A Platform for Evaluator-Centric Cybersecurity Training and Data Acquisition

Jaime C. Acosta*, Joshua McKee*, Alexander Fielder* and Salamah Salamah†
*U.S. Army Research Laboratory, White Sands Missile Range, NM 88002
†University of Texas at El Paso, El Paso, TX 79968

***Abstract*—Empirical-based models for security technologies in the commercial and military domain, including those that focus on protection, detection, and broader risk analysis, leverage data captured from sensors on network-connected devices including gateways, routers, and host nodes. Lacking, however, are datasets that contain specific state observations and actions from the evaluator (red/blue teammer) workstation; we call this the *inside*-view. This is largely due to issues associated with data ownership, data classification, and the lack of integrated evaluator-centric data-collection mechanisms.**

To enable and promote creation of open datasets that capture the inside-view, we introduce a scalable platform that consists of two main elements. First, the emulation sandbox, or EmuBox, is an open-source and portable (i.e., it can execute on a laptop) solution for creating small- to medium-sized heterogeneous scenarios for evaluators to set up practice environments and competitions and to hone their skills. Second, the evaluator-centric and extensible logger, ECEL, is a centralized management system that uses plugins for capturing and formatting evaluator data. We conclude the paper by providing a case study to demonstrate the setup and configuration of the platform along with a performance analysis.

I. Introduction

In the field of cybersecurity, data are a critical, yet limited, commodity that drive the advancement of intrusion detection and prevention systems, anti-virus software, and modeling and simulation, among others. Much focus has been on the *outside*-view of the system. This includes signature, rule, or anomaly-based intrusion alerts, incident reports, and network captures. The AFDA Intrusion Detection Dataset [1], NetReSec [2], and the DEFCON capture-the-flag event [3] network captures are examples of widely used resources. While there is still room for automated analysis on these datasets, including labeling of ground truth and benign and malicious events, these datasets are growing in number and are accessible to the scientific community.

These outside-view datasets provide a black-box view of network and host phenomena. Lacking is a way to collect the inside- view; i.e., the observed state and actions executed on a particular host by a particular evaluator; this also applicable to operators and defenders. By filling this gap, we hope to enable accurate and efficient associations from protection and detection alerts to particular user actions. This can then lead to human skill level and motivation characterization which can be used to improve defense tactics as well as security evaluations.

We have designed and implemented a platform that facilitates scenario-development and data acquisition from the evaluator perspective.

In this paper, we provide the following contributions:

1) EmuBox, an open-source, lightweight cybersecurity testbed[1] that facilitates the creation of heterogeneous scenarios (e.g., combined tactical/strategic) that serve simultaneous, isolated, training environments.
2) ECEL, an open-source, and extensible software package[2] that enables centralized data acquisition and management using custom or off-the-shelf collection tools.
3) A use case describing the platform setup and performance metrics.

II. Related Work

The rapidly evolving nature of technology requires cybersecurity analysts to undergo extensive training in order to stay current with emerging tools, techniques, and strategies. This training consists of detecting and exploiting vulnerabilities on networks, hosts, and applications using public and in-house testing tools. Data-collection mechanisms typically use tools to log events on machines that are part of the test environment.

A. Training

With respect to training, there are several commercial options; security companies such as InfoSec and SANS offer educational courses, bootcamps, and certifications. Free resources include Internet sites such as *hack.me* and *vulnhub.com* that provide intentionally vulnerable virtual machines (VMs) that can be set up using VirtualBox, VMWare, and others. These sites often include walkthroughs that document the specific steps that are required to test the vulnerable VMs. At a grander scope, cybersecurity capture-the-flag (CTF) is a widely known competitive affair where participants must complete a set of tasks to gain points. DEFCON [3] has hosted yearly CTF events for over 20 years. After each event, the tools, data, write-ups, and source code for the challenges and CTF engine are released to the public. The iCTF [4] has held CTF events at an international scope over the past 12 years. Participants must maintain a secure environment while also attempting to compromise opponent systems. The iCTF framework is free and open-source. Similarly, Facebook released an open-source platform [5] that provides users with a graphical interface featuring a scoreboard display with progress and task descriptions. Facebook provides several pre-built challenges and also allows customization.

