BOTFLEX REFERENCE MANUAL

SHEHARBANO KHATTAK
SYSNET, NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD, PAKISTAN.

---

1. INTRODUCTION

Despite efforts to detect and contain botnets, the phenomenon continues to plague our cyber infrastructure. Several reports **reference** bear testimony to the fact that botnet operators share information and services to faciliate their propagation and survivability. Unfortunately, the defense landscape lacks the degree of cooperation required to address the botnet phenomenon. We have managed to churn hundreds of reports and research papers to address the issue of botnet measurement, detection and defense. However, **no** open source tool for botnet detection exists. This is particularly surprising in view of the fact that botnets have been around for a while. Some schools of thought may argue that releasing botnet specific detection and defense tools in open source will help botnet creators to evade them. This approach advocates 'security by obscurity' which has been proved wrong time and again. To address the issue highlighted above, we propose to build an open source bot detection tool, BotFlex.

BotFlex is an open source tool for bot detection and analysis. Its detection methodology is similar to BotHunter **reference**. A bot carries several activities during its existence, which when combined, comprise its lifecycle. These can be broadly categorized as Inbound scanning, Host exploit, Egg download, CnC and Attack...

2. DESIGN

2.1 To-Do

3. CAPABILITIES

3.1 What BotFlex can do

—Should be deployed close to home.

3.2 What BotFlex cannot do

—Won't do well in Bro Cluster setting because of the scan scripts. Even otherwise, this tool is not meant for monitoring large traffic.

3.3 To-Do

—Replace the file reading script in bro.bif, util .h and util.cc with the input framework that is expected to be released with Bro 2.1

—If the input framework comes with Bro 2.1, arrange for the values in config.txt to be changed at runtime.

—Can we run BotFlex interactively?

,,,,

—Package it as a proper software.

—Arrange for the values in config.txt to be set via switches in command line.

—Make skeleton attack.bro

—Test communication with a Broccoli enabled remote server

4. EXTENDING BOTFLEX

There is a standard way to interface scripts with the major events *Scan*, *Exploit*, *Egg Download*, *CnC* and *Attack*. Similarly, a new phase like the ones just enumerated can be integrated into the *Correlation* module in a distinct way. How to write the new script is a decision left to the coder. Nevertheless, there are some standard guidelines which can be utilized for writing scripts that adhere to the design philosophy of BotFlex.

As the core of BotFlex rests upon various events, one needs to decide forehand whether the addition that has to be made justifies a full-fledge script of its own, or a simple event in the main module of the phase under question. For example, a number of services on the Internet distribute blacklists of CnC servers based on information from their globally deployed sensors (similar to honeypots). A very simple check in the context of the *CnC* module is to check if an IP within the monitored network established communication with an IP from a CnC blacklist. It would be superfluous to write a script *cnc_match.bro* that has only a few lines of code to generate the event *cnc-match*. On the other hand, spam generation is an important aspect of botnet *Attack* phase, particularly in the context of spambots. Similarly, some botnets [Stewart 2008] try to spread by injecting malicious links into random websites through SQL injection. It would be too conservative to force SQL injection and spam detection logic into the same script. By nature, these deserve independent, stand-alone scripts of their own. This decision can be further facilitated by asking oneself the following questions:

—The logic to be added is an independent phenomenon (case B) or can be used to increase evidence of another independent phenomenon (case A).

—Does it need a separate data structure to maintain pertinent information, which can potentially be further extended.

—Does its information need to go into data structure of another module?

—Will you describe it as a complex event or a simple event?

Once that has been decided, we can look at how to handle the individual cases A and B.

4.1 Case A: Simple Event

Every phase of botnet life cycle is placed in a folder named after the phase to which it corresponds. Within the folder, the script with the name *¡dir_name¿.bro* accomodates simple events. The best way to understand the process is to make use of an example.

Lets say we need to add a simple event e to the phase X of botnet lifecycle. This can be done by following these simple steps:

—Browse to the folder X and edit *X.bro*.

