Development of *broctld* for *Bro*

Chesleah Kribs Kira Lessin Robert Minter
https://github.com/KiraJ42/broctld

ABSTRACT

Intrusion detection systems are a large part of everyday life, from a small bell that rings over a shop door to alert an owner to the arrival of customers to a network monitoring system that analyzes each and every data packet traveling across a company's network for any intention of a malicious actor with a harmful intent. In this paper, we first present our research into generalized intrusion detection systems and the complexities they uncover for the facilitation of our understanding into the network monitoring system of Bro and our comprehensive discussion on the facets of this monitoring system. We then present a brief description of the Bro network monitor that includes the original goals of the creator, architecture, and current modern-day uses. It is followed by our groups implementation of *broctld*, a subset of Bro which eventually shall replace the existing system, *broctl*. Though our *broctld* implementation presents a simple basic structure of the service, upon completion, it will easily be able to integrate with the existing Bro system.

1 INTRODUCTION

Intrusion detection systems are widely used throughout the world to alert for malicious data found in network traffic. Network administrators are having an increasingly difficult time protecting their infrastructure from malicious intrusions. An intrusion is described as a set of actions that attempt to compromise the confidentiality, integrity, or availability (CIA) of a resource by exploiting the vulnerabilities of a system or resource [20]. Open-source IDS projects such as Snort, Suricata, and Bro share common signatures/rules that provide the flexibility needed to defend against emerging threats. These systems have become a staple in ensuring safety of systems and networks by using the signatures above to inspect incoming packets. By monitoring the system or network, an anomaly detection or misuse detection attempts to find the abnormal behavior using statistical methods. Misuse detection, which is also described as signature-based detection, uses a description of the known attacks to form the policies. The IDS contains modules that capture network packets, modules to decode each packet to classify a portion of the packet, and a module to detect whether the packet is benign or malicious based on the rules of the system. Therefore examining with the signatures to identify malicious traffic [18]. The main goals of these intrusion detection systems would be to provide good performance–without packet loss, and minimizing the false positives and false negatives.

In the sections that follow we present the research needed to understand the Bro network monitor system: networks and its security which includes the need for a overlapping system that sits on it, otherwise known as an intrusion detection system. The research we embarked on allowed us to understand each particular facet into the requirements to the *broctld* project, a subgroup of the Bro IDS. The basic structure for the *broctld* project is specified in the projects and ideas section of the Bro repository. The main goals of the this project include:

- *broctld* starts up immediately on start up.
- *broctld* monitors processes spawned by it and restarts them if necessary.
- restarting the main *broctld* process should not restart the rest of the bro processes - they should keep running and once the main *broctld* process comes back online it should regain control of the rest of the bro processes.

2 INTRUSION DETECTION SYSTEMS

Intrusion detection is the process of observing the activities in a network and then analyzing them for any attempts to compromise the CIA (confidentiality, integrity, availability) triad. The IDS sits on the side of a network and monitors the traffic at the entrance points and provides visibility into the security of the network.

These can identify things such as security policy violations, infections on the system, information leakage, and configuration errors.

In comparison, an intrusion prevention system (IPS) monitors the network traffic by watching traffic patterns and individual packets and has the ability to take immediate action, based on a set of rules established by a network administrator, whereas a true IDS does not have this option. A similar mechanism would be a firewall, that sits between a network and controls the traffic coming in.

An IDS can analyze packets from the network or analyze data generated from a host, Network-based IDS or Host-based IDS. A network-based IDS scans the network traffic to identify hostile activity. It sits on the entry and exit points of the network to monitor what is going in and out. It looks for the attack signatures in the network traffic via an promiscuous interface that receives all packets on the same network. The advantages of these NIDS is that there is an availability for packet analysis, evidence collection, real-time detection and response, and malicious intent classification. These can be deployed passively without requiring massive modifications to the system or the network. A host-based IDS monitors specific files, logs and registry settings on a single PC. They raise an alert when any access, modification, detection and copying of the monitored object occurs. The main objective of a HIDS is to flag any tampering of a specific PC and can automatically replace altered files when changed to ensure data integrity. The gist of these systems is to parse the system logs and watch the users logins and processes. These are widely available to customize since there are many workstations that can run with a HIDS. While a host-based IDS is better than a NIDS for detecting activities for a particular host, they have a limited view of the entire network and cannot detect attacks that are targeted for a host in a network that does not have HIDS installed.

