Practical Attack Detection, Analysis, and Response using Big Data, Semantics, and Kill Chains within the OODA Loop

GIAC (GCIA) Gold Certification

Author: Brian Nafziger, brian @ nafziger.net
Advisor: Barbara L. Filkins
Accepted: May 30 2015

Abstract

The traditional approach to using toolsets is to treat them as independent entities – detect an event on a device with one tool, analyze the event and device with a second tool, and finally respond against the device with a third tool. The independent detection, analysis, and response processes are traditionally static, slow, and disjointed.

The modern approach to using toolsets must leverage them in an adaptive, synergistic, and agile manner. Colonel John Boyd's decision cycle or OODA loop (Observe-Orient-Decide-Act) "favors agility over raw power" and is potentially apropos for synergistic, agile, and rapid incident detection, analysis, and response. Layering Boyd's OODA loop on a framework of Big Data, Semantics, and Kill Chains is potentially, the choice for not only detecting modern attacks, but also for augmented, analysis, and response in an adaptive, synergistic, and agile manner.

The objective is to show that Big Data, Semantics, Kill Chains, and the OODA loop offer the ability to augment the human in detection, analysis, and response with adaptivity, synergy, and agility.

1. Introduction

"What is strategy? A mental tapestry of changing intentions for harmonizing and focusing our efforts as a basis for realizing some aim or purpose in an unfolding and often unforeseen world of many bewildering events and many contending interests" (Boyd, 2006).

A groundswell of heterogeneous cyber security strategies, operations, tactics, and tools now exist (Vincent, 2014). Navigating this complex ecosystem is a requirement for security operations incident detection, analysis, and response (Flynn, 2012). Colonel John Boyd's decision cycle or OODA loop framework is often applied successfully in strategic combat operations to augment human decision-making using the elements of observation, orientation, decision, and action (Bailer, 2007). Boyd's OODA loop favors agility or adaptability over power (Bailer, 2007). Layering Boyd's OODA loop on a framework of Big Data, Semantics, and Kill Chains potentially offers, the tool, methods, model and decision cycle of choice for augmented, adaptive, synergistic, and agile incident detection, analysis, and response (Nafziger, 2014).

The simple adage "crawl, walk, run", applies to the maturity of security operations. Crawling and walking are akin to event data (observation) and the manual analysis and response processes (Martin, 2014). Running, the next stage in maturity, is the ability to contextualize the data (orient), decide based on that data (decide) and act based on that data (act) in an augmented, adaptive, synergistic, and agile manner (Schneier, 2014). That next stage in maturity is becoming necessary because of the complexity of the operational ecosystem and the limited capacity of the human mind to cope with complexity (Heuer, 1999). We are at the beginning of the era of augmented, adaptive, synergistic, and agile incident analysis and response (Schneier, 2014)

The objective of this paper is to show a practical framework for detection, analysis and response across the ecosystem. The framework must allow for augmented, adaptive, synergistic, and agile decisions and actions (Schneier, 2014). The objective of the paper is to deliver that starting point.

1.1. Detection – Big Data, Semantics, and the Kill Chain

Nafziger's proposed framework using Big Data, Semantics, and Kill Chains lays the foundation for attack detection (Nafziger, 2014).

"Big Data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization" (Laney, 2012). Digital data abounds and capturing it in a useful manner requires careful planning and execution (Duncan, 2014). Applying this concept to security, Big Data captures and organizes this digital data, typically referenced as events, from across diverse domains into a common information model (Splunk, 2014). In that model, data is normalized into a lingua franca, for instance, a field named src or src_ip references event source ip no matter its origination. Big Data also typically offers the ability to search and manipulate events, especially in complex ways (Alspaugh, S., Ganapathi, A., Hearst, M., & Katz, R., 2013).

Semantics originate in grammar. Grammar consists of rules of syntax and semantics. Syntax rules concern sentence structure and semantic rules concern sentence meaning (Anderson, 1990, p.g. 352, 396). Applying this concept to security, syntax rules detect an attack based on an event element such an ip or file hash while semantic rules detect an attack based on events signifying the adversary's tactics, techniques, and procedures (Bianco, 2013). Semantic detection is the preferred form of detection since it is easy for an adversary to change a file hash or an ip and increasingly difficult to change tactics, techniques, and procedures (Bianco, 2013). Semantic detection uses a variety of methods such as state, behavior, baseline, statistics, machine learning, and or data mining-based methods including the use of disciplines outside the normal security realm (Talabis, 2007). Semantic detection may use a combination of methods known as an ensemble to increase the reliability (Xin, 2013).

The Kill Chain originated with Air Force General Ronald Fogleman as a targeting concept of Find, Fix, Track, and Target and was later amended with Engage and Assess – fully known as F2T2EA (Tirpak, 2000). Applying a variation of this concept to security, the Kill Chain signifies an adversary's chain of progressive actions in an intrusion and the defenders ability to not only detect the intrusion as it progresses along the chain but also mitigate it (Hutchins, 2010). One of the critical Kill Chain ideas is that it is possible to stop an attack by simply stopping the attack at a single point along the chain (Olesker, 2012). Decomposing the Kill Chain steps into its processes - reconnaissance is target selection, weaponization is exploit creation, delivery is exploit conveyance, exploitation is exploit detonation, installation is exploit installation, command and control is exploit persistence, and actions on objectives are the final actions on the target (Hutchins, 2010). The final actions on objectives can include access to, disclosure of, modification of, destruction of, or withholding of information (Benson, n.d.).

1.2. Analysis and Response – the OODA Loop

This proposed framework revision adds the potential for automated or augmented (partially automated with a human in the loop) analysis and response using OODA.

The OODA Loop originated with Air Force Fighter Pilot Colonel John Boyd, known for numerous contributions to military strategy and tactics (Cowan, 2000). Boyd's theories originated from his experience in air-to-air combat and his scientific and historical research. The cornerstone of his theory is "Patterns of Conflict" which describes one of the concepts used in idea air-to-air combat - "fast transients suggests that, in order to win, we should operate at a faster tempo or rhythm than our adversaries - or, better yet, get inside adversary's observation-orientation-decision-action time cycle or loop" (Boyd, 2007). The concept became known as the OODA loop or the larger theme of adaptability (Osinga, 2013). The rationale for the OODA loop: appear unpredictable and, therefore, generate confusion as the adversary attempts to comprehend the events (Boyd, 2007). Boyd also describes the concepts of ambiguity – creating competing views of events, deception – creating views of events that are not, and novelty – creating views of events that have never been seen before - the pay-off of these concepts being the disorientation, disruption, and overload of the adversary (Richards, 2001).

Though born as a theory of combat, the OODA loop has migrated into varied domains of decision-making. Decomposing the OODA Loop steps - Observation is the process of understanding of one's environment; Orientation is the process of analysis and synthesis through understanding heritage, tradition, current circumstances, and previous experience; Decision is the mental process of selecting an action from among the options presented in observation and orientation; and Action is the process of performing the action (Cowan, 2000). Applying these concepts to security, Schneier suggests in "The Future of Incident Response" that Observation means understanding our network including but not limited to events and metadata from the boundary to the endpoint, Orientation means understanding our network within the context of the company, Decision means determining the proper action with the proper authority, and Action means performing the decision quickly and effectively (Schneier, 2014). Keanini suggests the OODA loop is an essential component of "A Holistic Approach to Cyber Security" where Observation and Orientation are the intelligence providing situational awareness and Decision and Action are the execution (Keanini, 2014).

2. Growing the Framework

The objective of this paper is to revise the Big Data, Semantic, and Kill Chain framework hereafter known as the Framework. Observation, the first letter in OODA, exists in the original and revised Frameworks - defining events, utilizing semantic methods to detect potential attack patterns, and utilizing kill chains to detect potential chain traversal. Orientation, the second letter in OODA, exists in the original and revised Frameworks – enriching events, adding context to determine the relevance and impact of events. Decision, the third letter in OODA, is new to the Framework – augmenting human decision or automating decisions by recommending potential actions. Action, the fourth and final letter is new to the Framework – performing the approved actions across a set of varied tools. The Decision and Action stages offer feedback into the original Observation and Orientation stages thereby allowing or completing the OODA loop. The resulting sum of new features augments human decisions, allows automated decisions, allows adaptive actions, allows for synergy across toolsets, and creates agility for moving from observation to action - the pay-off of these being disorientation, disruption, and overload of the adversary.

