PROCEEDINGS OF SPIE

SPIEDigitalLibrary.org/conference-proceedings-of-spie

Realistic computer network simulation for network intrusion detection dataset generation

Garrett Payer

SPIE.

Realistic Computer Network Simulation for Network Intrusion Detection Dataset Generation

Garrett Payer*[a]
[a]ICF International, 7125 Thomas Edison Dr. #100, Columbia, MD 21046

ABSTRACT

The KDD-99 Cup dataset is dead. While it can continue to be used as a toy example, the age of this dataset makes it all but useless for intrusion detection research and data mining. Many of the attacks used within the dataset are obsolete and do not reflect the features important for intrusion detection in today's networks. Creating a new dataset encompassing a large cross section of the attacks found on the Internet today could be useful, but would eventually fall to the same problem as the KDD-99 Cup; its usefulness would diminish after a period of time.

To continue research into intrusion detection, the generation of new datasets needs to be as dynamic and as quick as the attacker. Simply examining existing network traffic and using domain experts such as intrusion analysts to label traffic is inefficient, expensive, and not scalable. The only viable methodology is simulation using technologies including virtualization, attack-toolsets such as Metasploit and Armitage, and sophisticated emulation of threat and user behavior.

Simulating actual user behavior and network intrusion events dynamically not only allows researchers to vary scenarios quickly, but enables online testing of intrusion detection mechanisms by interacting with data as it is generated. As new threat behaviors are identified, they can be added to the simulation to make quicker determinations as to the effectiveness of existing and ongoing network intrusion technology, methodology and models.

Keywords: Machine Learning, Cyber Security, Intrusion Detection.

INTRODUCTION

Intrusion detection and cyber security are difficult problems to solve. One of the major issues plaguing enterprise networks is the detection of attacks against their systems. During the 2000s, the use of signature based intrusion detection was sufficient in protecting the enterprise against most types of attacks (Sommer & Paxson, 2003). During this period, malicious software, or malware, was significantly less advanced and exploited insecure network services (Zou, Gong, & Towsley, 2002). These services were largely exploitable due to improper or nonexistent vulnerably assessment and remediation within organizations and the lack of security during the design lifecycle of software.

One could argue that the state of software vulnerability is not much improved and perhaps it is even worse than the early 2000s.However, this is not due the same or worse vulnerabilities in today's software, but the increased sophistication of the attacker's tactics. In the past, attackers could send a single packet to a network service and expect to remotely exploit the system and be granted administrative access (Porras, Saïdi, & Yegneswaran, 2009). Due to the emphasis of security during software development and hardware enforcement of security policies (Andersen & Abella, 2004), these types of vulnerabilities have become significantly less common. Today's attacker must chain a number of different vulnerabilities together to achieve the same goal. While significant increase in attacker capability would be expected in order to execute these attacks, once a set of vulnerabilities is exploited, it becomes largely trivial to code these actions for less skillful individuals or for automated processes such as those incorporated into tool kits such as Metasploit (http://www.metasploit.com/, n.d.) and crimeware (Binsalleeh, et al., 2010).

The use of more sophisticated attack methods such as chaining has made the detection of such attacks much more difficult. Additionally, attackers employ obfuscation techniques such as encrypted payloads (Porras, Saïdi, & Yegneswaran, 2009) and packers. These methods obscure the actual attack payloads making it difficult to determine their nature until after they have been executed. This makes static analysis techniques (Wagner & Dean, 2001) such as signature detection more like playing a game of whack a mole, where once you are able to detect one threat, a completely new undetectable threat emerges immediately after.. Worse yet is polymorphic malware that can change

Next-Generation Robotics II; and Machine Intelligence and Bio-inspired Computation: Theory and Applications IX,
edited by Dan Popa, Muthu B. J. Wijesundara, Misty Blowers, Proc. of SPIE Vol. 9494, 94940V

aspects about itself (Polychronakis, Anagnostakis, & Markatos, 2009) so that signatures would need to constantly be updated to detect additional variants.

INTRUSION DETECTION METHODS

The most popular systems for intrusion detection is Snort. Snort is an implementation of signature-based intrusion detection for network traffic. Its greatest strength, is also its greatest weakness. While many behavior or anomaly detection techniques can lead to significant amounts of false positives (Patcha & Park, 2007), signatures defined for use by Snort can be as general or as specific as needed to detect packets of malicious intent. If these signatures are too broad, they too can lead to excessive false positives. The use of more specific signatures, looking for key features within network packets at network speeds, is very effective at detecting malicious traffic. If an attack or piece of malware uses a specific string or payload in order when compromising a network service to gain unauthorized access to the machine, network packets can be searched to determine if they too have this string. However, the lack of generalization in this approach can lead to similar attacks with perhaps as little as a single bit flipped, traversing networks completely undetected.