[1]Code available at https://github.com/ARL-UTEP-OC/emubox
[2]Code available at https://github.com/ARL-UTEP-OC/ecel/

978-1-5386-0595-0/17/$31.00 ©2017 IEEE

B. Scenario Development

A small set of scenario development tools support mixed tactical and strategic networks and use emulation to allow execution of real software binaries. The Common Open Research Emulator (CORE) [6] is capable of emulating hosts and network devices at the network layer and above. CORE provides a graphical user interface (GUI) which allows for trivial creation of network nodes which are capable of executing various applications, services, and network protocols. CORE supports hardware-in-the-loop (HIL) as well as connections to other CORE instances and live networks. The independent nature of CORE network instances allows for integration and portability among various platforms and systems. In regard to the assessment and defense community, these CORE scenarios can be used to enhance training and data analysis.

The Applied Communication Sciences (ACS) and the U.S. Army Research Laboratory (ARL) developed the Cyber Virtual Ad-hoc Network (CyberVAN) to provide larger-scale modeling and network testing [7]. CyberVAN uses a hybrid approach allowing arbitrary physical endpoint devices to communicate through a simulated network. This approach targets discrepancies that sometimes arise when relying solely on virtualization and emulation.

Large-scale network scenarios present additional challenges that require specialized infrastructures and expertise. Configuration and maintenance are critical for representing these environments. The National Cyber Range (NCR) [8] is a government-funded effort to address current needs for cybersecurity testing, evaluation, and training processes. The NCR is capable of virtualizing complex network topologies of up to 40,000 nodes. The range is capable of hosting scenarios of different classification levels and is able to restore all systems to a clean state after testing and scenario execution is completed.

C. Data Acquisition

Event loggers are widespread and exist to provide audit trails, backtracking, accountability, intrusion detection, and analysis, among others. As examples, Microsoft provides an API for the Windows Event Log, the Windows Event Tracing, and also a suite of tools to collect and view these data [9]. Linux and Mac OS have similar mechanisms with, for example, syslog, logger, Snoopy [10]. Wireshark, tshark, and tcpdump are among some widely used sniffers for collecting network traffic. Analysis engines such as the Bro, Snort, OSSEC, and HBSS Intrusion Detection Systems (IDS) monitor data and issue alerts for potentially malicious activities [11].

In summary, training systems mainly focus on the competitive aspect (score keeping) or do not holistically provide a tactical element, the ability to host simultaneous scenarios on portable systems, and collect evaluator-centric data. Also lacking is a system to integrate the various existing data-collection tools and facilitate data management and analysis.

III. Platform Overview

To address the gaps described in the previous section, we designed a platform by focusing on the following training and data acquisition goals.

A. Goals and Objectives

First, for training purposes and portability, the platform needs to support the execution of multiple scenarios simultaneously on a single, low to medium-end, system, such as a standard laptop or desktop computer. Participants should have an environment (VMs, traffic, tools, etc.) that is completely isolated from the other scenario environments. The environment must only have relevant processing and networking artifacts in the scenario (e.g., if remote desktop is used to connect to the scenario, this traffic should be inhibited from the evaluator's environment). This will facilitate analysis when studying the effects of evaluator actions on the scenario environment. The platform must support tactical components, e.g., a mobile ad-hoc network (MANET) alongside strategic components and should also allow mixed physical and virtual devices.

Second, the platform must allow participants to connect to a pre-configured "evaluator" machine using remote desktop software. The evaluator machine will be connected to the scenario network and participants will not have to modify their own physical machine settings. This will also facilitate the setup and data acquisition as the "evaluator" VM will have all necessary collection tools pre-installed. The platform should also allow users to connect their own machines to the scenario using a virtual private network (VPN). In this setup, participants may use their own custom tools in the scenario.

Lastly, the platform must include a robust and extensible data acquisition mechanism to record environmental states and evaluator actions when connected to the scenario. The collected data can be used to analyze techniques that are used during assessments and then to harden systems. Additionally, researchers will have a knowledge base on which to develop skill-based attacker models, decision support, and intelligent agents.

B. Platform High-Level Design

The platform consists of two key components: the EmuBox for creating scenarios and the ECEL for data acquisition as shown in Figure 1. Together EmuBox and ECEL enable the comprehensive collection of inside- and outside-view of scenario phenomena.