,,,

—Make entries in the data structure *XRecord* that reflect information relevant to your logic which you want to be logged and shared with other scripts if/when required.

—Update *function getXRecord* to define intialization of the newly added entries.

—Add name of the simple *event e* to the enum *X_tributary* in *export* block.

—Add information about the recently added data items to *record Info* so that it can be logged.

—If you want your additional data items to be available in *Correlation* module, you will need to edit signature of the main *event X* in *export* block. Moreover, modify the signature wherever *event X* is handled. At the moment, *event X* is handled only in *Correlation* module and chances are that it will remain this way for some time.

—Write logic that contributes to *event X*. Usually this involves handling other events and triggering the *evaluate* function when certain conditions are met. As mentioned earlier, *function evaluate* returns a boolean value to indicate whether or not the major *event X* was triggered.

—This step pertains to when *function evaluate* yields 'True'. Clean up the data relevant to the simple *event e* from *table_X*. Additionally, delete the entry of *event e* as defined in *enum X_tributary* from *tb_tributary* in *table_X*.

If simple *event e* uses some threshold values, the following additional steps need to be taken to ensure they can be tweaked by editing the *config.txt* file.

—Define the thresholds in *export* block with some default value.

—Make entry in the file *botflex/config.txt* in this format:/sl¡X¿ ¡threshold_name¿ ¡value¿

—In *event bro_init*, arrange for the threshold value to be pulled from the data structure that holds configurable values as specified in *botflex/config.txt*. The typical syntax is:

```
if ( <threshold_name> in Config::table_config["X"] )
{
<threshold_name> = func(Config::table_config["X"]["threshold_name"]);
}
```

Note that if *threshold_name* is of type *string*, *func* in the above snippet might be skipped altogether. In most cases, the *string* value needs to be converted into proper format before handing it over from *Config::table_config* to the script variable. The appropriate conversion function can be found in Bro's own list of utility functions in *bro/src/bro.bif* and *bro/src/strings.bif*. Additionally, BotFlex also offers a few useful functions of its own which can be found in *botflex/utils/types.bro*.

4.2 Case B: Complex Event

Things are rather easy in this case as all you have to do is to copy paste an existing script for a complex event (*attack/spam.bro* would be a good start) and replace things with your own. The only consideration is to decide how you want to interface

your main event with *Correlation* module. For ease of discussion, lets assume that the complex event that has to be added to phase *X* is called *Y*. If *Y* is so complex that it has a dozen data items that need to be reported to the *Correlation* module. It doesn't make sense to add these to the main *event X*. It not only makes the code look ugly (imagine an event with a signature that spans multiple lines), but also it's difficult to keep track of the order of signature data items wherever *event X* is handled. In this case, it's best to report *event Y* to the *Correlation* module as it is. Remember, we can tell the *Correlation* module to tie different events with the same botnet lifecycle phase. On the other hand, if *Y* is a complex event but not entirely independent, you can report *event Y* to *X.bro*. This is the same as handling *event Y* in *X.bro*. Obviosuly, you will need to make appropriate entries in *X_tributary*, *XRecord* and *getXRecord()*.

4.3 Modifying Correlation Module

This step is common to both cases A and B. Once the basic logic has been implemented, you have to let *Correlation* module know so that it can be added to the evaluation process that decides whether or not a given host is a bot. The correlation module has two main parts. One that relates to making the 'bot or not' decision, and the other is responsible for maintaining a data structure that can be shared with a remote entity, lets say a correlation server.

4.3.1 *Modifications related to 'Decision'.* Lets continue with our previous example of adding event Y—y to the correlation logic. Whether *event Y* piggybacks on *event X* or reports to correlation module independent *event Y*, a common set of actions is applicable. Cases that are distinct to y or Y have been specifically identified:

—(For *event Y*) Write event handler for *event Y*.

—Make additional entries in *CorrelationRecord*.