After gathering the data the IDS analysis kicks in and can have different methods of practice: Misuse Detection, Anomaly detection, or Stateful Protocol Analysis. Misuse detection uses a large database of known attacks and matches them with the events occurring. It looks at the signature of the attack and thus misuse detection can also be classified as signature-based detection or knowledge-based detection. These signature-based IDS are time-consuming as a new signature must be made for every attack. As a result, novel attacks are not detected. But, the signatures are easy to develop and understand and the pattern matching can be performed quickly. Anomaly-based detection centers on a baseline for network behavior which can be learned by the network or specified by a network administrator. Any events that are outside the normal network noise is assumed to be an intrusion. The basis of the anomaly detection focuses on this clear fact: it must be able to determine normal from malicious. Anomaly based detection has an advantage of being able to recognize new attacks without signatures; however, if an attack falls within the normal usage it is very hard to distinguish malicious activity, and it is very difficult to define the rules. Bhosale and Mane also describe stateful protocol analysis as a system that defines a set of constraints for a correctly behaving program or protocol[3]. It monitors the operations of the program or protocol against the constraints, and thus, helps to detect unknown attacks with a minimal false positive rate than anomaly detection.

3 BRO

Bro is described by its' developers as a UNIX-based network monitoring framework, developed in 1998 by Vern Paxson from the Network Research Group at Lawrence Berkeley National Lab (LBNL) and the International Computer Science Institute (ICSI). It was named Bro as an Orwellian reference to the assumption that big Brother is always watching you, whereas Bro network monitor is always monitoring the system. A large number of physical sites deploy Bro for the protection of their cyber-infrastructure: Universities, research labs, super-computing centers, open-science communities and major corporations. It specifically targets high-speed, high-volume networks in the range of 10GE to 100GE.

Bro can be used to build a network intrusion detection system, collecting network measurements, conducting forensic investigations, traffic base-lining, and more. [11] It has been compared to the combination of tcpdump, Snort, netflow, python, wireshare, and the Perl language all in one program. It is originally designed to protect the IDS from attacks itself. It is a protocol analyzer, which will accept network events, and it can turn network events into useful metadata to find and

take action against potential threats. Without customizing Bro, you are able to do massive deployment, analysis and interfacing. Bro has default scripts that mainly perform misuse-detection and logs activity comprehensively without assessment. Since the Bro IDS has a main emphasis on network security it conducts network traffic searches and determines if the anomalies detected are normal while inspecting. On the lowest most layer which processes the most amount of data, the work is performed to a minimum, as we go higher up through the layers the data stream diminishes. It is made up of modules such as libpcap, Event Engine, and a Policy Script Interpreter (see below image). The libpcap uses this for the packet capture library to capture packets within a network. This is used by tcpdump and isolates Bro from the details of the network link, and aids Bro to different UNIX variants. If the host operating system provides a powerful kernel packet filter, then the libpcap downloads the filter used to reduce the traffic to the kernel. It strains insignificant traffic coming from network interfaces at the network layer. These filtered packets are then passed onto the next layer deemed the event engine which performs integrity checks to ensure the packets are normal by verifying the IP headers checksum. IP fragments are put together and thus it is able to analyze entire IP packets. (cite is the comparative study and analysis of network detection tools). Finally the Bro policy script interpreter executes event handlers.