IV. EmuBox

EmuBox is written in Python and is cross-platform. It has been tested on Kali Linux 2016.1 and 2016.2 (32 and 64 bit). EmuBox leverages VirtualBox and CORE to support mixed virtual/physical systems, virtual remote desktop connection (VRDP), and heterogeneous (e.g., mixed MANET and wired) networks. The other two main components are the *Workshop Creator* and the *Workshop Manager*.

These components process scenario VMs as *Workshop Units* and *Workshop Groups*. Workshop units contain the set of

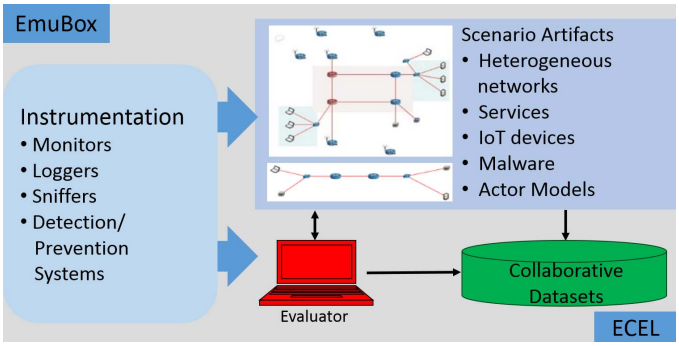

Fig. 1. Platform Architecture

VMs that make up a single scenario. At least one of these VMs must have the virtual remote desktop protocol (VRDP) enabled (this is a feature of the VirtualBox extensions pack). These machines use ECEL to collect evaluator-centric data. VirtualBox handles the VRDP so that the traffic associated with the remote desktop connection is not visible within the VM.

As part of the unit, other machines may be used to compose the scenario. We use VM with CORE to create the network topology. The topology may consist of Linux containers, Docker containers, and other VMs or external hardware through the HIL capability. Vulnerable systems, scripted actors (e.g., operators/defenders), and instrumentations may also be incorporated into the scenarios.

Network isolation is implemented using the VirtualBox internal network adapters. After a workshop unit is configured, the machines are started and a snapshot is taken as needed for the scenario. For example, the snapshot may be taken after a user logs in and starts the sshd service or after all routes converge in the network topology. This unit is then cloned to provide multiple simultaneous and isolated scenarios. The Workshop Creator automates the cloning process. During the cloning process, this component adjusts VRDP ports and internal network adapter names so that each group is isolated and uniquely accessible by participants.

The Workshop Manager component of EmuBox is a multi-threaded process that monitors VRDP connections for each workshop unit. It also contains a web service with a simple front-end that is implemented using the Flask micro web development framework. When participants navigate to the front-end they are shown the VRDP-enabled workshop units (those that are available and not currently in use). The front-end also provides participants with a unique connection string (IP address and VRDP port pair) to use in a remote desktop client, such as MS-RDP on Windows, Mac OS, iOS, and rdesktop on Linux (see Figure 2).

When a participant connects to a unit, it becomes unavailable and will no longer be shown in the web interface. After a participant disconnects from the unit, the system will automatically restore the associated VMs from snapshot and make them available.

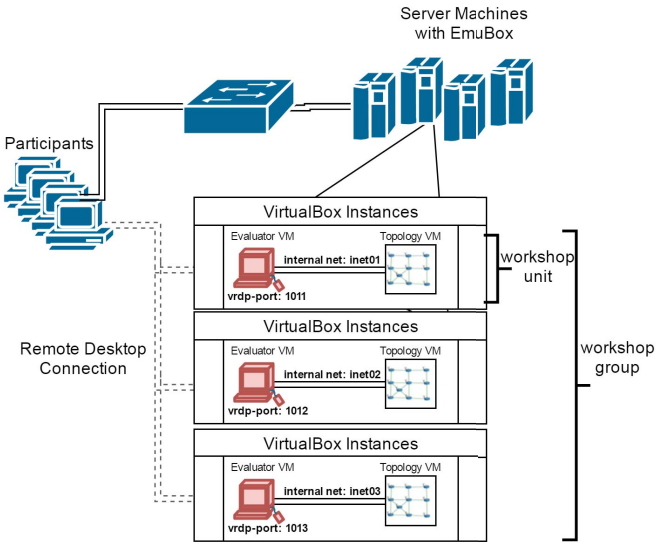

Fig. 2. EmuBox Usage

V. The ECEL

We designed ECEL using a plugin architecture to enable extensibility and to also leverage existing logging tools. We implemented ECEL in Python due to its rich framework and cross-platform compatibility. Figure 3 shows the ECEL architecture.

