An Overview of the Botnet Simulation Framework

Leon Böck[1] and Shankar Karuppayah[1,2] and Max Mühlhäuser[1] and Emmanouil Vasilomanolakis[3]

[1] Technische Universtiät Darmstadt, Germany, [2] Universiti Sains Malaysia, Malaysia, [3] Aalborg Universtiy, Copenhagen, Denmark

This paper was presented at Botconf 2020, Nantes (online webinar), 1-4 December 2020, www.botconf.eu
It is published in the Journal on Cybercrime & Digital Investigations by CECyF, https://journal.cecyf.fr/ojs

Abstract

Conducting research on botnets is oftentimes limited to the analysis of active botnets. This prevents researchers from testing detection and tracking mechanisms on potential future threats. Specifically, in the domain of P2P botnets, the protocol specifics, network churn and anti-tracking mechanisms greatly impact the success or failure of monitoring operations. Moreover, experiments on real world botnets, commonly lack ground truth to verify the findings. As developing and deploying botnets of sufficient size is accompanied by large costs and administration efforts, this paper attempts to address this issue by introducing a simulation framework for P2P botnets called Botnet Simulation Framework (BSF). BSF can simulate monitoring operations in botnets of more than 20.000 bots to evaluate tracking mechanisms or simulate takedown efforts. Moreover, communication traces can be exported to inject traffic into arbitrary PCAP files for training and evaluation of intrusion detection systems.

Keywords: Botnets, P2P Botnet, Simulation, Dataset Generation.

1 Introduction

A botnet is a network of inter-connected malware-infected machines called bots. Bots can be remotely controlled by a *botmaster* to carry out a multitude of malicious activities including Distributed Denial of Service (DDoS), credential theft, sending spam mail, distributing ransomware or stealing personal information.

While the specific functionalities of a botnet may vary, a Command and Control (C2) channel is always present to facilitate remote control access by the botmaster. Up until today, many botnets use a centralized C2 architecture, leveraging existing protocols and services such as IRC, HTTP or HTTPS. However, centralized C2 represents a single point of failure that makes it easier to take down such botnets. To overcome this drawback, many advanced botnets use mechanisms such as Domain Generation Algorithms (DGAs) or fast-flux DNS to hide and distribute the C2 infrastructure. Some botnets such as Gameover Zeus [1], Sality [2] or Hide'n Seek [3] leverage Peer-to-Peer (P2P) networks, to fully distribute the C2 among all infected machines, making them highly resilient against takedown attempts.

A common way to track P2P botnets is reverse engineering the malware and re-implementing the protocol in tracking mechanisms, namely crawlers and sensors, to monitor the botnet. To hamper monitoring efforts, botmasters oftentimes implement countermeasures such as rate limiting [2] and automated blacklisting mechanisms [1]. Evaluating the impact of such countermeasures, or developing more advanced monitoring mechanisms is difficult, as experiments on real botnets are limited to active botnets and oftentimes lack ground truth for comparison. A simulation framework presents a method to alleviate the aforementioned drawbacks. Such a framework can provide a flexible environment in which monitoring mechanisms can be evaluated against a variety of different settings of botnets and monitoring countermeasures. In addition, these evaluated settings can also be verified against the ground truth, something that was not easily doable in a live botnet. For instance, a botnet takedown strategy can be carefully planned and evaluated for its effectiveness before carrying it out on a targeted active botnet.

In this paper, we present an overview of the Bot-

net Simulation Framework (BSF) and its functionalities to simulate P2P botnets.[1] BSF is capable of simulating botnets of more than 20.000 bots and evaluate monitoring mechanisms against advanced countermeasures and novel botnets. Moreover, BSF can be used in conjunction with an external tool called Intrusion Detection Dataset Toolkit (ID2T) [4] to inject P2P botnet traffic into existing Packet Capture (PCAP) files, to test and improve Intrusion Detection System (IDS) and botnet detection mechanisms.

The remainder of this paper is structured as follows. First, Section 2 provides the reader with background information and the related work. Section 3 provides an overview of BSF. Section 4 describes how traffic can be injected into network captures using ID2T. Section 5 provides an evaluation of BSF and finally Section 6 concludes the paper.

2 Background and Related Work

In the following, we present some background and related work which are useful to guide readers in understanding our contribution in the remainder part of this paper.

2.1 Unstructured P2P Botnets

A P2P botnet may be deployed as either a structured or an unstructured P2P botnet. Considering the fact that the majority of currently known P2P botnets are unstructured P2P botnets, we detail some of the common inner workings of the unstructured P2P botnets in this subsection. Please note that here onward, we shall refer to unstructured P2P botnets simply as P2P botnets for ease of readability.

As bots in a P2P botnet do not rely on any central entities, e.g., servers, they solely depend upon other bots in the botnet to remain connected with and to the botmaster, i.e., for command and updates. Bots form an *overlay* that consists of neighborhood relationships between a bot and a subset of other bots in the overlay. The neighborhood relationship of a bot is stored locally and often referred as the *Neighborlist (NL)* in the literature [5].