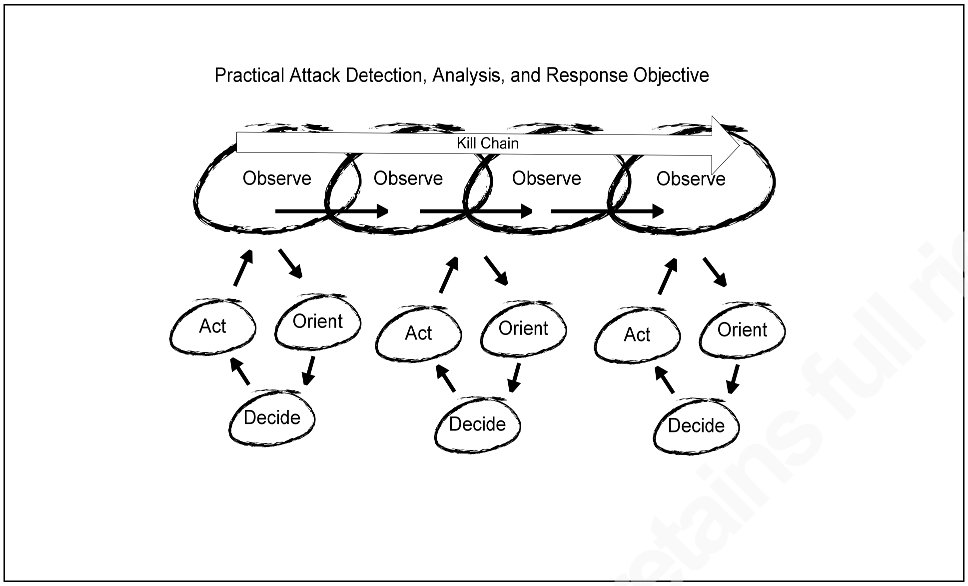

Figure 1: Objective

This paper strives to focus solely as a proof of concept of the value of *combining* big data, semantics, kill chains and the OODA loop into a framework. The individual components alone are topics of past and continuing future research. This paper strives to present the case for future integration methodologies. WARNING Splunk queries and Python code presented throughout this paper are derived from working framework queries and code, however, they are proof of concept (and both simplified and obfuscated) and as such, do not follow best practices. In production, please ensure proper design and coding including but not limited to security, logging, and error handling.

2.1. Observation and Orientation aka Detection

The Observation and Orientation steps build on the Data Mining concepts of Knowledge Discovery and Feature Selection (Brownlee, 2014). The Framework starts with events, detects potential features or indicators using semantic methods and then detects potential chains across semantic indicators. (Nafziger, 2014)

Observation requires events. "Event[s] can be defined as any detectable or discernable occurrence that has significance" (UCISA, n.d.). Events originate across the enterprise landscape from the web sites that customers use to the laptops that employees use. Domains, models, and elements organize events. Domains organize collections of models in similar operational spaces such as boundary, identity, and endpoint domains

(Robb, 2011). Models organize collections of elements such as a boundary filtering model consisting of a network firewall or proxy. Elements are a precise unit of knowledge such as an ip address or bytes or time. Splunk as the Big Data environment naturally organizes these events (Splunk, 2014). Reviewing from Nafziger's prior work, Figure 2 in the appendix shows a simple proxy query.

Orientation requires context - understanding our network within the context of the company (Schneier, 2014). Context associates an event with a proper understanding of the event value and significance. Context often begins with assets, identities, and vulnerabilities but can and will include a multitude of contexts (Chuvakin, 2010). Figure 3 in the appendix shows the creation of the dynamic asset context table periodic query capturing DHCP events and then the resulting value added to previous simple proxy query by deriving context from the dynamic asset context table (Nafziger, 2014). Figure 4 in the appendix shows several suggested contexts (Nafziger, 2014).

Orientation also requires semantics – understanding what is normal and what is not normal – using methods such as base lining length of connections, number of packets, or amount data (Cole, 2013). Figure 5 in the appendix shows the creation of a semantic method identifying abnormally large outbound proxy traffic (and using the dynamic asset context table) which then saves the results as a trigger for a Kill Chain table (Nafziger, 2014). Figure 6 in the appendix shows several suggested semantics and where the semantic resides within the Kill Chain (Nafziger, 2014).

Observation and Orientation - events, contexts, and semantics – culminates in a Kill Chain table. Mining the Kill Chain table provides a list of potential attacks. Figure 7 in the appendix shows a complete Kill Chain query of the events, contexts, and semantics (which are continuously populating the Kill Chain table) to identify potential attacks (Nafziger, 2014).

2.2. Decision aka Analysis

The Decision step builds on the Data Mining concept of Decision Trees. The Framework starts with events, contexts, and semantics detecting potential chains across the semantics. The Framework now focuses on mapping these semantics to decisions for the purpose of driving analysis and response.

Decision trees are a common modeling technique easily incorporated into the Framework. Simplistically described, decision trees are models which input variables and predict results. The model organizes into a tree structure consisting of nodes and leaves where variables are iteratively compared against nodes resulting in a leaf (Shalizi, 2009). Decision trees are classification trees that use finite values or regression trees that use continuous variables. Decision trees use data with known or expected results to train the tree.

The Framework uses the CART (Classification and Regression Tree) algorithm from the popular book, "Programming Collective Intelligence", by Toby Segaran (Segaran, 2007). Figure 8 quickly shows how to download and use the CART decision tree that is available on GitHub (Matt, 2014). The steps are: 1) download and install the PIL library; 2) download the decision tree. Viewing treepredict.py shows the training data. Loading python, importing treepredict, and then loading the training data allows simple testing of data against the decision tree and simple printing of the decision tree. The decision tree works as expected.

```
$ wget http://effbot.org/media/downloads/PIL-1.1.7.tar.gz
$ tar zxf PIL-1.1.7.tar.gz
$ cd PIL-1.1.7
$ python setup.py install

$ wget https://github.com/sirMackk/collective_intelligence_examples/archive/master.zip
$ unzip master.zip
$ cd collective_intelligence_examples-master/chap7

$ more treepredict.py

# Referrer, Location, Read FAQ, Pages Viewed, Service Chosen
my_data=[['slashdot', 'USA', 'yes', 18, 'None'],
        ['google', 'France', 'yes', 23, 'Premium'],
        ['digg', 'USA', 'yes', 24, 'Basic'],
        ['kiwitobes', 'France', 'yes', 23, 'Basic'],
        ['google', 'UK', 'no', 21, 'Premium'],
        ['(direct)', 'New Zealand', 'no', 12, 'None'],
        ['(direct)', 'UK', 'no', 21, 'Basic'],
        ['google', 'USA', 'no', 24, 'Premium'],
        ['slashdot', 'France', 'yes', 19, 'None'],
```

Brian Nafziger, brian @ nafziger.net

```
        ['digg', 'USA', 'no', 18, 'None'],
        ['google', 'UK', 'no', 18, 'None'],
        ['kiwitobes', 'UK', 'no', 19, 'None'],
        ['digg', 'New Zealand', 'yes', 12, 'Basic'],
        ['slashdot', 'UK', 'no', 21, 'None'],
        ['google', 'UK', 'yes', 18, 'Basic'],
        ['kiwitobes', 'France', 'yes', 19, 'Basic']]

$ python
>>> import treepredict
>>> tree = treepredict.build_tree(treepredict.my_data)
>>> treepredict.classify(['google', 'USA', 'no', 23], tree)
{'Premium': 3}
>>> treepredict.print_tree(tree)
0:google?
T-> 3:21?
  T-> {'Premium': 3}
  F-> 2:yes?
    T-> {'Basic': 1}
    F-> {'None': 1}
F-> 0:slashdot?
  T-> {'None': 3}
  F-> 2:yes?
    T-> {'Basic': 4}
    F-> 3:21?
      T-> {'Basic': 1}
      F-> {'None': 3}
>>>
```

Figure 8: Installing and Using CART (Matt, 2014).

