DRAFT: USER DOCUMENTATION FOR THE STIDE SOFTWARE PACKAGE

JULIE REHMEYER

1. Software Purpose

STIDE stands for Sequence Time-Delay Embedding, and it implements the time-delay embedding method of anomaly detection. Its primary function is to accept as input a time series (or a set of time series), divide it into a set of fixed-length sequences, compare that set of sequences with an existing database of fixed length sequences, and report on the consistency of the time series with the existing database. It can also be used to create a database of fixed-length sequences from scratch, or to add to a pre-existing database.

The STIDE software was originally developed by Steve Hofmeyr, a graduate student in the Computer Science Department at the University of New Mexico, as part of a research program that is applying ideas from immunology to problems in computer security. In particular, STIDE was written to assist in detecting intrusions by identifying the unusual sequences of system calls that may be created during an attempted intrusion [1, 2, 3, 4]. In this context, the time series being considered consists of the system calls made by a single process. We first record the system calls made by a process exhibiting normal behavior (i.e., in non-exploited situations), and then use STIDE to divide that continuous stream of system calls into sequences of a given length and store them in a database. Subsequently, when we want to know if another instance of the same program has been attacked, we record the system calls the process has generated and use STIDE to compare the resulting sequences of system calls with the database of normal sequences. A large number of sequences created by the potentially attacked process that weren't created by the uncompromised processes suggests that the process may have been exploited.

In practice, because of limitations in available system call tracing mechanisms, it is far easier for us to record simultaneously the system calls generated by several processes that are running at the same time. STIDE is designed to handle this sort of situation. It can simultaneously process multiple interwoven time series by requiring that each element in the input stream be preceded by an identifier to specify which series it comes from. In our work, that identifier is the process ID.

The simplest way that STIDE can analyze information about the consistency of new data with an existing database is to report the number of anomalous sequences, i.e., the number of sequences in the input which do not exist in the database.

It can also report the minimum Hamming distance [1]. Given a sequence from the data stream and a sequence from the database, we can compute the number of entries that are different between the two sequences and get the Hamming distance between those two sequences. The minimum of the Hamming distances between

Date: April 16, 1998.

the input sequence and all of the sequences from the database is the minimum Hamming distance for the input sequence.

The final option is that it can report a "locality frame count" [2]. When a process is exploited, there may be a short period of time (a locality) when the percentage of anomalous sequences is much higher. Although ten anomalies over the course of a long run may not be cause for concern, ten anomalies within thirty sequences might be. Thus it can be useful to observe how many anomalies there are *locally*. The number of sequences that are considered to be "local" to one another is called the size of the locality frame. In this mode, STIDE reports the largest number of anomalies it finds within any locality frame.

An additional advantage of calculating locality frame counts is that it provides an "on-line" measure. Ultimately, we are interested in a system which would detect intrusions as the system is running. Because locality frame counts are calculated locally, one can immediately be notified when an intrusion may be occurring.

2. Input Data Format

The input data consists of the time series to be analyzed. It is read from standard input. It is expected to be a series of pairs of positive integers, one pair per line, where the first integer identifies the data stream and the second integer is the element of the data stream. The end of the data stream can either be designated by the end of the file or by an occurrence of the number -1 as a stream identifier. In our work, the stream identifier is the process identification number (PID), and the elements of the data stream are system call numbers.

The following is a small example of an input file, tracking three processes, with PID's 744, 1069 and 9.

```
744   24
744   13
1069  4
1069  24
1069  4
744   5
9     24
1069  13
744   81
9     13
9     2
1069  5
1069  18
-1
```

If the number -1 occurs as a data element, STIDE interprets that as a missing data element. It does not form any sequences going through that data element. It clears the sequence and starts from scratch.

For example, suppose that the sequence length is 3 and the input is as follows:

```
220 14
220 185
220 20
220 -1
220 2
220 20
220 3
220 2
-1
```

STIDE would derive three sequences from this input: 14, 185, 20; 2, 20, 3; and 20, 3, 2.

