Regular Expression Searching

Regular expressions allow forensics analysts to search through large quantities of text information for patterns of data such as the following:

- Telephone Numbers
- Social Security Numbers
- Computer IP Addresses
- Credit Card Numbers

This data can be extracted because it occurs in known patterns. For example, credit card numbers are typically sixteen digits in length and are often stored in the following pattern or format: xxxx–xxxx–xxxx–xxxx.

This appendix explains the following:

- Understanding Regular Expressions
- Predefined Regular Expressions
- Going Further with Regular Expressions

Understanding Regular Expressions

Forensics analysts specify a desired pattern by composing a regular expression. These patterns are similar to arithmetic expressions that have operands, operators, sub-expressions, and a value. For example, the following table identifies the mathematical components in the arithmetic expression, 5/((1+2)*3):

Component	Example
Operands	5, 1, 2, 3
Operators	/, ( ), +, *
Sub-Expressions	(1+2), ((1+2)*3)
Value	Approximately 0.556

Like the arithmetic expression in this example, regular expressions have operands, operators, sub-expressions, and a value. How these expressions are created and used is explained using simple expressions followed by more complex regular expressions.

Note: Unlike arithmetic expressions which can only have numeric operands, operands in regular expressions can be any characters that can be typed on a keyboard, such as alphabetic, numeric, and symbolic characters.

Simple Regular Expressions

A simple regular expression can be made up entirely of operands. For example, the regular expression *dress* causes the search engine to return a list of all files that contain the sequence of characters *d r e s s*. The regular expression *dress* corresponds to a very specific and restricted pattern of text, that is, sequences of text that contain the sub-string *dress*. Files containing the words “dress,” “address,” “dressing,” and “dresser,” are returned in a search for the regular expression *dress*.

The search engine searches left to right. So in searching the regular expression *dress,* the search engine opens each file and scans its contents line by line, looking for a *d*, followed by an *r*, followed by an *e*, and so on.

Complex Regular Expressions—Visa and MasterCard Numbers

Operators allow regular expressions to search patterns of data rather than specific values. For example, the operators in the following expression enables the FTK's search engine to find all Visa and MasterCard credit card numbers in case evidence files:

\<((\d\d\d\d)[\–]){3}\d\d\d\d\>

Without the use of operators, the search engine could look for only one credit card number at a time.

Note: The credit card expression discussion in this section is included in FTK and is used here primarily for the explanation of advanced regular expressions.

The following table identifies the components in the Visa and MasterCard regular expression:

Component	Example
Operands	*d*, \–, spacebar space
Operators	\d, \, <, ( ), [ ], {3}, \>
Sub-Expressions	(\d\d\d\d), ((\d\d\d\d)[\–])
Value	Any sequence of sixteen decimal digits that is delimited by three hyphens and bound on both sides by non-word characters (xxxx–xxxx–xxxx–xxxx).

As the regular expression search engine evaluates an expression in left-to-right order, the first operand it encounters is the backslash less-than combination (\<). This combination is also known as the begin-a-word operator. This operator tells the search engine that the first character in any

search hit immediately follows a non-word character such as white space or other word delimiter.

Tip: A precise definition of non-word characters and constituent-word characters in regular expressions is difficult to find. Consequently, experimentation by FTK users may be the best way to determine if the forward slash less-than (\<) and forward slash greater-than (\>) operators help find the data patterns relevant to a specific searching task. The hyphen and the period are examples of valid delimiters or non-word characters.

The begin-a-word operator illustrates one of two uses of the backslash character (\), often called the escape character: the modification of operands and the modification of operators. On its own, the left angle bracket (<) would be evaluated as an operand, requiring the search engine to look next for a left angle bracket character. However, when the escape character immediately precedes the (<), the two characters are interpreted together as the begin-a-word operator by the search engine. When an escape character precedes a hyphen (–) character, which is normally considered to be an operator, the two characters (\–) require the search engine to look next for a hyphen character and not apply the hyphen operator (the meaning of the hyphen operator is discussed below).

The next operator is the parentheses ( ). The parentheses group together a sub-expression, that is, a sequence of characters that must be treated as a group and not as individual operands.

