iw-S0_20050106.20 | head -5
 148.19.251.179|   128.3.148.48| 2497|   80|  6|       16|      2631|FS PA
  |2005/01/06T20:01:54.119|
 148.19.251.179|   128.3.148.48| 2498|   80|  6|       14|      2159| S PA
  |2005/01/06T20:01:54.160|
 148.19.251.179|   128.3.148.48| 2498|   80|  6|        2|        80|F   A
  |2005/01/06T20:07:07.845|
  56.71.233.157|   128.3.148.48|48906|   80|  6|        5|       300| S
  |2005/01/06T20:01:50.011|
   56.96.13.225|   128.3.148.48|50722|   80|  6|        6|       360| S
  |2005/01/06T20:02:57.132|
$# Drop the head statement
$ rwcut --field=1-9 inweb/2005/01/06/iw-S0_20050106.20 --num-recs=5
            sIP|            dIP|sPort|dPort|pro|   packets|     bytes|   flags
|                  sTime|
 148.19.251.179|   128.3.148.48| 2497|   80|  6|       16|      2631|FS PA
|2005/01/06T20:01:54.119|
 148.19.251.179|   128.3.148.48| 2498|   80|  6|       14|      2159| S PA
|2005/01/06T20:01:54.160|
 148.19.251.179|   128.3.148.48| 2498|   80|  6|        2|        80|F   A
|2005/01/06T20:07:07.845|
  56.71.233.157|   128.3.148.48|48906|   80|  6|        5|       300| S
|2005/01/06T20:01:50.011|
   56.96.13.225|   128.3.148.48|50722|   80|  6|        6|       360| S
```

```
|2005/01/06T20:02:57.132|

$# Print only the third through fifth record

$ rwcut --field=1-9 inweb/2005/01/06/iw-S0_20050106.20 --start-rec-num=3

  --end-rec-num=5

           sIP|          dIP|sPort|dPort|pro|   packets|    bytes|   flags

|                sTime|

 148.19.251.179|   128.3.148.48| 2498|   80|  6|        2|       80|F   A

|2005/01/06T20:07:07.845|

  56.71.233.157|   128.3.148.48|48906|   80|  6|        5|      300| S

|2005/01/06T20:01:50.011|

   56.96.13.225|   128.3.148.48|50722|   80|  6|        6|      360| S

|2005/01/06T20:02:57.132|
```

A number of options manipulate output format. Tabulation is controllable with the --column-separator, --no-final-column, and --no-columns switches. --column-seperator will change the character used to distinguish columns, while --no-final-column drops the delimiter at the end of the line. --no-columns removes any space padding between columns. The --delimited switch combines all three: it takes a character as an argument, uses that character as a column separator, removes all padding in the columns, and drops the final column separator.

In addition, there are a variety of switches for changing column content:

--integer-ips

Converts IP addresses to integers rather than dotted quads. This switch is deprecated as of SiLK v3, and users should now use --ip-format=decimal.

--ip-format

The updated version of --integer-ips, --ip-format specifies how addresses are rendered. Options include canonical (dotted quad for IPv4, canonical IPv6 for IPv6), zero-padded (canonical, except zeroes are expanded to the maximal value for each format, so 127.0.0.1 is 127.000.000.001), decimal (print as the corresponding 32-bit or 128-bit integer), hexadecimal (print the integer in hexadeximal format), and force-ipv6 (prints all addresses in canonical IPv6 format, including IPv4 addresses mapped to the ::ffff:0:0/96 netblock).

--epoch-time

Prints timestamps as epoch values with floating-point millisecond precision.

--integer-tcp-flags

Converts TCP flags to their integer equivalents.

--zero-pad-ips

Pads the dotted quad IP address format with zeros, so that 128.2.11.12 is printed as 128.002.011.012. Deprec ated in favor of --ip-format in SiLK v3.

--icmp-type-and-code

Places the ICMP type in the source port and the ICMP code in the destination port.

--pager

Specifies the program to use for paging output.

Example 5-6 shows some of the preceding options.

Example 5-6. Other formatting examples

```
$# Change from fixed with columns to delims

$ rwcut --field=1-5 inweb/2005/01/06/iw-S0_20050106.20 --no-columns --num-recs=2

sIP|dIP|sPort|dPort|protocol|

148.19.251.179|128.3.148.48|2497|80|6|

148.19.251.179|128.3.148.48|2498|80|6|

$# Change the column separator

$ rwcut --field=1-5 inweb/2005/01/06/iw-S0_20050106.20 --column-sep=:

  --num-recs=2

            sIP:          dIP:sPort:dPort:pro:

 148.19.251.179:   128.3.148.48: 2497:   80:  6:

 148.19.251.179:   128.3.148.48: 2498:   80:  6:

$# Use --delim to change everything at once

$ rwcut --field=1-5 inweb/2005/01/06/iw-S0_20050106.20 --delim=: --num-recs=2

sIP:dIP:sPort:dPort:protocol

148.19.251.179:128.3.148.48:2497:80:6

148.19.251.179:128.3.148.48:2498:80:6

$# Convert IP addresses to integers

$ rwcut --field=1-5 inweb/2005/01/06/iw-S0_20050106.20 --integer-ip --num-recs=2

       sIP|       dIP|sPort|dPort|pro|

2484337587|2147718192| 2497|   80|  6|
2484337587|2147718192| 2498|   80|  6|

$# Use epoch time

$ rwcut --field=1-5,9 inweb/2005/01/06/iw-S0_20050106.20 --epoch --num-recs=2

           sIP|           dIP|sPort|dPort|pro|        sTime|

 148.19.251.179|   128.3.148.48| 2497|   80|  6|1105041714.119|

 148.19.251.179|   128.3.148.48| 2498|   80|  6|1105041714.160|

$# Zero pad IP addresses

$ rwcut --field=1-5,9 inweb/2005/01/06/iw-S0_20050106.20 --zero-pad --num-recs=2

           sIP|           dIP|sPort|dPort|pro|                sTime|

148.019.251.179|128.003.148.048| 2497|   80|  6|2005/01/06T20:01:54.119|

148.019.251.179|128.003.148.048| 2498|   80|  6|2005/01/06T20:01:54.160|
```

You will note that, as the command lines get more complex, I have truncated the longer options. SiLK uses GN U-style long options universally, so the only requirement for specifying an option is to type enough characters to m ake the name unambiguous. Expect more and more truncation as we build more and more complex commands.

Basic Field Manipulation: rwfilter

The most basic SiLK command with analytical values is rwcut paired with rwfilter through a pipe. Example 5-7 shows a simple rwfilter command.

Example 5-7. A simple rwfilter command