Considering the complexity of today's malware and the weaknesses of signature-based detection in identifying variants, extensive research has been focused on anomaly and behavior based detection mechanisms (Patcha & Park, 2007). With proper packing and encryption, it becomes much more difficult to simply observe a string of bytes and determine that it is of malicious intent. To make a more accurate determination, detection mechanisms must observe the state of machine and its behaviors. This can include not only network communication attributes such as destination and source IP addresses and ports, but also provide additional context. This would include host based information indicating the state of the system, including the actions taken upon it not directly correlated to observable network traffic (Yeung & Ding, 2003).

The major issue with using anomaly or behavior based protection is that instead of missing malicious activity, it becomes the false identification of benign behavior as malicious. Most anomaly detection results in significantly higher false positive rates (Patcha & Park, 2007). Investigations are no longer about finding new variants or unique malware but instead focus on properly modeling the system's behavior so as to limit the amount of false positives generated.

INTRUSION DETECTION DATASETS

There has been extensive research in the area of using Machine Learning (ML) in order to build these behavior models (Patcha & Park, 2007). One major issue when exploring the use of ML for cyber security is that there is a lack of standardized datasets. For years, the KDD'99 dataset had been the de facto standard when comparing performance of ML techniques (Kayacik, Zincir-Heywood, & Heywood, 2005). However, it is not viable for determining the effectiveness of techniques in regards to today's threats.

The KDD'99 labeled dataset was generated as a part of the DARPA Intrusion Evaluation Program and was derived almost 20 years ago. A large majority of the labels, attacks, and observable behaviors found within the dataset are completely out of date and are not a reflection of the types of behavior now found on most enterprise environments (Tavallaee, Bagheri, Lu, & Ghorbani, 2009). While the dataset can be used as a toy example for researchers to use in evaluation of their techniques, any models generated from this data would be absolutely useless in today's environment.

With the KDD'99 dataset being unsuitable for determining whether ML techniques are applicable for intrusion detection or cyber security in general, the next logical step is to identify other usable datasets. There are a number of different datasets available for download (Publicly available PCAP files, n.d.). As an example, the U.S. National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) (National CyberWatch Mid-Atlantic CCDC (MACCDC) , n.d.) releases the network traffic generated during their event as a set of Libpcap (TCPDUMP & LibPCAP, n.d.) packet captures (PCAP). These PCAP files can be downloaded for use in research.

For competition data such as the MACCDC, a number of major issues present themselves when utilizing the data for research: the data is unlabeled, it's heavily skewed as the majority of the traffic will be malicious in nature, and its relevance continues to wane with the passage of time. Without labeled data, it's difficult or impossible to tell what is benign or malicious, let alone assign more granular labels such as the type of attack. An expert could derive labels by examining the information. However, this can be impractical as there could be thousands of different sessions and

millions of packets to sort through. Additionally, without understanding the network in use, the assets being targeted, and the behavior of those systems in the presence of legitimate network traffic, the labels applied by expert may still be wrong.

Another method available to researchers to label data such as those coming from competitions is to simply run the PCAP file through existing signature based detection systems such as Snort, using a comprehensive set of signatures (Roesch, 1999). The problem with this approach is that the performance of the ML techniques is now tied to the tools providing the labels. The techniques will only be as good as the signature detection tool, rather than out performing it.

To adjust for the skew towards more malicious traffic in datasets similar to MACCDC, normal network communication on a campus or enterprise network could be captured and combined or injected into the PCAP containing the malicious network traffic. In this way, a researcher can introduce, at least what they believe to be known benign traffic with known malicious traffic. However, this approach can lead to additional problems for classification. Because the traffic is from two completely different networks, artifacts will be introduced. As an example, none of the benign traffic would share the same source and destination as the malicious traffic. In this example, a ML model may learn this artifact rather than the features necessary to identify malicious behavior. A similar issue was found in the KDD'99 dataset packets, malicious traffic had a Time to Live (TTL) of 126 or 253 while benign traffic used 127 or 254 (Tavallaee, Bagheri, Lu, & Ghorbani, 2009). It would be possible for ML techniques to weigh these features heavily, but would be completely useless on real world networks.

SIMULATION

Training ML classifiers requires close to real world, labeled datasets. Without this, ML techniques will continue to look great within in lab conditions, but perform much worse when used on real networks. Another method to generate labeled datasets is to perform a simulation. This can be achieved using several physical systems, installing and configuring all of the software, and then running a number of attacks against those systems. The data can be captured, and researchers would be reasonably certain as to which traffic is malicious or benign. However, this approach can be cumbersome as the systems would need to be reconfigured each time before an attack. This can be remedied by utilizing virtualization (Adams & Agesen, 2006), setting up virtual machines (VMs) rather than using physical systems. By utilizing virtualization, VMs can be snapshotted before attacks so that the state before experimentation can be captured and reapplied for different experiments. Depending on the complexity of the simulation, VMs will provide many roles such as a web server, database server, etc. and one or two VMs are designated to launch attacks using different penetration tools. The biggest advantage of this method is that the ground truth of what goes on within the simulation is known or at least predictable. When performing the simulation, any number of different attack techniques and exploits can be utilized. Because the nature of the attack, the targets, and when it occurs are monitored, the malicious sessions can be identified and a labeled as such.