3. Configuration Options

There are a number of options which affect STIDE's behavior. Every option has a default value. The values may be changed through command line arguments or through a configuration file. Values set by the configuration file override default values and values set by the command line override those set by either the configuration file or the defaults. The following options are available:

Short Name	Long name	Legitimate Values	Default Value
`a`	`add_to_db`	on or off	off
`c`	`config_name`	filenames	stide.config
`d`	`db_name`	filenames	default.db
`lf`	`lf_size`	1 – 999	1
`g`	`output_graph`	on or off	off
`l`	`seq_len`	1 – 199	6
`p`	`pair_offset`	integers	0
`s`	`write_db_stats`	on or off	off
`v`	`verbose`	on or off	off
`V`	`very_verbose`	on or off	off
`hd`	`compute_hdist`	on or off	off
`me`	`max_elements`	1 – 999	500
`ms`	`max_streams`	1 – 9999	100
`aof`	`add_output_format`	see below	see below
`cof`	`compare_output_format`	see below	see below

3.1. Descriptions of Options.

3.1.1. *Option* `add_to_db`. This flag indicates that you want the input data to be added to the database. If there is no pre-existing database, it indicates that you want to create a new database from the input data. Note that you cannot simultaneously compare data and add it to the database. If this switch is off, STIDE compares the input data with the database without adding it.

3.1.2. *Option* `config_name`. This is the name of the configuration file to be used. See Section 3.3 for more information about the configuration file.

3.1.3. *Option* `db_name`. This is the name of an existing database or the name under which to store a new database that will be created from the input data.

3.1.4. *Option* `lf_size`. This is the size of the locality frame (see Section 1 for an explanation of locality frame count). The value 1 effectively turns off locality frames.

3.1.5. *Option* `output_graph`. This causes STIDE to create a file `db_name.dot` containing a graph of the entire database forest formatted as input for the program Dot. Running Dot on the file translates it into PostScript format. The result is a graphical image of the database.

3.1.6. *Option* `seq_len`. A database stores trees of sequences of a set length. When building a new database, the length of the sequences to be stored is set with `seq_len`. When adding to or comparing with an existing database, one must use the same sequence length that was used when the database was generated. In those situations, STIDE will automatically figure out the correct sequence length and use it regardless of the user specification or the default.[1]

3.1.7. *Option* `pair_offset`. In `verbose` or `very_verbose` modes, STIDE reports on particular sequences of interest (see Sections 3.1.9 and 3.1.10). One of the pieces of information one might be interested in is where a particular sequence occurs in the input. Recall that the input data is a stream of pairs (stream number, element number), and each element in the sequence being considered came from one of those input pairs. STIDE reports on where the sequence occurred in the input by reporting the pair number of the last element of the sequence.

These numbers may be offset by a fixed amount by setting `pair_offset`.

3.1.8. *Option* `write_db_stats`. This flag causes STIDE to print out statistics on the database. The statistics it will print are the number of nodes in the database, the number of unique sequences, the number of branches, and the average database branch factor. See Section 4 for more information.

3.1.9. *Option* `verbose`. When adding to the database in `verbose` mode, STIDE will print information about each new sequence being added to the database, where the precise information is specified by the `add_output_format` parameter (see Section 3.1.14). When comparing the input data with an existing database in `verbose` mode, it will print information about each sequence that is itself a miss or whose locality frame contains a miss, where the precise information is specified by the `compare_output_format` parameter (see Section 3.1.15). In either case, when

[1]STIDE can do this for revision 1 databases only. STIDE can still process old-style databases, but cannot implement this feature. STIDE recognizes revision 1 databases by their initial line: `#DBrev: 1` and the following line: `#DBseq_len:` followed by an integer giving the sequence length. When STIDE processes an old-style database, it converts it to a revision 1 database if it is in `add_to_db` mode.

adding or comparing, STIDE will first print out a header with a list of the names of the variables being printed.

3.1.10. *Option* `very_verbose`. In `very_verbose` mode, STIDE will print out the information specified by `add_output_format` or `compare_output_format` for each sequence encountered in the input data, regardless of whether the sequence is new. As in `verbose` mode, STIDE will first print out a header with a list of the names of the variables being printed.

3.1.11. *Option* `compute_hdist`. This switch causes the Hamming distance [1] to be computed (see Section 1 for an explanation of Hamming distance).

3.1.12. *Option* `max_elements`. This is the maximum number of unique data elements that STIDE might encounter in the input data.