```
$ rwfilter --dport=80 inweb/2005/01/06/iw-S0_20050106.20 --pass=stdout

  | rwcut --field=1-9 --num-recs=5

           sIP|           dIP|sPort|dPort|pro|   packets|     bytes|   flags

  |                sTime|

 148.19.251.179|   128.3.148.48| 2497|   80|  6|        16|      2631|FS PA

  |2005/01/06T20:01:54.119|

 148.19.251.179|   128.3.148.48| 2498|   80|  6|        14|      2159| S PA

  |2005/01/06T20:01:54.160|

 148.19.251.179|   128.3.148.48| 2498|   80|  6|         2|        80|F   A

  |2005/01/06T20:07:07.845|

  56.71.233.157|   128.3.148.48|48906|   80|  6|         5|       300| S
  |2005/01/06T20:01:50.011|

   56.96.13.225|   128.3.148.48|50722|   80|  6|        6|      360| S

  |2005/01/06T20:02:57.132|
```

rwfilter with a single filter (the --dport option in this case), and a single redirect (the --pass=stdout) is about as simple as you can get. rwfilter is the workhorse of the SiLK suite: it reads input (directly from a file, using a set of g lobbing specifications, or through a pipe), applies one or more filters to each record in the data, and then redirects the records based on whether a record matches the filters (passes) or doesn't match (fails).

SiLK's rwfilter documentation is humongous, but primarily consists of repetitively describing the filter specifica tions for every field, so don't be intimidated. rwfilter options basically do one of three things: they specify how to fil ter data, how to read data, or how to direct the result of those filters.

Ports and Protocols

The easiest filters to start with are --sport, --dport, and --protocol. As the names imply, they filter on the source port, destination port, and protocol, respectively (see Example 5-8). These values can filter on a specific value (e. g., --sport=80 will pass any traffic where the source port is 80), or a range specified with a dash or commas (so --s port=79-83 will pass anything where the source port is between 79 and 83 inclusive, and could be expressed as --s port=79,80,81,82,83).

Example 5-8. Example filtering on sport

```
$ rwfilter --dport=4350-4360  inweb/2005/01/06/iw-S0_20050106.20

 --pass=stdout | rwcut --field=1-9 --num-recs=5

            sIP|          dIP|sPort|dPort|pro|   packets|    bytes|   flags

 |               sTime|

 218.131.115.42| 131.243.105.35|   80| 4360|  6|        2|       80|F   A

 |2005/01/06T20:24:21.879|

 148.19.96.160|131.243.107.239|   80| 4350|  6|       27|    35445|FS PA

 |2005/01/06T20:59:42.451|

 148.19.96.160|131.243.107.239|   80| 4352|  6|        4|      709|FS PA

 |2005/01/06T20:59:42.507|

 148.19.96.160|131.243.107.239|   80| 4351|  6|       15|    16938|FS PA

 |2005/01/06T20:59:42.501|

 148.19.96.160|131.243.107.239|   80| 4353|  6|        4|      704|FS PA

 |2005/01/06T20:59:42.544|

$ rwfilter --sport=4000-  inweb/2005/01/06/iw-S0_20050106.20
```

```
 --pass=stdout | rwcut --field=1-9 --num-recs=5
          sIP|           dIP|sPort|dPort|pro|   packets|     bytes|   flags
 |                 sTime|
 56.71.233.157|   128.3.148.48|48906|   80|  6|         5|       300| S
 |2005/01/06T20:01:50.011|
  56.96.13.225|   128.3.148.48|50722|   80|  6|         6|       360| S
 |2005/01/06T20:02:57.132|
  56.96.13.225|   128.3.148.48|50726|   80|  6|         6|       360| S
 |2005/01/06T20:02:57.432|
 58.236.56.129|   128.3.148.48|32621|   80|  6|         3|       144| S
 |2005/01/06T20:12:10.747|
  56.96.13.225|   128.3.148.48|54497|  443|  6|         6|       360| S
 |2005/01/06T20:09:30.124|
$ rwfilter --dport=4350,4352  inweb/2005/01/06/iw-S0_20050106.20
 --pass=stdout | rwcut --field=1-9 --num-recs=5
          sIP|           dIP|sPort|dPort|pro|   packets|     bytes|   flags
 |                 sTime|
 148.19.96.160|131.243.107.239|   80| 4350|  6|        27|     35445|FS PA
 |2005/01/06T20:59:42.451|
 148.19.96.160|131.243.107.239|   80| 4352|  6|         4|       709|FS PA
 |2005/01/06T20:59:42.507|
 148.19.96.160|131.243.107.239|   80| 4352|  6|         1|        40|    A
 |2005/01/06T20:59:42.516|
$ rwfilter --proto=1 in/2005/01/07/in-S0_20050107.01 --pass=stdout
 | rwcut --field=1-6 --num-recs=2
          sIP|           dIP|sPort|dPort|pro|   packets|
 35.223.112.236|    128.3.23.93|    0| 2048|  1|         1|
 62.198.182.170|    128.3.23.81|    0| 2048|  1|         1|
$ rwfilter --proto=1,6,17 in/2005/01/07/in-S0_20050107.01 --pass=stdout

 | rwcut --num-recs=2 --fields=1-6

           sIP|           dIP|sPort|dPort|pro|   packets|

  116.66.41.147|131.243.163.201| 4283| 1026| 17|         1|

  116.66.41.147|131.243.163.201| 3131| 1027| 17|         1|

$ rwfilter --proto=1,6,17 in/2005/01/07/in-S0_20050107.01 --fail=stdout

 | rwcut --num-recs=2  --fields=1-6

           sIP|           dIP|sPort|dPort|pro|   packets|

 57.120.186.177|   128.3.26.171|    0|    0| 50|        70|

 57.120.186.177|   128.3.26.171|    0|    0| 50|        81|
```

Note the use of --fail in the last example. Because there are 255 potential protocols, specifying "everything but TCP, ICMP, and UDP" could be expressed in two ways: either by specifying everything you want (--proto=0,2-5,7-16, 18-), or by using the --fail option. I'll discuss more advanced manipulation of --pass and --fail in the next chapter.

Size

Volume (size) options (bytes and packets) are similar to the protocol and port options in that you express the m numerically. Unlike the enumerations (ports and protocol), these numeric values can be expressed only as singl e digits or ranges, *not* as comma-separated values. So, --packets=70-81 is acceptable, but --bytes=1,2,3,4 is not.

IP Addresses

The simplest form of IP address filtering simply expresses the IP address directly (see Example 5-9). The follo wing examples show strict filtering on the source (--saddress) and destination (--daddress) address, and the --any-address option. --any-address will match *either*source or destination addresses.

Example 5-9. Filtering on IP addresses