The establishment and maintenance of this overlay is handled in a distributed manner by all participating bots through a *Membership Management (MM)* mechanism [5]. This mechanism ensures that bots regularly probe for the responsiveness of the neighbors in their NL. Neighbors that are found to be unresponsive, e.g., due to disinfection or churn effects, can be replaced by requesting for additional candidates from other responsive bots. Each entry in a NL minimally consists of an IP address and optionally the port of bots that are directly reachable over the Internet, i.e., superpeers. Bots that are behind NAT-like devices or stateful-firewalls, i.e., non-superpeers, would not remain in the neighborlists of bots as they are not directly reachable. Instead, they rely upon the superpeers to remain connected to the overlay and to get new updates from the botmaster.

In most P2P botnets, the process of probing for the responsiveness of the neighbors are also coupled with the feature to request or share the latest command from the botmaster with the neighbor. As such, new commands from the botmaster can be (eventually) disseminated to all bots connected to the overlay in a hop-to-hop fashion. As such, the features within a MM mechanism enables P2P botnets to inherit self-healing and self-organizing properties that make them valuable to the botmasters due to the low maintenance effort and operating cost [5].

2.2 Monitoring Unstructured P2P Botnets

Realizing the growing threat of P2P botnets, researchers and law enforcement agencies have actively started monitoring botnets to enumerate infections as well as to plan potential takedowns [6] . As a prerequisite to monitoring, the custom communication protocol of the botnets under scrutiny needs to be reverse engineered. Along with that, the MM mechanism of the botnet also needs to be understood before custom tools such as *crawlers* and *sensors* can be deployed within the botnet to monitor them [5].

Crawlers implement the neighborlist request feature of the botnet's MM mechanism to interact with the bots to discover the inter-connectivity between bots. However, only superpeers may be discovered by the crawlers. It has been reported in the literature that non-superpeers in a botnet typically consists of $40 - 60\%$ of the overall population [6].

Sensors are used to increase the visibility on both superpeers and non-superpeers in a botnet [7]. The main task of a sensor is to reliably respond to all incoming probing messages from the bots. After a sensor is injected into the NL of some superpeers, it has to wait until the information of itself being shared by superpeers to other nodes that may request them for their neighbors. Given enough time, the information about the sensor would be present in the NL of most bots in the overlay. Next, when a non-superpeer requests for neighbors, information of the sensor would eventually be returned by a superpeer. Subsequently, the sensor would be able to identify the non-superpeers based on the incoming messages, i.e., IP address information. The main drawback of a sensor is the fact that it can only enumerate bots but cannot obtain the inter-connectivity among them.

2.3 Network Simulators

P2P botnets are an interesting phenomenon to be studied. However, it is not always possible to study them in the wild, i.e., active botnets in the Internet. Any active interaction with the bots, e.g., testing strategies of a botnet takedown attempt, may be noticed by

[1] We want to point out, that even though the name is generic, BSF specifically addresses P2P botnets.

the botnet operators. Moreover, one could only study botnets that are out there in the wild. For those botnets that are no longer active or to study newer strategies implemented within botnets, it would be useful to study them in a simulation environment. Such a simulation environment would allow the evaluation of the influence of any past, current, or future botnet's design decision more thoroughly.

Nevertheless, to the best of our knowledge, there are no P2P botnet simulators that are available for this purpose, i.e., simulation of the P2P botnet communication overlay. However, there are several generic discrete event network simulators that could be leveraged to establish such a botnet simulator.

The network simulators ns-2 [8] and ns-3 [9] support simulating various network protocols, routing algorithms, and network types, e.g., wired/wireless networks. In addition, they also allow extending the simulator with user-defined protocols and devices that provide user interface support.

In addition to the above, OMNeT++ [10] is also another discrete event simulator that are widely used as network simulators. Moreover, thanks to modularity concept of OMNeT++, several interesting and useful frameworks, e.g., INET[2], have been developed using OMNeT++. Simulation using OMNeT++ only needs a configuration file to be filled up by the user before allowing the simulation to take place. This configuration file specifies the various parameters and settings that the simulation should be instantiated and executed with.

In the next section, we introduce our botnet simulator that is developed on top of OMNeT++.

3 Botnet Simulation Framework

The open source Botnet Simulation Framework (BSF)[3] is built on top of the OMNeT++ discrete event simulation framework [10]. Following the module-based structure of OMNeT++, BSF provides a flexible and extensible simulator for P2P botnets. Among its core features, bots can join P2P overlays, exchange messages, and experience churn behaviors, i.e., (re-)joining and leaving the network, following realistic distributions. Moreover, BSF facilitates experimentation with crawler and sensor nodes. Finally, several modules can be used to export the experimental results in the form of statistics, graph representations of the network topology or a message traces.

