CS 640 project, Fall 2005, Milestone 1

Cristian Estan

September 15, 2005

1 Introduction

For the first milestone (September 22), you will have to do two things: fix a bug in the existing Netpy code, and describe the interfaces between modules that can support the functionality in the revised feature list. Both things require that you get some level of understanding of how the current Netpy code works and how it is structured. For the bugfixes you should submit patch files with your changes to the code (one patch per bug fixed) and where applicable, a short explanation. After milestone 1 we will switch to using CVS and a real bugtracker. For the interfaces between modules you should submit one text file. In addition each team should submit a "contributions.txt" file that gives a short summary of what each team member worked on.

2 Bugs to fix for milestone 1

2.1 Database team

2.1.1 Deterministic sampling

The database module may perform some sampling of the flow record data after it is read out of the database and before it is handed over to the analysis engine. If the analysis engine uses the DNS plugin it can trigger some expensive DNS lookups for every IP address in the data it receives. Thus it is important that if the same query is run twice the database returns exactly the same flow records so that the results of the lookups can be found in the DNS cache. Of course this does not apply if the database was updated between the queries. Check to see whether this sampling performed after reading the database is deterministic and if not, make it deterministic.

2.1.2 Flow counts

Flow counts are partially implemented. Try to enable the flow count functionality. If this proves too hard write a few sentences on where you got stuck. Note: this will require you to create a new database. Use raw flow records from the netpy-demo/sampleinput directory. Make sure to set the `NETPY_DB` environment variable to the directory you want to create the new database in.

2.2 Analysis team

Start up the demo, switch to bidimensional analysis, set the threshold to 10 in the view menu, and hit refresh. Hover your mouse over the various panes in the output, and details on the traffic of the corresponding traffic group will appear at the bottom of the window. The traffic volumes indicated there are inconsistent with the darkness of the panes which should convey the same information. For example the fourth pane from the top row is grey (which means it must have some traffic), but the bottom of the window indicates it has no traffic. Find out where the problem is and fix it.

2.3 Plugin team

There are no plugins in the current codebase, so for the bug fixing part you will work on the user interface.

2.3.1 Layout

The labels for the horizontal side of the report overlap for bidimensional reports, especially when you use small thresholds. However, when the unidimensional report is displayed horizontally it does not have this problem. Fix the layout for the bidimensional report.

2.3.2 Preferences

Write a **short** report on what the various options in the preferences dialog do. Explain which work and which don't, and fix all the typos in the names of the options.

3 Interface specifications

The database and analysis teams will have to specify what files they own in the current source tree. For all interactions between the code of your team and the code of other teams, give a detailed description of function semantics (including arguments) and shared data structures. The interfaces should suffice to support all the features in the revised feature list. Even though we do not have a user interface team, describe the interface between your module and theirs too. For this milestone you need not coordinate between teams to make sure that both teams sharing an interface have a consistent view of how it works. That will happen for the next milestone. Wherever existing interaction between modules is adequate, just describe the existing interface instead of coming up with a new one that would force you to write a lot of code. The plugin team should also give the non-standard interfaces to the DNS plugin and the "prefix owner" plugin that one would need to facilitate the implementation of the "address and port helper" feature described in Tuesday's document in section 3.3.1. The `interface_partial_example.txt` file from the class directory `/p/course/cs640-estan/public` gives an example for the level of detail you have to get to when specifying a function in an interface. You need not use this specific function in your interface between the plugin and the analysis module. You can use different notation and naming conventions. The important thing is that your interfaces easy to understand and most importantly unambiguous.