An additional benefit to performing simulations is that they can be limited to exactly what needs to be studied as opposed to containing much more superfluous data. However, the biggest drawback to this process is that because of a researcher's limited time, experience, or resources, researchers will likely only create very small scale simulations. This smaller scale will often utilize simple user behavior models and simplified models of system interaction.

Due to the limited scale and the simplicity of these simulations, they do not properly model much of the behavior that actually takes place on a live network, even on smaller deployments such as branch offices. The methods being utilized by the researcher may succeed in in the face of limited simulation but completely fall apart when exposed to the complexity of a live network with many people and systems.

Further, to create highly relevant scenarios significant expertise is necessary; depending on the complexity, the expertise of a penetration tester may necessary to realize the model. Furthermore, tweaks to the experiment may require a few minutes to several days to effect the changes. When such major changes need to occur, the repeatability for previous experiments may be difficult or at least require the same amount of time to revert to a previous state.

SOFTWARE CONFIGURABLE SIMULATION

Experimentation should be a repeatable process that enables result verification and allows for minute changes in experimental design in order to test current and emerging scientific hypotheses. When performing cyber security

research, the same requirements are necessary to fully understand the effects of specific capabilities and observed behaviors. When exploring ML techniques for use in cyber security, it is important to be able to quickly adjust or change, and repeat simulations as easy as it is to tweak software configuration to explore as the problem space quickly. Therefore, we have been developing the Framework for Instantiation and Remote Execution of Lightweight Modular Platforms (FIRELAMP).

FIRELAMP is currently being developed to solve many of the issues described in earlier sections. Once FIRELAMP is out of early development, the system will be able to simulate a cyber-offensive tactical scenario: a network topology is created, a number of actors, either benign or malicious, will perform actions causing a number of events to occur. Each event will leave behind related and derivative network emissions such as application logs, network packet captures, and other telemetry which can be collected over the lifetime of the scenario. The scenario will end when one or more goals are complete. By having a record of every action that took place, the ground truth, it becomes possible to attribute an action or set of actions to a specific set of emissions and thus provide labeled datasets for further analysis.

FIRELAMP will implement a Scenario Definition in order to allow researchers to define how a scenario should be configured and executed. In order to facilitate human and computer readability of this information, we will utilize the Extensible Markup Language (XML) (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 1998) to express scenario information. The definition itself will consist of several major parts discussed below.

Topology Configuration

In this section, a scenario's various networks can be defined. OSI Layer 2 and Layer 3 aspects can be defined as well including a networks subnet, mask, default gateway, etc. This information will be used within the scenario not only for network configuration, but to provide additional configuration information to assets assigned to the scenario. For example, a host on a specific network can automatically be configured with the correct subnet defined by the network configuration. Figure 1 illustrates how a scenario definition file can be used to construct a simulation with a number of networks and VMs attached.

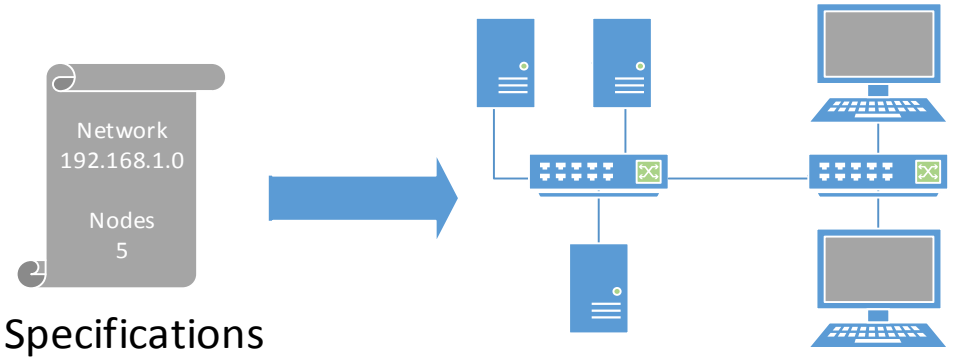

Figure 1. Network and Host configuration automatically generates a network topology and needed hosts for the experiment.

The current network definition includes an arbitrary network ID, a name, and the IP information including address, subnet and gateway. This information is used by the current FIRELAMP iteration to create virtual networks using OpenvSwitch (Pfaff, et al., 2009).The network name is referenced within the host definition in order to assign virtual networks to specific interfaces. OpenvSwitch provides the networking functionality including the ability to create virtual mirror ports that a sensor can tap in order to view all network traffic traversing a virtual switch. Configuration information is passed from the software via OpenvSwitch network service.