3.1 Framework Overview

Figure 1 presents an overview of BSF and its components. In the following, we describe the core components in greater detail.

[2]https://inet.omnetpp.org/

[3]https://github.com/tklab-tud/BSF

3.1.1 Configuration Files

As mentioned in Section 2.3, the OMNeT++ configuration file is used by modules to obtain the values for the various parameters and settings required for the simulations. BSF also leverages this configuration file to instantiate and provide the botnet-specific parameters for the simulation. This enables easy configuration and changing of simulations without modifying the code itself. Moreover, existing configuration files can be extended to carry out another experiment with minor changes without changing the existing configuration file. This in turn allows a user to carry out as many simulations as needed with the possibility to replicate the results with the specific configuration files.

3.1.2 Bots

The bot modules represent the core of the simulation framework. These modules generally extend a *NodeBase* module, that takes care of connecting to the simulated network and handles general message forwarding and receiving capabilities of each bot. To implement a specific botnet protocol, one may extend the NodeBase module, as it provides the greatest flexibility at the cost of additional effort in implementing all functionality. A quick way of implementing a botnet protocol is to use the provided *SimpleBot* module, that is highly configurable and follows the general concepts of unstructured P2P botnets such as Gameover Zeus [1], Sality [2] or Hide'n Seek [3]. More specifically, SimpleBot maintains and updates a neighborlist of known bots, frequently checks the availability of these bots and exchanges commands and updates using three different message pairs commonly implemented by unstructured P2P botnets.

Node IDs Each bot in BSF is assigned a unique identifier called *Node ID*. This identifier is used for routing messages, computing and exporting statistics, and addressing the nodes from global modules. In an abstract sense, the Node ID is a generic replacement for an IP address within BSF. Node IDs are assigned in a incremental fashion to all nodes present in the simulation, guaranteeing that all IDs $0 \leq ID < totalnodes$ exists. This is necessary for features such as retrieving a random bootstrap peer from the global nodelist.

Neighborlists NLs are separate modules within BSF. SimpleBot is designed to work with flexible neighborlist imlementations, that can be specified at the start of a simulation. The default implementation is the *SimpleNL* neighborlist. SimpleNL stores known bots in a map structure, mapping a bots ID to a *SimpleEntry* object. A SimpleEntry allows to keep track of when a bot last replied and how many requests were unanswered. Moreover, the SimpleNL can be freely

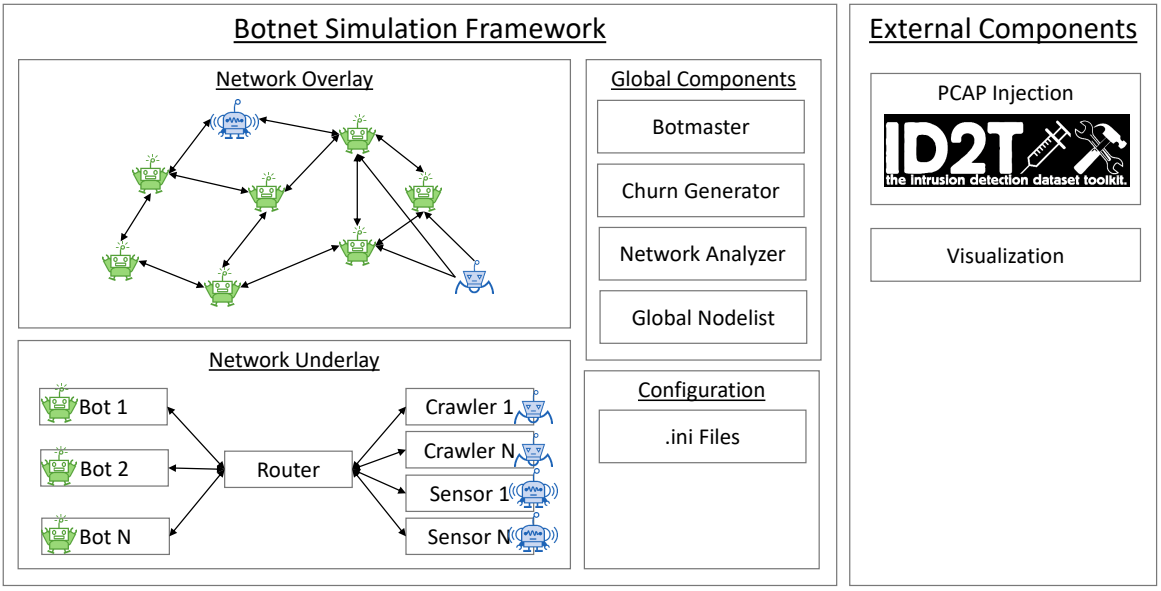

Figure 1: Overview of BSF and its components

configured in size and provides several methods to retrieve the stored entries, such as accessing by ID or retrieving a random entry.

Messages In its current form, *SimpleBot* provides six types of messages: Ping and Pong, command request and reply, and neighbor request and reply messages.

