ARL-TR-8175 ● SEP 2017

US Army Research Laboratory

Generating Artificial Snort Alerts and Implementing SELK: The Snort–Elasticsearch–Logstash–Kibana Stack

by Daniel E Krych, Joshua Edwards, and Tracy Braun

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8175 ● SEP 2017

US Army Research Laboratory

Generating Artificial Snort Alerts and Implementing SELK: The Snort–Elasticsearch–Logstash–Kibana Stack

by Daniel E Krych, Joshua Edwards, and Tracy Braun
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
September 2017	Technical Report	6/1/2016–8/12/2016

4. TITLE AND SUBTITLE
Generating Artificial Snort Alerts and Implementing SELK: The Snort–Elasticsearch–Logstash–Kibana Stack

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Daniel E Krych, Joshua Edwards, and Tracy Braun

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-CIN-D
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8175

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report details the development of an artificial Snort alert generator and the configuration of a Snort–Elasticsearch–Logstash–Kibana (SELK) stack for parsing, storing, visualizing, and analyzing Snort alerts. The first section covers the Snort alert-generation program, the methodology involved in developing it, and how it accelerates Snort-related research. The second section covers the development of configuration files and the pipeline for the SELK stack, followed by its deployment and uses. We develop the program, gen_alerts.py, which takes in a Snort rules file and generates artificial Snort alerts with a specified priority distribution for outputting high, medium, low, and very low alerts based on Snort's classifications. We construct the ELK pipeline, using Logstash to parse and organize Snort alerts. These generated alerts head this pipeline to create the SELK stack. To enable rapid deployment, we implement this system in a lightweight Lubuntu virtual machine that can be imported and used with VirtualBox or VMware. In addition, we provide an instructional guide on system setup. The methodologies described can be translated to the setup and use of the ELK stack for storing and visualizing any data.

15. SUBJECT TERMS
Snort, Elastic, Elasticsearch, Kibana, Logstash, ELK, SELK, data visualization, IDS, IPS, networking, traffic analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Tracy Braun
Unclassified	Unclassified	Unclassified	UU	36	19b. TELEPHONE NUMBER (Include area code) (301) 394-4954

Contents

List of Figures iv

List of Tables iv

1. Introduction 1

2. Methodology 2

2.1. Snort Alert Generation 2

2.2 The SELK Stack 8

3. Discussion and Conclusion 11

4. References 13

Appendix A. Using the Snort Alert Generator <gen_alerts.py> 15

Appendix B. Leveraging the Snort–Elasticsearch–Logstash–Kibana (SELK) Stack Using the Prebuilt, Lightweight Lubuntu Virtual Machine 17

Appendix C. Implementing Snort–Elasticsearch–Logstash–Kibana (SELK) in Your Environment 19

List of Symbols, Abbreviations, and Acronyms 28

Distribution List 29

List of Figures

Fig. 1 gen_alerts.py outputting 10 generated alerts 3
Fig. 2 gen_alerts.py usage outputted with "-help" flag..... 7
Fig. 3 The SELK stack pipeline 9
Fig. 4 "Default2" Dashboard we created and provide: dek_kibana_viz_setup_Aug-16.json 11

List of Tables

Table 1 Predefined priority distributions 3
Table 2 The header and options sections of a Snort rule used in the crafting of a Snort alert 5

1. Introduction

Snort is an open-source network intrusion prevention system (NIPS) and a network intrusion detection system (NIDS) developed by Sourcefire.[1,3] When running Snort in Intrusion-Detection mode, network traffic is monitored and a rules file (`.rules`) is used to set the traffic rules and generate alerts when one of these rules is satisfied. For example, a rule in the configuration file could tell Snort to generate an alert whenever it sees a transmission control protocol (TCP) connection established by a private network connecting to a public network. Users can also specify actions such as dropping the packet altogether. Snort rules can become very specific and target specific programs/malware by looking for known IP addresses, port numbers, byte values seen in packets, and so on. Snort uses a classification system with 38 classifications for alerts. These include things such as "trojan-activity", "shellcode-detect", "denial-of-service", "network-scan", and "misc-attack". Each classification has an associated priority level from 1 to 4, defined as high (1), medium (2), low (3), and very low (4). By default on a Linux system, alerts generated by Snort are stored in the file `/var/log/snort/alert`.[1] Alerts can then be examined by analysts to find patterns of misuse or indications of gradual attacks. This analysis can help network defenders stop some attacks before they succeed.

Elasticsearch, Logstash, and Kibana are free, open-source tools offered by the company Elastic. Elasticsearch is a search and analytics engine built on top of Apache Lucene, an information retrieval library, and enables efficient data storage and retrieval similar to a database. Logstash is a data collection and transportation platform that provides a way to ingest data from one source, filter and map the data as specified in a configuration file, and push them to another source (in this case, Elasticsearch). Kibana is a visualization platform, which reads Elasticsearch data and provides a graphical interface to query, analyze, transform, and monitor the data.[2] Each of these tools runs independently and communicates with its predecessor/successor in the pipeline, and together they form the ELK stack.

We have created the (Snort–Elasticsearch–Logstash–Kibana) SELK stack, which consists of the generating, parsing, storing, visualizing, and analyzing of Snort alerts with the ELK stack. An overview of the pipeline in a Linux system entails the following:

1) Snort IDS writes alerts as network traffic matches rules.

2) Logstash detects alerts being added to the alert file, ingests them, applies the rules specified in the configuration file to filter, and map values to variables, then pushes them to Elasticsearch in JSON format.

3) Elasticsearch stores the JSON-formatted Snort alerts.

4) Kibana connects to Elasticsearch and provides a graphical interface for viewing the data stored. Kibana can then generate graphs and visualizations displaying the information in useful ways. It also enables near-real-time monitoring of the data.

