Taking the Surprise out of Changes to a Bro Setup

Matthew Monaco, Alex Tsankov, Eric Keller
University of Colorado, Boulder
{matthew.monaco, alexander.tsankov, eric.keller}@colorado.edu

ABSTRACT
With network functions virtualization, an organization gains an ability to provide a much more agile security infrastructure. In this paper we focus on vulnerabilities and challenges created by this new flexibility itself. In particular, using Bro as a case study, we present i) a framework for testing Bro scripts using a packet traces, ii) a complementary framework for testing the performance impact of Bro scripts, iii) a continuous integration system for triggering automatic testing in response to code changes. With this system, security administrators are protected against logic errors in new and modified scripts as well as performance degradation.

1. INTRODUCTION
Cybersecurity threats have shown no signs of decreasing. As such, network administrators have an increasingly difficult task of protecting their infrastructure, with intrusion detection systems (IDS) playing a central role e.g., Bro [9], Snort [10]. In moving from physical appliances to software-based virtual network functions, we gain an ability to elastically scale and flexibly to deploy new network functionality. This flexibility can be a great asset in defending against emerging threats, but also poses a significant unresolved challenge – the development and deployment model. First, elasticity is not infinite. Organizations will typically deploy Bro (or in the future, more generally, a variety of network functions) as a cluster in their network. They will have a set number of servers which determines the limit of processing requirements such that traffic must be discarded or processing requirements such that traffic must be discarded or handled by an additional server. Likewise, will this new script, or modification, increase processing requirements such that traffic must be discarded or new hardware deployed? These questions are often answered simply by deploying and seeing what happens.

In this paper we argue that more formal management of network functions configuration and setup are essential. We use Bro as a case study to present our preliminary work in this space. As a step toward providing administrators with knowledge of how modifications to their network functions software and configuration will impact functionality and performance, before it is deployed in a production environment, we present a new, test-driven infrastructure with Bro as a case study. This framework provides (i) the ability to define and perform ad hoc testing (BroUnit §3), (ii) the ability to track the performance impact of configuration changes across a variety of traffic patterns (BroFiler §4), (iii) a continuous integration system for Bro which is integrated with GitLab §5, a git management platform that can, in response to code changes, oversee the execution of BroUnit and BroFiler, as well as perform some bookkeeping for later analysis and visualization (BroCI §6), and (iv) ultimately, we envision it serving as a central component to foster a community §6).

2. RELATED WORK
The Bro project provides a “simple driver for basic unit tests,” called btest [2]. Btest is a generic tool – not specific to, but appropriate for Bro – which adds syntactic sugar to shell scripts for evaluating the results of commands. Additionally, btest offers a baseline functionality for storing the results of an initial test and using them for comparisons on future tests. Btest is an imperative approach to functionality similar to the checks sub-component of BroUnit §3, however BroUnit checks are defined declaratively which simplifies things for administrators.

Pcapr [4] is a social networking site built around packet captures. Through pcapr, users can view, publish, and edit packet captures for educational purposes. In Section 6 we discuss taking a site like pcapr further with additional features for viewing, publishing, and editing Bro scripts paired with packet captures.

Automatic testing methodologies have been developed for IDSs which generate synthetic traffic by reversing traffic sig-
were altered or dropped [1]. Which resulted in Bro workers crashing until their scripts.

From version 2.2 to 2.3 there were syntax changes, additions, and removals to its overall Bro configuration. A test case, an organization can be sure that future changes, no matter how subtle, can be integrated with BroUnit as a traffic source.

3. BROUNIT

BroUnit is our testing framework for Bro centered around testing individual scripts as well as a collection of scripts. A key insight of this paper is that Bro scripts should be tightly married to test traffic. This contribution is embodied in BroUnit and its test case definitions.

As illustrated in Figure 1, the test cases instruct BroUnit how to configure and run test (Bro) workers, and when they are ready, what traffic to pass through them. The test cases also instruct BroUnit how to evaluate the resulting Bro log files to determine which tests passed and which tests failed.

In the remainder of this section we will elaborate on the two main testing mechanisms as well as describe the current state of our implementation.

3.1 Specific Traffic Patterns

This insight that scripts should be coupled to test traffic evolved from our desire to mitigate against a number of things that can go wrong while modifying Bro scripts or upgrading Bro itself. Formal and comprehensive testing is required to assure that malicious traffic does not go by unnoticed. By explicitly defining the purpose of a script through a test case, an organization can be sure that future changes, additions, and removals to its overall Bro configuration continue to honor the intent of administrators; a task which can be difficult as focus changes, personnel rolls over, etc.