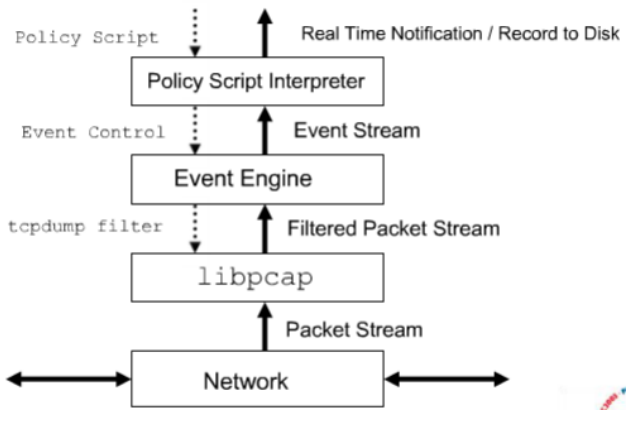

Figure 1: The Bro architecture [9]

Bro also comes with its own language which users can utilize to program the policy scripts. Policy scripts allow network administrators to fine tune the Bro installation to specifically search types and patterns of traffic, and define those as malicious. This interpreter checks whether the processing generated any events on a FIFO queue, and processes every event until empty. For each event passed to the interpreter, it retrieves compiled code for the corresponding handler and binds values of the events to the arguments of the handler and interprets the code. It can then do what is stated below, by generating new events and log notifications. Furthermore, developers can expand the capabilities of bro by having scrips execute in certain events to block, alert, or log information about certain network traffic. It only reports to log files and does not have a GUI to help users. By capturing log files for the network traffic, it makes it easier to read logs for different conditions. Some logs that bro creates when running include:

- dns.log, which contains DNS activities,
- conn.log, which contains TCP/UDP/ICMP connections,
- ssl.log, which contains SSL/TLS handshake information,
- http.log, which contains HTTP requests and replies,
- weird.log, which contains unexpected network-level activity.

To achieve the functionality that was desired, the developers of Bro were guided by a few goals. The first having the functionality of high speed and large volume monitoring. The greatest sources of threats for an environment are the external hosts connecting to servers over the internet. These networks need a single link connecting it to the remainder of the DMZ, and this can be monitored by passively by watching the DMZ link. The next is a no packet filter drop. If an application using a packet filter cannot consume packets as quickly as they arrive on the link, then the filter will buffer the packets for a later consumption, and if the filter runs out of the buffer the packets that arrive after will drop. Drops can completely alter the monitoring, since these missing packets could identify a network intruder that is deemed malicious by the monitor. Before Bro, there was a lengthy delay before an attack could be detected, so the developers wanted to create real time notification for Bro. If an attack is detected quickly then it is much easier to trace back to a certain attacker to minimize and prevent damage and to record the attackers network activity. Another requirement

was that the system must be designed in order to make it easy to add to it the knowledge of different new types of attacks. Finally, Bro wanted to avoid simple mistakes by defining the security policy that is as clear and error-free as humanly possible.

4 *BROCTLD*

As a subset of Bro, BroControl is a interactive interface tool for operating Bro installations. It has two modes of operation, a standalone mode for managing a traditional single system Bro setup; or a cluster mode for setting up and maintaining a multi-system setup for load-balancing across multiple independent machines. If a *broctld* command is specified directly on the command line, *broctld* performs the action associated with that command immediately. The files modified include broctl.cfg, node.cfg, and networks.cfg before running *broctld*.

On the same fabric, our group had the opportunity to create a similar single persistent system service titled *broctld*. It functions as an interactive shell-like interface, which allows the user to direct commands to specific nodes. It is started and stopped through the operating systems mechanism for managing scripts like the init.d, systemd, launchctl, for managing system services. When *broctld* is running, it manages all the processes running on the system and offers an API to the *broctl* client that allows the users to control a setup and user functionality similar to the above setup of*broctl*. *Broctld* is run primarily in cluster mode, where the master node controls all the other nodes through connections to the *broctld* instances. Within the master node is a main process which the user interfaces with to interact with the system. There is also a supervisor node which is responsible for maintaining all the bro processes, and for persisting when the main process restarts. Whereas the *broctl* node connects only to the master *broctld* which passes the commands on as necessary. The ability to restart the main *broctld* process without shutting down all the process allows for configuration changes, upgrading, and error recovery all without causing monitoring outages.