This work aims to promote and accelerate US Army Research Laboratory (ARL) research involving the use of Snort and the ELK stack for parsing, storing, analyzing, and visualizing Snort alerts or any data in several ways:

- Anyone looking for a better way to store and examine Snort alerts can use this system out of the box.
- End users can use the 2 parts to this system—the Snort alert generator and the ELK stack system—independently and gain insight into setting up a similar system with their own data by following the instructional guide in Appendix C, and altering the Logstash configuration file to parse their data specifically.
- The Snort Alert Generation program allows end users to generate realistic Snort alerts based on their rules files without having to run Snort. This also provides interesting test data for any system that ingests Snort alerts.
- The SELK stack demonstrates the capabilities of the open-source ELK stack suite, and outlines the methodology involved in filtering, storing, and visualizing the data. Anyone looking to expand upon their data collection, or looking to replace a less-robust storage and visualization system, will want to experiment with the ELK stack.

2. Methodology

2.1. Snort Alert Generation

The main idea is a program that takes in a Snort rules file and generates artificial alerts based on the rules found in the file provided. The name of this program is `gen_alerts.py`. End users can specify N, the number of artificial alerts to be generated. The program will then output “fast” formatted alerts. The fast format is

one of Snort's available formats, and it prints a quick one-line alert, whereas the "full" format prints alerts with full packet headers and is far more verbose.

Because Snort uses a classification-priority system to rank the severity of alerts, we also wanted to give the end user some freedom on specifying the priority distribution they desired. We defined 5 priority distributions from which end users can choose, which are shown in Table 1.

Table 1 Predefined priority distributions

Priority distribution	High alerts	Medium alerts	Low alerts	Very low alerts
1	5%	24%	69%	2%
2	10%	29%	59%	2%
3	15%	34%	49%	2%
4	20%	39%	39%	2%
5	33.33%	32.33%	32.33%	2%

By default, we set the priority distribution to "2," which generates 10% high, 29% medium, 59% low, and 2% very low alerts. Only 1 classification in Snort's default settings is specified as "very low," and this is for a "tcp-connection" seen. Therefore, we decide to limit the percentage of these to only 2%, because very little variability exists when generating alerts at this priority level. The final product of our program is seen in Fig. 1, which shows 10 alerts generated using the `community.rules` file Snort provides on their website. We use the "`-micros`" flag to output timestamps with microsecond precision, as Snort does by default (see Appendix A for more information).

```
$ ./gen_alerts.py -i community.rules -n 10 -micros
2016/09/26-10:50:09.902359 [**] [1:3138:9] "NETBIOS SMB-DS Trans2 QUERY_FILE_INFO andx attempt" [**] [Cla
ssification: protocol-command-decode] [Priority: 3] {tcp} 218.215.0.162:51403 -> 192.168.221.75:445
2016/09/26-11:06:27.718665 [**] [1:30779:3] "SERVER-OTHER OpenSSL TLSv1 large heartbeat response - possib
le ssl heartbleed attempt" [**] [Classification: attempted-recon] [Priority: 2] {tcp} 10.53.152.57:2484 -
> 230.129.27.236:60590
2016/09/26-11:06:55.967557 [**] [1:3136:9] "NETBIOS SMB Trans2 QUERY_FILE_INFO andx attempt" [**] [Classi
fication: protocol-command-decode] [Priority: 3] {tcp} 78.145.119.161:61930 -> 10.177.63.95:139
2016/09/26-11:10:41.720355 [**] [1:30516:8] "SERVER-OTHER OpenSSL TLSv1.1 large heartbeat response - poss
ible ssl heartbleed attempt" [**] [Classification: attempted-recon] [Priority: 2] {tcp} 10.18.52.227:21 -
> 173.152.31.138:52171
2016/09/26-11:40:50.576526 [**] [1:3142:9] "NETBIOS SMB-DS Trans2 FIND_FIRST2 andx attempt" [**] [Classif
ication: protocol-command-decode] [Priority: 3] {tcp} 224.125.60.254:49507 -> 192.168.26.57:445
2016/09/26-12:00:29.107917 [**] [1:30510:6] "SERVER-OTHER OpenSSL SSLv3 heartbeat read overrun attempt" [
**] [Classification: attempted-recon] [Priority: 2] {tcp} 6.3.92.248:63576 -> 10.19.1.12:25
2016/09/26-12:04:59.985186 [**] [1:3083:9] "MALWARE-BACKDOOR Y3KRAT 1.5 Connection confirmation" [**] [Cl
assification: misc-activity] [Priority: 3] {tcp} 10.82.11.141:5880 -> 232.202.26.219:55418
2016/09/26-12:18:19.164359 [**] [1:32852:2] "MALWARE-CNC Win.Trojan.Poolfiend variant outbound connection
" [**] [Classification: trojan-activity] [Priority: 1] {tcp} 192.168.129.156:49369 -> 9.184.109.35:8000
2016/09/26-12:20:24.454228 [**] [1:3138:9] "NETBIOS SMB-DS Trans2 QUERY_FILE_INFO andx attempt" [**] [Cla
ssification: protocol-command-decode] [Priority: 3] {tcp} 175.138.59.94:63869 -> 192.168.154.207:445
2016/09/26-12:37:37.078379 [**] [1:26558:3] "BLACKLIST User-Agent known Malicious user agent Brutus AET"
[**] [Classification: misc-activity] [Priority: 3] {tcp} 117.182.157.237:61145 -> 192.168.130.147:7144
```

Fig. 1 gen_alerts.py outputting 10 generated alerts

To better understand Snort alerts, we ran Snort in IDS mode and observed how Snort rules files were written and processed by Snort to create alerts. For this system, we used an Ubuntu 12.04 virtual machine (VM). After performing some tests in our environment, one being an ICMP rule that we could trigger by sending outbound pings, we found that metadata in the rule are used to craft the "fast" formatted alerts. The Snort alert is crafted from both the network traffic and metadata from the rule.