The purpose of the Ping and Pong messages is to probe if a bot is still active or responsive. Furthermore, it contains an optional *version* field, that can be used to simulate and track the current version of other bots in the botnet.

The second type of messages implemented within SimpleBot are command request and reply messages. These can be used to request the latest set of updates or commands from another bot. In its default setting, these messages are exchanged if a Ping Pong exchange reveals that one of the bots has an older version and therefore requests for the latest update using a command-request.

The third type of messages are neighbor-requests and neighbor-replies. These are used by bots to fill or update their neighborlists. A neighbor-request message is triggered, if the size of a bot's neighborlists falls below a configurable threshold. A bot will then contact one of its responsive neighbors and requesting additional peers. The contacted neighbor will then reply with a neighbor-reply message, that contains a configurable subset of bots from its own neighborlist.

Membership Maintenance The last core component of the SimpleBot implementation is its MM mechanism. It is triggered at a pre-specified interval and then iterates through all neighborlist entries, sending Ping-messages to check if they are still online. If peers have not replied within a certain period or have not replied to a configurable number of requests, they will be removed from the neighborlist. If the neighborlist is below the configured threshold, the maintenance loop will also actively send out neighbor-requests to try to fill up the empty slots in the neighborlist.

3.1.3 Networking

Most research in monitoring P2P botnets is focused on the overlay network, rather than the underlying Internet infrastructure. In order to reduce complexity and increase efficiency, BSF currently provides a simple star network topology consisting of a single router connecting all bots. We want to point out, that BSF is not limited to only this kind of network and one could implement more complex networks, if required. Connecting to the underlying network is handled automatically by the NodeBase module. Messages are routed by a central router based on the assigned NodeIDs. While we could theoretically establish direct connections between all bots without an intermediate router, the intermediate router provides several desirable benefits. First off, the central router observes all messages exchanged between any type of node in the network. This allows us to record and export the entire message exchange of the simulation for analysis purposes. Second, one could easily implement new features such as packet loss or man-in-the-middle attacks on top of the default router implementation. Lastly, the router implementation serves also as an ideal location to introduce latency into the message exchange between bots, i.e., to simulate delays.

3.1.4 Monitoring Nodes

One of the main goals of BSF is the development and evaluation of monitoring mechanisms in different environments. At the heart of any P2P botnet monitoring operation, we want to collect information about the infected bots, how they are interconnected, and when they are online. For that, BSF implements base classes

for collecting and exporting data of typical monitoring mechanisms such as crawlers and sensors. Essentially, a monitoring node is just another extension of the NodeBase module and has all the capabilities of sending and receiving messages of regular bots. However, thanks to the modularity and configuration options provided by OMNeT++, monitoring nodes can be customized widely. One example that is already implemented, is a crawler for the *SimpleBot* botnets.

The capabilities of simulating and evaluating different P2P botnets and monitoring mechanisms have already been showcased in two of our previous publications [11, 12], where we investigated the capabilities of monitoring mechanisms against advanced anti-monitoring mechanisms. A key advantage, even over real world experiments, is that we can compare the monitoring results to the ground truth extracted from the simulator. Therefore, this provides researchers with the unique opportunity to accurately measure the effectiveness and drawbacks of different (proposed) monitoring mechanisms in various settings.

3.1.5 Churn

One of the main challenges of simulation environments is to closely resemble the reality. In the case of P2P botnet simulation, even if one re-implements a botnet protocol to perfection, it is still necessary to accurately model the behavior of bots. While most of this behavior is described by the protocol, the churn behavior is influenced by various external factors such as diurnal patterns, network congestion or outages, IP churn or migration of mobile devices such as phones and laptops.

To address this issue, BSF has introduced an advanced method to recreate churn based on real world measurements [11]. This churn generator is called *LifetimeChurn* and can be freely configured using three parameters. The first parameter is the *target node count* and specifies the desired population of bots being active at any point in time. The specified value is an average, that dynamically affects the rate of bots (re-)joining or leaving the active population of the botnet. The remaining two parameters are the *shape* and *scale* of the Weibull [13] distribution specifying the lifetime behavior of bots. We chose a Weibull distribution, as related works on measuring churn in P2P networks and botnets have reported good fits of Weibull distributions on the churn behavior measured in the real world. This allows us to recreate the heavy tailed distributions of bots within BSF, i.e., many bots have shorter lifetimes, whereas lesser bots have very long lifetimes. The current default implementation uses parameters measured for the Sality botnet in 2015 [14], but can easily be replaced in the configuration file.

While LifetimeChurn aims to recreate real world churn behavior as close as possible, sometimes we do not want nodes to churn at all. The most common example is the use of monitoring nodes, that are commonly under our control and continuously online. For this scenario, we have another churn generator called *NoChurn*. As the name suggest nodes managed by this churn generator are not affected by churn at all.

