Repeatable Experimentation for Cybersecurity Moving Target Defense

Jaime C. Acosta[1(✉)], Luisana Clarke[2], Stephanie Medina[2], Monika Akbar[2], Mahmud Shahriar Hossain[2], and Frederica Free-Nelson[1]

[1] DEVCOM Army Research Laboratory, Adelphi, USA
jaime.c.acosta.civ@mail.mil
[2] University of Texas at El Paso, El Paso, USA

Abstract. The scientific method emphasizes that repeatable experimentation is critical for several reasons; to facilitate comparative analysis, to recreate experiments, to re-validate reported results, to critique and propose improvements, and to augment the work. In the field of cybersecurity moving target defense, where assets are shuffled to thwart attackers, it is critical to know what strategies work best, the success factors, and how these strategies may impact system performance. While some researchers make their algorithms, models, and tools available as open source, it is difficult and, in some cases, impossible to recreate studies due to the lack of the original operating environment or no support for software components used within that environment.

In this paper, we present the repeatable experimentation system (RES), which aids in creating and recreating networked virtual environments to conduct comparative network studies. Experiments are composed of virtual machines, containers, automation scripts, and other artifacts that are needed to recreate and re-run a study. This includes data collection and analysis. We provide a case study where we incorporate two publicly available moving target defense implementations that use different underlying software components. We present how RES can be used for fully automated experimentation along with an analysis on the results obtained from parallel and sequential executions. We have packaged the case study into a RES file that can be used by other researchers to repeat, modify, and improve on these and other works.

Keywords: Cybersecurity · Repeatable experimentation · Dynamic defense · Moving target defense

1 Introduction

Moving target defense (MTD) is a technique that holds much promise; it reduces the attack surface by making changes to system configurations when certain conditions are met. Network MTD approaches may change IP addresses at certain time intervals or based on observed throughput [6]. At the operating system

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2021
Published by Springer Nature Switzerland AG 2021. All Rights Reserved
J. Garcia-Alfaro et al. (Eds.): SecureComm 2021, LNICST 398, pp. 82–99, 2021.
https://doi.org/10.1007/978-3-030-90019-9_5

level, a prominent technique in wide use is address space layout randomization (ASLR), where the mapping of memory for processes is meant to be ever-changing and unpredictable [11]. Application-level MTD algorithms may, for example, attempt to randomize the code execution paths of a binary [23].

Attackers, especially those that are experienced or members of an organizations, follow certain tactics, techniques, and procedures (TTPs) depending on their objectives [22]. MITRE and others [27] have documented some of these attack scenarios through analyses of breaches and other incidents. TTPs consist of steps that include the gathering of information, such as the network devices and addresses on an internal network. An attacker that gains access will be more prepared, and likely need to reveal less (for example, eliminating the need for a noisy port scan) based on what they know beforehand. Using the IP Shuffling technique [3], a defender can attempt to eliminate this overmatch by dynamically changing the addresses of network nodes.

There are many considerations that must be addressed before claiming security measures as successful. Rigorous investigations are required to study the impact of a security measure in concrete scenarios and their effectiveness against the adversary's intentions. Additionally, a single solution will not likely work in all circumstances. For example, in a network where situational awareness is occasionally transmitted using short messages in a small network, high-frequency shuffling may not be an issue. This solution may not be suitable in scenarios where there is high-latency and high throughput. In short, a characterization of the impact of the algorithm in a relevant scenario must be performed. These results must be compared and analyzed against other relevant studies in relevant domains.

In this paper, we present the repeatable experimentation system (RES)[1] with the objective of enabling widely-accessible and shareable MTD experimentation. In summary, our contributions are the following.

- An open source and extensible tool, RES, along with demonstrations showing how it can be used to package experiments to facilitate repeatability.
- A case study in which we incorporate a MTD implementation[2] that uses software-defined networking and a randomized IP shuffling technique (similar to those reported in [5]).
- An analysis that shows that using RES and its parallel execution does not impact execution results and a characterization of the MTD implementation when used against port scans.

Section 2 outlines the literature relevant to this paper. We explain our Repeatable Experimentation System (RES) in Sect. 3. Section 4 provides the design of the case study we conducted in this paper. We provide experimental analyses with MTD characterization in Section 5. Section 6 concludes the paper with a brief discussion on future directions of the work.

[1] Available at: https://github.com/ARL-UTEP-OC/res.
[2] Available at: https://github.com/ARL-UTEP-OC/res-ryu-mtd.

2 Related Work

Many approaches have been proposed to create variability for achieving moving target defense. Some of these approaches work on reducing attack surfaces [15,21, 33], some focus on randomizing the network components [16,24], others propose randomizing types of services [28], or switching operating systems [4]. Among these, the software-defined network (SDN) has gained attention in randomizing network components by programmatically changing network configurations [2, 20,32].