In addition to configuring the network, the hosts also need to be defined. The software utilizes the Linux Kernel Virtual Machine (KVM) (Kivity, Kamay, Laor, Lublin, & Liguori, 2007) in order to define and run VMs for the simulation. The configuration information is provided via the scenario definition to configure the VMs that will be used during the scenario. Configuration information can include physical aspects such as assigned memory or the number of CPUs. Additionally, VM configurations can be explicit such as setting a specific IP address on an interface, or allowing automatic selection based on the assigned network. In Figure 2, we have an example of a definition file that currently builds a network.

To facilitate the use of multiple similar virtual machines within a simulation, rather create a virtual machine for each system to participate within the simulation, a number of virtual machine templates were created. These templates contain their representative operation systems and software needed to run the host based processes. However, at simulation run time, rather than run the templates, they are cloned, with the disk using qcow2 (Sin & Wong, 2007) virtual disks so that during the simulations, VMs will only save changes against the template baseline, rather than make a complete copy of the template itself. This significantly cuts down on setup time and allows for the easy destruction of the virtual machines after the simulation is performed. Every time a simulation is run, all VMs are created fresh so that the same starting state is exactly the same defined by the scenario definition. Figure 2 shows a sample scenario definition file.

```
<scenario>
 <host_template id="1">
  <name>winxp-template</name>
  <image>winxp-template.qcow2</image>
  <type>winxp</type>
 </host_template>
 <host_template id="2">
  <name>kali-template</name>
  <image>kali-template.qcow2</image>
  <type>debian</type>
 </host_template>
 <host_template id="3">
  <name>metasploitable-template</name>
  <image>metasploitable-template.qcow2</image>
  <type>debian</type>
 </host_template>
 <network id= "1">
  <name>central</name>
  <subnet type= "static">192.168.200.0</subnet>
  <mask type= "static">255.255.255.0</mask>
  <gateway>192.168.200.1</gateway>
 </network>
 <network id= "2">
  <name>management</name>
  <subnet type= "static">10.44.22.0</subnet>
  <mask type= "static">255.255.255.0</mask>
  <gateway>10.44.22.1</gateway>
 </network>
 <host id= "1">
  <name>victim1</name>
  <int id= "1">
   <IP>192.168.200.11</IP>
   <network-name>central</network-name>
  </int>
  <cpu-count>1</cpu-count>
  <template>winxp-template</template>
  <mem>512</mem>
 </host>
<host id= "2”>
  <name>attacker</name>
  <int id= "1">
   <IP>192.168.200.100</IP>
   <network-name>central</network-name>
  </int>
  <cpu-count>1</cpu-count>
  <template>kali-template</template>
```

Figure 2. Scenario definition of hosts, host templates, and networks.

In the definition file, each VM template is defined, which includes its virtual disk, the template name, and the type of VM. The type is needed in order to determine how configuration information from the VM is to be applied to the template. From there, each network is defined, including its network address, subnet, and gateway. Finally, each virtual host is defined by referencing a template, and then its specific configuration characteristics.

Attacker Behavior Configuration –

In order for the simulation to generate malicious behavior and network traffic, a simulated attacker will be introduced. The standard way of defining the actions to be taken by a hacker is through the use of an attack graph. An attack graph defines the set of capabilities that an attacker has, as well as the probabilities of executing specific actions. This can include directives based on discovered information or actions to take once a system is accessible via a reverse shell. Figure 3 shows how the attacker is introduced to the simulation. The attacker script is uploaded to a specific VM where the script is executed.

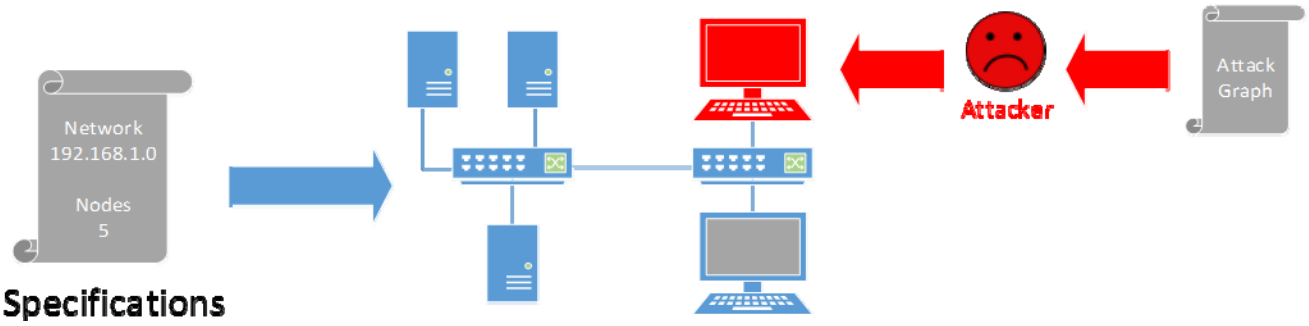

Figure 3. Shows the attacker being introduced to a virtual machine with an attack graph indicating the behavior he will follows for the simulation.