As it is desirable in many cases to have different nodes affected by different or no churn behavior, BSF allows to assign sets of nodes to specific parameterized churn generators. As an example, one could use two different LifetimeChurn instances for superpeers and non- superpeers respectively and a third NoChurn instance for monitoring nodes. This provides great flexibility in modeling churn behavior within BSF.

3.1.6 Statistics and Data Export

As OMNeT++ provides the full flexibility of the C++ programming language, one could implement data analysis features directly in BSF. However, at the current point in time, this is only implemented to a limited degree. For most purposes, we have decided to leverage existing tools and libraries from a broad spectrum of programming languages, such as networkx [15], numpy [16], scipy [17], R [18] or gephi [19]. To leverage these existing tools, BSF implements a variety of data exports such as, overlay graph snapshots, general and command message traces, or monitor node data. Table 1 provides a short overview of the available data exports and their format.

Description	Filetype
Message Trace	csv
Command Trace	csv
Monitor Data	Matrix representation
Graph Snapshots	graphviz format

Table 1: Available data export formats

While general purpose libraries for machine learning, visualization or statistics are abundant, the framework provides a simple python application for visualizing differences between monitoring and ground truth graphs in a browser. Figure 2 depicts a simple comparison of the entire simulated network vs the view obtained by a crawler.

Moreover, we integrated functionality into ID2T to inject botnet communication traces into real world PCAPs, which will be described in the following section.

4 Network Traffic Injection

While the core functionality of BSF is focused on P2P botnet simulation from an overlay perspective, this is insufficient for mechanisms that rely on a detailed representation of lower network layers. To address this issue, we have implemented functionality to export BSF traces, that can be injected into any PCAP file using an external tool called ID2T.

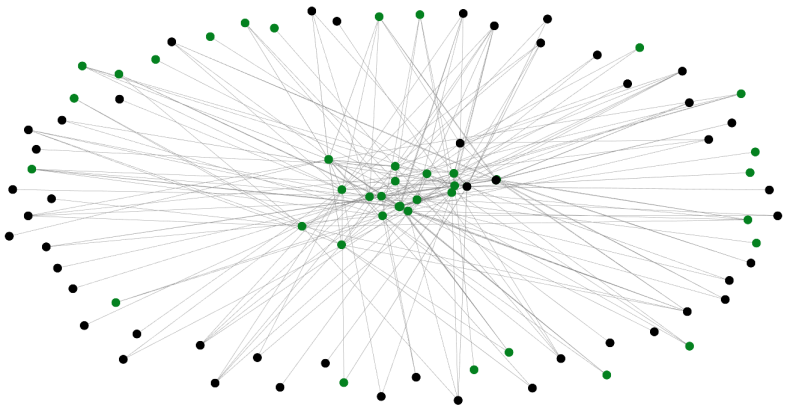

Figure 2: Visualization example of BSF: Comparison of all active nodes and the information obtained by a crawler in green.

4.1 Intrusion Detection Dataset Toolkit

The Intrusion Detection Dataset Toolkit (ID2T) [4, 20] is a framework for creating and injecting synthetic attack network traffic for the evaluation of intrusion detection algorithms and systems. In particular, ID2T receives as input a network Packet Capture (PCAP) file of arbitrary size and produces as output a new PCAP file that includes synthetically generated and labeled attacks. ID2T does not just merge PCAP files with simulated attacks; rather, to produce a realistic dataset, it analyzes the input traffic and replicates its properties when generating attacks [20].

Furthermore, ID2T supports the injection of a multitude of attacks, ranging from portscans and DDoS attacks to network exploitation and botnet traffic. In addition, ID2T offers an API for the further development of attacks.

4.2 Traffic Export and Injection

The message trace export functionality of BSF provides an accurate record of all messages exchanged during the botnet overlay simulation, including timestamp, message types, source node ID and destination node ID. We leverage this information in the implementation of the *P2PBotnet* attack in ID2T. In its current implementation, the P2PBotnet attack can only inject UDP based botnet packets. However, ID2T itself provides functionality and many other attacks using TCP based packets. At a high level, ID2T translates messages obtained from the BSF message trace into UDP packets. These packets are then *injected* into an existing PCAP. In order to do this, the P2PBotnet attack addresses three main tasks: 1) assigning IP addresses to BSF IDs 2) selecting which messages to inject, and 3) creating payloads for each message type. We will describe each of these three steps in the following subsections.

4.2.1 IP assignment

Before we describe the parameters and mechanisms for assigning IP addresses to the BSF message traces, we need to explain a differentiation of *local* and *external* bots within ID2T. A local bot is considered to be within the network for which a PCAP was recorded. That means, that all traffic originating and addressed towards that bot is visible within the PCAP. Contrary to that, an external bot is considered to be outside the scope of the network for which the PCAP was recorded. Therefore, only messages addressed to or originating from another bot within the local scope is visible in the PCAP.