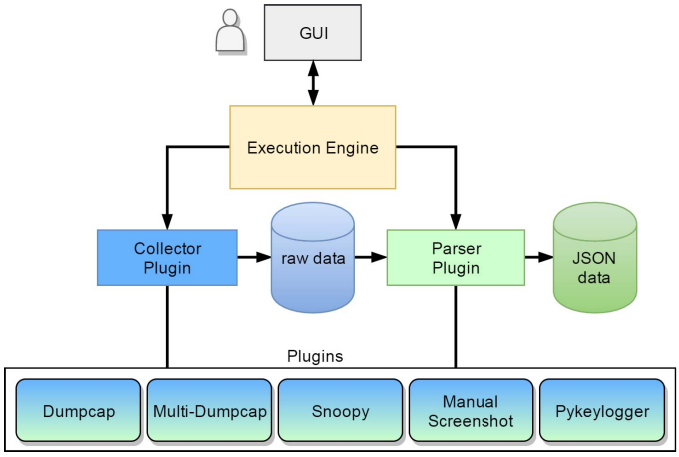

Fig. 3. ECEL Architecture

ECEL's execution engine handles the fundamental processing aspects of the system as well as the plugin infrastructure, which is made up of collectors and parsers. Users invoke the engine through the graphical interface.

A. Execution Engine

The underlying processes that display the interface and manage the plugins are implemented in the execution engine. The engine also archives data and starts, pauses, resumes, and terminates plugins. It keeps track of execution details such as process IDs and plugin execution commands. When

invoked, the engine first identifies all *enabled* plugins and their configuration information. The graphical interface is then instantiated and shown to the user. When a user chooses to start plugins from the graphical interface, the execution engine will invoke the script or system call associated with the plugin.

B. System Extensibility through Plugins

The execution engine provides an Application Programming Interface (API) that can be used to extend ECEL capabilities. We have implemented a collector and a parser plugin for each of the following:

- Dumpcap: network sniffer that stores packet captures in a pcap file named according to the start time.
- Multi-dumpcap: multiple dumpcap instances running on a set of specified interfaces. Data are stored in multiple pcap files all named according to the interface name and start time.
- PyKeylogger: collects clicks and timed screenshots as images named according to the time taken; keystrokes are stored in a text file and each character press contains metadata for the user, process id, thread id, and context window.
- Snoopy: system call logger that stores time-stamped commands in a text file.
- Manual Screenshot: collects on-demand screenshots and metadata. Screenshots are stored as images, named according to the time taken; metadata are stored in a text file with the same name as the screenshot.

In our design, both the collector and parser are software interfaces that implement generic behavior and configuration. Each individual plugin inherits and extend these behaviors.

C. Collector Plugin

All collector plugins must inherit from the collector class and, therefore, must implement methods for logging data and time stamp information associated with some phenomena. There are two types of collectors. Manual collectors are instantiated in an ad-hoc fashion by interacting with the graphical interface or the status icon. Automatic plugins will continuously execute in the background. For example, the manual screenshot plugin resides in the status icon context menu. When a user invokes this plugin, a separate text entry window will solicit IP address, initials, and an annotation. The Snoopy plugin, on the other hand, runs as a background process and produces a log consisting of all system calls.

In addition to their *type*, these plugins must also define additional values that are used by the engine during execution. *Autorestart* specifies whether the plugin should be restarted if it is terminated outside the application. An example where this is useful is when using the dumpcap collector. If a network interface is disabled or abruptly loses connectivity, the dumpcap process will terminate. If *autorestart* is enabled, the engine will continually attempt to restart the process. If the interface becomes enabled, data-collection will continue. *Parser path* points to a parser plugin that is capable of decoding the data gathered by the collector. *Enabled* determines whether the plugin should be loaded and displayed on the graphical interface. These configuration values are stored in a JSON file that is read when the processing engine executes.

Users may also define custom variables and values for plugins. As an example, the multi-dumpcap collector defines a variable called *Interfaces* that specifies the devices that will be used for data collection. In the collector source code, this value is read and then used as part of the execution command.

D. Parser Plugin

Parser plugins inherit from the parser class and, therefore, must implement a *parse* function, which takes input and output paths as parameters. This function can be written directly in the python code; alternatively, this function can call external scripts or executables. For performance reasons, we implemented the parsers using Java classes. Our parsers all generate structured JSON data because we plan to develop a visualization system to view data in a time line. Parsers may be executed as within ECEL using the GUI or as separate processes.