The primary requirement and challenge in creating a training tree or ruleset is creating a cohesive, comprehensive, and consistent taxonomy across the environment of events, contexts, semantics, kill chains (detection), decision tree training data (analysis) and actions (response). Figure 9 shows how to create a ruleset test environment. Quite simply, create the training and test csv files, load python, import treepredict and then load the training ruleset and testing data. Testing should include classifying ad-hoc test data and printing the trained decision tree. To best create the ruleset a bit of analysis and response foreknowledge must exist. The basic ruleset states, if the asset is in the exploit kill chain with any semantic (it is blank) and newly online (determined from the dynamic asset context table), then run the autorunsc tools as an action, if the asset has autorunsc results, then run an endpoint scan, etc. Once again, the decision tree works as expected, this time using contexts, semantics, and kill chains to produce the autorunsc action and the endpoint scan action.

```
# training data

$ cat treepredict-train.csv
chain, semantic, recentOnline, recentVMscan, recentEPscan, recentEPautoruns, action
NO,NO,NO,NO,NO,NO,noop
,,,,,, noop
Exploit,,YES,,,,autorunsc
Exploit,,YES,,,YES,epscan
Exploit,,YES,,YES,YES,vmscan
Exploit,,YES,YES,YES,YES,inform

# testing data

$ cat treepredict-test.bsn.csv
chain, semantic, recentOnline, recentVMscan, recentEPscan, recentEPautoruns, action
Exploit,,YES,,,,

# training and testing in action

$ python
>>> import treepredict
>>> train=[line.split(',') for line in file('./treepredict-train.csv')]
>>> for c, r in enumerate(train):
...     train[c]=[s.strip() for s in r]

>>> print train
[['chain', 'semantic', 'recentOnline', 'recentVMscan', 'recentEPscan', 'recentEPautoruns', 'action'],
['NO', 'NO', 'NO', 'NO', 'NO', 'NO', 'comment'], ['', '', '', '', '', '', 'comment'], ['Exploit', '', 'YES', '', '', '',
'autorunsc'], ['Exploit', '', 'YES', '', '', 'YES', 'epscan'], ['Exploit', '', 'YES', '', 'YES', 'YES', 'vmscan'],
['Exploit', '', 'YES', 'YES', 'YES', 'YES', 'inform']]

>>> tree = treepredict.build_tree(train)

>>> test=[line.split(',') for line in file('../lookups/treepredict-test.bsn.csv')]
>>> for c, r in enumerate(test):
...     test[c]=[s.strip() for s in r]

>>> treepredict.classify(test[0],tree)
{'autorunsc': 1}
```

```
>>> treepredict.classify(["", "", "", "", "", ""], tree)
{'noop': 2}

>>> treepredict.classify(["Exploit", "", "YES", "", "", ""], tree)
{'autorunsc': 1}

>>> treepredict.classify(["Exploit", "", "YES", "", "", "YES"], tree)
{'epscan': 1}
>>>

>>> treepredict.print_tree(tree)
5:YES?
T-> 3:?
   T-> 4:?
      T-> {'epscan': 1}
      F-> {'vmscan': 1}
   F-> {'inform': 1}
F-> 0:Exploit?
   T-> {'autorunsc': 1}
   F-> 0:chain?
      T-> {'action': 1}
      F-> {'comment': 2}
```

Figure 9: Decision Tree Testing Results

As previously stated, to best create the ruleset a bit of analysis and response foreknowledge must exist to grow the necessary contexts, semantics, and kill chains. Knowing the upcoming actions for a response, figure 10, 11, and 12 show new and old suggested contexts for the Framework. The context data will be populated later by the action command. Figure 13 shows the updated suggested semantics list.

```
# create an autorunsc context table periodic query

earliest=-1h index=autorunsc
| stats min(_time) as firstTime max(_time) as lastTime by
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256

| table firstTime lastTime
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256
```

```
| inputlookup append=T INFOSEC-CTX-ENDPOINT-AUTORUNS-DYNAMIC.csv
| where lastTime > relative_time(now(), "-30d")
| stats min(firstTime) as firstTime max(lastTime) as lastTime  by
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256
| table firstTime lastTime
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256
| outputlookup INFOSEC-CTX-ENDPOINT-AUTORUNS-DYNAMIC.csv

# query the content of autorunsc context table (populated later)

| inputlookup INFOSEC-CTX-ENDPOINT-AUTORUNS-DYNAMIC.csv
| table firstTime lastTime
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256
```

Figure 20: Autorunsc Context Query

```
# create a virus scan context table periodic query

earliest=-1h index=antivirus
| stats min(_time) as firstTime max(_time) as lastTime last(message) as lastMessageScan by host
| table host lastMessageScan firstTime lastTime

| inputlookup append=T INFOSEC-CTX-ENDPOINT-VIRUSSCAN-DYNAMIC.csv
| where lastTime > relative_time(now(), "-30d")
| stats min(firstTime) as firstTime max(lastTime) as lastTime  last(lastMessageScan) as
lastMessageScan  by host
| table host firstTime lastTime lastMessageScan
| outputlookup INFOSEC-CTX-ENDPOINT-VIRUSSCAN-DYNAMIC.csv

# query the content of the virus scan context table (populated later)

| inputlookup INFOSEC-CTX-ENDPOINT-VIRUSSCAN-DYNAMIC.csv
| table firstTime lastTime host  lastMessageScan
```

Figure 31: Virus Scan Context Query

```
# create a vuln scan context table periodic query
```

```
earliest=-1h index=vulnerabilities
| stats min(_time) as firstTime max(_time) as lastTime last(status) as status by id ip dns host os
type severity signature cve cvss
| table firstTime lastTime status ip host os type severity signature cve cvss

| inputlookup append=T INFOSEC-CTX-VULNERABILITY-DYNAMIC.csv
| where lastTime > relative_time(now(), "-30d")
| stats min(firstTime) as firstTime max(lastTime) as lastTime  last(status) as status by ip host os
type severity signature cve cvss
| table  firstTime lastTime status ip host os type severity signature cve cvss
| outputlookup INFOSEC-CTX-VULNERABILITY-DYNAMIC.csv

# query the content of the virus scan context table (populated later)

| inputlookup INFOSEC-CTX-VULNERABILITY-DYNAMIC.csv
| table firstTime lastTime status id ip dns host os type severity signature cve cvss
```

Figure 42: Vulnerability Scan Context Query

Context	#	Description
Assets	1	Asset DB Connection
	2	Assets Dynamic Collection
Endpoint	3	Endpoint Autoruns Dynamic Collection
	4	Endpoint Virus Scan Dynamic Collection
Identity	5	Identity DB Connection
	6	Identity Dynamic Collection
Vulnerability	7	Vulnerability DB Connection
	8	Vulnerability Dynamic Collection