The P2PBotnet attack provides four different parameters to specify how the BSF message trace should be injected into a given PCAP. The first parameter *src.bots.count* specifies how many local bots should be injected. ID2T will then select and map that number of BSF node IDs to real IP addresses. The other parameters related to IP assignment are *ip.reuse*, *ip.reuse.local* and *ip.reuse.external*, which specify the percentage of IPs to be reused from the original PCAP. As an example, specifying $ip.reuse.local = 0.5$ and $ip.reuse.external = 0.5$ would tell ID2T to assign 50% of both local and external bots to IP addresses already existing in the target PCAP and create new IP addresses for the remaining injected bots. This way, one can choose to add P2P traffic to already existing hosts in the PCAP or introduce new hosts with attack traffic only.

4.2.2 Message Selection

While we previously described how BSF node IDs are mapped to real IP addresses, we still have to specify which IDs and messages to inject. This is influenced by two parameters *interval.selection.strategy* and the *attack.duration*. The attack duration specifies the time period for which the botnet traffic should be injected into the PCAP. Additional parameters can also be used to specify a start and end time relative to the start time of the given PCAP. For the selection strategy, ID2T of-

fers either *random* or *optimal*. While random simply picks from the set of existing node IDs randomly, optimal tries to choose nodes with the highest amount of traffic. However, selecting the optimal parameter comes at significantly increased computation times, to identify the node IDs with the highest traffic amount.

Once a node ID is selected to be mapped to a local bot, its entire communication ranging the specified duration, is injected into the PCAP. For either node ID selection strategy, the period of highest traffic matching the specified duration is chosen for injection.

4.2.3 Payload creation

For payload creation, the P2PBotnet attack currently supports only random payload content representing encrypted traffic. Therefore, the main parameter is the length of the payload. A user can specify this using three parameters: *msg.map packet.padding* and *random.padding*. The *msg.map* defines a given length in bytes for each message type present in the BSF trace. Additionally, one can specify a padding length, that is added to each injected payload. Lastly, this padding can be defined to be random or static, i.e., a random value between 0 and the padding length is chosen, or the padding length is applied as is.

In Section 5 we will showcase, the described functionality and how BSF and ID2T can be used together, to recreate artificial traffic flows for a Sality botnet [2].

5 Evaluation

The aim of this section is to provide an overview of the capabilities and use cases of BSF. While example use cases for BSF itself can be found in previous publications [11, 12], this section provides a more generic evaluation of BSFs runtime performance. Furthermore, we showcase how BSF and ID2T can be used in conjunction to mimic Sality botnet P2P communication.

5.1 Runtime Performance

In order to measure the runtime performance of BSF, we are interested in how long it takes to simulate a botnet of a given size for a given simulation period. In order to do this, we compare the simulation progress against the runtime measured in hours. As BSF is a discrete event simulator, simulation time progresses as fast as the events are processed. For BSF, the number of events is heavily influenced by the amount of messages exchanged between bots. Therefore, any parameters influencing the amount of messages sent, has a direct correlation to the runtime of a simulation. Moreover, the specified time span of the simulation and the number of bots affect the runtime of a simulation. Depending on these factors, the runtime needed to simulate a botnet for 10 days can vary greatly. Within the following subsections, we will describe three different simulation scenarios and evaluate their runtime performances.

5.1.1 Experimental Setup

To evaluate the runtime performance of BSF, we used the SimpleBot implementation with three different sets of parameters, mimicking the unstructured P2P botnets: Sality, Gameover Zeus and Hide'n Seek. As mentioned previously, the runtime performance is mostly affected by the number of bots, duration and the amount of messages exchanged between bots. For the SimpleBot implementation, the number of messages is most affected by the NL-size and the MM-interval. The larger the NL-size of each bot, the more messages are exchanged during each MM cycle, increasing the amount of messages within the simulation. Similarly, the shorter the MM-interval of a bot, the more often it will exchange messages with all bots in its NL, again increasing the amount of messages in the simulation.

For our experimental setup, we picked three botnets with differing NL-size and MM-interval. Each botnet simulation is carried out for a duration of ten days and repeated eight times using different seeds for each run. To showcase that BSF is able to simulate large networks, we adjusted the total and active population in relation to the expected simulation load caused by the NL-size and MM-interval parameters. Table 2 provides an overview of the parameters used for each botnet.

All simulations were conducted on a server with an Intel E5-2640 v2 processor with 16 physical cores at 2.0 GHz and 64GB of DDR3 RAM at 1600MT/s.

5.1.2 Results and Discussion

The results of our evaluation are depicted in Figure 3. Due to the large NL-size of Sality, the simulation had an average runtime of 13.614 hours, surpassing the runtime of the other two botnets by at least 500%. For Gameover Zeus and HnS, the simulations have similar runtimes of 2.352 and 2.333 hours respectively. Nevertheless, we have to point out, that the number of bots simulated for Gameover Zeus is ten times higher than that of Hide'n Seek. This increase in nodes at similar runtimes is possible due to Gameover Zeus producing much less messages with its low MM interval and small NL size.