In the following passage, we provide an example of a rule seen in the Snort rules file `community.rules`[1], the associated alert generated by `gen_alerts.py`, and the fields used in crafting the alert:

RULE:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 53
(msg:"PROTOCOL-DNS dns zone transfer via TCP
detected"; flow:to_server,established; content:"|00 01
00 00 00 00 00|"; depth:8; offset:6;
byte_test:1,!&,0xF8,4; content:"|00 00 FC 00 01|";
fast_pattern; isdataat:!1,relative; metadata:ruleset
community, service dns; reference:cve,1999-0532;
reference:nessus,10595; classtype:attempted-recon;
sid:255; rev:23;)
```

ALERT:

```
2016/06/17-05:34:08.382267 [**] [1:255:23] "PROTOCOL-
DNS dns zone transfer via TCP detected" [**]
[Classification: attempted-recon] [Priority: 2] {tcp}
83.8.50.58:54630 -> 172.24.136.71:53
```

ALERT FIELDS:

```
timestamp [**] [gid:sid:rev] "msg" [**]
[Classification: classtype] [Priority: priority]
{protocol} sip:sport -> dip:dport
```

The program is written in Python for its rapid development and data parsing capabilities. To take in command-line arguments from the end user, we leverage the argparse module. First, we create the "`-i`" flag to read in a Snort rules file.

Next, we parse the rules file line-by-line, then split the line to differentiate the header section from the rest, which we dub the "options" section. The fields within

these sections are shown in Table 2. Many more fields are provided in the rule in the options section, such as the flow, content, and reference fields, but these are not used to generate the fast formatted alert.

Table 2 The header and options sections of a Snort rule used in the crafting of a Snort alert

Header	Options
protocol	msg
source IP	classtype
source port	priority
destination IP	gid
destination port	sid
	rev

Within the header section of the rule, we see the client/server IPs and ports, which can be specified in multiple ways. Both the IPs and ports can be set in the Snort rules file or configuration file using a list variable such as "HOME_NET" and "SSH_PORTS". They can also be provided as exact values, such as "146.186.33.5" and "335". Lists can also be provided within the fields, making the following possible: "[1.1.1.1, 2.2.2.2, 3.3.3.3, 4.4.4.4]" and "[111, 222, 333, 444]". In addition, shorthand enumeration can be used such as with classless inter-domain routing (CIDR)-addressed IPs, "1.1.1.0/24", and port ranges, "326:335". Furthermore, these can be used in combination provided within lists, and "!" can be added to represent the Boolean "NOT". All of these cases are handled in the parsing of the fields in `gen_alerts.py`. Since our program only takes in the Snort rules file, we assume that any variable declarations will be set there instead of a configuration file.

In terms of fields used in the alert, the options section of the rule usually provides the msg (message), classtype (classification), sid (signature ID), and rev (revision ID). According to the Snort manual,[3] the gid is the Generator ID, the sid is the Snort/Signature ID, and the rev is the revision ID. The gid provides the component of Snort that generated the alert and defaults to "1" since it is not commonly set. The sid is used as a signature ID, and the rev is the number of revisions that have been made to the rule. For more specific information, please see the 3.4.3 gid section of the Snort manual.[3] By default, Snort derives the priority from the classification field, but this can be overwritten by providing the priority field in the options section or by defining it in the Snort rules file or configuration file. For example:

```
'config classification: network-scan,Detection of a
Network Scan,1'.
```

By default, "network-scan" is priority 3 (i.e., low priority). Here, the user set network scan to 1, meaning high priority, to reflect the importance of this classification in their system.

By default, we preset all of the variable lists to Snort's default variable definitions. Users can define Server lists and Port lists using the keywords "`var`" , "`ipvar`", or "`portvar`", where "`var`" is universal. For example, "`var HOME_NET 192.168.1.0/24`" and "`var HTTP_PORTS 80,8080,8000`", which could also be defined with "`ipvar HOME_NET 192.168.1.0/24`" and "`portvar HTTP_PORTS 80,8080,8000`". Since Snort does not define the `HOME_NET`, if it is not set in the rules or configuration file, we define it as any private IP address (which in turn means `EXTERNAL_NET` is anything but private IPs). Snort also has over 50 ports listed in their `HTTP_PORTS` list, including but not limited to "80", "81", "311", "8000", "8008", "8080", and "8088".[3] While parsing the `.rules` file in `gen_alerts.py`, we also check for any user-defined changes to the default lists, and we override the preset lists.

In order to generate N realistic alerts when given a `.rules` file with only a small fraction of N rules, we find and enumerate mutable fields and introduce randomness. Generated alerts would be unrealistic if they repeated connection information and timestamps, thus this variability and randomness are necessary. In addition, to enforce the generation of realistic alerts, we cannot just randomly grab pieces from several alerts to generate new ones because this could result in illogical alerts. For example, a generated alert with "udp" for its protocol field combined with a message field clearly matching a TCP rule would not make sense. Thus, we needed to determine which fields must be grouped together to form a logical, realistic alert.

The combination of the gid/sid/rev fields uniquely describes the rule and is linked to the message, classification, priority, protocol, IPs, and ports fields. Unlike the previous fields, the timestamp is not linked to other fields and can be randomized to the extent desired by the end user. By default, we use a 2-h span with an end time of now and start time of 2 h before now. Since we are precise to the microsecond, we have already greatly reduced the chance of repeating alerts, even if we only provided a few rules. The sip, sport, dip, and dport fields can also be randomized within the boundaries of the rule definition. Fields can be variables, such as "HTTP_PORTS" or "AIM_SERVERS", defined as lists of possible values. This allows us to choose a random value from the list for the field. At other times, actual lists are provided in the rule, or CIDR-addressed IP addresses, or port ranges. All of these resolve to a list of IPs and ports to choose from randomly, and we still generate a realistic alert by adhering to the rule's original IP/port definitions.