The current version of the software doesn't support coding attacker behavior as attack graphs, but as a separate XML definition. Specific actions and conditions are defined and an attack script using Armitage's Cortana scripting engine is generated.

```
<campaign>
 <action id="1">
  <name>SubnetScan</name>
  <subnet>192.168.200.0</subnet>
  <mask>27</mask>
  <desc>A scan of the subnet 192.168.200.0/27 will be performed.</desc>
 </action>
 <action id="2">
  <name>exploit</name>
  <exploit>exploit/windows/smb/ms08_067_netapi</exploit>
  <payload>windows/meterpreter/bind_tcp</payload>
  <desc>compromise engagaged...</desc>
 </action>
 <event id="1">
  <name>on_start</name>
  <action>1</action>
 </event>
 <event id="2">
  <name>on_hosts</name>
  <os>Microsoft Windows</os>
  <action>2</action>
 </event>
</campaign>
```

Figure 4. The definition of an attack campaign to generate an attack script for a simulation.

The definition calls for defining actions and events. Since the definition is almost a 1 to 1 mapping to the Cortana Scripting language, the events defined correspond to definable events in a Cortana script. The actions are subroutines defined within Cortana that will execute an action. Those subroutines are then associated to specific events. Figure 5 shows the corresponding Cortana code used a template to generate an attack script.

```
sub {{ name }} {
      println("{{ desc }}");
      cmd($console, "db_nmap -O --osscan-guess {{ subnet }}/{{ mask }}");
}
sub {{ name }} {
      $host = $1;
      $newConsole = console();
      println("$host {{ desc }}");
      cmd($newConsole, "use {{ exploit }}");
      cmd_set($newConsole, %(RHOST => $host, PAYLOAD => "{{ payload }}"));
      cmd($newConsole, "exploit -j");
}
on ready {
  $console = console();
  println("Campaign started.");
  {{ action }}()
}
on hosts {
      println("{{ desc }}");
      %hostDB = hosts();
      foreach $host (keys(%hostDB)) {
         if (%hostDB[$host]["exploit_attempt_count"] < 1 && %hostDB[$host]["os_name"] eq
"{{ os }}") {
            {{ action }}($host)
         }
      }
}
```

Figure 5. The corresponding templates to generate the code from the attack definition

When using the above definition, the script that gets generated performs a scan of the defined subnet. If it discovers any host running “Microsoft Windows” it initiates a specific attacking exploiting the MS08-067 vulnerability

User Behavior Configuration –

Similar to how the attacker is introduced to the simulation, a user behavior script is generated based on a predetermine definition of behavior. This script is then run on the assigned VM. The purpose of introducing users to the simulation allows for the generation of benign or normal behaviors to exist on the simulated systems. It’s difficult and impossible to model the systems operation if nothing is operating it. Figure 6 shows how the user behavior is introduced to the simulation via a script to be run on an assigned VM.

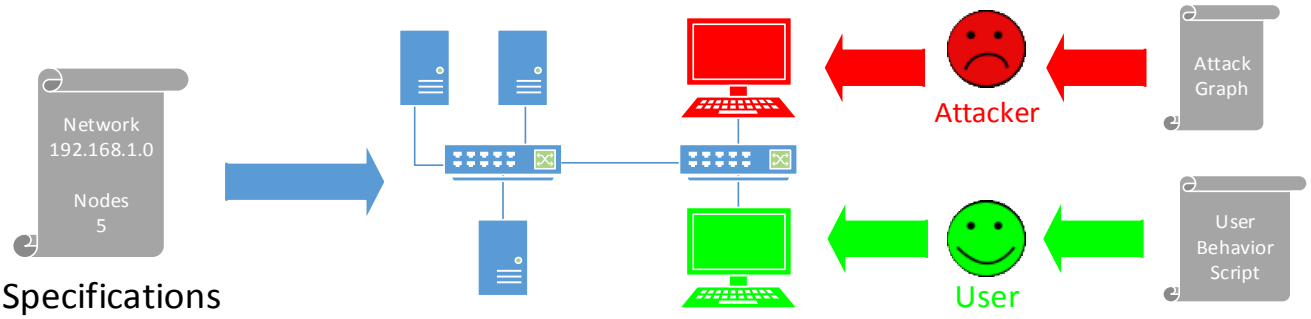

Figure 6. Shows the user being introduced to a virtual machine with a user behavior script guiding the action the user is to take during the simulation.

The current version of the software does not currently support user behavior scripts yet. No definition of user behaviors currently exists and this will be left to future work.