Figure 53: Updated Suggested Contexts

Integration into the Framework is accomplished using a Splunk custom command pattern. The Splunk command selects the ruleset, trains using the ruleset, parses the incoming search data, classifies the search data, and finally returns the classification to the search stream. Figure 14 shows how to create the Splunk command: 1) copy the previously created training and testing dataset to the lookups directory; 2) copy the treepredict code to the bin directory; 3) append the command stanzas and execute a debug refresh; 4) modify the treepredict algorithm to provide a reference the external PIL library; 5) create the proof of concept decision tree command; 6) validate the command.

```
# copy files to /opt/splunk/etc/apps/search

cp treepredict-train.csv/opt/splunk/etc/apps/search/lookups
cp treepredict-test.csv/opt/splunk/etc/apps/search/lookups
cp treepredict.py /opt/splunk/etc/apps/search/bin/

# append to file name/location /opt/splunk/etc/apps/search/local/commands.conf

[decision]
filename = _decision.py
streaming = false
retainsevents = true
overrides_timeorder = false

# replace within file name/location /opt/splunk/etc/apps/search/bin/treepredict.py
# since PIL is not installed under Splunk's Python, append and import it

# from PIL import Image, ImageDraw
import sys
sys.path.append("/usr/lib64/python2.6/site-packages/PIL")
import Image, ImageDraw

# create file name/location /opt/splunk/etc/apps/search/bin/_decision.py
# proof of concept implementation of a Splunk CART Decision Tree Command
# dependent on Toby Segaran. 2007. Programming Collective Intelligence
# dependent on https://github.com/sirMackk/collective_intelligence_examples/tree/master/chap7

import os,csv,sys,time,string,logging,splunk.Intersplunk
import treepredict

fwnull = open(os.devnull, "w")
frnull = open(os.devnull, "r")

LOG_FILENAME = '/opt/splunk/etc/apps/search/bin/_decision.log'
LOG_FORMAT = "[%(asctime)s] %(name)s %(levelname)s: %(message)s"

try:
        logging.basicConfig(filename=LOG_FILENAME, \
                level=logging.DEBUG,format=LOG_FORMAT)
        logging.info(sys.argv)

        keywords, options = splunk.Intersplunk.getKeywordsAndOptions()
        ruleset = options.get('tree', 'default')
        logging.info(ruleset)
```

```
results,dummy,settings = splunk.Intersplunk.getOrganizedResults()
logging.info(results)

# training data

filt = lambda s: s.replace('\"','').replace('\'','')
fill = lambda s: s or ""

f = open("/opt/splunk/etc/apps/search/lookups/"+str(ruleset),"r")
data = f.read()
f.close()

keys = '\n'.join(data.split('\n')[:1])
keys = filt(keys).replace(' ', '').replace('\r','').split(',')
logging.info(keys)

train = '\n'.join(data.split('\n')[1:data.count('\n')])
logging.info(train)

train=[line.split(',') for line in train.split('\n')]
logging.info(train)

for col, row in enumerate(train):
        train[col]=[s.strip() for s in row]
logging.info(train)

tree = treepredict.build_tree(train)

for r in results:
        logging.info(r)

        # application data

        row = []
        for key in r:
                if key in keys:
                        row.append(r[key].strip())

        decision = treepredict.classify(row, tree)
        logging.info(decision)

        r["result"] = decision

splunk.Intersplunk.outputResults( results )

logging.info("exiting")
```

Brian Nafziger, brian @ nafziger.net

```
except:
        import traceback
        stack =  traceback.format_exc()
        results = splunk.Intersplunk.generateErrorResults("Error : Traceback: " + str(stack))

# execute the newly created command

| inputlookup treepredict-test.csv
| decision tree=treepredict-train.csv
| table chain,  recentOnline,  result

chain     recentOnline        result
Exploit   YES                 {'autorunsc': 1}
```

Figure 64: Splunk Decision Tree Command with Results

2.3. Action aka Response

The Action step builds on typical response tools. All that is necessary is to integrate the tools into the Framework. Many tools were available, and several tools were integrated. These few tools documented reflect a variety of methods to demonstrate the ease and flexibility of integration.

Integration into the Framework is accomplished using a Splunk custom command pattern. The logic, in particular, is inspired by Mark Baggett and the Impacket Collection of Python classes from Core Labs (Baggett, 2013; CoreLabs, 2013). The first Splunk command copies autorunsc to the remote asset using smbclient.py and then executes autorunsc using wmiexec.py. The results are syslogged as key-value pairs for easy capture and extraction by the previously defined autorunsc context query. Figure 15 shows how to create the Splunk command: 1) download and install the asn, crypto and impacket classes; 2) define the command via the commands stanza and execute a debug refresh; 3) create the proof of concept autorunsc command; 4) validate the command.

```
# inspired by
http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Impacket
https://isc.sans.edu/forums/diary/Automating+Incident+data+collection+with+Python/19025/
http://pen-testing.sans.org/blog/2013/03/27/psexec-python-rocks
```

```
$ wget http://sourceforge.net/projects/pyasn1/files/pyasn1/0.1.8/pyasn1-0.1.8rc1.tar.gz
$ tar -xvf pyasn1-0.1.8rc1.tar.gz
$ cd pyasn1-0.1.8rc1
$ python setup.py install

$ wget http://pypi.python.org/packages/source/p/pycrypto/pycrypto-2.6.tar.gz
$ tar xzf pycrypto-2.6.tar.gz
$ cd pycrypto-2.6
$ python setup.py install

$ wget --mirror http://impacket.googlecode.com/svn/trunk/      #impacket-0.9.13-dev
$ cd impacket.googlecode.com/svn/trunk
$ python setup.py install

$ wget https://download.sysinternals.com/files/Autoruns.zip
$ unzip Autoruns.zip



# file name/location /opt/splunk/etc/apps/search/local/commands.conf
# refresh via https://splunk01.company.com:8443/en-US/debug/refresh
[autorunsc]
filename = _autorunsc.py
streaming = true
retainsevents = true
overrides_timeorder = false

# proof of concept implementation of a Splunk Autoruns Command
# file name/location /opt/splunk/etc/apps/search/bin/_autorunsc.py

import os,sys,csv,string,socket,StringIO,splunk.Intersplunk
import shlex,subprocess
import logging

LEVEL = {
        'emerg': 0, 'alert':1, 'crit': 2, 'err': 3,
        'warning': 4, 'notice': 5, 'info': 6, 'debug': 7 }

FACILITY = {
        'kern': 0, 'user': 1, 'mail': 2, 'daemon': 3,
        'auth': 4, 'syslog': 5, 'lpr': 6, 'news': 7,
        'uucp': 8, 'cron': 9, 'authpriv': 10, 'ftp': 11,
        'local0': 16, 'local1': 17, 'local2': 18, 'local3': 19,
        'local4': 20, 'local5': 21, 'local6': 22, 'local7': 23 }

FILENAME = '/opt/splunk/etc/apps/search/bin/_autorunsc.log'
FORMAT = "[%(asctime)s] %(name)s %(levelname)s: %(message)s"
```

```
filt = lambda s: s.replace('\"','').replace('\'','')
fill = lambda s: s or ""

fwnull = open(os.devnull, "w")
frnull = open(os.devnull, "r")

try:
        logging.basicConfig(filename=FILENAME,level=logging.DEBUG,format=FORMAT)
        logging.info("entering")

        # get options
        keywords, options = splunk.Intersplunk.getKeywordsAndOptions()

        # get results
        results,dummy,settings = splunk.Intersplunk.getOrganizedResults()
        logging.info(results)

        f = open('./_autorunsc.put','w')
        f.write('use admin$\nput autorunsc.exe\nexit\n')
        f.close()

        for r in results:

                # IMPACKET classes can be imported and directly used

                cmd = "wmiexec.py USER:PASSWORD@"+filt(r['ip'])+" \"hostname\""
                cmd = subprocess.Popen(shlex.split(cmd), stdin=frnull,
                        stdout=subprocess.PIPE, stderr=fwnull)
                data, err = cmd.communicate()
                hostname = '\n'.join(data.split('\n')[3:4])
                logging.info(data)

                cmd = "smbclient.py USER:PASSWORD @"+filt(r['ip'])+" -f _autorunsc.put"
                cmd = subprocess.Popen(shlex.split(cmd), stdin=frnull,
                        stdout=subprocess.PIPE, stderr=fwnull)
                data, err = cmd.communicate()
                logging.info(data)

                cmd = "wmiexec.py USER:PASSWORD @"+filt(r['ip'])+" \"autorunsc.exe
/accepteula -acfv\""
                cmd = subprocess.Popen(shlex.split(cmd), stdin=frnull,
                        stdout=subprocess.PIPE, stderr=fwnull)
                data, err = cmd.communicate()

                data = data.decode('utf-16','ignore').encode('ascii','ignore')
```

```
                        keys = '\n'.join(data.split('\n')[:1])
                        keys = filt(keys).replace(' ', '').replace('\r','').split(',')

                        body = '\n'.join(data.split('\n')[1:data.count('\n')])

                        reader = csv.DictReader(StringIO.StringIO(body), fieldnames=keys,
skipinitialspace=True, delimiter=b',', quoting=csv.QUOTE_MINIMAL, quotechar=b'"')

                        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

                        for value in reader:
                                    kvtext = "hostname="+hostname.replace('\r','')+" "+"ip="+filt(r['ip'])+"
"
                                    for key in keys:
                                                kvtext += filt(key) + '=\"' + filt(value[key]).replace("\n","\\n")
+ '\" '
                                    data = '<%d>%s:
%s'%(LEVEL['notice']+FACILITY['daemon']*8,"autorunsc",kvtext)
                                    sock.sendto(data,  ("10.1.1.254", 514))

                        sock.close()

            # output results
            splunk.Intersplunk.outputResults( results )

            logging.info("exiting")

except:
   import traceback
   stack = traceback.format_exc()
   results = splunk.Intersplunk.generateErrorResults("Error : Traceback: "+str(stack))

# execute the newly created command

$ cat test.csv
host,ip
HOST1,10.1.1.1

| inputlookup test.csv | search ip=10.1.1.1 | autorunsc

# query the autorunsc context table

| inputlookup INFOSEC-CTX-ENDPOINT-AUTORUNS-DYNAMIC.csv
| table firstTime lastTime
```

```
hostname ip Time EntryLocation Entry Enabled Category Profile Description Publisher ImagePath
Version LaunchString MD5 SHA-1 PESHA-1 PESHA-256 SHA-256

firstTime  lastTime  hostname ip  Time  EntryLocation  Entry  Enabled
     Category  Profile  Description  Publisher ImagePath  Version
     LaunchString  MD5  SHA-1  PESHA-1 PESHA-256  SHA-256

1428890237  1428976502  HOST1  10.1.1.1  10/26/2011  12:23  PM
     HKLM\Software\Wow6432Node\Classes\CLSID\{083863F1-70DE-11d0-BD40-
00A0C911CE86}\Instance  AVI Decompressor  enabled  Codecs  System-wide
     DirectShow Runtime.  (Verified)  Microsoft  Windows
     c:\windows\syswow64\quartz.dll  6.6.7601.17713  HKCR\CLSID\{CF49D4E0-
1115-11CE-B03A-0020AF0BA770}  0ae0c4955e1de29ccdc9da1b816fe5ee  NULL  NULL
     NULL  NULL
```