```
$ rwfilter --saddress=197.142.156.83 --pass=stdout

   in/2005/01/07/in-S0_20050107.01 | rwcut --num-recs=2

           sIP|           dIP|sPort|dPort|pro|   packets|     bytes|   flags|

                 sTime|      dur|                 eTime|sen|

 197.142.156.83|  224.2.127.254|44510| 9875| 17|        12|      7163|        |

2005/01/07T01:24:44.359|   16.756|2005/01/07T01:25:01.115|  ?|

 197.142.156.83|  224.2.127.254|44512| 9875| 17|         4|      2590|        |

2005/01/07T01:25:02.375|    5.742|2005/01/07T01:25:08.117|  ?|

$ rwfilter --daddress=128.3.26.249 --pass=stdout
```

```
  in/2005/01/07/in-S0_20050107.01 | rwcut --num-recs=2

          sIP|          dIP|sPort|dPort|pro|   packets|     bytes|   flags|

                  sTime|      dur|                 eTime|sen|

211.210.215.142|   128.3.26.249| 4068|   25|  6|         7|       388|FS PA   |

  2005/01/07T01:27:06.789|    5.052|2005/01/07T01:27:11.841|  ?|

 203.126.20.182|   128.3.26.249|51981| 4587|  6|        56|      2240|F   A   |

  2005/01/07T01:27:04.812|   18.530|2005/01/07T01:27:23.342|  ?|

$ rwfilter --any-address=128.3.26.249

  --pass=stdout in/2005/01/07/in-S0_20050107.01 | rwcut --num-recs=2

          sIP|          dIP|sPort|dPort|pro|   packets|     bytes|   flags|

                  sTime|      dur|                 eTime|sen|

211.210.215.142|   128.3.26.249| 4068|   25|  6|         7|       388|FS PA   |

  2005/01/07T01:27:06.789|    5.052|2005/01/07T01:27:11.841|  ?|

 203.126.20.182|   128.3.26.249|51981| 4587|  6|        56|      2240|F   A   |

  2005/01/07T01:27:04.812|   18.530|2005/01/07T01:27:23.342|  ?|
```

Address options accept a variety of range descriptors. Each quad in an IP address can be expressed using the same comma-dash format that protocols and ports use. IP addresses will also accept the character *x* to mean *0-2 55*. This expression can be used within each quad; SiLK will match each quad separately. In addition to this comm a-dash format, SiLK can match on CIDR blocks.

SiLK supports IPv6 by using IPv6’s colon-based notation. The following are all examples of valid IPv6 filters in SiLK, and Example 5-10 shows how to filter them:

```
::ffff:x

::ffff:0:aaaa,0-5

::ffff:0.0.5-130,1,255.x
```

Example 5-10. Filtering IP ranges

```
$#Filtering on the last quad

$ rwfilter --daddress=131.243.104.x inweb/2005/01/06/iw-S0_20050106.20

  --pass=stdout | rwcut --field=1-5 --num-recs=5

          sIP|          dIP|sPort|dPort|pro|

 150.52.105.212|131.243.104.181|   80| 1262|  6|
 150.52.105.212|131.243.104.181|   80| 1263|  6|
  59.100.39.174| 131.243.104.27|   80| 3188|  6|
  59.100.39.174| 131.243.104.27|   80| 3191|  6|
  59.100.39.174| 131.243.104.27|   80| 3193|  6|
# Filtering a range of specific values in the third quad
$ rwfilter --daddress=131.243.104,107,219.x inweb/2005/01/06/iw-S0_20050106.20
   --pass=stdout | rwcut --field=1-5 --num-recs=5
            sIP|            dIP|sPort|dPort|pro|
  208.122.23.36|131.243.219.201|   80| 2473|  6|
205.233.167.250|131.243.219.201|   80| 2471|  6|
   58.68.205.40| 131.243.219.37|   80| 3433|  6|
208.233.181.122| 131.243.219.37|   80| 3434|  6|
   58.68.205.40| 131.243.219.37|   80| 3435|  6|
# Using CIDR blocks
$ rwfilter --saddress=56.81.0.0/16 inweb/2005/01/06/iw-S0_20050106.20
  --pass=stdout | rwcut --field=1-5 --num-recs=5
            sIP|            dIP|sPort|dPort|pro|
   56.81.19.218|131.243.219.201|   80| 2480|  6|
    56.81.16.73|131.243.219.201|   80| 2484|  6|
    56.81.16.73|131.243.219.201|   80| 2486|  6|
    56.81.30.48|131.243.219.201|  443| 2490|  6|
   56.81.31.159|131.243.219.201|  443| 2489|  6|
```

Time

There are three time options: --stime, --etime, and --active-time. These fields require a time range, which in SiLK is written in the format:

YYYY/MM/DDTHH:MM:SS-YYYY/MM/DDTHH:MM:SS

Note the T separating the day and hour. The --stime and --etime fields filter exactly what it says on the can, whi ch can be a bit counterintuitive; specifying --stime=2012/11/08T00:00:00-2012/11/08T00:02:00 filters any record whose *start time* is between midnight and two minutes after midnight on November 8, 2012. Records that started *before* midnight and are still being transmitted during that range will not pass. To find records that occurred within a particular period, use the --active-time filter.

TCP Options

Flows are aggregates of packets, and in the majority of cases, this aggregation is relatively easy to understand. For example, the number of bytes in a flow is the sum of the number of bytes of all the packets that comprise the flow. TCP flags, however, are a bit more problematic. In NetFlow v5, a flow's flags are the bitwise OR of the flags in its constituent packets—meaning that a flow indicates that a flag was present or absent in the *entire* flow, but not *where*. A flow could conceivably consist of a gibberish sequence of flags such as a FIN, then an ACK and SYN. Monitoring software such as YAF expands NetFlow to include additional flag fields, which SiLK can take advantage of.

The core flag filtering switches are --flags-initial,--flags-all, and --flags-session. These options accept flags in the form *high flags*/*mask flags*. If a flag is listed in the mask, SiLK always parses it. If a flag is listed in the high flags, SiLK passes it *only* if the value is high. The flags themselves are expressed using the characters in Table 5-2.

Table 5-2. Expressing TCP flags in rwfilter

Character	**Flag**
F	FIN
S	SYN
R	RST
P	PSH
A	ACK
U	URG
E	ECE
C	CWR

The combination of high flags and mask flags tends to confuse people, so let's review some examples. Remember that the basic rule is that in order to evaluate a flag, it *must* be in the mask. A flag specified as high but not specified in the mask will be ignored.

§ Setting the value to S/S will pass any record where the SYN flag is high.

§ Setting the value to S/SA will pass any record where the SYN flag is high *and* the ACK flag is low.

§ Setting the value to SA/SA will pass any record where *both* SYN and ACK flags are high.

§ A combination like SAF/SAFR will return any record where the SYN, ACK, FIN flags are high *and* the RST flag is low, which would be expected of a normal TCP connection.

In addition to these options, SiLK provides a set of flag-specific options in the form of --syn-flag, --fin-flag, and so on for each potential flag. These options take a 1 or 0 as an argument: setting the value to 1 will pass records where the flag is high, 0 will pass records where the flag is low, and not including the option will pass all records.

WHAT SHOULD TCP FLAGS LOOK LIKE?

The combination of TCP flags in any particular flow can be a useful indicator of the flow's behavior, and there a re certain flag combinations that raise suspicion.

Almost all TCP flows should pass *either* SAF/SAFR or SAR/SAFR, *without* passing SAFR/SAFR. This is because most sessions will end in a FIN, with aberrations ending in a RST. If both FIN and RST are seen, that's suspicious.

A TCP session without an ACK flag is curious, *especially* if that session has four or more packets. Stacks are u sually hardcoded to give up after *n* packets, where *n* tends to be in the neighborhood of three.

For a client, the initial flag should be a SYN, while a server should have a SYN+ACK. You should never see a SY N after the initial flag. Resynchronization would mean a new session started using the same ephemeral port, whic h is weird for TCP.

The PSH and URG flags are, in my mind, the universal indicator of boring sessions. If I see a session *without* P SH, especially if the session is long, it strikes me as curious. In my mind, a "normal" TCP session will have FSPA hi gh. A flow with just PA high is usually a keep-alive and an indication of a broken flow—look in the repository for the same address combination and you'll probably find a SAP flow occurring before it.

Backscatter/response messages include A, SA, and RA flows. A good number of RA packets will arrive on any l arge network due to backscatter from spoofed DDoS attacks. There isn't really anything you can do about these pa ckets; they're not even directly aimed at your network.

Helper Options

If you compare rwfilter's option-based filtering against tcpdump's BPF filtering, it's immediately obvious that rw filter's approach is much more primitive. This was an intentional decision: rwfilter is focused on processing large v olumes as quickly as possible, and the overhead involved in processing some kind of parseable language was dee med too expensive.

The place where this usually trips people up is the lack of obvious not and or operators. For example, if you wa nt to filter out all web sessions, you may try to filter traffic where one port is 80, and the other is ephemeral. The ini tial attempt might be:

```
rwfilter --sport=80,1024-65535 --dport=80,1024-65535 --pass=stdout
```

The problem is that this will also pass any flows where the source and destination port are both 80, and flows where the source and destination port are both ephemeral. To deal with problems like this, rwfilter has a collection of helper functions, which combined with the --fail option and multiple filters should be able to address any of thes e problems.

In the case of ports, the --aport option refers to either the source *or* the destination port. Using --aport and two filters, you can identify the appropriate sessions as follows:

```
rwfilter --aport=80 --pass=stdout | rwfilter --input-pipe=stdin

    --aport=1024-65535 --pass=stdout
```

The first filter identifies anything engaged in port 80 traffic, and the second takes that set and identifies anythi ng that also used an ephemeral port.

A number of IP address helper options are available. --anyaddress filters across source and destination addres ses simultaneously. --not-saddress and --not-daddress pass records with addresses that *don't* match the option sp ecification.

Miscellaneous Filtering Options and Some Hacks

rwfilter has a couple of direct text output options: --print-stat (see Example 5-11) and --print-volume-stat. Thes e can be used to print a summary of the traffic without having to resort to cut, count, or other display tools. They al so will print volumes of records that did *not* pass a filter.

Example 5-11. Using --print-stat

```
$ rwfilter --print-volume-stat in/2005/01/07/in-S0_20050107.01 --proto=0-255

     |           Recs|         Packets|            Bytes|     Files|

Total|            2019|         2730488|         402105501|        1|

 Pass|            2019|         2730488|         402105501|         |

 Fail|               0|               0|                 0|         |

$ rwfilter --print-stat in/2005/01/07/in-S0_20050107.01 --proto=0-255

Files     1.  Read      2019.  Pass      2019. Fail         0.
```

Note in Example 5-11 the use of the --proto=0-255 option. In almost all invocations, rwfilter expects *some* form of filtering applied to it, so when you need a filter that passes everything, the easiest approach is just to specify all the protocols. --print-stat and --print-volume-stat output to stderr, so you can still use stdout for pass, fail, and all c hannels.

Like rwcut, rwfilter has a record limit command. --max-pass-records and --max-fail-records can be used to limit the number of records passed through a pass or fail channel.

rwfileinfo and Provenance

SiLK filter files contain a fair amount of metadata, which can be accessed using the rwfileinfo command (see E xample 5-12). rwfileinfo can work with files, as seen in the examples below, or directly on stdin by using stdin or - a s an argument.

Example 5-12. Using rwfileinfo

```
$ rwfileinfo in/2005/01/07/in-S0_20050107.01

in/2005/01/07/in-S0_20050107.01:

  format(id)         FT_RWAUGMENTED(0x14)
```

```
 version          2
 byte-order       littleEndian
 compression(id)  none(0)
 header-length    28
 record-length    28
 record-version   2
 silk-version     0
 count-records    2019
 file-size        56560
 packed-file-info  2005/01/07T01:00:00 ? ?
$ rwfilter --print-stat in/2005/01/07/in-S0_20050107.01 --proto=6
 --pass=example.rwf
Files    1.  Read      2019.  Pass      1353. Fail       666.
$ rwfileinfo example.rwf
example.rwf:
 format(id)       FT_RWGENERIC(0x16)
 version          16
 byte-order       littleEndian
 compression(id)  none(0)
 header-length    156
 record-length    52
 record-version   5
 silk-version     2.1.0
 count-records    1353
 file-size        70512
 command-lines
                 1  rwfilter --print-stat --proto=6 --pass=example.rwf
 in/2005/01/07/in-S0_20050107.01
```

```
$ rwfilter --aport=25 example.rwf --pass=example2.rwf --fail=example2_fail.rwf

$ rwfileinfo example2.rwf

example2.rwf:

 format(id)         FT_RWGENERIC(0x16)

 version            16

 byte-order         littleEndian

 compression(id)    none(0)

 header-length      208

 record-length      52

 record-version     5

 silk-version       2.1.0

 count-records      95

 file-size          5148

 command-lines

          1  rwfilter --print-stat --proto=6 --pass=example.rwf

 in/2005/01/07/in-S0_20050107.01

          2  rwfilter --aport=25 --pass=example2.rwf

 --fail=example2_fail.rwf example.rwf
```

The fields reported by rwfileinfo are as follows:

example2.rwf

The first line of every rwfileinfo dump is the name of the file.

format(id)

SiLK files are maintained in a number of different optimized formats; the format value is a C macro describing the type of the file, followed by the hexadecimal ID of that type.

version

The version of the file format.

byte-order

The order in which bytes are stored on disk; SiLK maintains distinct little- and big-endian formats for faster rea ding.

compression(id)

Whether the file is natively compressed, again for faster reading.

header-length

The size of the file header; a SiLK file with no records will be just the size of the header-length.

record-length

The size of individual file records. This value will be 1 if records are variable length.

record-version

The version of the records (note that record versions are distinct from file versions and SiLK versions).

silk-version

The version of the SiLK suite used to create the file.

count-records

The number of records in the file.

file-size

The total size of the file; if the file is uncompressed, this value should be equivalent to the header length added to the product of the record length and record count.

command-lines

A record of the SiLK commands used to create the file.

Example 5-13 shows how to use the --note-add command.

Example 5-13. Using --note-add

```
$ rwfilter --aport=22 example.rwf --note-add='Filtering ssh' --pass=ex2.rwf

$ rwfileinfo ex2.rwf

ex2.rwf:

  format(id)       FT_RWGENERIC(0x16)

  version          16

  byte-order       littleEndian

  compression(id)  none(0)

  header-length    260

  record-length    52
```

```
record-version     5

silk-version       2.1.0

count-records      10

file-size          780

command-lines

            1  rwfilter --print-stat --proto=6 --pass=example.rwf

in/2005/01/07/in-S0_20050107.01

            2  rwfilter --aport=22 --note-add=Filtering ssh

--pass=ex2.rwf example.rwf

annotations

            1  Filtering ssh
```

Combining Information Flows: rwcount

rwcount can produce time series data from the output of an rwfilter command. It works by placing counts of b ytes, packets, and flow records into fixed-duration *bins*, which are equally sized time periods specified by the use r. rwcount is a relatively straightforward application. Most of its complexity comes from relating the flows, which t hemselves have a duration, to the bins.

The simplest invocation of rwcount is shown in Example 5-14. The first thing to notice is the use of the --bin-si ze option. In this example, the bins are half an hour, or 1,800 seconds. If --bin-size isn't specified, rwcount will defa ult to 30-second bins. Bin sizes don't have to be integers; floating-point specifications with a resolution down to th e millisecond are acceptable for people who like *lots* of bins in their output.

Example 5-14. Simple rwcount invocation