A simple example is in the context of Bro version upgrades. From version 2.2 to 2.3 there were syntax changes which resulted in Bro workers crashing until their scripts were altered or dropped [1].

Figure 1: BroUnit configures and runs Bro against multiple traffic sources. It then analyzes Bro’s log files and determines the pass/fail status of each test case.

Whereas replaying a specific traffic pattern can be used to test whether a single condition is met (and therefore essential step in testing), we also need the ability to test the script(s) as a whole with a mix of traffic that is more representative of real traffic. In doing so, we can uncover any issues that arise under load, or that may arise from unspecified test cases (i.e., conditions for which there is not a specific traffic trace, or that invalidate assumptions made when constructing test cases [8]).

Further, organizations’ assumptions about the traffic that they are scanning must be tested as the statistical breakdown of packets evolves over time. For example, a cluster might contain a script with a high performance cost for analyzing FTP traffic. Such a script was originally acceptable because there was only one, rarely used FTP server, however, over time, a department increases its FTP footprint without necessarily notifying its network security team.

For such a scenario, BroUnit allows tests to be run against (i) the live traffic itself, (ii) synthetic traffic matching the characteristics of the live traffic, or even (iii) synthetic traffic with traffic mixes that can be used to test behavior under sudden traffic changes. (This also benefits performance profiling, which is discussed further in Section 4). Live traffic can be scanned from a real Ethernet interface to run tests against an organization’s live traffic. Traffic generators such as D-ITG can also be used to run test cases under load.

Here, the checks on the resulting logs will be statistical based (rather than exact matches) because we cannot be certain of the exact number of times a certain portion of a script will be triggered. We can, however, know roughly what we would expect for the chosen traffic mix (live or synthetic). These statistics can be minimum and maximum thresholds, a multi-modal threshold (e.g., in-attack we might expect one result and otherwise we might expect another result), or even relative to information from baseline scripts (which are assumed to be good, such as a script that simply tracks the number of connections).

A test failure, in this case may be due to a number of factors such as the traffic mix having changed since the script was created. It is then a trigger to the administrator to seek understanding behind the result and adjust the test or deployment accordingly (e.g., remove scripts targeted at traffic no longer seen).

3.2 Representative Traffic Mix

Whereas playing a specific traffic pattern can be used to test whether a single condition is met (and therefore essential step in testing), we also need the ability to test the script(s) as a whole with a mix of traffic that is more representative of real traffic. In doing so, we can uncover any issues that arise under load, or that may arise from unspecified test cases (i.e., conditions for which there is not a specific traffic trace, or that invalidate assumptions made when constructing test cases [8]).

Further, organizations’ assumptions about the traffic that they are scanning must be tested as the statistical breakdown of packets evolves over time. For example, a cluster might contain a script with a high performance cost for analyzing FTP traffic. Such a script was originally acceptable because there was only one, rarely used FTP server, however, over time, a department increases its FTP footprint without necessarily notifying its network security team.

For such a scenario, BroUnit allows tests to be run against (i) the live traffic itself, (ii) synthetic traffic matching the characteristics of the live traffic, or even (iii) synthetic traffic with traffic mixes that can be used to test behavior under sudden traffic changes. (This also benefits performance profiling, which is discussed further in Section 4). Live traffic can be scanned from a real Ethernet interface to run tests against an organization’s live traffic. Traffic generators such as D-ITG can also be used to run test cases under load.

Here, the checks on the resulting logs will be statistical based (rather than exact matches) because we cannot be certain of the exact number of times a certain portion of a script will be triggered. We can, however, know roughly what we would expect for the chosen traffic mix (live or synthetic). These statistics can be minimum and maximum thresholds, a multi-modal threshold (e.g., in-attack we might expect one result and otherwise we might expect another result), or even relative to information from baseline scripts (which are assumed to be good, such as a script that simply tracks the number of connections).

A test failure, in this case may be due to a number of factors such as the traffic mix having changed since the script was created. It is then a trigger to the administrator to seek understanding behind the result and adjust the test or deployment accordingly (e.g., remove scripts targeted at traffic no longer seen).