Figure 75: Splunk Autoruns Command with Results

The next Splunk command executes an antivirus scan application on the remote asset using wmiexec.py. The results are naturally logged via the endpoint application and captured by the previously defined virus scan context query. Figure 16 shows how to create the Splunk command. The steps are: 1) download and install the asn, crypto, and impacket classes; 2) define the command via the commands stanza and execute a debug refresh; 3) create the proof of concept virus scan command; 4) validate the command.

```
#inspired by
http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Impacket
https://isc.sans.edu/forums/diary/Automating+Incident+data+collection+with+Python/19025/
http://pen-testing.sans.org/blog/2013/03/27/psexec-python-rocks

$ wget http://sourceforge.net/projects/pyasn1/files/pyasn1/0.1.8/pyasn1-0.1.8rc1.tar.gz
$ tar -xvf pyasn1-0.1.8rc1.tar.gz
$ cd pyasn1-0.1.8rc1
$ python setup.py install

$ wget http://pypi.python.org/packages/source/p/pycrypto/pycrypto-2.6.tar.gz
$ tar xzf pycrypto-2.6.tar.gz
$ cd pycrypto-2.6
$ python setup.py install

$ wget --mirror http://impacket.googlecode.com/svn/trunk/ #impacket-0.9.13-dev
$ cd impacket.googlecode.com/svn/trunk
$ python setup.py install
```

```
# file name/location /opt/splunk/etc/apps/search/local/commands.conf
# refresh via https://splunk01.company.com:8443/en-US/debug/refresh
[virusscan]
filename = _virusscan.py
streaming = true
retainsevents = true
overrides_timeorder = false

# proof of concept implementation of a Splunk Virus Scan Command
# file name/location /opt/splunk/etc/apps/search/bin/_virusscan.py

import os,csv,sys,string,socket,splunk.Intersplunk
import shlex,subprocess
import logging

fwnull = open(os.devnull, "w")
frnull = open(os.devnull, "r")

LOG_FILENAME = '/opt/splunk/etc/apps/search/bin/_virusscan.log'
LOG_FORMAT = "[%(asctime)s] %(name)s %(levelname)s: %(message)s"

try:
	logging.basicConfig(filename=LOG_FILENAME,level=logging.DEBUG,format=LOG_FO
RMAT)
	logging.info("entering")

	# get options
	keywords, options = splunk.Intersplunk.getKeywordsAndOptions()
	mode = options.get('mode', 'unknown')
	logging.info(mode)

	# get results
	results,dummy,settings = splunk.Intersplunk.getOrganizedResults()
	logging.info(results)

	for r in results:

		# IMPACKET classes can be imported and directly used

		cmd = "wmiexec.py USER:PASSWORD@"+r['ip']+" \"wmic process call create
\'C:\Progra~1\Endpoint\Scan.exe /drive c\'\""
		cmd = subprocess.Popen(shlex.split(cmd),
				stdin=frnull, stdout=subprocess.PIPE, stderr=fwnull)
		data,err = cmd.communicate()
```

```
                        logging.info(cmd)
                        logging.info(data)

                # output results
                splunk.Intersplunk.outputResults( results )

                logging.info("exiting")

except:
    import traceback
    stack =  traceback.format_exc()
    results = splunk.Intersplunk.generateErrorResults("Error : Traceback: " + str(stack))

# execute the newly created command

$ cat test.csv
host,ip
HOST1,10.1.1.1

| inputlookup test.csv | search ip=10.1.1.1 | virusscan

# query the virus scan context table

| inputlookup INFOSEC-CTX-ENDPOINT-VIRUSSCAN-DYNAMIC.csv
| table firstTime lastTime host  lastMessageScan

firstTime  lastTime  host        lastMessageScan
1427566743        1427566743          HOST1    Scan Complete: Risks: 0 Scanned: 123788
```

Figure 86: Splunk Virus Scan Command with Results

The next Splunk command executes a vulnerability scan application on the remote asset using a REST API call. The results are naturally logged via the vulnerability application logging and captured by the previously defined vulnerability scan context query. Figure 17 shows how to create the Splunk command: 1) define the command via the commands stanza and execute a debug refresh; 2) create the proof of concept virus scan command; 3) validate the command.

```
# file name/location /opt/splunk/etc/apps/search/local/commands.conf
# refresh via https://splunk01.company.com:8443/en-US/debug/refresh
[vulnscan]
filename = _vulnscan.py
```

```
streaming = true
retainsevents = true
overrides_timeorder = false

# proof of concept implementation of a Splunk Vuln Scan Command
# file name/location /opt/splunk/etc/apps/search/bin/_vulnscan.py

import os,sys,csv,string,socket,datetime,StringIO,splunk.Intersplunk
import shlex,subprocess
import logging

filt = lambda s: s.replace('\"','').replace('\'','')
fill = lambda s: s or ""

fwnull = open(os.devnull, "w")
frnull = open(os.devnull, "r")

try:
        logging.basicConfig(filename=FILENAME,level=logging.DEBUG,format=FORMAT)
        logging.info("entering")

        # get options
        keywords, options = splunk.Intersplunk.getKeywordsAndOptions()
        mode = options.get('mode', 'unknown')
        logging.info(mode)

        # get results
        results,dummy,settings = splunk.Intersplunk.getOrganizedResults()
        logging.info(results)

        scanlist=""
        for r in results:

                scanlist += r['ip']+","

        scanlist = scanlist[:-1]
        scannerlist = "VULNSCANNER01"

        logging.info(scanlist)
        logging.info(scannerlist)

        setupScan = "curl -H 'X-requested-With: curl' -X 'POST' -u USER:PASSWORD
'https://SCANMGR/definegroup.php?action=edit&title=DynamicScan&ips=" + scanlist +
"&scanners=" + scannerlist + "'"

        cmd = subprocess.Popen(shlex.split(setupScan), stdin=frnull,
```