Data Capture Configuration

Performing evaluations on cyber capabilities in various scenarios indelibly leads to significant amount of emission information. This information will need to be analyzed using automated processes as well as through human interaction. A simulation system will need to automatically process and store cyber scenario telemetry in a storage location and initiate automated analysis. The information can include raw PCAP from the simulation, event logs from the participating virtual machines, as well use both the user and attacker behavior logs. As Figure 7 shows, after the scenario is complete, the various telemetry information generated by the simulation is collected into the data storage for the software. PCAP is collected via a Sensor VM attached to the mirror port on the virtual switch configured through OpenvSwitch.

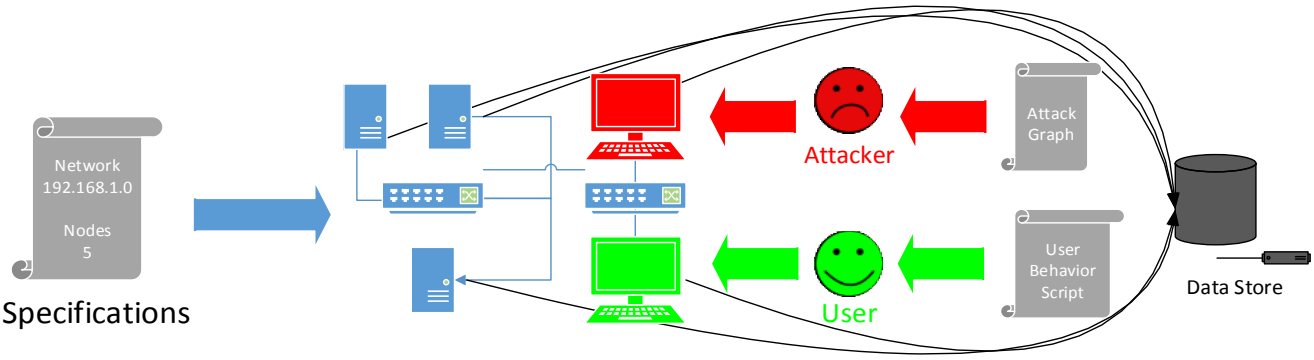

Figure 7. Telemetry data I pulled into the data store for analysis after the simulation has ended.

The information produced by the simulation is saved in the data store in order to persist after the scenario has ended. This data store is currently configured as a standalone instances HDFS (Borthakur, 2008) along with integration with Apache Spark (Zaharia, Chowdhury, Franklin, Shenker, & Stoica, 2010). Data capture is currently defined as a number operations within the scenario definition file. Figure 8 shows an example of how operations are defined.

```
<operation name="sensor-download">
 <action id="1">
  <command>download</command>
  <param1>/home/sensor/sensor.pcap</param1>
  <param2>{$dest}/{$host}.pcap</param2>
 </action>
 <action id="2">
  <command>tar-out</command>
  <param1>/var/log</param1>
  <param2>{$dest}/log-{$host}.tar</param2>
 </action>
 <action id="3">
  <command>untar</command>
  <param1>{$dest}/log-{$host}.tar</param1>
  <param2>{$dest}/log-{$host}</param2>
 </action>
</operation>
<operation name="attacker-upload">
 <action id="1">
  <command>upload</command>
  <param1>attack.cna</param1>
  <param2>/root/attack.cna</param2>
 </action>
</operation>
<operation name="attacker-download">
 <action id="1">
  <command>download</command>
  <param1>/root/attacker.pcap</param1>
  <param2>{$dest}/{$host}.pcap</param2>
 </action>
 <action id="2">
  <command>tar-out</command>
  <param1>/var/log</param1>
  <param2>{$dest}/log-{$host}.tar</param2>
 </action>
</operation>
<operation name="winxp-download">
 <action id="1">
  <command>tar-out</command>
  <param1>/WINDOWS/system32/config</param1>
  <param2>{$dest}/log-{$host}.tar</param2>
```

Figure 8. Operations defined within the scenario definition file in order to scrap telemetry information from the simulation.

Rather than define every specific action needed to upload and download information from the simulation, the scenario definition can allows for operations to be defined as set of actions. A set of actions are needed to carry out an operation such as gathering the event longs from a Linux system. Each operation can be assigned to a pre and post operations queue on each defined VM. Operations in the pre operations queue are executed before the VM is started, while the post operations queue is executed after the simulation has ended and the VM is turned off.

Operations can also be defined for the scenario itself. As an example, the operations in the post operation queue for the scenario are executed after all of the VM's post operation queue have been executed and the VMs have been destroyed. Actions can include the use of outside commands, so our data labeler program gets run as post operation for the scenario and ingests the log and PCAP information pulled from the VMs in the simulation. Currently, the labeling on supports PCAP and its limited to labeling based on source and destination IP addresses.

MACHINE LEARNING FOR INTRUSION DETECTION

After the simulation has ended, the information pulled from the scenario is stored within the data store and labeled. The end goal of FIRELAMP is to provide a relevant data set, formatted for use in experimenting with Machine Learning

techniques. The data sets will contain the raw data as well as aggregate information such as flow data, all of which will be labeled to indicate their malicious or benign intent and eventually the granular details such as whether a specific entry in a log can be attributed to an attacker pivoting on a victim machine.