Mimicked Botnet	Total Bots	Active Bots	Duration	NL Size	MM Interval	Runs	Runtime	Std.
Gameover Zeus	20000	5000	10d	50	30m	8	2.352h	0.201h
Sality	5000	1250	10d	1000	40m	8	13.614h	0.507h
Hide'n Seek	2000	500	10d	116	32s	8	2.333h	0.158h

Table 2: Simulation parameters and runtime performance results.

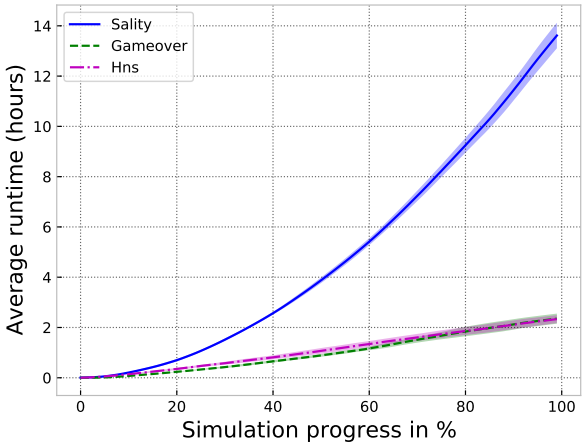

Figure 3: Runtime comparison of different parameter settings within BSF

Another important observation, is that in the early stages of the simulation, progress is much faster until it grows linearly in the later stages of the simulation. The main factors influencing this is the way bots join the network within BSF. To facilitate a natural formation of the overlay network, bots join the network one after another following the churn patterns. Therefore, only few bots are active in the beginning of the simulation, generating less messages or events to be processed.

We hope, that these examples provided a good overview of the size of simulations and their runtimes within BSF. We want to point out once more, that given sufficient computation power and time, any size of botnet may be simulated.

5.2 Data Injection

Within this section, we want to showcase the suitability of injecting simulated communication traces into a PCAP file. For that, we try to reproduce the conversation statistics of a real world Sality communication trace provided by Sebastián García as part of the Malware Capture Facility Project[4].

5.2.1 Experimental Setup

For this experiment, we used a simulation configuration mimicking the Sality botnet settings for SimpleBot and exported the communication trace of the last five hours of the ten day simulation. In a second step, we used ID2T to inject the communication of ten randomly selected bots into a PCAP file using the option *inject.empty=True* to remove all packets previously present in the PCAP. Due to the random selection of bots and the churn behavior, some injected bots only have limited communication for the data export period. Therefore, in a second step, we analyzed the PCAP and only exported all packets of the bot with the most messages exchanged. The resulting PCAP file contains traffic of one simulated bot consisting of $7,841$ packets over the timespan of $10,092.904$ seconds and is referred to as *Mimicked Sality*.

To compare the simulated traffic against a real-world trace of the Sality botnet, we use the PCAP recording of a Sality malware from the Malware Capture Facility Project. From this we extracted a $10,000$ seconds slice of traffic to compare against our simulated trace. We then used the Sality dissector module for Wireshark[5] to extract only the P2P communication of Sality. We refer to this PCAP as *Real Sality* from here onwards. To compare the mimicked and real Sality traces, we exported the Wireshark conversation statistics for each PCAP and compared the average over all conversations.

5.2.2 Results and Discussion

The results of the comparison between the mimicked and real Sality PCAPs, are presented in Table 3. Except for the number of conversations, all values are averaged across all conversations of a respective PCAP. The first noteworthy result is, that the real Sality has only 621 conversations compared to 1059 for the mimicked Sality. As Sality has a NL-size of up to 1000 [2] we expected the real trace to have an amount of conversations close to that number. Based on our investigations of the sample used by the Malware Capture Facility Project, we found out, that the bootstrap peers provided by the malware are only 740. Therefore, we assume that only a limited number of peers was active, and the malware was not able to quickly fill up its NL to the maximum value. Another observation is that the average bytes and packets exchanged between two peers is higher for the real trace compared to our mim-

[4] García, Sebastián. Malware Capture Facility Project. Retrieved from https://stratosphereips.org
[5] https://github.com/tillmannw/sality-dissector

icked trace. We speculate that this is caused by the low number of NL-entries for the real trace, leading to additional NL-request messages being sent to fill up the neighborlist. Normalizing the number of bytes by dividing with the number of packets, we get similar values for both the real and the mimicked traces, supporting our previous assumption.

	Real Sality	Mimicked Sality
Conversations	621	1059
Bytes	668.29	543.94
Packets	8.768	7.404
Bytes/Packets	76.218	73.464
Duration	8443.383	9067.351
Packets A -> B	3.676	2.888
Packets B -> A	5.092	4.517
Bytes/Packets A -> B	75.035	73.91
Bytes/Packets B -> A	77.072	73.179

Table 3: Comparison of average conversation statistics of a real and mimicked Sality botnet's PCAP.