Given the rise in MTD research, there are usually multiple different MTDs to choose from under any given threat scenario. However, there is no standard metric for evaluating MTDs, which makes it difficult to estimate the cost or assess the effectiveness of MTDs when deciding the best or most suitable MTD. Several projects used attack surface [7,21] and attack graphs [13] to evaluate MTDs. Others used security [18], performance [8], and various network and system properties [14,29] to assess MTDs.

When it comes to measuring the effectiveness or evaluating the performance of an MTD, most MTDs present a customized approach. There are a few ongoing efforts towards building theoretical assessment frameworks for MTDs [6,31]. However, much of these efforts are geared towards specific types of attacks or MTDs. There are no standard mechanisms for analyzing different MTDs and evaluating and comparing their effectiveness, to the best of our knowledge.

3 Repeatable Experimentation System

The development of RES is based on several interactions with cybersecurity professionals and previous experience in developing cybersecurity-inclusive moving target defense network scenarios. Even when moving target defense algorithms are made available, it still takes a significant amount of work to replicate the experiments described in scientific papers — to allow building upon the work. This is primarily due to software dependencies or software being inoperable; where newer versions of packages are incompatible with other software, including the operating system. Other reasons include gaps in the intricate details related to set up and configuration that are omitted, possibly due to paper length constraints. We developed a novel software tool – RES – to mitigate these issues and to facilitate the scientific experimentation process.

3.1 Capabilities

RES is available as open source software to encourage its wide use and to allow analysts to share their experiments without having to purchase software licenses. It is extensible and it leverages several other tools including Oracle VirtualBox [30], Apache Guacamole [10], and, optionally, HashiCorp Vagrant [12]. Python 3 is the primary implementation language and we use the PyQt5 module for the

graphical frontend. RES runs on Linux, Mac, and Windows with only minimal graphical differences. The underlying base code is the same.

Expansion and flexibility are critical for repeatable MTD experiments, especially due to the ever-changing and ever-improving underlying virtualization technologies. To support expansion and flexibility of existing virtual environments, we used a plugin design to allow analysts to add, swap, and improve interfaces to underlying components such as the hypervisor, remote desktop provider, and packaging system.

To ease the process of replicating experiments – including those developed by other analysts – RES allows analysts to import a single all-inclusive file that contains all the required artifacts necessary for an experiment. This includes the virtual machines (VMs), which are automatically loaded into the hypervisor, as well as any other instructional materials associated with the experiment. After import, the number of steps required to repeat the experiment are minimal. Exporting of experiments is trivial. Users are able to specify virtual machines, materials, and also their experiment configurations (number of experiments, scripts, remote display information) and then create an all-inclusive package.

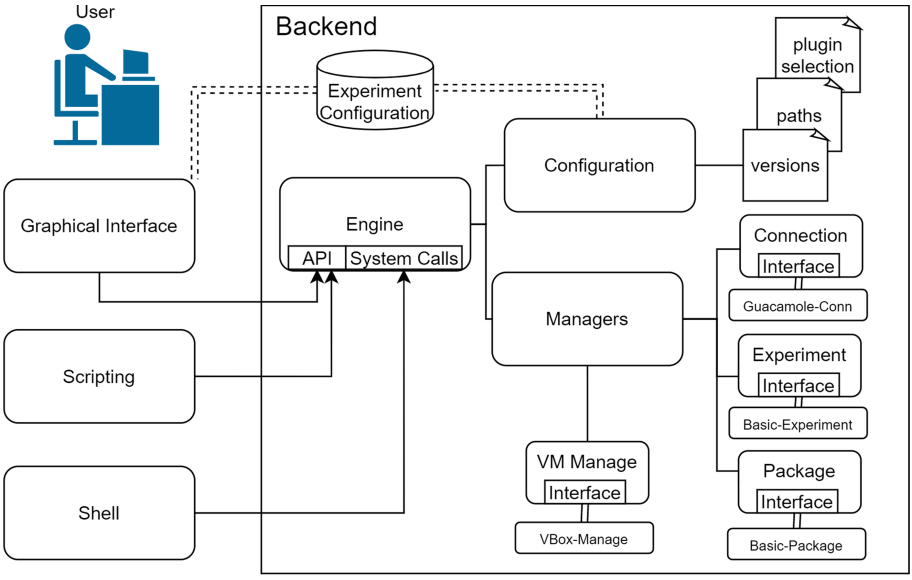

Fig. 1. RES architecture

Experiments may be cloned to enable the parallel or sequential execution of multiple runs. An analyst may include specific commands that run on start. As an example, a user may have an experiment that requires Mininet [17] to run with certain arguments; pointing to the scenario that will execute. The scenario

needs to run for a certain length of time, then quit and copy the resulting data to a network attached storage device. All of these commands can be added to the experiment during configuration. All of the hypervisor invocations are multi-threaded, including calls to start, stop, pause, snapshot, and delete clones; this is done to improve performance.