3.1.13. *Option* `max_streams`. This is the maximum number of different data streams that STIDE might encounter in the input data.

3.1.14. *Option* `add_output_format`. When adding to the database in `verbose` or `very_verbose` modes, STIDE will print the `add_output_format` string for every sequence of interest (see Sections 3.1.9 and 3.1.10). Substitutions are made for control characters as follows:

Control Char	Meaning
%s	Stream Identification Number
%d	Database Size
%p	Pair number of last data element of sequence in the whole input stream
%i	Pair number of last data element of sequence in its particular data stream
\t	Tab
\n	Newline

See section 3.1.7 for more information about the meaning of the %p and %i control characters.

The default value of `add_output_format` is:

```
"DB Size: %d\tStream: %s\tPair Number: %p\n"
```

3.1.15. *Option* `compare_output_format`. When comparing data in `verbose` mode, STIDE will print the `compare_output_format` string for every sequence which is itself an anomaly or whose locality frame contains an anomaly. In `very_verbose` mode, STIDE will print the string indicated for *every* sequence, regardless of whether it is an anomaly. Substitutions are made for control characters as follows:

Control Char	Meaning
%s	Stream Identification Number
%p	Pair number of last data element of sequence in the whole input stream
%i	Pair number of last data element of sequence in its particular data stream
%a	1 if this sequence is an anomaly, 0 otherwise
%c	locality frame count of this sequence
%h	Hamming distance
\t	Tab
\n	Newline

See section 3.1.7 for more information about the meaning of the %p and %i control characters.

The default value of `compare_output_format` is:

```
"Pair Number: %p\tStream Number: %s\n"
```

3.2. **Command-Line Arguments.** All parameters may be set using the command line, in one of two ways. The short name may be used, preceded by a hyphen and followed by a value (if appropriate). The long name may also be used, but it must be preceded by *two* hyphens and followed by a value (if appropriate). Values set by the command line override those set in any other way.

Switches which are simply turned on or off need not be followed by a value. Parameters may be set in any order. There must be space between the parameter name and the value. Flags may not be combined.

STIDE expects the input data to come from standard input.

3.2.1. *Examples.* To use STIDE to create a database called "our_data.db" from the input file "input1.dat" with sequences of length 10, using the default configuration file name, in verbose mode, with output format "`%p\t%s\t%d\n`", one could type:

```
stide -d our_data.db -a -l 10 -v -aof "%p\t%s\t%d\n" < input1.dat
```

To add the data from the file "input2.dat" to that database, using the same configuration file, not in verbose mode, and to create a graph in dot format, one could type:

```
stide -d our_data.db --output_graph --add_to_db -l 10 < input2.dat
```

Then to compare the data in file "input3.dat" to the database and have the results reported using locality frame counts with locality frame size 20, using the configuration file "run3.config", one would type:

```
stide -d our_data.db -f 20 -l 10 -c run3.config < input3.dat
```

3.3. **Configuration File.** All parameters may be set using a configuration file. The first line of a configuration file must be:[2]

```
#ConfigFileRev: 1
```

After the first line, lines may be commented out using a "#" sign. Each parameter is set on its own line, using the long name followed by a colon, followed by the value. Lines may be continued by putting a backslash as the last character of the line. White space at the beginning of lines will be ignored. Parameters which are simple switches may be set with the value "on" or "off", or with no value at all (which will turn them on).

Configuration file values override default values and are overridden by command-line values.

3.3.1. *Example.* The following is a sample configuration file:

```
#ConfigFileRev: 1
#Sample STIDE configuration file containing default values.

 db_name: default.db     # name of database
 seq_len: 6              # length of sequences
 max_elements: 500       # maximum number of unique elements
 # in input
 max_streams: 100        # maximum number of unique streams
   # in input
 pair_offset: 0          # offset for pair number count
 add_output_format:  \
         "DB Size: %d\tStream: %s\tPair Number: %p\n"
 compare_output_format:  \
         "Pair Number: %p\tStream Number: %s\n"
 lf_size: 1              # 1 causes locality frame counts not
                         # to be computed
 add_to_db: off          # Add this data to the database, or,
                         # if there is no database, create a
                         # new one -- do not do comparisons
 output_graph: off       # Outputs graphing information in Dot
                         # format
 compute_hdist: off      # Compute Hamming distances
 write_db_stats: off     # At end, print out statistics about
   # database
 verbose: off            # Verbose mode
 very_verbose: off       # Very verbose mode
```