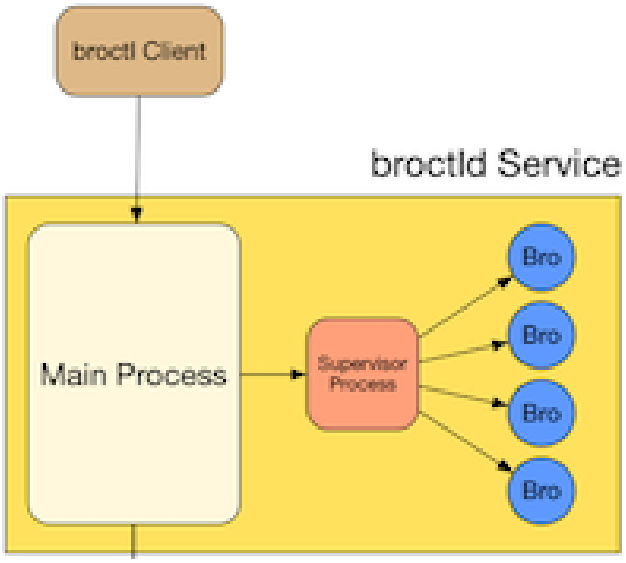

Figure 2: The proposed *broctld* architecture [13]

When the *broctld* service starts up, the main process receives the control first; however, no supervisory process will be running at this time. The main process then spawns the supervisor process and sets the communication between the two. The supervisory process will not yet control any bro processes. Then the supervisor will start the bro processes that it will monitor.

5 DEVELOPMENT

Our development of *broctld* began with the examination of the tentative schedule provided on the Bro website which laid out, as follows, their suggested steps for the implementation of *broctld*:

Build out single-node deployment

(1) Break up the current*broctl* into a client/server application. The API will probably largely mimic the current set of *broctl* commands.
(2) Remove all the logic that copies Python code around. Assume *broctl*/*broctld* is installed system-wide through standard OS installation mechanisms.
(3) Factor out the code that controls Bro processes so that there is a clear API for their management in terms of simple primitives (start/stop/etc.).
(4) Introduce the Supervisor Process. Remove the process management code from *broctld* and replace it with communication to the Supervisor Process.

Build out the cluster deployment

(1) Set up inter-node communication between broctld services on different systems
(2) Design the API for the master to control slave node, including pushing out configuration and receiving status information.[13]

Our goal was to follow the listed schedule as closely as possible in order to stay true to the suggested design of *broctld*. To that effect we wanted to understand how Bro worked and, specifically, how it was implemented so that our design could slide seamlessly into the already existing architecture. Unfortunately that proved more difficult than expected, resulting in several unusable virtual machines on which we attempted to install Bro. Although we learned a considerable amount both from our research, as described previously, and our attempts to produce a stable, running Bro product, it was decided, in the interest of time and in order to improve the possibility of producing a complete project, that we would attempt to implement the basic structure of the *broctld* system outside of the Bro environment. With that being the case, part of our design goal was to make it so that the structure could easily be implemented once it was put into an environment with bro in it.

Once we moved beyond trying to setup and environment to support Bro, it became clear that the basic idea of *broctld* was strikingly similar to a system almost entirely associated with malware, a command and control server. Command and control servers are most often used in the control of botnets for some malicious purpose, such as distributed denial of service attacks, stealing user data, sending spam email messages, or even taking control of devices remotely. The owner of the server can maintain control all of the infected devices in their botnet from a single location, but should the server ever be taken down the bots would continue to run, completely unaffected. This clear cut comparison gave us some insights for the development of *broctld*.