The next operator is the *\d*. This operator, which is another instance of an operand being modified by the escape character, is interpreted by the search engine to mean that the next character in search hits found may be any decimal digit character from *0-9*.

The square brackets ([ ]) indicate that the next character in the sequence must be one of the characters listed between the brackets or escaped characters. In the case of the credit card expression, the backslash-hyphen-spacebar space ([\-*spacebar space*]) means that the four decimal digits must be followed by a hyphen or a spacebar space.

Next, the *{3}* means that the preceding sub-expression must repeat three times, back to back. The number in the curly brackets ({ }) can be any positive number.

Finally, the forward slash greater-than combination (\>), also know as the end-a-word operator, means that the preceding expression must be followed by a non-word character.

Other Variations on the Same Expression

Sometimes there are ways to search for the same data using different expressions. It should be noted that there is no one-to-one correspondence between the expression and the pattern it is supposed to find. Thus the preceding credit card regular expression is not the only way to search for Visa or MasterCard credit card numbers. Because some regular expression operators have related meanings, there is more than one way to compose a regular expression to find a specific pattern of text. For instance, the following regular expression has the same meaning as the preceding credit card expression:

```
\<(\d\d\d\d)(\–| ){3}\d\d\d\d\>
```

The difference here is the use of the pipe (|) or union operator. The union operator means that the next character to match is either the left operand (the hyphen) or the right operand (the spacebar space). The similar meaning of the pipe (|) and square bracket ([ ]) operators give both expressions equivalent functions.

In addition to the previous two examples, the credit card regular expression could be composed as follows:

```
\<\d\d\d\d(\–| )\d\d\d\d(\–| )\d\d\d\d(\–| )\d\d\d\d\>
```

This expression explicitly states each element of the data pattern, whereas the {3} operator in the first two examples provides a type of mathematical shorthand for more succinct regular expressions.

Predefined Regular Expressions

FTK provides the following predefined regular expressions:

- U.S. Social Security Numbers
- U.S. Phone Numbers
- U.K. Phone Numbers
- IP Addresses
- Visa and MasterCard Numbers

The Social Security Number, U.S. Phone Number, and IP Address expressions are discussed in the following sections.

Note: The U.K. Phone Number expression is similar enough to the U.S. Phone Number that it does not warrant a separate discussion.

Social Security Number

The regular expression for Social Security numbers follows a relatively simple pattern:

```
\<\d\d\d[\– ]\d\d[\– ]\d\d\d\d\>
```

This expression reads as follows: find a sequence of text that begins with three decimal digits, followed by a hyphen or spacebar space. This sequence is followed by two more decimal digits and a hyphen or spacebar space, followed by four more decimal digits. This entire sequence must be bounded on both ends by non-word characters.

U.S. Phone Number

The regular expression for U.S. phone numbers is more complex:

```
((\<1[\–\. ])?(\([\<)\d\d\d[\)\.\–/ ] ?)?\<\d\d\d[\.\– ]\d\d\d\d\>
```

This expression demonstrates that regular expressions can be used to find more complex data patterns than simple credit card and Social Security number patterns.

The first part of the above expression,
`((\<1[\–\. ])?(\([\<)\d\d\d[\)\.\–/ ] ?)?`,
means, in effect, that an area code may or may not precede the

seven digit phone number. This meaning is achieved through the use of the question mark (?) operator. This operator requires that the sub-expression immediately to its left appear exactly zero or one times in any search hits. Therefore, the U.S. Phone Number expression finds telephone numbers with or without area codes.

This expression also indicates that if an area code is present, a number one (1) may or may not precede the area code. This meaning is achieved through the sub-expression (\<1[\–\.])?, which says that if there is a "1" before the area code, it will follow a non-word character and be separated from the area code by a delimiter (period, hyphen, or spacebar space).

The next sub-expression, (\([|\<)\d\d\d[\)\.\–/] ?, specifies how the area code must appear in any search hits. The \([|\<) requires that the area code begin with a left parenthesis or other delimiter. (Note that the left parenthesis is, of necessity, escaped.) The initial delimiter is followed by three decimal digits, then another delimiter—namely, a right parenthesis, a period, a hyphen, a forward slash, or a spacebar space. Lastly, the question mark (?) means that there may or may not be one spacebar space after the final delimiter.