Brian Nafziger, brian @ nafziger.net

```
                        stdout=subprocess.PIPE, stderr=fwnull)
            data, err = cmd.communicate()

            logging.info(setupScan)
            logging.info(data)

            startScan = "curl -H 'X-Requested-With: curl' -X 'POST' -u USER:PASSWORD
'https://SCANMGR/scan.php?action=launch&group=DynamicScan'"

            cmd = subprocess.Popen(shlex.split(startScan), stdin=frnull,
                        stdout=subprocess.PIPE, stderr=fwnull)
            data, err = cmd.communicate()

            logging.info(startScan)
            logging.info(data)

            # output results
            splunk.Intersplunk.outputResults( results )

            logging.info("exiting")

except:
   import traceback
   stack = traceback.format_exc()
   results = splunk.Intersplunk.generateErrorResults("Error : Traceback: "+str(stack))

# execute the newly created command

$ cat test.csv
host,ip
HOST1,10.1.1.1

| inputlookup test.csv | search ip=10.1.1.1 | vulnscan

# query the vuln scan context table

firstTime   lastTime    status  ip      host    os      type    severity    signature cve
            cvss
1428568576          1429007194          Active  10.x.x.x    HOST2   Windows 7
            Confirmed       2       Internet Explorer Vulnerability CVE-20XX-XXXX   5
1428564792          1428564792          Active  10.x.x.x    HOST2   Windows 7
            Confirmed       2       Internet Explorer Vulnerability CVE-20XX-XXXX   5
```

Figure 97: Splunk Vulnerability Scan Command with Results

2.4. OODA in Action

The original Kill Chain logic detects multiple events completing a kill chain by using transactions. Figure 18 shows remnants of the original Kill Chain logic with the new created and appended decision and action logic. The decision logic resulting actions feed into the Splunk map command thereby executing the selected actions using the Splunk custom action commands called dynamically via the Splunk script command. Figure 18 shows commentary, and the results of each stage interspersed throughout the singular query. The kill chain with decisions and actions works as expected.

```
# create a kill chain trigger event with decision and actions!

# read existing events
|inputlookup INFOSEC-CHAIN.csv | sort - lastTime

# null incomplete fields for transitive transactions
| eval host=upper(host) | eval user=upper(user)
| eval host = if(host="-","NULL",host) | eval user = if(user="-","NULL",user)
| eval host = if(host="None","NULL",host) | eval user = if(user="None","NULL",user)
| eval thost = if(isnull(host),"NULL",host) | eval tuser = if(isnull(user),"NULL",user)

# create raw event
| eval _time=lastTime | eval _raw = "time=\""+strftime(lastTime , "%Y-%m-%d %H:%M:%S")+"\" host="+thost+" user="+tuser+" semantic=\""+semantic+"\" chain=\""+chain+"\""

# simplified single transaction with search to find chain
| transaction host user connected=f mvlist=t mvraw=t delim="\n\n" maxspan=-1
| search *Delivery*Exploit*Exfiltrate*

| table _time _raw chain semantic host user closed_txn eventcount field_match_sum duration

# kill chain trigger events containing bundled events
```

_time	_raw	chain	semantic	host	user
2015-[illegible]	time="2015-[illegible]" host=[illegible]1 user=H[illegible] semantic="Mail Recipient Vulnerable" chain="Delivery" time="2015-[illegible]4" host=W[illegible]1 user=H[illegible] semantic="Endpoint Load Unique" chain="Exploit" time="2015-[illegible]53" host=W[illegible]1 user=H[illegible] semantic="Proxy Dest Unique" chain="Exfiltrate" time="2015-04-[illegible]7" host=WC[illegible]1 user=HC[illegible]3 semantic="Endpoint Risk Found" chain="Exploit"	Delivery Exploit Exfiltrate Exploit	Mail Recipient Vulnerable Endpoint Load Unique Proxy Dest Unique Endpoint Risk Found	W[illegible] W[illegible] W[illegible] W[illegible]	H[illegible]3 H[illegible]3 H[illegible]3 H[illegible]03
2015-[illegible]	time="2015-[illegible]0" host=W[illegible]1 user=H[illegible] semantic="Mail Sender Unique" chain="Delivery" time="2015-[illegible]" host=W[illegible]1 user=H[illegible]02 semantic="Endpoint Load Unique" chain="Exploit" time="2015-[illegible]2" host=W[illegible]1 user=H[illegible]02 semantic="Proxy Dest Unique" chain="Exfiltrate"	Delivery Exploit Exfiltrate	Mail Sender Unique Endpoint Load Unique Proxy Dest Unique	W[illegible] W[illegible] W[illegible]1	H[illegible]2 H[illegible]2 H[illegible]2

```
# singularize host and user
```

Brian Nafziger, brian @ nafziger.net

```
| eval host= mvindex(host, 0)
| eval user= mvindex(user, 0)

# perform context lookups
| lookup INFOSEC-CTX-IDENTITY-DB.csv user OUTPUT name department
| lookup INFOSEC-CTX-ASSET-DYNAMIC.csv host OUTPUT lastTime AS assetTime
| lookup INFOSEC-CTX-VULNERABILITY-DYNAMIC.csv host OUTPUT lastTime AS vmscanTime
| lookup INFOSEC-CTX-ENDPOINT-VIRUSSCAN-DYNAMIC.csv host OUTPUT lastTime AS
epscanTime
| lookup INFOSEC-CTX-ENDPOINT-AUTORUNS-DYNAMIC.csv host OUTPUT lastTime AS
epautorunsTime

| table _time user host ip assetTime vmscanTime epscanTime autorunsTime

# context added to kill chain trigger event
```

_time	user	host	ip	assetTime	vmscanTime	epscanTime	autorunsTime
2015-[illegible]	H[illegible]403	W[illegible]X1	10.[illegible]24	14[illegible]62 14[illegible]020			
2015-[illegible]	HF[illegible]	W[illegible]R1	1[illegible]73	14[illegible]421 14[illegible]06 14[illegible]07			

```
# binarize the context for decision-making
| eval recentOnline=if (floor((now()-assetTime)/60/60) <= 12,"YES","NO")
| eval recentVMscan=if (floor((now()-vmscanTime)/60/60) <= 24,"YES","NO")
| eval recentEPscan=if (floor((now()-epscanTime)/60/60) <= 24,"YES","NO")
| eval recentEPautoruns=if (floor((now()-epautorunsTime)/60/60) <= 24,"YES","NO")

# execute the decision
| table _time user host ip recentOnline recentVMscan recentEPscan recentEPautoruns
| decision tree=treepredict-train.csv
| table _time user host ip recentOnline recentVMscan recentEPscan recentEPautoruns result
| rex field=result "'(?P<cmd>[^']+)'"

# loop the decision command to the script command to execute the action
| map search="| stats count
| eval _time=now() | eval user=\"$user$\" | eval host=\"$host$\" | eval ip=\"$ip$\"
| eval recentOnline=\"$recentOnline$\" | eval recentVMscan=\"$recentVMscan$\"
| eval recentEPscan=\"$recentEPscan$\" | eval recentEPautoruns=\"$recentEPautoruns$\"
| eval result=\"$result$\" | eval cmd=\"$cmd$\" | script $cmd$
" maxsearches=10

| table  _time user host ip recentOnline recentVMscan recentEPscan recentEPautoruns result cmd

# resulting binarized decision context and resulting action
```

_time	user	host	ip	recentOnline	recentVMscan	recentEPscan	recentEPautoruns	result	cmd
2015-0[illegible]	H[illegible]03	W[illegible]	1[illegible]4	YES	NO	NO	NO	{'autorunsc': 1}	autorunsc
2015-0[illegible]	H[illegible]02	W[illegible]	1[illegible]	NO	NO	NO	NO	{'comment': 2}	comment

Figure 108: Splunk Kill Chain with OODA Decisions and Actions

3. Conclusion

Several noteworthy challenges occurred. As previously noted dirty data and latent data was always present and required correction (Nafziger, 2014). Challenges unique to this investigation were many: fickleness of the decision logic – the original ID3 decision algorithm did not handle missing data; fickleness of the action logic – occasionally tools broke due to conditions in the environment beyond control emphasizing the need for agility with multiple available actions (and the need for error handling); abstractions of the action logic – the desire to abstract the actions lead to several competing approaches. The first approach was a pass-through action command --a Splunk command directly called the OS command line. This approach was unwieldy passing a variety of arguments. The second approach was a remote action command --a Splunk command pushed (via psexec.py) a compiled Python script (via Pyinstaller) bundled with any necessary binaries to the target for execution. This approach, though successful, was complex. The chosen approach was using the Splunk map and script commands along with Splunk custom commands. The final and primary challenge as noted earlier was complexity - creating a cohesive, comprehensive, and consistent taxonomy across the environment.