The Bro project consists of a combination of code written in both Python and C++. Since *broctld* is intended to take the place of the current *broctl* system and *broctl* is written in Python, we decided to implement *broctld* in Python as well. The initial step was to build a simple shell-like environment which was similar to standalone *broctl* and would provide the user with an interface to create, destroy, and control all instances of Bro running on their network. This shell program began as the main process for our implementation, however over time, as we got a clearer idea of what the main process needed to do the user interactive shell program was split away from the main process entirely. Now, the main process starts up both the shell process, for user interaction, and the supervisor process (when necessary), for node interaction. The supervisor process is currently capable of spawning bro processes, gathering data from the nodes running those processes (and reporting that data back to the main process when the user requests it) and connecting to nodes which may have been running before the supervisor process had started. By using the Python networking module and working in a client-server structure, we were able to create a system where a failure of the server process (the main program) does not effect the client process (the supervisor program). Additionally, by using the Python multiprocessing library we gave the supervisor process the ability to spawn, connect to, and control the nodes running Bro.

6 FUTURE IMPROVEMENTS

The major upcoming development for our implementation of *broctld* is the integration of the project into the existing framework of Bro. Before the final integration can begin, however, there are some additional improvements we intend to make. Currently, upon restart of the main process, the user must manually instruct the program to reconnect with the supervisor process, and by extension the running instances of Bro, but this process should be automatic. Additionally we shall implement the master and slave node system for *broctld* as described on the *broctld* project page and as depicted in figure 3 which can be found on the following page. With this set-up a master node will be able to copy itself, and control the copies in such a way that a vast network of Bro instances can be maintained and governed from that single master node.

7 CONCLUSION

We intended to implement the *broctld* service, as presented on the Bro development web-page. This already large project was made even more monumental by a lack of experience working with networks and intrusion detection systems, Bro specifically, as well as the lack of clarity which was present in the code of Bro. After several failed attempts to implement a functioning Bro environment, we decided to create

the structure of *broctld* outside of Bro with the goal that it could be transferable to a system with Bro on it. *broctld* is made up of nodes controlled by a supervisor connected to a main process where the main process can be ended, either intentionally or otherwise, without affecting the supervisor process or any processes running on the nodes and can regain contact to the supervisor process upon restarting. To that end, this project was a success and once future improvements have been completed as well as the additional functionality necessary to interact flawlessly with existing Bro system, our implementation will provide all the desired function of *broctld*.

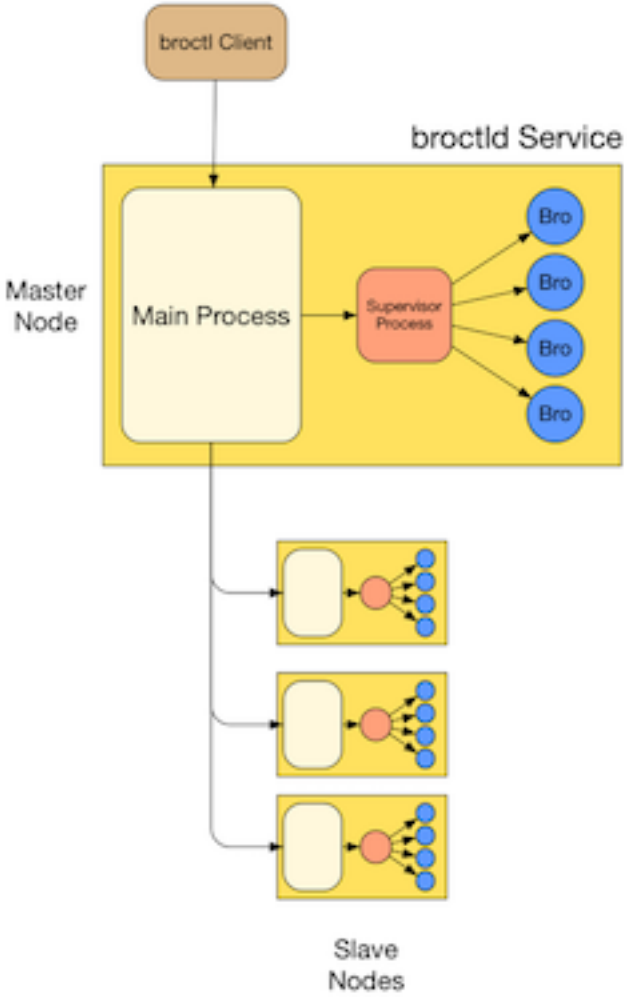

Figure 3: Master-slave node architecture for *broctld*[13]

REFERENCES

[1] Mikhail Belkin and Partha Niyogi. Using manifold stucture for partially labeled classification. In *Advances in neural information processing systems*, pages 953–960, 2003.