```
$ rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |

  rwcount --bin-size=1800

               Date|     Records|           Bytes|        Packets|

2005/01/07T01:00:00|      257.58|     42827381.72|      248724.14|

2005/01/07T01:30:00|     1589.61|    211453506.60|     1438751.93|

2005/01/07T02:00:00|      171.81|    147824612.67|     1043011.93|
```

As Example 5-14 shows, rwcount outputs four columns: a date column in SiLK's standard date format (YYYY/ MM/DDTHH:MM:SS), followed by record, byte, and packet columns. The floating-point values are a function of rwc ount interpolating how much traffic should be in each bin;rwcount calls this a *load scheme*.

The load scheme is an attempt by rwcount to approximate how much of a flow took place over the period spec ified by the bins. In the default load scheme, rwcount splits each flow proportionally across all the bins during whi ch the flow was taking place. For example, if a flow takes place from 00:04:00 to 00:11:00, and bins are five minute s long, 1/7 of the flow will be added to the first (00:00:00-00:04:59) bin, 5/7 to the second bin (00:05:00-00:09:59), and 1/7 to the third (00:10:00-00:14:59) bin. rwcount takes an integer parameter in the --load-scheme option, with t he following results:

0	Split the traffic evenly across all bins covered. In the previous example, the flow would be split into third s, and a third added to each bin.
1	Add the entire flow to the first bin covered by the flow. In the previous example, 00:00:00-00:04:59.
2	Add the entire flow to the last bin covered by the flow. In the previous example, 00:10:00-00:14:59.
3	Add the entire flow to the middle bin covered by the flow. In the previous example, 00:05:00-00:09:59.
4	The default load scheme.

rwcount uses the flow data provided to guess which time bins are required, but sometimes you have to explicit ly specify the time, especially when coordinating multiple files. This can be done using the --start-epoch and --end-epoch options to specify starting and ending bin times. Note that these parameters can use the epoch time or yyy y/mm/dd:HH:MM:SS format. rwcount also has an option to print dates using epoch time: the --epoch-slots option.

The --skip-zero option (see Example 5-15) is one of a number of output format options. Normally, rwcount print s every empty bin it has allocated, but --skip-zero causes empty bins to be omitted from the output. In addition, rw count supports many of the output options mentioned for rwcut: --no-titles, --no-columns, --column-separator, --no-final-delimter, and --delimited.

Example 5-15. Using epoch slots and the --skip-zero option

```
rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |

  rwcount --bin-size=1800.00 --epoch

          Date|      Records|          Bytes|       Packets|

    1105059600|       257.58|    42827381.72|     248724.14|

    1105061400|      1589.61|   211453506.60|    1438751.93|

    1105063200|       171.81|   147824612.67|    1043011.93|

$ rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |

 rwcount --bin-size=1800.00

        --epoch --start-epoch=1105057800

          Date|      Records|          Bytes|       Packets|
    1105057800|         0.00|              0.00|           0.00|
    1105059600|       257.58|       42827381.72|      248724.14|
    1105061400|      1589.61|      211453506.60|     1438751.93|
    1105063200|       171.81|      147824612.67|     1043011.93|
$ rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |
  rwcount --bin-size=1800.00
       --epoch --start-epoch=1105056000
          Date|      Records|             Bytes|        Packets|
    1105056000|         0.00|              0.00|           0.00|
    1105057800|         0.00|              0.00|           0.00|
    1105059600|       257.58|       42827381.72|      248724.14|
    1105061400|      1589.61|      211453506.60|     1438751.93|
    1105063200|       171.81|      147824612.67|     1043011.93|
$ rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |
 rwcount --bin-size=1800.00
        --epoch --start-epoch=1105056000 --skip-zero
          Date|      Records|             Bytes|        Packets|
    1105059600|       257.58|       42827381.72|      248724.14|
    1105061400|      1589.61|      211453506.60|     1438751.93|
    1105063200|       171.81|      147824612.67|     1043011.93|
```

rwset and IP Sets

IP sets are SiLK's most powerful capability, and something that distinguishes the toolkit from most other analy tical tools. An IP set is a binary representation of an arbitrary collection of IP addresses. IP sets can be created fro m text files, from SiLK data, or by using other binary SiLK structures.

The easiest way to start with IP sets is to create one, as in Example 5-16.

Example 5-16. Creating IP sets with rwset

```
$ rwfilter in/2005/01/07/in-S0_20050107.01 --all=stdout |
  rwset --sip-file=sip.set --dip-file=dip.set
$ ls -l *.set
```

```
-rw-r--r--  1 mcollins  staff    580 Jan 10 01:06 dip.set
-rw-r--r--  1 mcollins  staff  15088 Jan 10 01:06 sip.set
$ rwsetcat sip.set | head -5
0.0.0.0
32.16.40.178
32.24.41.181
32.24.215.49
32.30.13.177
$ rwfileinfo sip.set
sip.set:
  format(id)          FT_IPSET(0x1d)
  version             16
  byte-order          littleEndian
  compression(id)     none(0)
  header-length       76
  record-length       1
  record-version      2
  silk-version        2.1.0
  count-records       15012
  file-size           15088
  command-lines
                  1  rwset --sip-file=sip.set --dip-file=dip.set
```

rwset takes flow records and produces up to four output files. The file specified with --sip-file will contain source IP addresses from the flow, --dip-file will contain destination addresses, --any-file will contain source and destination IP addresses, and nhip-file will contain next hop addresses. The output is binary and read with rwsetcat, and as with all SiLK files, the file can be examined using rwfileinfo.

The power of IP sets comes when they're combined with rwfilter. rwfilter has eight commands that accept IP sets (--sipset, --dipset, --nhipset, --anyset, and their negations). Sets are explicitly designed so rwfilter can rapidly query using them, enabling a variety of useful queries, as seen in Example 5-17.

Example 5-17. Set manipulation and response

```
$ # First, we create IP sets; I use aport=123 (NTP on UDP) to filter down
$ # to a reasonable set of addresses.  NTP clients and servers use the same
$ # port.
$ rwfilter in/2005/01/07/in-S0_20050107.01 --pass=stdout --aport=123 |
  rwset --sip-file=sip.set --dip-file=dip.set
$ # Now, let's see how many IP addresses are created
$ rwsetcat --count-ip sip.set
15
$ # Generating output using rwfilter; note the use of the --dipset file as the
$ # sip set; this means that I'm now looking for messages that responded to
$ # these addresses.  This means that I've seen ntp going to and from the
$ # address, meaning it's likely to be a legitimate speaker, as opposed to a
$ # scan on port 123.
$ rwfilter out/2005/01/07/out-S0_20050107.01 --dipset=sip.set --pass=stdout
 --aport=123 | rwcut | head -5
          sIP|          dIP|sPort|dPort|pro|   packets|    bytes|   \
flags|                  sTime|     dur|                 eTime|sen|
  128.3.23.152|    56.7.90.229|  123|  123| 17|         1|        76|   \
    |  2005/01/07T01:10:00.520|    0.083|2005/01/07T01:10:00.603|  ?|
  128.3.23.152|  192.41.221.11|  123|  123| 17|         1|        76|   \
    |  2005/01/07T01:10:15.519|    0.000|2005/01/07T01:10:15.519|  ?|
  128.3.23.231| 87.221.134.185|  123|  123| 17|         1|        76|   \
    |  2005/01/07T01:24:46.251|    0.005|2005/01/07T01:24:46.256|  ?|
  128.3.26.152| 58.243.214.183|  123|10123| 17|         1|        76|   \
    |  2005/01/07T01:27:08.854|    0.000|2005/01/07T01:27:08.854|  ?|
$ # Let's look at statistics; using the same file, I look at the hosts
$ # that responded
$ rwfilter out/2005/01/07/out-S0_20050107.01 --dipset=sip.set  --aport=123
```