Several excellent growth areas exist. There are additional concepts that need integrated such as F3EAD. There are additional events, contexts, and semantics that need integration such as firewalls and intrusion systems. There are additional interrogation and interdiction actions (scripts) that need to be integrated. There are Splunk architecture abstractions that need integrated such as event types, models, summaries, and macros. Moreover, finally, rigorous testing needs to be completed.

In the end Practical Attack Detection, Analysis, and Response with Big Data, Semantics, Kill Chains, and OODA appears viable in small controlled scenarios. The Framework shows the potential to augment, to be adaptive, to be synergistic, and to be agile.

4. References

Alspaugh, S., Ganapathi, A., Hearst, M., & Katz, R. (2013). Building Blocks for Exploratory Data Analysis Tools. Retrieved March, 2015, from http://www.eecs.berkeley.edu/~alspaugh/papers/lsa_idea_2013.pdf

Anderson, J. R. (1990). *Cognitive psychology and its implications* (3rd edition). New York: W. H. Freeman & Co.

Bagget, M. (2015). Automating Incident Collection. Retrieved April, 2015, https://isc.sans.edu/forums/diary/Automating+Incident+data+collection+with+Python/19025/

Bailer, J. (2007, June). Army Business Transformation: The Utility of Using Corporate Business Models within the Institutional Army. Retrieved January, 2015, from http://www.dtic.mil/get-tr-doc/pdf?AD=ADA471102

Benson, C. (n.d.). Security Threats. Security Threats. Retrieved March, 2015, from http://technet.microsoft.com/en-us/library/cc723507.aspx

Bianco, D. (2013, March 1). Enterprise Detection & Response: The Pyramid of Pain. Retrieved March, 2015, from http://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

Boyd, J. (2005, February). Organic Design. Retrieved March, 2015, from http://www.dnipogo.org/boyd/organic_design.pdf

Boyd, J. (2006, June). Strategy of ? and ?. Retrieved March, 2015, from http://www.dnipogo.org/boyd/strategic_game.pdf

Boyd, J. (2007, November). Patterns of Conflict. Retrieved March, 2015, from http://www.dnipogo.org/boyd/patterns_ppt.pdf

Boyd, J. (2010, November). The Essence of Winning and Losing. Retrieved March, 2015, from http://pogoarchives.org/m/dni/john_boyd_compendium/essence_of_winning_losing.pdf

Brownlee, J. (2014, January). What is Data Mining and KDD. Retrieved April, 2015, from http://machinelearningmastery.com/what-is-data-mining-and-kdd/

Chuvakin, A., Knapp, E. (2010, January). Content Aware Siem Define. Retrieved April, 2015, from http://www.enterprisemanagement360.com/wp-content/files_mf/white_paper/content_aware_siem_defined.pdf

Brian Nafziger, brian @ nafziger.net

Cole, E. (2012). Advanced Persistent Threat: Understanding the Danger and How to Protect your Organization. Waltham, MA: Syngress.

Core Labs. (2003). What is Impacket? Retrieved April, 2015, http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Impacket

Cowan, J. (2000). From Air Force Fighter Pilot to Marine Corps Warfighting: Colonel John Boyd, His Theories on War, and their Unexpected Legacy. Retrieved March, 2015, from http://www.dnipogo.org/fcs/boyd_thesis.htm

Duncan, A. (2014, February 1). The ABC of Data Governance: Driving Information Excellence. Retrieved March, 2015, from http://www.slideshare.net/AlanDDuncan/the-abc-of-data-governance

Flynn, J. (2012, July). Intrusions along the Kill Chain. Retrieved January, 2015, from https://media.blackhat.com/bh-us-12/Briefings/Flynn/bh-us-12-Flynn-intrusion-along-the-kill-chain-WP.pdf

Heuer, R. (1999). *Psychology of Intelligence Analysis* (2nd ed.). Washington, D.C.: Center for the Study of Intelligence, Central Intelligence Agency.

Hutchins, E. M., Cloppert, M. J., & Amin, R. M. (2010, November 21). Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains. . Retrieved March, 2015, from http://www.lockheedmartin.com/content/dam/lockheed/data/corporate/documents/LM-White-Paper-Intel-Driven-Defense.pdf

Keanini, T. (2014, March). The OODA Loop: A Holistic Approach to Cyber Security. Retrieved March, 2015, from http://www.slideshare.net/Lancope/ooda-loop-webinar2

Laney, D. (2012, June). 3D Data Management: Controlling Data Volume, Velocity, and Variety. Retrieved February, 2015, from https://www.gartner.com/doc/2057415/importance-big-data-definition

Martin, G. (2014, August). The Security Operations Holy Grail. Retrieved January, 2015, from http://threatstream.com/blog/socholygrail

Matt. (2014, April). Collective Intelligence Examples. Retrieved April, 2015, https://mattscodecave.com/posts/programming-collective-intelligence-chapter-7-

notes &
https://github.com/sirMackk/collective_intelligence_examples/tree/master/chap7

Nafziger, B. (2014, October). Practical Attack Detection using Big Data, Semantic-based methods, and the Kill Chain model. Retrieved January, 2015, from http://www.sans.org/reading-room/whitepapers/warfare/practical-big-data-kill-chain-framework-35487

Olesker, A. (2012, August) Questing the Offense – Defense Balance in Cyberspace. Retrieved January, 2015, from http://www.oodaloop.com/technology/2012/08/06/questioning-the-offense-defense-balance-in-cyberspace/

Osinga, F. (2006). Science, Strategy and War: The Strategic Theory of John Boyd. London ; New York, N.Y.: Routledge.

Osinga, F. (2013). 'Getting' A Discourse on Winning and Losing: A Primer on Boyd's 'Theory of Intellectual Evolution'. Retrieved March, 2015, from http://www.contemporarysecuritypolicy.org/assets/CSP-34-3%20Osinga.pdf

Richards, C. (2001). A swift, elusive sword: What if Sun Tzu and John Boyd did a national defense review? Washington, D.C.: Center for Defense Information.

Robb, C. (2011, October 1). The Heart of the Matter A Core Services Taxonomy for State IT Security Programs. . Retrieved March, 2015, from http://www.nascio.org/publications/documents/NASCIO_CoreSecuritySevices.pdf

Schneier, B. (2014, August). The State of Incident Response. Retrieved March, 2015, from https://www.youtube.com/watch?v=u54Radu2bF0

Schneier, B. (2014, November). The Future of Incident Response. Retrieved January, 2015, from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6924685

Segaran, T. (2007). Programming collective intelligence: Building smart web 2.0 applications. O'Reilly.