Researchers would be able to vary the scenarios not by creating all new simulations with different VMs, but by through subtle changes within the scenario definition. With the introduction of configurable attack graphs and user behavior maps, simulations can contain near realistic representations of behaviors found within enterprise environments. This will increase the confidence in the ML techniques being tested and evaluated as to their sufficiency in live environments. Figure 9 shows the ML would operate directly on the data within the data store.

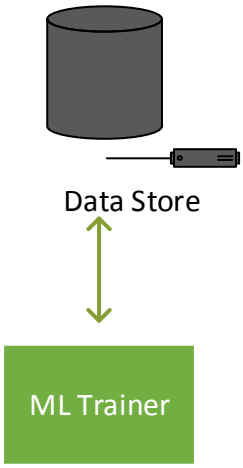

Figure 9. Telemetry stored in the data store can be used to train/test the Machine Learning algorithms.

In order to test techniques within the simulation, this can be achieved by installing the detection software within template, and assign one or more VMs to use this template within in the scenario. Figure 10 shows an example of how an ML technique can learn from the data generated from previous simulations, and use this information to interact with a new simulation based on the previous scenario definition.

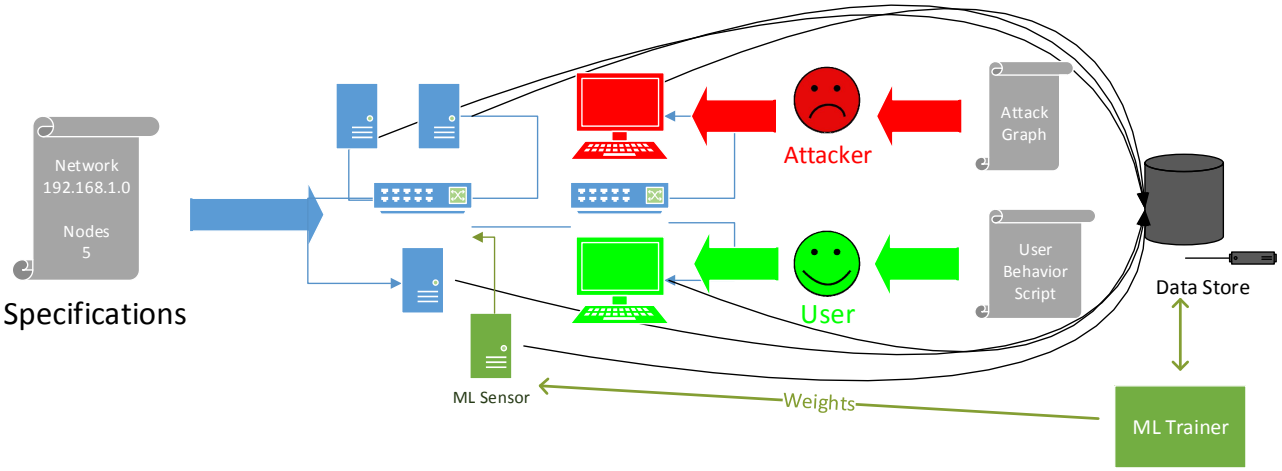

Figure 10.The state from the ML algorithm after learning from previous data used to create a sensor within in a similar scenario.

In this scenario, a new sensor template is created and loaded with the sensor software that will be utilizing the specific ML algorithm designed to detect specific behaviors within this scenario. Data collected from previous simulations has

been used to train the ML algorithm. The weights or state is then transferred via an operation to the sensor. The log data generated by the sensor software can then be collected at the end of the simulation and itself analyzed.

FUTURE WORK

The current version of software in the FIRELAMP project is still very much a prototype. For this system to be useful for future research into cyber security, the scope of the software will need to expand to encompass greater and more complicated simulations. This can be accomplished by pursuing the following:

Introduction of User Behavior Scripts – User behavior simulation needs to be added in order generate benign traffic. Not only doe scripts need to be introduced, but a mechanism developed in order to define and execute behavior that you would find that normal humans operating on the simulated systems would perform

Increase the complexity attacker capabilities - .Currently, the software only performs scans and executes exploits. Additional capabilities typically found within attack graphs need to be explicitly defined. Furthermore, campaigns that exist outside of Cortana scripts will need to be added as well.

Robust labeling mechanism – Considering that the current supported simulations only utilize an attacker from a single IP, labeling currently involves identifying the IP belonging to the attacker. Labeling will need to be augmented to support not only the attribution of specific actions taken by the attacker, but also simulated users. In addition, event logs and other host based telemetry will need to be labeled and attributed as well.