```
  --print-stat
Files     1.  Read      12393.  Pass        21. Fail       12372.
$ # Now I look at everyone else; not-dipset means that I'm looking at everything
$ # on port 123 that doesn't go to these addresses.
$ rwfilter out/2005/01/07/out-S0_20050107.01 --not-dipset=sip.set  --aport=123
   --print-stat
Files     1.  Read      12393.  Pass       337. Fail       12056.
```

Sets can also be generated by hand using rwsetbuild, which takes text input and produces a set file as the outp ut. The rwsetbuild specification takes any of the IP address specifications used by the --saddress option in rwfilter: literal addresses, integers, ranges within dotted quads, and netmasks. Example 5-18 demonstrates this.

Example 5-18. Building a set using rwsetbuild

```
$ cat > setsample.txt
# Comments in set files are prefaced with a hashmark
# Literal address
255.230.1.1
# Note that I'm putting addresses in some semi-random order; the output
# will be ordered.
111.2.3-4.1-2
# Netmask
22.11.1.128/30
^D
$ rwsetbuild setsample.txt setsample.set
$ rwsetcat --print-ip setsample.set
22.11.1.128
22.11.1.129
22.11.1.130
22.11.1.132
111.2.3.1
111.2.3.2
```

111.2.4.1

111.2.4.2

255.230.1.1

Sets can also be manipulated using the rwsettool command, which provides a variety of mechanisms for addin g and removing sets. rwsettool supports four manipulations:

--union

Creates a set that includes any address that appears in any of the sets.

--intersect

Creates a set that includes only addresses that appear in all the sets specified.

--difference

Removes addresses in the latter sets from the first set.

--sample

Randomly samples a set to produce a subset.

rwsettool is generally invoked using an output path (--output=_file_), but if nothing is specified, it will dump to s tdout. As with rwfilter, rwsettool output is binary, so a pure terminal dump triggers an error. Example 5-19 shows a manipulation with rwsettool.

Example 5-19. Set manipulation with rwsettool

$ rm setsample2.set

$ cat > setsample2.txt

# Build a set that covers our original setsample file to

# see what happens with various functions

22.11.1.128/29

$ rwsetbuild setsample2.txt setsample2.set

$ rwsettool --union setsample.set setsample2.set | rwsetcat

22.11.1.128

22.11.1.129

22.11.1.130

22.11.1.131

22.11.1.132

```
22.11.1.133

22.11.1.134

22.11.1.135

111.2.3.1

111.2.3.2

111.2.4.1

111.2.4.2

255.230.1.1

$ rwsettool --intersect setsample.set setsample2.set | rwsetcat

22.11.1.128

22.11.1.129

22.11.1.130

22.11.1.131

$ rwsettool --difference setsample.set setsample2.set | rwsetcat

111.2.3.1

111.2.3.2

111.2.4.1

111.2.4.2

255.230.1.1
```

rwuniq

rwuniq is the utility knife of counting tools. It allows an analyst to specify a key containing one or more fields, and will then count a number of different values, including total number of bytes, packets, flow records, or unique I P addresses matching the key.

rwuniq's default configuration counts the number of flows that occurred for a particular key. The key itself mus t be specified using the --field option, which accepts the field specifiers in Table 5-1. rwuniq can accept multiple fie lds, and the key will be generated in the order specified in the command line. Example 5-20 demonstrates the key f eatures of the --field option. As it shows, field order in the option affects field ordering in the output.

Example 5-20. Various field specifiers using rwuniq

```
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |

  rwuniq --field=sip,proto | head -4
```

```
          sIP|pro|   Records|
 131.243.142.85| 17|        1|
131.243.141.187| 17|        6|
   128.3.23.41| 17|        4|
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=1,2 | head -4
          sIP|          dIP|   Records|
  128.3.174.158|    128.3.23.44|        2|
    128.3.191.1|239.255.255.253|        8|
   128.3.161.98|131.243.163.206|        1|
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=sip,sport | head -4
          sIP|sPort|   Records|
 131.243.63.143|53504|        1|
 131.243.219.52|61506|        1|
131.243.163.206| 1032|        1|
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=sport,sip | head -4
sPort|          sIP|   Records|
55876|  131.243.61.70|        1|
51864|131.243.103.106|        1|
50955| 131.243.103.13|        1|
```

Also, note that when fields' orders are changed, the order in which records are output also changes. rwuniq does *not* guarantee record ordering by default; sorting can be ordered by using the --sort-output option.

rwuniq provides a number of count switches that instruct it to count additional values (see Example 5-21). The counting switches are --bytes, --packets, --flows, --sip-distinct, and --dip-distinct. Each of these fields can be used on their own, or by specifying a threshold (e.g., --bytes, --bytes=10, or --bytes=10-100). A single-value threshold (--bytes=10) provides a minimum, while a two-value threshold (--bytes=10-100) provides a range with a minimum and maximum. If you don't specify an argument, then the switch returns all values.

Example 5-21. Field spec with rwuniq

```
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=sport,sip --bytes --packets | head -5
sPort|          sIP|          Bytes|   Packets|
55876|  131.243.61.70|           308|        4|
51864|131.243.103.106|           308|        4|
50955| 131.243.103.13|           308|        4|
56568|  128.3.212.145|           360|        5|
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=sport,sip --bytes --packets=8 | head -5
sPort|          sIP|          Bytes|   Packets|
    0| 131.243.30.224|          2520|       30|
  959|   128.3.215.60|           876|       19|
 2315|131.243.124.237|           608|        8|
56838| 131.243.61.187|           616|        8|
$ rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwuniq --field=sport,sip --bytes --packets=8-20 | head -5
sPort|          sIP|          Bytes|   Packets|
  959|   128.3.215.60|           876|       19|
 2315|131.243.124.237|           608|        8|
56838| 131.243.61.187|           616|        8|
  514|   128.3.97.166|          2233|       20|
```

rwbag

The last set of tools to discuss in this chapter are *bag tools*. A *bag* is a form of storage structure. It contains a key (which can be an IP address, a port, the protocol, or an interface index), and a count of values for that key. Bag s can be created from scratch or from flow data using the rwbagcommand (see Example 5-22).

Example 5-22. An rwbag call, creating an IP address bag

```
$rwfilter out/2005/01/07/out-S0_20050107.01 --all=stdout |
  rwbag --sip-bytes=sip_bytes.bag
$rwbagcat sip_bytes.bag | head -5
```