Overall, we can summarize that it is possible to mimic the general characteristics of a P2P botnet such as Sality using BSF and ID2T. However, the lack of the ability to decrypt the payload may limit deep packet inspection related applications on the artifical trace. Nevertheless, we think that this tool can be useful to evaluate P2P botnet detection mechanisms or general IDS against a wide variety of simulated P2P botnet behaviors.

6 Conclusion

BSF attempts to bridge a gap in P2P botnet research, by enabling researchers and other defenders to evaluate monitoring mechanisms, takedown strategies and general P2P botnet behavior in a simulation setting. Among the major benefits, research can be conducted without interfering with other ongoing operations, novel ideas can be evaluated against future botnet designs, and ground truth is available to verify against any analysis or experiments. Moreover, the ability to recreate real-world churn, provides a more realistic environment, than traditional approaches to simulate churn. Lastly, the ability to export the simulation data for processing in other tools makes BSF flexible for P2P botnet research. Specifically, the interoperability with ID2T [4] to inject P2P botnet traffic into existing network traffic even facilitates testing of IDSs and botnet detection approaches.

For future work on BSF, an interesting addition would be integrating support for structured P2P botnets and other non-P2P C2 channels. Furthermore, improving the existing base classes and introducing additional monitoring nodes are planned in the near future. Lastly, adding a more realistic underlay network to the simulation may facilitate more accurate message traces and recreation of payloads for injection with ID2T.

Acknowledgements

This research work has been funded by the Royal Bank of Canada within the project Novel P2P Botnet Detection under Grant Agreement 2761478.4 and the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

Author details

Leon Böck

Telecooperation Group
Technische Universität Darmstadt, Germany
boeck@tk.tu-darmstadt.de

Shankar Karuppayah

Telecooperation Group
Technische Universität Darmstadt, Germany
and
National Advanced IPv6 Centre
Universiti Sains Malaysia, Malaysia
kshankar@usm.my

Max Mühlhäuser

Telecooperation Group
Technische Universität Darmstadt, Germany
max@tk.tu-darmstadt.de

Emmanouil Vasilomanolakis

Cyber Security Network
Aalborg University, Copenhagen, Denmark
emv@es.aau.dk

References

[1] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of gameover zeus," in *2013 8th International Conference on Malicious and Unwanted Software:" The Americas"(MALWARE)*, pp. 116–123, IEEE, 2013.

[2] N. Falliere, "Sality: Story of a peer-to-peer viral network," *Rapport technique, Symantec Corporation*, vol. 32, 2011.

[3] B. Botezatu, "New hide 'n seek iot botnet using custom-built peer-to-peer communication spotted in the wild." https://labs.bitdefender.com/2018/01/new-hide-n-seek-iot-botnet-using-custom-built-peer-to-peer-communication-spotted-in-the-wild/, 2018.

[4] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser, "Id2t: A diy dataset creation toolkit for intrusion detection systems," in *2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 739–740, IEEE, 2015.

[5] S. Karuppayah, *Advanced Monitoring in P2P Botnets: A Dual Perspective*. Singapore: Springer Singapore, springerbr ed., 2018.

[6] C. Rossow, D. Andriesse, T. Werner, B. Stone-gross, D. Plohmann, C. J. Dietrich, H. Bos, and D. Secureworks, "P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets," in *IEEE Symposium on Security & Privacy*, 2013.

[7] B. Kang, E. Chan-Tin, and C. Lee, "Towards complete node enumeration in a peer-to-peer botnet," *Proceedings of International Symposium on Information, Computer, and Communications Security (ASIACCS)*, 2009.

[8] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1st ed., 2010.

[9] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*, pp. 15–34. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[10] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 60, ICST (Institute for Computer Sciences, Social-Informatics and . . . , 2008.

[11] L. Böck, E. Vasilomanolakis, M. Mühlhäuser, and S. Karuppayah, "Next generation p2p botnets: Monitoring under adverse conditions," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 511–531, Springer, 2018.

[12] L. Böck, S. Karuppayah, K. Fong, M. Mühlhäuser, and E. Vasilomanolakis, "Poster: Challenges of accurately measuring churn in p2p botnets," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, (New York, NY, USA), p. 2661–2663, Association for Computing Machinery, 2019.

[13] W. Weibull, "Wide applicability," *Journal of applied mechanics*, 1951.

[14] S. Karuppayah, *Advanced monitoring in P2P botnets*. PhD thesis, Technische Universität Darmstadt, 2016.

[15] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference* (G. Varoquaux, T. Vaught, and J. Millman, eds.), (Pasadena, CA USA), pp. 11 – 15, 2008.

[16] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[18] R Core Team, *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017.

[19] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Third international AAAI conference on weblogs and social media*, 2009.

[20] C. G. Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, "On generating network traffic datasets with synthetic attacks for intrusion detection," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 2, 2020.