E. User Interface

Users invoke the processing engine functions through the GUI (see Figure 4).

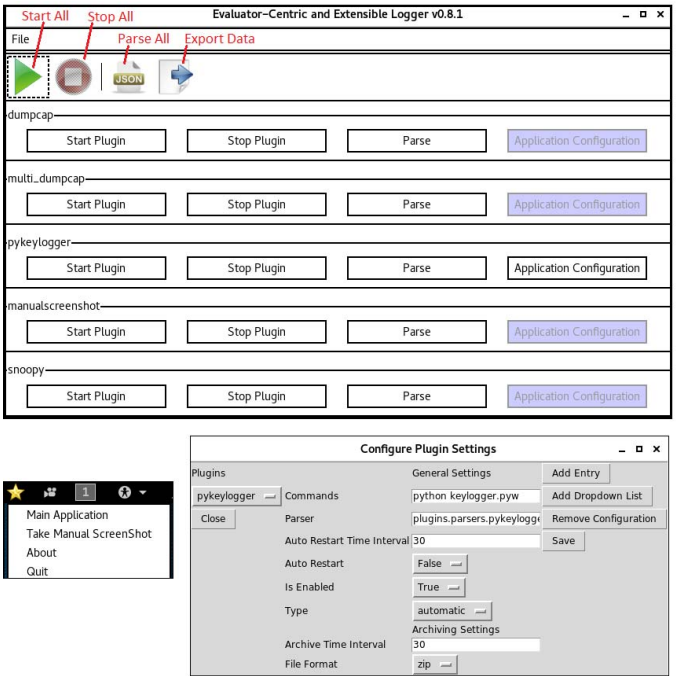

Fig. 4. The ECEL graphical interface. The main window is shown at the top, the context menu bottom left, and the configuration sub-window bottom right.

The interface has both a window view and a context menu (shown as a status icon in the task bar). Through the main window, collector plugins may be started individually or in batch mode. The configuration sub-windows are used to view and modify the processing engine and plugin configurations. The parsed data may also be compressed and copied to an

output path through the export window. The context menu is used to open the main window, start *manual* collector plugins, and exit the application.

VI. Case Study: Network Layer Route Hijacking Scenario

To demonstrate the use of the platform and to determine the scalability of our approach, we developed a small challenge scenario based on a pen-testing tutorial from [12].

A. Platform Setup

We installed EmuBox on a Dell blade server running Windows Server 2012 with 48 processors and 128GB of system RAM. We monitored system resource statistics with 8 simultaneous participants. The server and the 8 participant laptops were all connected to a gigabit switch. Participants used Linux rdesktop to connect to the scenario using a fixed resolution of 1280 x 768.

The Evaluator VM is a Kali 2016.1 64 bit machine and includes the Loki framework for testing routing protocols. The rest of the topology is implemented using CORE. We simulate a human user by executing a python script on the webclient (11.0.0.2) that communicates with the webserver (10.0.4.10). The webserver is hosting a website using nginx and is also running an OpenSSH sshd service (with no active connections). The client/server traffic hops through five routers. These routers decide paths using the routing information protocol version 2 (RIPv2) with null authentication. The client first requests an unencrypted authentication page. If the page loads correctly (i.e, the page contains a form with a text entry and a password entry fields), then the client will supply a user name and password. The sshd service allows remote logins. The user name and password in the web traffic can be used to log in to the sshd service. If the connection between the client and server is interrupted, the client will continually attempt to reconnect and supply credentials.

The evaluator VM is connected to the scenario through a CORE RJ-45 node, which is subsequently connected to a router in the client/server traffic path (see Figure 5).

Participants are given no other information (IP address scheme, outside networks, etc.) about the network. The ultimate objective of this challenge scenario is for the participant (using the evaluator machine) to successfully log in to the sshd service. The steps for completing this challenge are described below.

- Identify the gateway and a valid and unused IP address using Wireshark. Assign this address to the local interface.
- Identify the web and ssh server on 10.0.4.10 using a network scanner.
- Clone the website using the social engineering toolkit (SET).
- Inject a RIP route advertisement to the 10.0.4.0/24 subnetwork and start a virtual interface using the address 10.0.4.10. Host the spoofed website and sniff credentials.