The latter portion of this expression, \<\d\d\d[\.\–]\d\d\d\d\>, requests a seven-digit phone number with a delimiter (period, hyphen, or spacebar space) between the third and fourth decimal digit characters. Note that typically, the period is an operator. It means that the next character in the pattern can be any valid character. To specify an actual period (.), the character must be escaped (\.). The backslash period combination is included in the expression to catch phone numbers delimited by a period character.

IP Address

An IP address is a 32-bit value that uniquely identifies a computer on a TCP/IP network, including the Internet. Currently, all IP addresses are represented by a numeric sequence of four fields separated by the period character. Each field can contain any number from 0 to 255. The following regular expression locates IP addresses:

\<[1-2]?[0-9]?[0-9]\.[1-2]?[0-9]?[0-9]\.[1-2]?[0-9]?[0-9]\.[1-2]?[0-9]?[0-9]\>

The IP Address expression requires the search engine to find a sequence of data with four fields separated by periods (.). The data sequence must also be bound on both sides by non-word characters.

Note that the square brackets ([ ]) still behave as a set operator, meaning that the next character in the sequence can be any one of the values specified in the square brackets ([ ]). Also note that the hyphen (–) is not escaped; it is an operator that expresses ranges of characters.

Each field in an IP address can contain up to three characters. Reading the expression left to right, the first character, if present, must be a *1* or a *2*. The second character, if present, can be any value *0–9*. The square brackets ([ ]) indicate the possible range of characters and the question mark (?) indicates that the value is optional; that is, it may or may not be present. The third character is required; therefore, there is no question mark. However, the value can still be any number *0–9*.

Going Further with Regular Expressions

You can begin building your own regular expressions by experimenting with the default expressions in FTK. You can modify the default expressions to fine-tune your data searches or to create your own expressions.

Locating More Information on Regular Expressions

The World Wide Web contains many other reference materials and tutorials for regular expression searching. For example, the Website http://www.regular-expressions.info/ provides a regular expression for finding e-mail addresses. Keep in mind, however, that there is some variation among the search engines. Some of them differ in expression syntax, i.e., in the way that they form and use operands and operators.

Tip: Regular expression operators are often referred to as metacharacters in the regular expression literature.

See http://www.boost.org/libs/regex/syntax.htm#syntax for a definitive reference on the syntax employed by Regex++, the regular expression search engine bundled with FTK.

Note: The regular expression search engine used by FTK is called Regex++. It was created by Dr. John Maddock, a contributor to www.boost.org.

Common Operators

The following is a list of common operators:

Operators	Description
+	Matches the preceding sub-expression one or more times. For example, "ba+" will find all instances of "ba," "baa," "baaa," and so forth; but it will not find "b."
$	Matches the end of a line.
*	Matches the preceding sub-expression zero or more times. For example, "ba*" will find all instances of "b," "ba," "baa," "baaa," and so forth.
?	Matches the preceding sub-expression zero or one times.
[]	Matches any single value within the square brackets. For example, "ab[xyz]" will find "abx," "aby," and "abz." A hyphen (-) specifies ranges of characters with the brackets. For example, "ab[0-3]" will find "ab0," "ab1," "ab2," and "ab3." You can also specify case specific ranges such as [a-r], or [B-M].
[^ftk]	Matches any character except those bound by the [^ and the].

\<	Matches the beginning of a word. In other words, the next character in any search hit must immediately follow a non-word character.
\>	Matches the end of a word.
|	Matches either the sub-expression on the left or the right. For example, A|u will requires that the next character in a search hit be “A” or “u.”
\d	Matches any decimal digit.
\l	Matches any lowercase letter.
\s	Matches any white space character such as a space or a tab.
\u	Matches any uppercase letter.
\w	Matches any whole word.
^	Matches the start of a line.
{*n*,*m*}	Matches the preceding sub-expression at least *n* times, but no more than *m* times.
{*n*}	Matches the preceding sub-expression *n* times.