Feature, number of section describing it	Importance	Amount of work for each team		
		Database	Analysis	Plugin
Timestamps in database 2.5	High	Medium	None	None
Sampled NetFlow 3.5.1	High	Small	None	None
Faster analysis engine 3.1.1	High	None	Large	None
Plug-in hierarchy support 2.1.2	High	Small	Large	None
Strengthening filters 2.2.1	High	Medium	None*	None
User defined categories 2.1.3	High	None	None	Large
DNS IP hierarchy 2.1.4	High	None	None	Large
Packaging 3.4	High	Medium		
Faster database 3.1.2	Medium	Large	None	None
Flow counts 2.4.1	Medium	Medium	None*	None
SYN counts 2.4.2	Medium	Medium	None*	None
Accuracy feedback 3.3.3	Medium	Medium	None*	None
Automatic time selection 3.2.1	Medium	Medium	None*	None
Comparison reports 2.3	Medium	None	Large	None
Dest. port hierarchy 2.1.1	Medium	None	Small	None
Manual prefix hierarchy 2.1.5	Medium	None	None	Medium
Accuracy selection 3.3.4	Low	Medium	None*	None
Filter extensions 2.2.2	Low	Medium	None	Small
Packet header traces 3.5.2	Low	Small	None	None
Database size management 3.6.1	Low	Small	None	Small
Address and port helper 3.3.1	Low	Medium		
Navigation shortcuts 3.2.2	Low	Medium		
Filter drill-down work 3.2.3	Low	Medium		

Table 1: Shortened and re-prioritized feature table. Note how the scope of 2.1.5 has been reduced: interaction with the WHOIS servers is no longer required, the user will add the data about prefix ownership manually to a text file. The amount of work for 3.3.1 decreased because it can now rely on higher priority features: 2.1.4 and 2.1.5.

Example for specifying a function in an interface

The get_hierarchy_position is part of the interface between the analysis engine and a plugin. The analysis engine calls this function before performing the hierarchical heavy hitter analysis. Through this function the plugin maps a flow identifier to the internal encoding of its position in the hierarchy implemented by the plugin. This function must be implemented by every plugin, and the analysis module will get a function pointer to it during initialization.

```
void get_hierarchy_position(const struct flow_identifier* flowID_p,
                            int instanceID,
                            struct internal_encoding* result_p);
```

The flowID_p argument to this function is a pointer to the structure that holds the identifier of the flow.

```
struct flow_identifier{
  unsigned int src_IP;
  unsigned int dest_IP;
  unsigned short int src_port;
  unsigned short int dest_port;
  unsigned char proto;
};
```

The instanceID argument has the instance pointer which the analysis module obtains during initialization. The type of data it points to depends on the plugin. The plugin uses this pointer to differentiate the operation of the get_hierarchy_position() function for various instances of the plugin. For example one instance would map the source address to DNS names while the other would map the destination address.

The result_p argument holds the internal encoding of the position of the flow in the hierarchy implemented by the plugin. The length field gives the number of characters used by the actual hierarchy position encoding. The analysis module must make a copy of those bytes because the validity of the data pointed to by the hierarchy_position field is not guaranteed across function calls.

```
struct internal_encoding{
  unsigned char* hierarchy_position;
  unsigned short int length;
};
```

Example for describing a data structure used across an interface

The internal_encoding structure stores the internal encoding of the position of the a flow in the hierarchy implemented by the plugin. This structure is used by the analysis engine for all plugins. The length field gives the number of characters used by the actual internal encoding of the hierarchy position pointed to by the hierarchy_position field. This internal encoding of a position in the hierarchy is an explicit list of more and more specific groups (nodes in the tree representation of the hierarchy).

```
struct internal_encoding{
  unsigned char* hierarchy_position;
  unsigned short int length;
};
```

The bytes pointed to by hierarchy_position have a list of groups the traffic is part of starting with the most general and ending with the most specific. The group names can have any number of bytes and can contain all characters except the special separator character. Each module declares its separator character at initialization. The "*" group of all traffic (the root of the tree representation of the hierarchy) includes all flows and as such it is conceptually at the beginning of all lists of groups, but it is redundant so it should not be included. For example the DNS plugin can declare "." as its separator character and encode "www.cs.wisc.edu" as {hierarchy_position:"edu.wisc.cs.www", length:15}. The get_hierarchy_position function should map a flow identifier down to the most specific group it is part of, but the hierarchical heavy hitter analysis can produce encodings that end in groups for which there are more specific groups. These are represented with a trailing separator. For example {hierarchy_position:"edu.wisc.cs.", length:12} refers to various computers within the cs.wisc.edu domain while {hierarchy_position:"edu.wisc.cs", length:11} refers to the one computer called cs.wisc.edu.