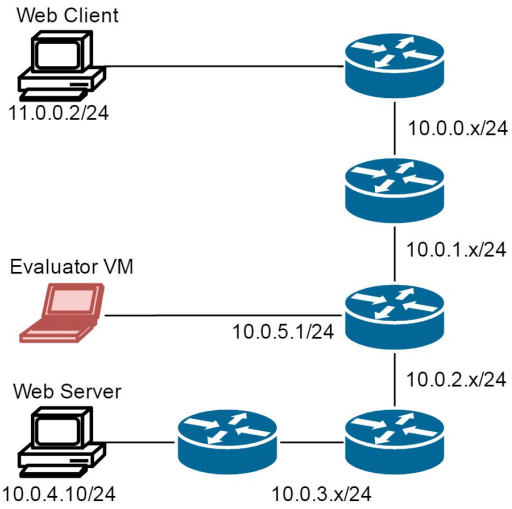

Fig. 5. Route Hijacking Scenario

- Bring down the virtual interface and terminate the route injection processes.
- Wait until the original route to 10.0.4.10 is restored and then log in to the ssh server.

B. Performance Metrics

We used the Windows Performance Analysis Toolkit to collect resource-utilization data during the scenario challenge. Each workshop unit was configured to use 3GB of RAM–1GB for the network topology VM and 2 GB for the evaluator VM. Participants were familiar with the scenario and were able to finish the challenge in roughly 15 minutes–which helped us to record a more synchronized view of resource utilization. Alternatively, participants that are unfamiliar with the scenario usually take much longer (up to 80 minutes), varying with experience and skill level. Performance graphs can be seen in Figure 6.

While the graphs show that the system did not induce substantial performance degradation, it is clear that the network throughput would be the first bottleneck. While VirtualBox has some options for compressing video streams from a remote host, using the *vrdevideochannel* option, the rest is transmitted uncompressed.

The total number of transmitted bytes during the workshop was 1.9GB in a total of 214,361 packets; there was a peak throughput of 22.5MB/s at 187 seconds. The majority of traffic was sent from the EmuBox server to the participant machines; in fact, all but 1.3MB of the total throughput. Most of the throughput occurred within the 300- and 450-second time frame. This is likely when participants simultaneously used tool GUIs (Wireshark, Iceweasel, and Loki) the most.

The CPU usage averaged at 1.92% and stayed below 5% for most of the workshop. The peak near 900 seconds marks the point when some of the VMs were shut down and restored to a previous snapshot state. Memory usage was between 24GB