CONCLUSION

Signature-based intrusion detection has been relegated to whack a mole in identifying new malware. Not only are methods used by attackers significantly more complicated, but intrusion detection evasion techniques are also significantly more sophisticated. Machine Learning for behavior and anomaly detection can offer a possible solution, but research in this area is hindered by the lack of data.

The KDD'99 data set should not be used in order to evaluate the effectiveness of ML approaches to cyber security. Not only is the dataset out of date, but many of the attacks are completely irrelevant to cyber security today. While additional datasets are available, they are typically unbalanced and unlabeled, making supervised learning techniques difficult to utilize. Researchers are either required to label the data themselves or utilize existing tools to identify threats within the data sets. This can lead to inconsistency in how this information is labeled. Additionally, these datasets become obsolete as well.

Simulation is most likely the best option when attempting to train machine learning classifiers for intrusion detection. Simulations allow researchers to know the ground truth about what actions are actually occurring. Knowing the truth would allow for accurate labeling of data. However, this process can be a very complex and tedious process prone to error.

The FIRELAMP project seeks to simply simulation to the point of simply changing a configuration file can allow for drastic changes within a scenario. Additionally, the project would allow for the automatic collection of data and labeling as well as simulated user and attacker behavior. The current version of the software currently meets some of these goals. More will be needed before the needed level of complexity is achieved in simulations in order to thoroughly vet ML techniques for use in live environments.

REFERENCES

Adams, K., & Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices* (pp. 2-13). ACM.

Andersen, S., & Abella, V. (2004, September). *Changes to functionality in microsoft windows xp service pack 2, part 3: Memory protection technologies, Data Execution Prevention.*

Retrieved from Microsoft TechNet Library: https://technet.microsoft.com/en-us/library/bb457151.aspx

Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Debbabi, M., & Wang, L. (2010). On the analysis of the zeus botnet crimeware toolkit. *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)* (pp. 31-38). IEEE.

Borthakur, D. (2008). *HDFS architecture guide.* Retrieved from Hadoop Apache Project: http://pristinespringsangus.com/hadoop/docs/hdfs_design.pdf

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (1998). *Extensible markup language (XML).* Retrieved from World Wide Web Consortium: http://www. w3. org/TR/1998/REC-xml-19980210

http://www.metasploit.com/. (n.d.). (Rapid7) Retrieved April 3, 2015, from http://www.metasploit.com/

Kayacik, G. H., Zincir-Heywood, A. N., & Heywood, M. I. (2005). Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets. *Proceedings of the third annual conference on privacy, security and trust.*

Kivity, A., Kamay, Y., Laor, D., Lublin, U., & Liguori, A. (2007). kvm: the Linux virtual machine monitor. *Proceedings of the Linux Symposium*, (pp. 225-230).

National CyberWatch Mid-Atlantic CCDC (MACCDC) . (n.d.). Retrieved from http://maccdc.org/

Patcha, A., & Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks, 51*(12), 3448-3470.

Pfaff, B., Pettit, J., Amidon, K., Casado, M., Koponen, T., & Shenker, S. (2009, October). Extending Networking into the Virtualization Layer. *Hotnets.* New York, NY: ACM SIGCOMM.

Polychronakis, M., Anagnostakis, K. G., & Markatos, E. P. (2009). An empirical study of real-world polymorphic code injection attacks. *Proceedings of the 2nd USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET).*

Porras, P., Saïdi, H., & Yegneswaran, V. (2009). A foray into Conficker's logic and rendezvous points. *USENIX Workshop on Large-Scale Exploits and Emergent Threats.*

Publicly available PCAP files. (n.d.). Retrieved from Netresec: http://www.netresec.com/?page=PcapFiles

Roesch, M. (1999). Snort: Lightweight Intrusion Detection for Networks. *LISA*, (pp. 229-238).

Sin, C., & Wong, D. (2007). Image baby image!: making pc cloning more efficient. *5th annual ACM SIGUCCS fall conference* (pp. 314-317). New York, NY: Association for Computing Machinery.

Sommer, R., & Paxson, V. (2003). Enhancing byte-level network intrusion detection signatures with context. *Proceedings of the 10th ACM conference on Computer and communications security* (pp. 262-271). ACM.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009.* IEEE.

TCPDUMP & LibPCAP. (n.d.). Retrieved from tcpdump: http://www.tcpdump.org/

Wagner, D., & Dean, D. (2001). Intrusion detection via static analysis. *Proceedings. 2001 IEEE Symposium on Security and Privacy* (pp. 156-168). IEEE.

Yeung, D.-Y., & Ding, Y. (2003). Host-based intrusion detection using dynamic and static behavioral models. *Pattern recognition*, 229-243.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: cluster computing with working sets. *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (pp. 10-10). USENIX .

Zou, C. C., Gong, W., & Towsley, D. (2002). Code red worm propagation modeling and analysis. *Proceedings of the 9th ACM conference on Computer and communications security* (pp. 138-147). ACM.