—Add initialization logic for these entries in *get_correlation_record*

—Update relevant entries in the event handler and if certain conditions are met, update *tb_tributary* in *table_tributary* and trigger the *evaluate* function. If the evaluation succeeds (i.e. returns True), clean up.

4.3.2 *Modifications related to 'Data'.* A number of data structures are used through out BotFlex to hold pertinent data items. A common thread to all these data structures (tables, to be more specific) is that they expire after fixed time windows. This behavior is useful for analyzing events in a temporal fashion. The flipside is that we do not have persistent data anywhere. Imagine at some point we want to share a summary of the activities of a host till the time it was declared as a bot with a remote party. One might suggest sharing the final log file with the interested party. There are two main disadvantages to this approach:

—The log file can possibly contain a number of entries for the same host. This burdens the receiving party with the responsibility to use text processing tools to extract information of interest from the file.

—If only a small piece of information is required by the remote party, it is not justifiable to send the entire log file to R.

,,,

For this reason, we maintain a data structure *table_bot* that contains a summary of activities per host that was classified as a bot. To add Y or y to this table, make appropriate entries in *BotRecord*, *get_bot_record*, and other get_* records if applicable. Wherever event related to Y or y is handled, make sure you update the corresponding values in *table_bot*.

5. INSTALLATION

BotFlex has not been packaged per se. This will happen soon, though. Until that happens, here is a quick guide that explains how to get BotFlex to work (Nix only :-)). The good news is that BotFlex does not have any additional dependencies except for the ones it inherited from Bro.

(1) Download botflex from `https://github.com/sheharbano/bsg/tree/master/botflex`
(2) Download Bro source (which is a compressed folder of the form bro.tar.gz). For details, please look at Appendix (this is not optional).
(3) Extract to a place of your choice, but remember the path.
(4) cd to the folder where Bro was extracted and modify bro source code as described in botflex/bro_modifications (5 minutes of copy pasting :-)).
(5) Assuming you are still in the directory where bro was extracted (bro), do the usual (i) ./configure (ii) make (iii) make install
(6) Move botflex to /usr/local/bro/share/bro/site. Note: The above is the typical directory hierarchy of Bro unless you have specifically changed the path during ./configure stage. If that's the case, use the path you specified during configuration.
(7) 5. For best results, considering modifying /usr/local/bro/share/bro/site/botflex/config.txt according to your own environment. In particular, take time to specify local network. The default thresholds will be effective unless changed in the file mentioned above.
(8) Add the following line to /usr/local/bro/share/bro/site/local.bro @load botflex/detection/correlation/correlation
(9) We are all set now. For live traffic analysis, do bro -i ¡interface¿ local.bro For analyzing a previously generated trace file, do bro -r ¡file.pcap¿ local.bro
(10) Log files will be generated in your current directory. Bots are listed in correlation.log (TO-DO). Other log filenames are self explanatory.

6. CONCLUSION

A. APPENDIX A: DOWNLOADING BRO

The detailed quickstart guide can be found at `http://bro-ids.org/documentation/quickstart.html` To cut a long story short, we further summarize Bro download here:

(1) Download Bro:

```
git clone --recursive git://git.bro-ids.org/bro
```

(2) Get dependencies: (RPM/RedHat based Linux)

,,,

```
sudo yum install cmake make gcc gcc-c++ flex bison libpcap-devel
openssl-devel python-devel swig zlib-devel file-devel
```

(DEB/Debian-based Linux)

```
sudo apt-get install cmake make gcc g++ flex bison libpcap-dev
libssl-dev python-dev swig zlib1g-dev libmagic-dev
```

(Others) Check the above page. Haven't tested on other platforms.

(3) Set path. (Bourne-shell syntax)

```
export PATH=/usr/local/bro/bin:\$PATH
```

(C-shell syntax)

```
setenv PATH /usr/local/bro/bin:\$PATH
```

REFERENCES

STEWART, J. 2008. Danmec/asprox sql injection attack tool analysis. http://www.secureworks.com/research/threats/danmecasprox/. Online accessed: 12/5/2012.

,,,