3.3 Implementation

BroUnit (like much of the tooling ancillary to Bro) is implemented in Python. At this time, each unit is an entire script. It is designed to be modular and extensible, while also being easy to use. As such, test cases are declarative but are themselves valid Python and stored in either brounit.py
or brounit/*.py, so could, for example, even call arbitrary code to setup each test.

The traffic source types are modular and can be easily extended by writing sub-classes of BroUnitSource. Currently, traffic sources are defined for packet captures, live traffic, D-ITG [5], and composites thereof. The test case checks are also modular and similarly to traffic sources are derived from the base class BroUnitCheck. Currently, the primary check is BroUnitRegex for performing regular expression matching within Bro log files.

4. BROFILER

Understanding the performance profile of Bro script library is just as important as being confident of its ability to correctly alert on specific traffic patterns. There are three possible outcomes in situations where a Bro cluster can not keep up with the traffic it is tasked to analyze: i) packets can be discarded (the default behavior if Bro is incapable of storing the packets properly in its buffer), ii) more hardware can be thrown into the cluster, and/or iii) the active set of scripts can be modified or reduced.

This first option is undesirable because without prioritization, important traffic can go by unnoticed, and there is not always a clear path to prioritization in the first place without an expensive hardware frontend or specialized network interface cards. As a general rule of thumb each Bro worker can handle 100Mbit/s. Peering points of 10Gbit, 40Gbit, and even 100Gbit are common; therefore without any pre-filtering of packets a cluster might require 1,000 cores to guarantee zero packet drops. This itself is on the scale of small datacenter. Therefore the second option can quickly become cost-prohibitive. The third option, optimizing the Bro script configuration, is often the best as it both elegantly and deterministically manages the demand on a cluster. For the third option, with the changing demands on a network at different times, it’s possible for security administrators to leverage information provided by BroFiler to develop different, and dynamic security profiles – performing less analysis at peak hours or deeper packet-analysis with the same devices without dropped packets in off-hours.

4.1 Design and Implementation

BroFiler, like BroUnit (§3) is implemented and configured in python. It is used in conjunction with BroUnit to collect statistics. These include packet drop rate, CPU utilization, memory utilization, etc. Conceptually, BroFiler is quite simple. It currently knows two sets of configuration options, defined in profile.py. Absolute limits can be defined which cause an error to be raised if the performance while running BroUnit is not acceptable. Relative thresholds can also be defined which trigger an error if performance impact of the current commit is too pronounced (it is the job of BroCI (§5) to pass along the previous commit’s results).

4.2 Analysis with BroFiler

BroFiler enables two levels of analysis through graphs on a web frontend. First, time-based plotting of individual runs provide insight into how specific traffic patterns impact performance of a Bro cluster. This is tightly coupled to the set of scripts which are configured. For example a performance spike can indicate a particularly troublesome pattern

![Figure 2: BroFiler results obtained by running the same test case against various script configurations. Higher lines (more packets processed), are better.](image)

needs to be worked around, or that it’s simply not worth attempting to detect it.

As an example evaluation that we performed, figure 2 shows the number of packets processed over time from the same test case for multiple script configurations. It also includes the total number of packets sent. The test traffic consists of four packet captures, replayed in parallel, for 100 iterations each. The data shown in Figure 2 indicates that the custom scripts used in this test do not actually have a large performance impact. Rather, Bro’s default configuration affects the drop rate significantly for this test. These results would indicate to a new installation that defaults should be evaluated and thinned out based on what’s important to the local site.

A second type of analysis, historical-based graphs, allow administrators to track performance from commit to commit in the source repository. Knowing the relative performance impact of each commit is an important tool when a Bro cluster begins to drop too many packets and it is necessary to go back and evaluate if any scripts should be dropped from the cluster. BroFiler gives administrators the ability to balance the value of targets in their network vs. the cost of protecting them; for example there might be a 5% performance impact from doing deep analysis on FTP traffic, but the organizations servers are not high-value targets because they are read-only archives of public files.

As future work (§8) we would like to provide finer-grained detail with BroFiler. Currently script-by-script and change-by-change statistics are available, the latter of which can potentially be used to infer profiling information from within a script itself. Explicit, per-event statistics are also desirable.