```
usage: gen_alerts.py [-h] -i I -n N [-pd PD] [-s S] [-e E] [-v] [-debug]
                     [-uncomment] [-o O] [-noyear] [-micros]

Provided with a Snort .rules file, the desired number of alerts,
and optionally: the starting and ending timestamps, the priority
distribution (2 by default), output file location, verbosity
flag, debug flag, noyear flag, and uncomment flag, we generate N
fake alerts by enumerating through variable fields in real-world
alerts (using the rules provided), forcing a specified
distribution of alerts based on Snort's defined priority
levels (1-4) -- where 1 is the highest

optional arguments:
  -h, --help  show this help message and exit
  -i I        The name of the Snort .rules file
  -n N        The number of alerts to be generated
              *Note: Due to the priority distribution and generating alerts
              for each priority, the actual number of alerts generated can
              vary some from the desired number
  -pd PD       The priority distribution (1 - 5) allows you to choose from a
               preset list of distributions.  Snort classifies alert priority
               on a 4-point scale (1-4) where 1 is the highest.  Below, choose
               from the 5 distributions which determines the percentage of the
               N alerts generated which would be classified in Snort as high,
               medium, low, or very low.  By default #2 is set, so if the
               number of alerts to be generated N = 100,
               10 alerts will be high, 29 medium, 59 low, 2 very low.