[2] Pierre Bérard, Gérard Besson, and Sylvain Gallot. Embedding riemannian manifolds by their heat kernel. *Geometric & Functional Analysis GAFA*, 4(4):373–398, 1994.

[3] Dhanashri Ashok Bhosale and Vanita Manikrao Mane. Comparative study and analysis of network intrusion detection tools. In *Applied and Theoretical Computing and Communication Technology (iCATccT), 2015 International Conference on*, pages 312–315. IEEE, 2015.

[4] Daniel B. Cid. Host intrusion detection for everyone, June 2008. [Online Document] Available URL http://www.ossec.net./.

[5] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[6] Ronald R Coifman, Stephane Lafon, Ann B Lee, Mauro Maggioni, Boaz Nadler, Frederick Warner, and Steven W Zucker. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the national academy of sciences*, 102(21):7426–7431, 2005.

[7] Abdulla Dakhgan, Ali Hadi, Jaafer Al Saraireh, and Doaa Alrababah. Passive dns analysis using bro-ids. In *New Trends in Computing Sciences (ICTCS), 2017 International Conference on*, pages 121–126. IEEE, 2017.

[8] Joseph M Dalton, Ali Ahmad, and Kay Stanney. Command and control resource performance predictor (c 2 rp 2). In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 321–322. ACM, 2004.

[9] Stephen Lau. The bro instrusion detection, Nov 2003. PowerPoint Presentation given at Lawrence Berkley NAtional Laboratory.

[10] Matthew Monaco, Alex Tsankov, and Eric Keller. Taking the surprise out of changes to a bro setup. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 49–52. ACM, 2016.

[11] Ryan Nolette. Threat hunting with bro, Jan 2018. [Online Document] Available URL https://sqrrl.com/threat-hunting-bro/.

[12] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23-24):2435–2463, 1999.

[13] Veron Paxson. The bro network monitor, 1999. [Online Document] Availabe URL https://www.bro.org/.

[14] InfoSec Resources. Ids: An introduction to snort., June 2017. [Online Document] Available URL http://www.garydewey.com/ids-an-introduction-to-snort/.

[15] InfoSec Resources. Basic snort rules syntax and usage., May 2018. [Online Document] Available URL http://www.resources.infosecinstitute.com/snort-rules-workshop-part-one/gref.

[16] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[17] Anshu Sharma and Monika Sharma. Analysis and implementation of bro ids using signature script. In *Soft Computing Techniques and Implementations (ICSCTI), 2015 International*

Conference on, pages 57–60. IEEE, 2015.

[18] Kittikhun Thongkanchorn, Sudsanguan Ngamsuriyaroj, and Vasaka Visoottiviseth. Evaluation studies of three intrusion detection systems under various attacks and rule sets. In *TENCON 2013-2013 IEEE Region 10 Conference (31194)*, pages 1–4. IEEE, 2013.

[19] Robert Udd, Mikael Asplund, Simin Nadjm-Tehrani, Mehrdad Kazemtabrizi, and Mathias Ekstedt. Exploiting bro for intrusion detection in a scada system. In *Proceedings of the 2nd ACM International Workshop on Cyber-Physical System Security*, pages 44–51. ACM, 2016.

[20] Patrick Wheeler and Errin Fulp. A taxonomy of parallel techniques for intrusion detection. In *Proceedings of the 45th annual southeast regional conference*, pages 278–282. ACM, 2007.