```
  128.3.2.16|     10026403|
  128.3.2.46|        27946|
  128.3.2.96|       218605|
  128.3.2.98|          636|
 128.3.2.102|         1568|
```

Like sets, bags are a second-order binary structure for SiLK, meaning that they have their own toolkit (rwbagca t, rwbagtool, and rwbagbuild), the data is binary (so it can't be read with *cat* or a text editor), and they can be deriv ed from flow data or built from a datafile.

The basic bag generation tool is rwbag, which as seen in Example 5-22, takes flow data and produces a bag fil e from it. rwbag can generate 27 types of bags, simultaneously if you're so inclined. These 27 types comprise thre e types of counting (bytes, packets, and flows), and nine types of key (sip, dip, sport, dport, proto, sensor, input, ou tput, nhip). Combine the key and the counting type, and you have a switch that will create a bag. For example, to c ount all packets from source and destination IP addresses, call rwbag --sip-packets=b1.bag --dip-packets=b2.bag.

Advanced SiLK Facilities

In this section, we discuss more advanced SiLK facilities, in particular, the use of PMAPs and the collection an d conversion of SiLK data.

pmaps

A SiLK *prefix map* (PMAP) is a binary file that associates specific subnetworks (prefixes) with tags. PMAPs are used to record various mappings of a network, such as whether a network belongs to a particular organization or ASN, or country code lookup. Using a source such as GeoIP, you can build a PMAP that associates IP addresses w ith their country of origin.

The SiLK tool suite expects some basic PMAPs:

address_types.pmap

Describes an address's type, conventionally indicating whether the address is inside or outside of the network you are monitoring. Specify the default filesystem location for this PMAP using the SILK_ADDRESS_TYPES environ mental variable.

country_codes.pmap

This PMAP describes the country code for an address. Specify the default location for this PMAP using the SIL K_COUNTRY_CODES environmental variable.

PMAPs, like set files, can be created from text. Example 5-23 shows a simple PMAP file. Note the following att ributes:

§ The set of labels at the beginning. PMAPs do not store strings, but enumerable types identified by an intege r. This enumeration is defined using the labels. You can see that the PMAP in Example 5-23, for instance, stores a 3 to mark normal traffic.

§ The default key. Any value that doesn't match one of the network blocks listed in the map is given the defaul t value.

§ The actual declarations. Each declaration consists of a network specification, such 192.168.0.0/16, followe d by a label.

Example 5-23. PMAP Input

```
# This is a simple PMAP file that tracks some of the standard RFC 1918

# reserved addresses

#

# First we create some labels

label 0 1918-reserved

label 1 multicast

label 2 future

label 3 normal

#

# Specify the mode; this must be either ip or proto-port. ip in this case

# refers to v4 addresses

#

mode ip

#

# Everything otherwise not specified is normal

default normal

# Now the maps

192.168.0.0/16    1918-reserved

10.0.0.0/8        1918-reserved

172.16.0.0/12     1918-reserved

224.0.0.0/4       multicast

240.0.0.0/4       future
```

Once you've created a text representations of the PMAP, you can compile the binary PMAP file using the rwpm apbuild command. rwpmapbuild has two mandatory arguments: an input filename, with the file in the text format d escribed above, and a name for the output file. As with most SiLK commands, rwpmapbuild will not overwrite an e xisting output file. For example:

```
$ rwpmapbuild -i reserve.txt -o reserve.pmap
$ ls -l reserve.*

 -rw-r--r-- 1 mcollins staff 406 May 27 17:16 reserve.pmap

 -rw-r--r-- 1 mcollins staff 526 May 27 17:00 reserve.txt
```

Once a PMAP file is created, it can be added to rwfilter and rwcut using the pmap-file argument. Specifying the use of a PMAP file effectively creates a new set of fields in the filter and cut commands; since PMAP files are expli citly related to IP addresses, these new fields are bound to IP addresses.

Consider Example 5-24, which uses rwcut. In this example, the --pmap-file argument is colon-delimited; the val ue before the colon (reserve in the example) is a label, and the value after is a filename. rwcut binds the term reser ve to the pmaps for the source and destination IP address, creating two new fields: src-reserve (for the mapping of the source address to the PMAP) and dst-reserve (for the mapping of the destination address) to the PMAP.

Example 5-24. Creating the src-reserve and dst-reserve fields

```
$ rwcut --pmap-file=reserve:reserve.pmap --fields=1-4,src-reserve,dst-reserve

 traceroute.rwf | head -5

        sIP|         dIP|sPort|dPort|   src-reserve|   dst-reserve|

 192.168.1.12|  192.168.1.1|65428|   53| 1918-reserved| 1918-reserved|

 192.168.1.12|  192.168.1.1|56126|   53| 1918-reserved| 1918-reserved|

 192.168.1.12|  192.168.1.1|52055|   53| 1918-reserved| 1918-reserved|

  192.168.1.1|  92.168.1.12|   53|56126| 1918-reserved| 1918-reserved|

$ # Using the pmap in filter; note that rwcut is not using the pmap

$ rwfilter --pmap-file=reserve:reserve.pmap --pass=stdout traceroute.rwf

  --pmap-src-reserve=1918-reserved  | rwcut --field=1-5

  | head -5

sIP| dIP|sPort|dPort|pro|

192.168.1.12| 192.168.1.1|65428| 53| 17|

192.168.1.12| 192.168.1.1|56126| 53| 17|

192.168.1.12| 192.168.1.1|52055| 53| 17|

192.168.1.1| 192.168.1.12| 53|56126| 17|
```

Collecting SiLK Data

There are a number of different tools for collecting data and pushing it into SiLK. The major ones are YAF, whic h is a flow collector, and rwptoflow and rwtuc, which convert other data into SiLK format.

YAF

Yet Another Flowmeter (YAF) is the reference implementation for the IETF IPFIX standard, and is the standard flow collection software for the SiLK toolkit. YAF can read *pcap* data from files or capture packets directly, which it then assembles into flow records and exports to disk. It hasonline documentation. The tool itself can be entirely c onfigured using command-line options, but the number of options is fairly daunting. At its simplest, a YAF comma nd looks like this:

```
$ sudo yaf -i en1 --live=pcap -out /tmp/yaf/yaf
```

This reads data from interface en1 and drops it to the file in the temporary directory. Additional options control how data is read and how it is converted into flow and output format

yaf output is specified via the --out switch in tandem with the --ipfix and --rotate switches. By default, --out outp uts to a file; in the example above, the file is */tmp/yaf/yaf*, but any valid filename will do (if --out is set to -, then ya f will output to stdout).

When --out is specified with --rotate, yaf writes the output to files that are rotated by a delay specified by the --r otate switch (e.g., --rotate 3600 will update files every hour). In this mode, yaf uses the name specified by --out as a base filename, and attaches a suffix specified in YYYYMMDDhhmmss format, along with a decimal serial numbe r and then a *.yaf* file extension.

When yaf is specified with the --ipfix switch, it communicates IPFIX data to a daemon located elsewhere on the network. In this case (the most complicated option), --ipfix takes a transport protocol as an argument, while --out t akes the IP address of the host. The additional --ipfix-port switch takes a port number when needed. Consult the d ocumentation for more information.

The most important options are:

--live

Specifies the type of data being read; possible values formats are pcap, dag, or napatech. dag and napatech re fer to proprietary packet capture systems, so unless you have that hardware, just set --live to pcap.

--filter

Applies a BPF filter to the *pcap* data.

--out

The output specifier, discussed above. The output specifier will be a file, a file prefix, or an IP address dependin g on whatever other switches are used.

--ipfix

Takes a transport protocol (tcp, udp, sctp, or spread) as an argument, and specifies that output is IPFIX transp orted over the network. Consult the yaf documentation for more information.

--ipfix-port

Used only if --ipfix is specified. It specifies the port that the IPFIX data is sent to.

--rotate

Used only with files. If present, the filename in --out is used as a prefix, and files are written with a timestamp a ppended to them. The --rotate option takes an argument and the number of seconds before moving to a new file.

--silk

Specifies output that can be parsed by SiLK's rwflowpack tools.

--idle-timeout

Specifies the idle timeout for flows in seconds. If a flow is present in the flow cache and isn't active, it's flushe d as soon as it's been inactive for the duration of the idle timeout. Defaults to 300 seconds (five minutes).

--active-timeout

Specifies the active timeout for flows; the active timeout is the maximum amount of time an active flow will be stored in cache before being flushed. Defaults to 30 minutes (1,800 seconds). Note that the active timeout determ ines the maximum observed duration of collected flows.

YAF has many more options, but these are the basic ones to consider when configuring flows. Consult the YAF manpage for more details.

COOKBOOK: YAF

YAF has a ton of options, and how they operate together can be a bit confusing. Here are some examples of Y AF invocations:

Read yaf from an interface (en1) and write to a file on disk:

```
$sudo yaf -i en1 --live=pcap -o /tmp/yaf/yaf
```

Rotate the files every five minutes:

```
$sudo yaf -i en1 --rotate 300 --live=pcap -o /tmp/yaf/yaf
```

Read a file from disk and convert it:

```
$yaf <example.pcap >yafout
```

Run a BPF filter on the data, in this case for TCP data only

```
$ sudo yaf -i en1 --rotate 300 --live=pcap -o /tmp/yaf/yaf --filter="tcp"
```

Export the YAF data over IPFIX to address 128.2.14.11:3059

```
$ sudo yaf --live pcap --in eth1 --out 128.2.14.11 --ipfix-port=3059

  --ipfix tcp
```

rwptoflow

SiLK uses its own compact binary formats to represent NetFlow data that tools such as rwcut and rwcount pre sent in a human-readable form. There are times when an analyst needs to convert other data into SiLK format, suc h as taking packet captures from IDS alerts and converting it into a format where IP set filtering can be done on th e data.

The go-to tool for this task is rwptoflow. rwptoflow is a packet data to flow conversion tool. It does *not* aggreg ate flows; instead, each flow generated by rwptoflow is converted into a one-packet flow record. The resulting file c an then be manipulated by the SiLK suite as any other flow file.

rwptoflow is invoked relatively simply with an input filename as its argument. In Example 5-25, the *pcap* data fr om a traceroute is converted into flow data using rwptoflow. The resulting raw file is then read using rwcut and you can see the correspondence between the traceroute records and the resulting flow records.

Example 5-25. Converting pcap data with rwptoflow

```
$ tcpdump -v -n -r traceroute.pcap  | head -6

reading from file traceroute.pcap, link-type EN10MB (Ethernet)

21:06:50.559146 IP (tos 0x0, ttl 255, id 8010, offset 0, flags [none],

        proto UDP (17), length 64)

  192.168.1.12.65428 > 192.168.1.1.53: 63077+ A? jaws.oscar.aol.com. (36)

21:06:50.559157 IP (tos 0x0, ttl 255, id 37467, offset 0, flags [none],

        proto UDP (17), length 86)

  192.168.1.12.56126 > 192.168.1.1.53: 30980+ PTR?

  dr._dns-sd._udp.0.1.168.192.in-addr.arpa. (58)

21:06:50.559158 IP (tos 0x0, ttl 255, id 2942, offset 0, flags [none],

        proto UDP (17), length 66)

  192.168.1.12.52055 > 192.168.1.1.53: 990+ PTR? db._dns-sd._udp.home. (38)

$ rwptoflow traceroute.pcap > traceroute.rwf

$ rwcut --num-recs=3 --fields=1-5 traceroute.rwf

   sIP|  dIP|sPort|dPort|pro|

 192.168.1.12|  192.168.1.1|65428|   53| 17|

 192.168.1.12|  192.168.1.1|56126|   53| 17|

 192.168.1.12|  192.168.1.1|52055|   53| 17|
```

rwtuc

When correlating data between different sources, you will occasionally want to convert it into SiLK's format. rw tuc is the default tool for converting data into SiLK representation, as it works with columnar text files. Using rwtu c, you can convert IDS alerts and other data into SiLK data for further manipulations.

The easiest way to invoke rwtuc is to use it as an inverse of rwcut. Create a file with columnar entries and mak e sure that the titles match those used by rwcut:

```
$cat rwtuc_sample.txt
```

```
sIP        |dIP       |proto

128.2.11.4  | 29.3.11.4 | 6

11.8.3.15   | 9.12.1.4  | 17

$ rwtuc < rwtuc_sample.txt > rwtuc_sample.rwf

$ rwcut rwtuc_sample.rwf --field=1-6

 sIP| dIP|sPort|dPort|pro|   packets|

  128.2.11.4|  29.3.11.4|    0|    0|  6|         1|

   11.8.3.15|   9.12.1.4|    0|    0| 17|         1|
```

As the following fragment shows, rwtuc will read the columns, use the headers to determine column content, a nd stuff any unspecified fields with a default value if no column is provided. rwtuc can also take column specificati ons at the command line using the --fields and --column-separator switches, as so:

```
$cat rwtuc_sample2.txt

128.2.11.4  x 29.3.11.4 x 6 x 5

7.3.1.1    x  128.2.11.4 x 17 x 3

$ rwtuc --fields=sip,dip,proto,packets --column-sep=x < rwtuc_sample2.txt

  > rwtuc_sample2.rwf

$ rwcut --fields=1-7 rwtuc_sample2.rwf

  sIP|  dIP|sPort|dPort|pro|   packets|     bytes|

 128.2.11.4|  29.3.11.4|    0|    0|  6|         5|         5|

    7.3.1.1| 128.2.11.4|    0|    0| 17|         3|         3|
```

SiLK's binary format requires values for every field, which means that rwtuc makes a best guess for field value s that it doesn't have. For instance, the previous example specifies packets as a field but not bytes, so rwtuc just d efines the packet value to be identical to the byte value.

If there exists a common default value (e.g., all traffic has the same protocol), this value can be defined using one of a number of field-stuffing options in rwtuc. These options are identical to the field filtering options in rwfilte r, except they only take single values. For example, --proto=17 sets the protocol of every entry to 17.

In the fragment below, we use the field stuffing command --bytes=300 to set a value of 300 bytes for every entr y in *rwtuc_sample2.txt*:

```
$ rwtuc --fields=sip,dip,proto,packets --column-sep=x --bytes=300 <

  rwtuc_sample2.txt > rwtuc_sample2.rwf

$ rwcut --fields=1-7 rwtuc_sample2.rwf
```

```
   sIP|  dIP|sPort|dPort|pro|   packets|     bytes|
 128.2.11.4|   29.3.11.4|    0|    0|  6|         5|       300|
    7.3.1.1|  128.2.11.4|    0|    0| 17|         3|       300|
```

The resulting RWF file will contain a value of 300 bytes, even though the byte value is not in the original text fil e. The packet values, which are specified in the file, are set to whatever was specified there.

Further Reading

1. Time Shimeall, Sid Faber, Markus DeShon, and Drew Kompanek, “Using SiLK for Network Traffic Analysis,” Software Engineering Institute.

[5] You’ll notice that there are two datasets, one with scans and one without. To understand why, read Pang *et al.*, “The Devil and Packet Trace Anonymization,” ACM CCR 36(1), January 2006.