               PRIORITY DISTRIBUTIONS:
               (The higher the number, the more balanced the low-high
               priorities are)
                   * = DEFAULT
               (1) 5% high, 24% medium, 69% low, 2% vlow
              *(2) 10% high, 29% medium, 59% low, 2% vlow
               (3) 15% high, 34% medium, 49% low, 2% vlow
               (4) 20% high, 39% medium, 39% low, 2% vlow
               (5) 33.33% high, 32.33% medium, 32.33% low, 2% vlow
  -s S        The starting year/month/day or just month/day for the range of
              alerts generated in one of the following formats:
                  (1) yyyy/mm/dd
                  (2) mm/dd
              *leading zeros not required
  -e E        The ending year/month/day or just month/day for the range of
              alerts generated in one of the following formats:
                  (1) yyyy/mm/dd
                  (2) mm/dd
              *leading zeros not required
  -v          Verbosity
  -debug      Provide debug info
  -uncomment  Parse commented rules as well
  -o O        Append generated alerts to specified file
  -noyear     Don't include the year in generated alerts
  -micros     Use microseconds instead of milliseconds in timestamps
              **WARNING**: DO NOT USE this flag if you plan to store
              alerts in Elasticsearch and use the ELK stack.
              Elasticsearch only stores timestamps up
              to millisecond precision.  Using this flag will
              cause Elasticsearch and ELK stack issues.
```

Fig. 2 gen_alerts.py usage outputted with “-help” flag

Our program first parses the entire Snort rules file provided, creating/updating any values defined that overwrite Snort’s defaults, and stores the nonmutable fields of each parsed rule together as one item in a dictionary indexed by the priority level. It then loops a number of times for each priority, based on N, the number of alerts to be generated, and the chosen priority distribution that specifies the percentage of N that is high, medium, and so on. Next, it grabs a random rule at that priority level and generates an alert filled with the rule’s metadata and random values that meet the rule’s parameters. For example, we may see “HTTP_PORTS” or “any” provided as the source port, so we resolve this to the list of ports it entails and select one of these ports at random. Each alert is stored as it is crafted, and after N alerts have been created, it sorts them by timestamp and outputs them to stdout or a

specified output file. Several helper-functions are used to handle repeated tasks throughout the parsing and random parameter generation sections, including "Parse_var", "Gen_IP", and "Gen_Port".

After implementing the aforementioned parsing techniques and developing functions to handle repeated tasks, we focused more on the usability and the flags available when calling the program. These are detailed in Fig. 2 and can be outputted using "`python gen_alerts.py –help`", assuming Python is installed and the user is in the current directory. See Appendix A for more information on using `gen_alerts.py`.

2.2 The SELK Stack

Having a program that can generate Snort alerts will provide great test data for anyone working with Snort, or a system that ingests Snort alerts, but at this point we only have raw text. We wanted a way to investigate and monitor real Snort alerts. We decided to use Elasticsearch as the back-end to store the Snort alerts, and Kibana as the front-end to visualize them. Elasticsearch stores data in an efficient way and enables fast retrieval through querying by creating "nodes" and "indices," and splitting up the data intelligently. Kibana provides an easy-to-use graphical interface for performing these queries, and creating graphs and charts to view and monitor the data as they are stored. Adding in Logstash, the additional tool that ingests, filters, and forwards data to Elasticsearch, as well as the gen_alerts.py program for generating Snort alerts, we now form the SELK stack and create a full system for generating, filtering, storing, visualizing, and monitoring our data.[2] This pipeline can be seen in Fig. 3 and begins with either Snort firing rules and generating alerts or `gen_alerts.py` generating fake alerts, and it ends with Kibana visualizations and monitoring of these alerts. This system will mostly benefit end users who are hoping to store, visualize, and investigate their Snort alerts in depth, but it can also make Snort data testing much more involved. In addition, the SELK stack and the methodology used to create it can be seen as a great example of how to set up a similar system with any data of your choosing.

Going into this part of the project, we developed a system in which end users could find trends in their Snort data. Perhaps a specific type of malware or malware in general was on the rise, or maybe they would be able to classify patterns of attacks and use these to decrease the attack surface on their front and thwart future attacks.

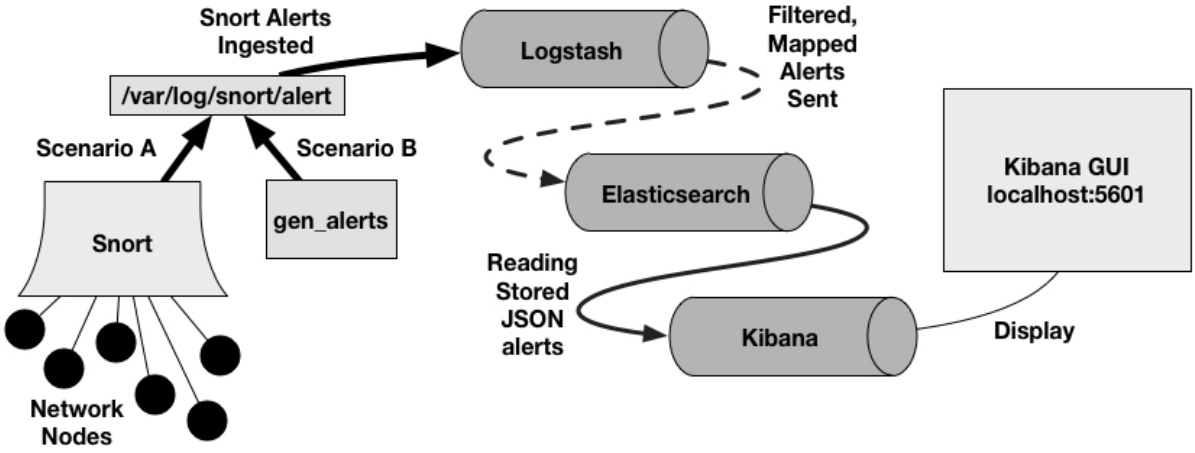

Fig. 3 The SELK stack pipeline

As discussed earlier, Elasticsearch, Logstash, and Kibana are all tools provided by the company Elastic, and thus work well together out of the box. Getting this stack up and running with their default settings is not too difficult, but the setup gets more complex when trying to customize settings such as altering Elasticsearch's storage methods or Kibana's custom index patterns. Logstash required the most work since we needed to instruct the system on how to parse and filter our Snort alerts. One tutorial we found useful for learning about the ELK stack and its configuration was by DigitalOcean.[4]

The Logstash configuration file (`snort-alert.conf` in our case) is used to instruct Logstash on where it is ingesting data from, how it is filtering/parsing data, and where to send them to. We ingest data from the file `/var/log/snort/alert` since it is the default location for Snort to store alerts. We use Grok patterns to filter Snort alerts and map/typecast fields in the alert to variables we will store them as in Elasticsearch. Grok filtering is the best method for Logstash to parse arbitrary text and structure it so that it is easily queryable. This requires writing Grok patterns to match the exact log format and providing variable names to the data parsed and typecast information to overwrite the default storage of the field as a string. Logstash provides approximately 120 Grok patterns, so there is a good chance the logs' patterns may have already been established, but in our case we needed to develop our own. For more specific information on writing custom Grok patterns, see the Grok guide provided by Elastic.[5] To test custom Grok patterns and simplify debugging, we recommend using an online interpreter such as the "Grok Debugger".[6]

Once our Grok patterns were created and we instructed Logstash to output our data to Elasticsearch, as well as stdout to view the stream of logs as they are processed,

we were able to get the SELK pipeline up and running. As shown in Fig. 3, the SELK stack was built to work alongside Snort and can ingest data directly from Snort's alert output, or, in our case, it can be tested with `gen_alerts.py` and ingest its outputted alerts. The alerts are ingested, filtered/mapped, and outputted by Logstash to Elasticsearch. Elasticsearch stores and indexes the data, and then Kibana reads the data in Elasticsearch and displays them in a user-friendly graphical interface. In Kibana, the "Discover" tab can be used to query data and display them by fields in the alert or in raw JSON. Visualizations of the data can be created, including pie charts, bar graphs, tables, Geo-coordinate maps, and more. To create Geo-coordinate maps, GeoIP databases can be leveraged, which enable the translation of IP addresses to coordinates. We found a DigitalOcean tutorial to be helpful in enabling this feature.[7] After creating visualizations, one can use Kibana's "Dashboard" feature to view several of them at once. This provides a great way to quickly view data and monitor them.

An example of several Kibana visualizations and a dashboard we created with Snort alert data can be seen in Fig. 4. This dashboard is saved as "Default2" and can be imported into Kibana through our settings file `dek_kibana_viz_setup_Aug-16.json` by following instructions in Appendix C. In the upper left corner, we have a pie chart embedded in another pie chart, where the inner chart shows the Snort alert priority levels (1–4) seen and the outer chart shows the classifications seen. To the right of this, we see a line graph displaying the log timestamps over time to view trends in the amount of alerts seen, and to the right of that we see a world map showing malware traffic. In this case, we are mapping coordinates based on the locations of the source IPs seen within alerts containing malware keywords. On the bottom left, we have a table displaying the top 10 destination ports seen in alerts, along with the protocol and classification seen in alerts with those ports. Moving to the right again, we can see the countries of destination IPs binned by month. Finally, in the bottom right corner, we see another set of embedded pie charts, but now they are showing the protocols seen with the classifications.

This enables us to quickly see trends in data, such as the fact that most alerts with priority 1 (high) are related to a "default-login-attempt," or that a few days after "2016-06-11," the number of Snort alerts we saw jumped up drastically (Fig. 4). We also note that the United States accounts for most of the malware traffic alerts seen in our dataset, specifically from the Kansas/Montana area.

Fig. 4 "Default2" Dashboard we created and provide: dek_kibana_viz_setup_Aug-16.json

3. Discussion and Conclusion

Throughout these projects, we faced multiple challenges such as 1) the constant discovery of edge cases and customizations in Snort `.rules` files to handle when parsing Snort rules and 2) the ELK stack causing our system to run out of file descriptors and errors saying the Java language was out of memory/heap space. When first setting up the ELK stack, it appeared that our Logstash Grok patterns were working when they actually were not. This resulted in a lot of confusion and the inability to create Kibana index patterns indexed by the log timestamp. By default, Kibana index patterns are indexed on the "@timestamp" field, which denotes the time the log was actually stored in Elasticsearch. For a system running Snort in real time and immediately storing alerts in Elasticsearch, this would not make a noticeable difference, but when a system only submits logs on a set schedule, or when older logs are input along with newer ones, this results in false data.

These limitations led us to storing the logs in Elasticsearch based on the log timestamp, but this led to even more issues because by default Logstash creates an index for each day. This resulted in hundreds of Elasticsearch indices for only a few hundred random logs over the past year and caused the system to hang, Kibana to time out, and more Elasticsearch errors to occur. The solution to all of this was to index on the log timestamp but continue to store the logs in Elasticsearch based

on the day they were submitted. Now, instead of hundreds of indices per day, we only have one, resulting in a stable, easy-to-use system.

In order to promote and accelerate Army research involving the use of Snort and the ELK stack for parsing, storing, analyzing, and visualizing Snort alerts or any data, we have provided a lightweight Lubuntu Virtual Machine in `.ova` format. This VM has the SELK stack setup and is ready to go along with our `gen_alerts.py` program. This `.ova` file can be imported into either VirtualBox or VMware with ease, and more details can be found in Appendix B. In addition, in Appendix C, we have provided a guide for setting up the SELK stack in one's own environment. The methodology described can also be used to set up an ELK stack with any data.

The SELK system provides end users with a system for parsing, storing, analyzing, and visualizing Snort alerts, but the methodology can be applied to any data. The benefits of these projects and a system such as this are clear. We can now generate artificial Snort alerts without ever running Snort. This benefits any research involving Snort data, and it allows for detailed, realistic datasets. In addition, we can now do so much more with our Snort alert data and easily analyze them as a whole or by specific days, months, years, and so forth. If Snort is set to run in real time, we can monitor our Snort alert data live and more easily find attacks, analyze them, and stop them before they escalate.

4. References

1. Snort. [accessed 2017 May 3]. https://www.snort.org/.
2. The Open Source Elastic Stack. [accessed 2017 May 22]. https://www.elastic.co/products.
3. SNORT Users Manual 2.9. Martin Roesch; c1998–2003. Chris Green; c2001–2003. Sourcefire, Inc.; c2003–2013. Cisco and/or its affiliates. All rights reserved; c2014–2016. [accessed 2017 May 3]. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/.
4. Anicas M. How to install ELK: Elasticsearch, Logstash, and Kibana. [accessed 2017 May 23]. https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14-04.
5. Grok. [accessed 2017 May 22]. https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html.
6. Grok Debugger. [accessed 2017 May 23]. http://grokdebug.herokuapp.com/.
7. Anicas M. How to map user location with GeoIP ELK. [accessed 2017 May 24]. https://www.digitalocean.com/community/tutorials/how-to-map-user-location-with-geoip-and-elk-elasticsearch-logstash-and-kibana.

INTENTIONALLY LEFT BLANK.

Appendix A. Using the Snort Alert Generator <gen_alerts.py>

When planning to use `gen_alerts.py` to generate Snort alerts and use them with another system (not the Snort–Elasticsearch–Logstash–Kibana [SELK] System which is used to store/visualize/monitor alerts), then use the following instructions.

Running the Program:

```
python gen_alerts.py –i X.rules –n Y -micros
```

Where **X** is the name of the rules file and **Y** is the number of alerts to be generated.

Note: By default, this program outputs Snort alerts with timestamps with millisecond precision, whereas Snort, itself, uses timestamps with microsecond precision. Use the "`-micros`" flag to produce timestamps in microseconds in order to have the most realistic, accurate Snort alerts. This design is by choice since Elasticsearch only stores timestamps with a maximum of millisecond precision. Because of this, issues arise when using microseconds in timestamps and having Logstash attempt to parse them and store them in Elasticsearch. In Kibana, the timestamps are used as a way of organizing the alerts by the time the timestamps say they were created, instead of the time they were stored in Elasticsearch. Therefore, when planning to use the SELK stack, *do not use this flag*, and instead understand that the best precision one can achieve at present is milliseconds.

For more information on running `gen_alerts.py` and to see the list of available command line options, use the following command or see Fig. 2:

```
python gen_alerts.py –help
```

For an example run of `gen_alerts.py`, see Fig. 1.

Appendix B. Leveraging the Snort–Elasticsearch–Logstash–Kibana (SELK) Stack Using the Prebuilt, Lightweight Lubuntu Virtual Machine

1. Download and install VirtualBox or VMware (VirtualBox is suggested).

   https://www.virtualbox.org/wiki/Downloads

2. Download the OVA file (`.ova`). Available from several locations. Contact the authors for details.

3. Import the OVA file (`.ova`) into the virtual machine environment.

   a. VirtualBox import instructions can be found here:

      https://www.virtualbox.org/manual/ch01.html#ovf

   b. VMware import instructions can be found here:

      https://pubs.vmware.com/fusion7/index.jsp?topic=%2Fcom.vmware.fusion.help.doc%2FGUID-275EF202-CF74-43BF-A9E9-351488E16030.html

4. Run the virtual machine and jump to Appendix C section "Start up the ELK stack in different tabs/screens/terminals".

5. If planning to use this environment for a more permanent setup, increase the hard disk size to allow for the storage of more data.

   a. VirtualBox hard-drive increase

      ```
      VBoxManage modifyhd vdi_file_full_path_and_name
      --resize <size in MB>
      ```

      Example to resize to 40 GB:

      ```
      VBoxManage modifyhd vdi_file_full_path_and_name
      --resize 40000
      ```

   b. VMware hard-drive increase:

      https://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1004047

Appendix C. Implementing Snort–Elasticsearch–Logstash–Kibana (SELK) in Your Environment

The following principles can be applied to setup ELK with any data.

Download the ELK stack:

Grab the appropriate files for your OS, and download directly or use `wget`

Then unzip files (and un-tar if on Linux)

<https://www.elastic.co/downloads/elasticsearch>

<https://www.elastic.co/downloads/logstash>

<https://www.elastic.co/downloads/kibana>

Install Java JRE

```
sudo apt-get install openjdk-8-jre
```

```
java -version
```
(Test to see Java version & ensure installation)

Grab dependencies and setup environment for gen_alerts.py program

```
sudo apt-get install python-ipy
```

```
sudo mkdir /var/log/snort
```

```
sudo touch /var/log/snort/alert
```

Move Logstash config file into the 'logstash-x.x.x/' folder

```
mv snort-alert.conf logstash-2.3.3/
```
(Logstash version 2.3.3)

GeoIP setup

```
mkdir /etc/logstash
```

```
cd /etc/logstash
```

```
sudo curl –O
http://geolite.maxmind.com/download/geoip/database/Geo
LiteCity.dat.gz
```

```
sudo gunzip GeoLiteCity.dat.gz
```

Start up the ELK stack in different tabs/screens/terminals