Shalizi, C. (2009, November). Data Mining 26-350. Classification and Regression Trees Lecture. Retrieved April, 2015, from http://www.stat.cmu.edu/~cshalizi/350/lectures/22/lecture-22.pdf

Splunk. (2014). Common Information Model. Retrieved March, 2015, from http://docs.splunk.com/Documentation/CIM/latest/User/Overview

Talabis, M. (2007). Security Analytics Project. Retrieved March, 2015, from https://www.blackhat.com/presentations/bh-usa-07/Del_Moral_Talabis/Presentation/bh-usa-07-del_moral_talabis.pdf

Tirpak, J. (2000, July 1). Find, Fix, Track, Target, Engage, Assess. Retrieved March, 2015, from http://www.airforcemag.com/magazinearchive/pages/2000/july%202000/0700find.aspx

UCISA. (n.d.). ITIL – A Guide to Event Management. Retrieved March, 2015, from https://www.ucisa.ac.uk/~/media/Files/members/activities/ITIL/service_operation/eventm_management/ITIL_a%20guide%20to%20event%20management%20pdf.ashx

Vincent, A. (2014, April). What is a Threat Intelligence Platform. Retrieved January, 2015, from http://www.threatconnect.com/news/threat-intelligence-platform-2-2/

Xin, Kai, (2013, October). Good Enough Analytics. Retrieved March, 2015, from http://www.slideshare.net/KaiX/good-enough-analyticsfinal

Yen, T. F., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., & Kirda, E. (2013, December). Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In Proceedings of the 29th Annual Computer Security Applications Conference (pp. 199-208). ACM.

4.1. Appendix

```
# simple proxy query

index=proxy | table _time src dest bytes_in bytes_out

_time                   src       dest            bytes_in  bytes_out
2015-04-01 13:55:17     10.x.x.x  mail.yahoo.com  39        228
```

Figure 2: Proxy Event Query (Nafziger, 2014)

```
# create a dynamic asset context table periodic query

earliest=-1h index=dhcp GrantLease OR RenewLease

| stats min(_time) as firstTime, max(_time) as lastTime by host ip mac
| table firstTime  lastTime host ip mac

| inputlookup append=T INFOSEC-CTX-ASSET-DYNAMIC.csv
| where lastTime > relative_time(now(), "-7d")

| stats min(firstTime) as firstTime max(lastTime) as lastTime by  host ip mac
| table firstTime  lastTime host ip mac
| outputlookup INFOSEC-CTX-ASSET-DYNAMIC.csv

# query the proxy using the dynamic asset context table

index=proxy
| lookup INFOSEC-CTX-ASSET-DYNAMIC ip as src OUTPUT host mac
| table _time src dest host mac

_time                   src       dest            host    mac
2015-04-12 15:43:44     10.x.x.x  ssl.gstatic.com HOST1   dd241dd368dd
```

Figure 3: Proxy Context Query (Nafziger, 2014)

Context	#	Description
Assets	1	Asset DB Connection
	2	Assets Dynamic Collection
Identity	3	Identity DB Connection
	4	Identity Dynamic Collection

Brian Nafziger, brian @ nafziger.net

Vulnerability	5	Vulnerability DB Connection
	6	Vulnerability Dynamic Collection

Figure 4: Suggested Contexts (Nafziger, 2014)

```
# create an abnormal proxy semantic event for the kill chain table

earliest=-1h index=proxy

| bucket _time span=1m
| stats sum(bytes_out) as bytes_out by _time src dest
| eventstats min(_time) as firstTime max(_time) as lastTime
        avg(bytes_out) as avg stdev(bytes_out) as stdev
| eval notable=avg + 3*stdev
| where bytes_out > notable

| lookup INFOSEC-CTX-ASSET-DYNAMIC ip as src OUTPUT host

| table _time host dest bytes_out notable

| eval chain = "Exfiltrate" | eval semantic = "Proxy Large Outbound"
| stats min(firstTime) as firstTime  max(lastTime) as lastTime by host semantic chain
| table firstTime lastTime host semantic chain

| inputlookup append=t INFOSEC-CHAIN.csv
| where lastTime > relative_time(now(), "-24h")
| stats first(firstTime) as firstTime last(lastTime) as lastTime  by host semantic chain
| table firstTime lastTime host semantic chain
| outputlookup INFOSEC-CHAIN.csv

# query the abnormal proxy semantic event within the kill chain table

| inputlookup INFOSEC-CHAIN.csv
| table firstTime lastTime host semantic chain

firstTime       lastTime        host    semantic                chain
1428861600      1428797700      HOST1   Proxy Large Outbound    Exfiltrate
1428839100      1428842700      HOST8   Proxy Large Outbound    Exfiltrate
```

Figure 5: Proxy Semantic Query – Abnormal Outbound Traffic (Nafziger, 2014)

Kill Chain	#	Description
Delivery	1	Mail Recipient Vulnerable

	2	Mail Sender Unique
Exploit	3	Endpoint Load Unique
	4	Endpoint Risk Found
Exfiltrate	5	Proxy Long Connect
	6	Proxy Frequent Connect
	7	Proxy Large Outbound
	8	Proxy Destination IP

Figure 6: Suggested Semantics (Nafziger, 2014)

```
# create a kill chain trigger event

# read existing events
|inputlookup INFOSEC-CHAIN.csv

# null incomplete fields for transitive transactions
| eval host=upper(host) | eval user=upper(user)
| eval host = if(host="-","NULL",host)
| eval user = if(user="-","NULL",user)
| eval host = if(host="None","NULL",host)
| eval user = if(user="None","NULL",user)

| eval thost = if(isnull(host),"NULL",host)
| eval tuser = if(isnull(user),"NULL",user)
| eval _time=lastTime

# create raw event from chain event
| eval _raw = strftime(lastTime , "%Y-%m-%d %H:%M:%S")+" host="+thost+" user="+tuser+" semantic="+semantic+" chain="+chain

# transactions to find chain
| transaction host user connected=f mvraw=t delim="\n\n" maxspan=-1 keepevicted="t" startswith="Delivery" endswith="Exploit"
| transaction host user connected=f mvraw=t delim="\n\n" maxspan=-1 keepevicted="t" startswith="Exploit" endswith="Exfiltrate"
| transaction host user connected=f mvraw=t delim="\n\n" maxspan=-1 startswith="Delivery" endswith="Exfiltrate"
| table _time _raw chain semantic host user closed_txn eventcount field_match_sum
| search closed_txn=1

# perform context lookups
| lookup INFOSEC-CTX-ASSET-DYNAMIC host OUTPUT lastTime ip mac
| eval lastAssetTime = strftime(lastTime , "%Y-%m-%d %H:%M:%S")
| lookup INFOSEC-CTX-IDENTITY-DYNAMIC  host OUTPUT lastTime src_host src_ip
```

```
| eval lastIdentTime = strftime(lastTime , "%Y-%m-%d %H:%M:%S")
| lookup INFOSEC-CTX-VULNERABILITY-DYNAMIC host OUTPUT lastTime signature status
| eval lastScanTime = strftime(lastTime , "%Y-%m-%d %H:%M:%S")

| table _time _raw  host user
lastAssetTime ip mac
lastIdentTime src_host src_ip
lastScanTime signature status

# the triggered kill chain event

_time
2015-04-12 13:18:05

_raw
2015-04-12 13:18:05 host=HOST1 user=NULL semantic=Mail Sender Unique chain=Delivery
2015-04-12 13:19:35 host=HOST1 user=NULL semantic=Endpoint Risk Found chain=Exploit
2015-04-12 14:38:32 host=HOST1 user=USER1 semantic=Proxy Dest Unique chain=Exfiltrate

host            user
HOST1           USER1

lastAssetTime           ip          mac
2015-04-12 13:17:18     10.x.x.x    dd241dd368dd

lastIdentTime           src_host    src_ip
2015-04-12 13:19:22     SERVER99    10.x.x.99
2015-04-12 13:20:23     SERVER23    10.x.x.23

lastScanTime            signature               status
2015-04-09 16:36:27     Microsoft - Zero Day    Active
2015-04-09 16:36:27     Adobe Multiple Vulns    Active
```

Figure 7: Kill Chain combining Events, Contexts, and Semantics (Nafziger, 2014).

Brian Nafziger, brian @ nafziger.net