Perdix: A Query Language for Security Logs

Orr Bibring
ob2135@columbia.edu

Justin Prosco
jp2415@columbia.edu

Bing Wu
bw2236@columbia.edu

Angelika Zavou
az2172@columbia.edu

September 24, 2007

Abstract

This paper introduces Perdix, a high-level scripting language for the analysis of Linux `iptables` firewall logs. The language is designed to make parsing of logs more intuitive and easily scripted without the use of shell utilities such as `grep`, `uniq`, and `sort`, which are typically called in other scripting languages such as `bash`.

1 Introduction

Perdix was designed for the use of writing compact scripts to parse through security log files such as Linux `iptables` logs. These logs can get very large if they are not rotated, and sorting through them often requires the use of several different Unix utilities.

Microsoft released Log Parser, an application that allows the use of standard SQL syntax to parse through security logs. The product only works on Windows, however, and can only be used to parse Windows related logs. Perdix attempts to provide similar functionality to this application on the Unix platform.

Awk is also a similar language to Perdix, though awk is more generic in scope. Perdix is designed with the goal of parsing through security logs, and as such, the data types and syntax have been tailored to the type of information contained in these logs, such as IPv4 addresses, port numbers, and timestamps. A sample security log entry from the Linux `iptables` firewall is found below:

```
Aug 12 23:05:02 localhost kernel:  IPTABLES DROP (IN): IN=eth1 OUT=
MAC=00:04:5a:80:db:54:00:0d:66:27:34:54:08:00 SRC=10.100.100.10
DST=10.100.100.11 LEN=40 TOS=0x00 PREC=0x00 TTL=30 ID=49421
PROTO=TCP SPT=43091 DPT=80 WINDOW=3072 RES=0x00 URG PSH FIN URGP=0
```

Perdix is a scripting language for parsing through very large plain-text security logs. It is essentially a query language designed specifically for the purpose of gathering information from these log files.

The compiler for Perdix will be implemented in Java and will convert Perdix code into equivalent Java code, making extensive use of the Java String data type and utilities.

2 Functionality

Perdix is designed to provide a simple and convenient language for scripting the parsing of security log files. An administrator does not need to know how the log is formatted, but only the type of information that they want to obtain. With awk, for instance, the programmer would have to know that the timestamp appears first in the logfile so they could refer to column `$1`. In Perdix, however, the focus is on content rather than structure, and the programmer only needs to know that they are looking for a date.

There are also a limited amount of statistical features included, such as `count` and `unique`, which will count the amount of results returned or only return the unique results respectively, further enhancing the usability of the language.

The final output of a Perdix script is the results from the log file that match the specified query. There are several options for displaying the results, such as printing the results to the screen or writing them to an output file.

3 Language Features

The data types in Perdix have been optimized so that they match the type of network security information typically found in these log files. The data types correspond with the content typically found in these logs, rather than with generic data types.

Code in Perdix is written using English statements. Whitespace is irrelevant to the final executable and is stripped by the compiler. Each statement is completed by a semicolon (;), though function delcarations contain an implicit semicolon delineated by the right curly brace (}).

3.1 Data Types

- `date` : The date type is a calendar date in the format "Month Day Hour:Min:Sec" where Month is a three letter abbreviation for the calendar month and Hour is a 24-hour clock.
- `port`: The port data type is an integer.
- `address`: The ip datatype is an IPv4 address in the dotted format, such as 192.168.1.1.
- `protocol`: The protocol field is one of the following values: TCP, UDP, ICMP.
- `status`: The status field is one of the following values: ACCEPT, REJECT, or DROP.
- `log`: This data type is a reference to a specific log file that will be used for input/output operations.
- `string`: This datatype is used for string matching operations. This data type is used to provide string literals to search for within the log entries.

3.2 Operators

Expressions are formed from operators and operands; any expression including any assignment or a function call, can be a statement.

Perdix contains the following arithmetic operators, which may be used within a conditional statement:

- `=` : equals; evaluates to true when the lvalue equals the rvalue
- > : greater than; evaluates to true when the lvalue is strictly greater than the rvalue
- < : less than; evaluates to true when the lvalue is strictly less than the value of the rvalue
- >= : greater than or equal; evaluates to true when the lvalue is greater than or equal to the rvalue
- <=: less than or equal; evaluates to true when the lvalue is less than or equal to the rvalue
- `!=` : not equal; evaluates to true when the lvalue does not equal the rvalue

Logical operators are also defined in order to add or subtract results from the returned queries. The following logical operators are defined:

- `AND` : evaluates to true when both the lvalue and rvalue are true
- `OR` : evaluates to true when either the lvalue and rvalue are true
- `NOT` : evaluates to true when neither the lvalue and rvalue are true

3.3 Conditional Logic

Perdix, similarly to C, offers straightforward, single-threaded control. Perdix provides the fundamental control-flow *if-then-else* construct, which can be used with the above conditional logic for program control flow.

3.4 Comments

Comments are a single line beginning with the `#` character. Anything following a `#` character is assumed to be a comment and will be discarded by the compiler. An example of this is shown below:

```
# This is a comment
```

4 Functions

Functions are called using a function name and a set of parentheses. The beginning of every program contains the `start()` function. Functions do not have a return value, as they always return a string of text. Function bodies are contained within a set of curly braces { }, much in the same way that functions are contained in C. All variables within a Perdix program are local to the function in which they are defined in.

Additionally, since Perdix aims to be a compact scripting language, all functions must be defined in the same source file. There is no support for linking external files or headers at this time.

Function declarations are the only statement in Perdix that do not end with a semicolon, as they contain an implicit semicolon delineated by the right curly brace.

5 Example Program

The code below opens an `iptables` log file and searches for traffic that was allowed on port 80 to a target machine with the IP address 192.168.100.1. It then displays the results to standard output.

```
start()
{
     log file1 = "/var/log/iptables"
     open(file, R);

     address server_ip = "192.168.100.1";
     port server_port = "80";

     if (dst_ip =  sever_ip AND dst_port = server_port) then
     {
          # Print the values out to the screen
          print();
     }
     else
     {
          print("No results found");
     }
}
```

6 Limitations

Perdix is currently designed and optimized for reviewing Unix firewall log files, specifically those created by the Linux `iptables` firewall. While there are several other popular log file formats, including kernel messages from Unix syslog, Apache httpd logs, Microsoft Windows event logs and IIS logs, this language was designed to handle Linux `iptables` logs specifically. The data types have been designed to correspond to the specific fields contained within the `iptables` log.

While Perdix may be adapted to address parsing these other formats in the future, it can currently only be used for parsing Linux `iptables` logs. Nonetheless, Perdix is an extremely simple, effective and helpful tool for gathering crucial information from Linux security log files.