5. BROCI

Continuous integration testing is an increasingly common practice in modern software development. It is a powerful concept that automatically gives developers near-instantaneous feedback in response to code changes. The overhead of setting up a CI system is negligible as most free and public git hosting services such as GitLab and Bitbucket provide out-of-the-box support. Further, CI systems open the door for structured and automated paths from development to production when combined with sufficient unit test cases (§3).

Our BroCI implementation is written in Python with a few lines of Bash for interfacing with GitLab. As illustrated
De Facto

industry grows and more formal requirements take shape. A Bro cluster. This is especially true as Bro’s footprint in
improvement to the workflow surrounding the management of
performance thresholds can be evaluated (§)
BroFiler with the previous run’s test results so that relative
Filer and then storing results in a database. It provides
subsequent actions from GitLab such as automated merging.

BroCI itself is in charge of running both BroUnit and Bro-
Filer and then storing results in a database. It provides
BroFiler with the previous run’s test results so that relative
performance thresholds can be evaluated (§4).

6. TOWARDS A BETTER COMMUNITY

The components previously described are a significant im-
provement to the workflow surrounding the management of
a Bro cluster. This is especially true as Bro’s footprint in
industry grows and more formal requirements take shape.

6.1 A De Facto Repository

We believe that these tools build towards a more cohesive
de facto community as well. A major pain point we have
identified both for novice users as well as established instal-
lations is the acquisition of an appropriate set of scripts with-
out a standardized, and centralized source. For new users,

it can be an overwhelming process to pull together, with
confidence, an effective and performant configuration. For
established sites, which are ostensibly higher value targets,
keeping pace with new attack patterns is a high priority.

BroCI can serve as a community where organizations can
publish and download scripts and, importantly, test cases
and packet captures to accompany them. Having multiple
alternatives of attacks is important as traffic can be con-
structed in such a way to deliberately evade IDSs [6]. Rat-
ing systems, alerts to new attacks, discussion, and improve-
ments to others’ scripts can all take place in a centralized,
trustworthy location. This, we feel, will lower the barrier to
entry of Bro as well as increase its effectiveness as an IDS.

6.2 BroCI as a Service

Beyond a community for the sharing and discussion of
scripts, packet captures, and test cases, BroCI can easily
be launched as a service. Most obviously, BroCI as a Ser-
vice lowers the barrier to entry for organizations wishing to
transition to formalized testing. Some may wish to use the
service indefinitely, while others as a means to gain experi-
ence before deploying a private testbed.

Conversely, organizations can donate workers to the pub-
lic testbed. This is how the public GitLab-CI suite operates.
The service itself just performs bookkeeping and coordina-
tion but does not actually provide compute resources for
servicing testing. Users must associate their own runners,
and can choose for them to be private or public.

Finally, BroCI as a Service offers the ability for dynamic
feedback of script, packet capture, and test case changes.
For example, a user forks a script with an ostensibly more
efficient implementation to detect malicious behavior in a
pcap file. Rather than waiting for users to give feedback
about whether or not the fork works, BroCI can test auto-
matically, for both correctness and performance.

7. ACKNOWLEDGMENTS

This research was supported in part by the National Sci-
ence Foundation award number 1406192.

8. CONCLUSIONS AND FUTURE WORK

With the great flexibility afforded by NFV and SDN to
deploy new security functionality comes the challenge of en-
suring that what is being deployed does not make the secur-
ity infrastructure less secure. In this paper, we presented
an initial step toward a solution where administrators can
understand the impact changes have on both traffic han-
dling as well as performance. Together, BroUnit, BroFiler,
and BroCI fill a need in the management workflow of Bro as
well as in the community at large. As future work, we hope
to continue the development with richer set of capabilities,
generalize the platform, and ultimately, develop enough out-
side interest in these tools to foster a community where users
can browse, comment on, and change a library of scripts and
packet captures, as well as provide BroCI as a service.

9. REFERENCES

sphinx-git/install/release-notes.html#bro-2-3.
[2] Btest - a simple driver for basic unit tests.
https://www.bro.org/sphinx/components/btest/
README.html.
 generation of realistic network workload for emerging
 networking scenarios. Computer Networks,
 intrusion detection: Evasion, traffic normalization, and
 end-to-end protocol semantics. In USENIX
 developing an ids stimulator for the black-box testing
 of network intrusion detection systems. In Computer
 modeling as a basis for security requirements. In Symposium
 on requirements engineering for information security (SREIS),
 2005.
 intruders in real-time. Computer networks,
detection for networks. In LISA, volume 99, pages