4. Output Data

For every run, STIDE will first output the final configuration data assembled from the defaults, the configuration file and the command-line arguments, in a

[2] Old-style configuration files lack this line. STIDE will assume that configuration files that lack this line are old-style, and will try to parse them accordingly, issuing a warning to the user.

format which could be used as a configuration file. It then prints the number of different streams in the input, the total number of pairs read from the input, and the total number of sequences read from the input. The subsequent output depends on whether STIDE was adding to the database or making comparisons.

4.1. **Output Data About Comparisons.** If you have run the program to compare sequences, at the end STIDE will print out the number of sequences that were anomalous, and the percentage of sequences that were anomalous. If locality frame counts were being computed, STIDE reports the maximum locality frame count encountered in any stream, and if Hamming distances were being computed, STIDE reports the largest minimum Hamming distance of any sequence in any stream.

If the `verbose` switch was on and the `compare_output_format` parameter is set appropriately, STIDE will print out information about each sequence which is either itself an anomaly or whose locality frame contains an anomaly (if locality frames are being computed). If the `very_verbose` switch was on and the `compare_output_format` parameter is set appropriately, STIDE will print out information about each sequence, regardless of whether it is an anomaly. The precise information to be output is specified by the user in `compare_output_format`. See Section 3.1.15 for details on what information `compare_output_format` may request.

4.2. **Output Data About The Database.** If you are adding to the database, STIDE will print out the number of unique sequences in the input that were not previously in the database. One can get further information about the growth of the database by turning on `verbose` or `very_verbose` modes, and one can get information about the shape and complexity of a database using the `write_db_stats` switch.

4.2.1. *Database Growth Information.* In `verbose` mode, STIDE will print out information on each new sequence which is added to the database. In `very_verbose` mode, STIDE will print out information on each sequence read in, regardless of whether it is new. The information that STIDE produces is determined by the `add_output_format` parameter. See Section 3.1.14 for details on what information may be requested.

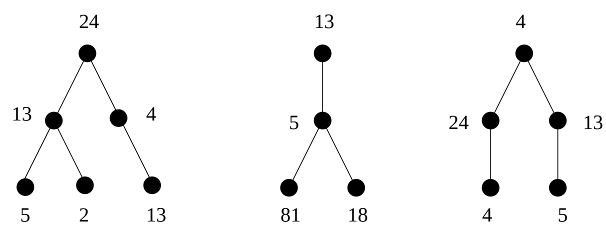
4.2.2. *Database Statistics.* The `write_db_stats` switch causes STIDE to print out information about the shape and complexity of the database. The `write_db_stats` switch may be used either when adding to the database or when making comparisons.

The sequences are stored as forests (groups of trees). Each path down each tree represents a sequence that STIDE has encountered. STIDE can compute the number of nodes on the trees, the number of leaves (leaves are the ends of the trees, i.e., the last element in a sequence), the number of branches, and the average branch factor, which is the number of branches divided by the difference between the number of nodes and the number of sequences.

For example, consider the sequences derived from the first sample input file in Section 2:

24, 13, 5
13, 5, 81
4, 24, 4
24, 4, 13
4, 13, 5
13, 5, 18
24, 13, 2

We can represent those sequences by the forest:



In this database, the number of nodes is 15, the number of leaves is 7, and the number of branches is 12. There are 7 unique sequences. The average branch factor is $12/(15-7) = 1.5$.

References

[1] S. Hofmeyr, S. Forrest, and A. Somayaji "Lightweight intrusion detection for networked operating systems." Submitted to *Journal of Computer Security* (July, 1997).

[2] S. Forrest, S. Hofmeyr, and A. Somayaji "Computer immunology" *Communications of the ACM* Vol. 40, No. 10, pp. 88-96 (1997).

[3] A. Somayaji, S. Hofmeyr, and S. Forrest "Principles of a Computer Immune System." New Security Paradigms Workshop (presented September, 1997).

[4] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff "A sense of self for Unix processes." In Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy, IEEE Computer Society Press, Los Alamitos, CA, pp. 120-128 (1996).