```
elasticsearch-2.3.3/bin/elasticsearch

kibana-4.5.3-linux-x64/bin/kibana

logstash-2.3.3/bin/logstash -f logstash-2.3.3/snort-alerts.conf
```

*Note the '`-f`' flag is used to provide the .conf file

*By default it pulls alerts from `/var/log/snort/alert` so if alerts are in this file that you do not want to be stored, consider changing the paths. See the "Changes to Make Based on File Locations, etc." section for details.

***All three will output information to stdout as they are running.**

Look for some lines explaining the status of the servers and if they are communicating.

Logstash will just say:

```
Settings: Default pipeline workers: 1
Pipeline main started
```

Test the setup

Check on Elasticsearch server

```
curl localhost:9200/
```

This should return something along the lines of:

```
{
      "name" : "Tom Cassidy",
      "cluster_name" : "elasticsearch",
      "version" : {
            "number" : "2.3.3",
            "build_hash"                          :
            "218bdf10790eef486ff2c41a3df5cfa32dadc
            fde","build_timestamp"    :    "2016-
            0517T15:40:04Z",
            "build_snapshot" : false,
            "lucene_version" : "5.5.0"
      },
      "tagline" : "You Know, for Search"
}
```

Check on Kibana server

Browse to 'localhost:5601' in your browser of choice.
You should see the Kibana main page, but you will not be able to see any data yet since we have not added any.

***If any issues occur, it is most likely due to Elasticsearch – query ES to investigate**

Query Elasticsearch to get its status, health, indices, shards, nodes, etc.

- Use the following command to see more options

```
curl localhost:9200/_cat
```

- Add '?v' after the chosen command to see column headers with output

```
curl localhost:9200/_cat/X?v
```

where 'X' is 'health', 'shards', 'indices', 'recovery', etc. and '?v' provide output with column headers

- View more data in Elasticsearch

```
curl localhost:9200/_all?pretty
```

Changes to make based on file locations, etc.

- In `log_alerts.sh` and `log_alerts_dates.sh` change the filepaths
  - Change the paths to `gen_alerts.py`
  - Change the name and path to the output file if desired
- In `snort-alerts.conf` change the input file and path accordingly

Now that everything is setup – basic test run

(1) **Spin up Elasticsearch**

```
elasticsearch-2.3.3/bin/elasticsearch
```

(2) **Spin up Kibana**

```
kibana-4.5.3-linux-x64/bin/kibana
```

(3) **Spin up Logstash**

```
logstash-2.3.3/bin/logstash -f logstash-
2.3.3/snort-alerts.conf
```

(4**) Run the script to continuously generate alerts**

```
./log_alerts_dates.sh
```

You will see 'Generating X alerts' and 'Sleeping for X seconds…' indicating that alerts have been appended to the file `/var/log/snort/alert` (ensure file exists)

(5) **Switch to the console running Logstash and within approximately 10 seconds you should see alerts being filtered**

*This is because we are sending alerts to Elasticsearch and stdout (specified in `snort-alerts.conf`)

(6) **Once your Logstash input file contains alerts, open up the browser and access Kibana (localhost:5601)**

(i) In Kibana you will first need to 'Configure an index pattern'
Under 'Index name or pattern' replace '`logstash-*`' with '`snort-alert-*`' as is defined in the Logstash configuration file

(ii) Under 'Time-field name' using the drop-down menu select 'logts' to store the alerts based on their log timestamps or select '@timestamp' to store the alerts based on the time the alerts were added to Elasticsearch.

*The former is highly recommended since you will most likely not always be submitting alerts on the same day they were created.

(iii) Click the 'Create' button, which should now be green

* If this is greyed out, uncheck the checkbox next to 'Index contains time-based events' and check it again to refresh or refresh the page altogether

(iv) Under the tabs Settings —> Objects select the 'Import' button
Select the file `dek_kibana_viz_setup_Aug-16.json`

(v) Navigate to 'Dashboard' then click the 'Load Saved Dashboard' folder button, and select 'Default' to open and view the Dashboard. You can also go to 'Discover' to view data (change the time range in the upper right corner).

Additional Elasticsearch and Kibana commands/options

- **General Kibana introduction**
  - <https://www.elastic.co/guide/en/kibana/current/index.html>
- **Create new visualizations**
  - <https://www.elastic.co/guide/en/kibana/current/visualize.html>
- **Create new searches / hone in on specific data in the database**
  - <https://www.elastic.co/guide/en/kibana/current/discover.html>

- **Remove all data stored in Elasticsearch (ES must be running, Kibana off)**

  Run the following command a few times:

  ```
  curl -XDELETE 'http://localhost:9200/_all'
  ```

  You should see `{"acknowledged":true}` responses to the query.

  ctrl + c the Elasticsearch process, then start it back up again.

  You should see 'recovered [0] indices into cluster_state' in the stdout info, and if you see that [x] indices have been recovered, repeat the process.

  * If Kibana is left running during this process an index for Kibana will be created, and in this case you will see a nonzero number of indices. To ensure the index is for Kibana -

  ```
  curl localhost:9200/_cat/indices?v
  ```

- **Create a snapshot of your entire Elasticsearch database**
  ***We suggest installing 'Sense' – an app for Kibana to make larger queries easier**

  < https://www.elastic.co/guide/en/sense/current/installing.html>

  Setup – register a snapshot repository in Elasticsearch:

```
PUT /_snapshot/backup
{
      "type": "fs",
      settings": {
            "location":    "/home/username/backup-
      folder",
            "compress": true,
            "chunk_size": "10m"
      }
}
```

Add following line to elasticsearch.yml:

path.repo: /home/username/backup-folder

Create:

curl -XPUT localhost:9200/_snapshot/backup/snapshot-name

Check:

curl -XGET localhost:9200/_snapshot/backup/snapshot-name?pretty

Restore:

curl -XPOST localhost:9200/_snapshot/backup/snapshot-name/_restore

Delete:

curl -DELETE /_snapshot/backup/snapshot_name

For more specific details:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-snapshots.html>

Example output from Elasticsearch, Kibana, and Logstash

Elasticsearch:

[2016-08-05 14:33:35,290][INFO][node] [Aurora] version[2.3.4], pid[7957], build[e455fd0/2016-06-30T11:24:31Z] [2016-08-05 14:33:35,297][INFO][node] [Aurora] initializing ...
[2016-08-05 14:33:35,905][INFO][plugins] [Aurora] modules [reindex, lang-expression, lang-groovy], plugins [], sites [] [2016-08-05 14:33:35,920][INFO][env] [Aurora] using [1] data paths, mounts [[/ (/dev/sda1)]], net usable_space [3.1gb], net total_space [6.7gb], spins? [possibly], types [ext4]
[2016-08-05 14:33:35,920][INFO][env] [Aurora] heap size [1015.6mb], compressed ordinary object pointers [true]
[2016-08-05 14:33:38,293][INFO][node] [Aurora] initialized
[2016-08-05 14:33:38,293][INFO][node] [Aurora] starting ...
[2016-08-05 14:33:38,411][INFO][transport] [Aurora] publish_address {127.0.0.1:9300}, bound_addresses {[::1]:9300}, {127.0.0.1:9300}
[2016-08-05 14:33:38,414][INFO][discovery] [Aurora] elasticsearch/8nba3kiqR0SoEJjfv5Serg [2016-08-05 14:33:41,520][INFO][cluster.service] [Aurora] new_master {Aurora}{8nba3kiqR0SoEJjfv5Serg}{127.0.0.1}{127.0.0.1:9300}, reason: zen-disco-join(elected_as_master, [0] joins received)
[2016-08-05 14:33:41,557][INFO][gateway] [Aurora] recovered [0] indices into cluster_state
[2016-08-05 14:33:41,558][INFO][http] [Aurora] publish_address {127.0.0.1:9200},

bound_addresses {[::1]:9200}, {127.0.0.1:9200}
[2016-08-05 14:33:41,559][INFO][node] [Aurora] started [2016-08-05 14:33:53,162][INFO][cluster.metadata] [Aurora] [.kibana] creating index, cause [api], templates [], shards [1]/[1], mappings [config]
[2016-08-05 14:33:53,462][INFO][cluster.routing.allocation] [Aurora] Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[.kibana][0]] ...]).
[2016-08-05 14:34:32,990][INFO][cluster.metadata] [Aurora] [logstash-2016.08.05] creating index, cause [auto(bulk api)], templates [logstash], shards [5]/[1], mappings [_default_, logs]
[2016-08-05 14:34:33,274][INFO][cluster.routing.allocation] [Aurora] Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[logstash 2016.08.05][4]] ...]).
[2016-08-05 14:34:33,353][INFO][cluster.metadata] [Aurora] [logstash-2016.08.05] update_mapping [logs]
[2016-08-05 14:34:33,428][INFO][cluster.metadata] [Aurora] [logstash-2016.08.05] update_mapping [logs]

Kibana:

log [14:33:47.833] [info][status][plugin:kibana] Status changed from uninitialized to green – Ready log [14:33:47.863] [info][status][plugin:elasticsearch] Status changed from uninitialized to yellow – Waiting for Elasticsearch
log [14:33:47.881] [info][status][plugin:kbn_vislib_vis_types] Status changed from uninitialized to green - Ready
log [14:33:47.887] [info][status][plugin:markdown_vis] Status changed from uninitialized to green – Ready
log [14:33:47.898] [info][status][plugin:metric_vis] Status changed from uninitialized to green – Ready
log [14:33:47.901] [info][status][plugin:spyModes] Status changed from uninitialized to green – Ready
log [14:33:47.903] [info][status][plugin:statusPage] Status changed from uninitialized to green – Ready
log [14:33:47.906] [info][status][plugin:table_vis] Status changed from uninitialized to green – Ready
log [14:33:47.918] [info][listening] Server running at http://0.0.0.0:5601
log [14:33:52.971] [info][status][plugin:elasticsearch] Status changed from yellow to yellow - No existing Kibana index found
log [14:33:56.073] [info][status][plugin:elasticsearch] Status changed from yellow to green – Kibana index ready

Logstash output when processing an alert (*Not all will have GeoIP data):


```
{
        "message" => "2016/08/05-13:12:44.182 [**] [1:1625:14] \"PROTOCOL-FTP SYST
        overflow
        attempt\" [**] [Classification: protocol-command-decode] [Priority: 3] {tcp}
        144.243.61.226:60277 -> 172.31.24.253:21",
        "@version" => "1",
        "@timestamp" => "2016-08-05T18:35:29.087Z",
        "path" => "/var/log/snort/alert",
        "host" => "elk-VirtualBox",
        "logts" => "2016-08-05T17:12:44.182Z",
        "gid" => "1",
        "sid" => "1625",
         "rev" => "14",
        "msg" => "PROTOCOL-FTP SYST overflow attempt",
        "classification" => "protocol-command-decode",
        "priority" => "3",
        "protocol" => "tcp",
        "sip" => "144.243.61.226",
        "sport" => "60277",
        "dip" => "172.31.24.253",
        "dport" => "21",
        "geoip" => {
                "ip" => "144.243.61.226",
                "country_code2" => "US",
                "country_code3" => "USA",
                "country_name" => "United States",
                "continent_code" => "NA",
                "region_name" => "MD",
                "city_name" => "Annapolis",
                "postal_code" => "21401",
                "latitude" => 38.987899999999996,
                "longitude" => -76.5454,
                "dma_code" => 512,
                "area_code" => 410,
                "timezone" => "America/New_York",
                 "real_region_name" => "Maryland",
                "location" => [
                        [0] -76.5454,
                        [1] 38.987899999999996
                ]
        }
}
```

List of Symbols, Abbreviations, and Acronyms

ARL	US Army Research Laboratory
CIDR	classless inter-domain routing
IP	Internet Protocol
NIDS	network intrusion detection system
NIPS	network intrusion prevention system
SELK	Snort–Elasticsearch–Logstash–Kibana
TCP	transmission control protocol
VM	virtual machine

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

3 (PDF) DIR USARL RDRL CIN D D KRYCH T BRAUN J EDWARDS

INTENTIONALLY LEFT BLANK.