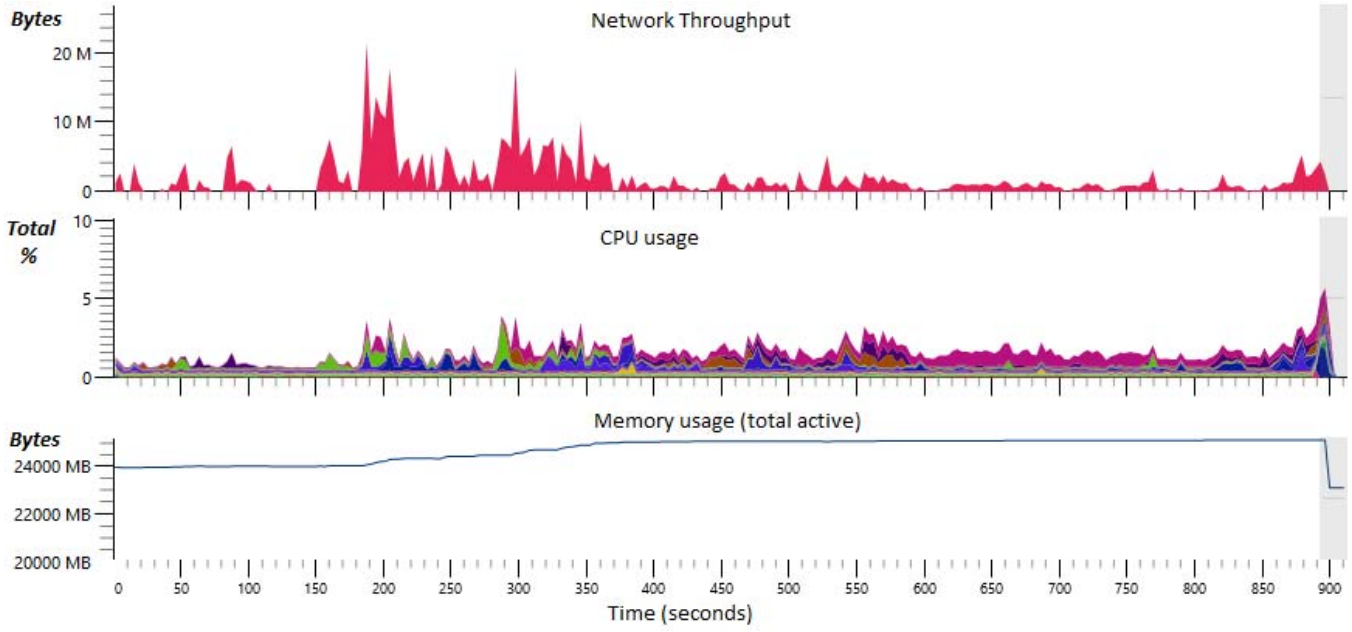

Fig. 6. Platform resource utilization during the hijacking scenario challenge serving 8 simultaneous participants. A total of 16 virtual machines were active.

and 25GB. Each workshop unit had a total of 3GB allocated to the VMs. Together, the 8 units make up the 24GB.

We also tested our platform and scenario on a laptop with an i7 processor with 16GB of RAM. There were no noticeable performance issues with 4 simultaneous workshop units. With 5 units, the system became unusable due to memory usage.

VII. Future Work

For the ECEL, we plan to develop a video-recorder plugin and plugins for assessment tools such as Nmap and Nessus. Currently under development is a visualization system that is capable of displaying, editing, and annotating the ECEL generated JSON data.

With respect to EmuBox, we will allow VPN connections by using the HIL feature of CORE and the port forwarding feature of VirtualBox. We are currently using our platform on an isolated network at a university, but we will eventually allow connections from the Internet. Frontend improvements will provide user management, usage statistics, and support for curriculum development, such as longer-term and persistent connections and surveys. We plan to conduct a formal user study to determine the effectiveness of our challenges and to solicit feedback for improvements. We are also working on automated mechanism for copying ECEL data to external storage when participants complete the exercises.

References

[1] AFDA. Afda intrusion detection datasets. [Online]. Available: https://www.unsw.adfa.edu.au/australian-centre-for-cyber-security/cybersecurity/ADFA-IDS-Datasets/

[2] NetReSec. Netresec publicly available pcap files. [Online]. Available: http://www.netresec.com

[3] E. Nunes, N. Kulkarni, P. Shakarian, A. Ruef, and J. Little, "Cyber-deception and attribution in capture-the-flag exercises," in *Cyber Deception*. Springer, 2016, pp. 151–167.

[4] G. Vigna, K. Borgolte, J. Corbetta, A. Doupé, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili, "Ten years of ictf: The good, the bad, and the ugly," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, 2014. [Online]. Available: https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna

[5] F. LLC. Facebook capture the flag. [Online]. Available: https://www.facebook.com/notes/protect-the-graph/facebook-capture-the-flag/1466163253623821/

[6] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "Core: A real-time network emulator," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*. IEEE, 2008, pp. 1–7.

[7] R. Chadha, T. Bowen, C.-Y. J. Chiang, Y. M. Gottlieb, A. Poylisher, A. Sapello, C. Serban, S. Sugrim, G. Walther, L. M. Marvel *et al.*, "Cybervan: A cyber security virtual assured network testbed," in *Military Communications Conference, MILCOM 2016-2016 IEEE*. IEEE, 2016, pp. 1125–1130.

[8] B. Ferguson, A. Tall, and D. Olsen, "National cyber range overview," in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 123–128.

[9] D. Schauland and D. Jacobs, "Managing the windows event log," in *Troubleshooting Windows Server with PowerShell*. Springer, 2016, pp. 17–33.

[10] M. B. Marius Aamodt Eriksen and B. Skufca. Snoopy logger. [Online]. Available: https://github.com/a2o/snoopy

[11] A. Milenkoski, M. Vieira, S. Kounev, A. Avritzer, and B. D. Payne, "Evaluating computer intrusion detection systems: A survey of common practices," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 12, 2015.

[12] R. Wood. Exploiting RIPv1. [Online]. Available: https://digi.ninja/blog/rip_v1.php