Lastly, we focused on giving analysts the ability to interact with RES using different modes of operation. The system can be invoked through a graphical interface, but also through Python scripting that communicates directly with the backend as shown in Fig. 1.

3.2 Design

The system, shown in Fig. 1, consists of a segregated frontend and backend. The frontend is a way to abstract and facilitate access to backend capabilities. Users of the graphical application can run an experiment using the point-and-click interface; the developer of an experiment can create a script that automates all of the actions required to execute and collect data; an interactive shell provides a user with an interactive terminal-session to conduct the same actions available through the graphical interface. Communication between the frontend and backend occurs through either a Python application programming interface, essentially function calls with results provided through return values, or by executing a companion application that makes calls to the engine and writes results to standard output. Examples are provided with the application; including unit tests and a fully working graphical user interface that implements the system capabilities.

The graphical interface (shown in Fig. 2) contains a mechanism to create and modify experiment configurations (each is stored as a XML file). When started, RES displays the Configuration tab (shown in the left window) where a user may select, import, and export experiments. They can also specify a server IP address (used for remote display), group name, number of clones (or instantiations of the experiment), among several others. It is also possible to add virtual machines and any additional files that should belong to the experiment from this view. Selecting a virtual machine (shown in the bottom window) will display additional configuration options, including whether remote display should be enabled for the machine, commands that should be run on the machine when it first starts, and the interfaces on the machine. Specifying network interface names that match across other virtual machines will connect the machines together (similar to them being on a shared network switch). In Fig. 2, the Ubuntu18-core-cdes machine has 8 different interfaces; the first is intnet1. The ATTACKER_10.0.0.2 machine has only a single interface called intnet1. These two machines, when cloned, will share a common interface and can therefore communicate directly. This functionality is preserved across clones so that each set will only be able to communicate amongst themselves. This allows for building larger, complex networks when combined with network emulation software such as the Common Open Research Emulator [1] or Mininet. In this case, machine's traffic can be forced to traverse through the emulated network, which can be e.g., an SDN, before reaching a

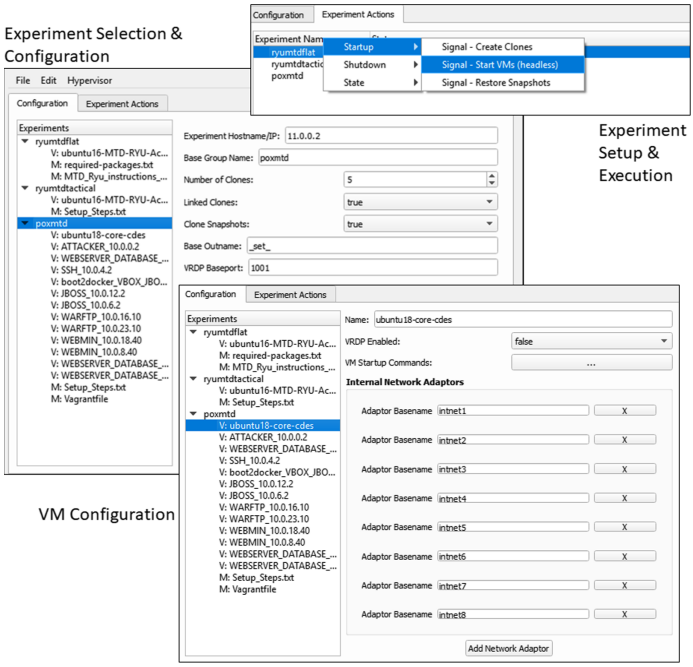

Fig. 2. RES graphical interface

recipient. The second GUI tab (shown in the top window) invokes various functions of the Experiment Manager. It is worth noting that these XML files can be manually created and modified; the GUI is merely a convenient way to do so. The backend modules are included in Fig. 1.

The Configuration module reads and parses the XML experiment files and then makes data structures available to the Managers. The Configuration module also reads and writes information related to specific plugins (e.g., the concrete VirtualBox implementation), paths to external programs (such as the path to VBoxManage), and values representative of specific versions of packages associated with the system.

The Managers module uses a plugin design to enable flexible integration with external components. The VMManage interface module consists of all of the tasks associated with virtualization components at the virtual machine level; including starting, pausing, stopping, snapshot, and restoring of individual virtual machines. The software currently implements these functions with VirtualBox plugins, but it can easily be extended to support others, such as VMWare, ESX, XenServer, and Proxmox; as these all have mechanisms for controlling virtual machines as well as several interfaces and concrete implementation classes to handle remote desktop functions. We also chose to implement VMManage at this granularity to enable concurrent execution; e.g., to allow several virtual

machines to snapshot, clone, start, stop, etc. simultaneously; many of which are not concurrent through the standard VirtualBox GUI.

The Package interface module primarily handles importing and exporting. Using the information associated with the experiment, provided by the Experiment Configuration, it calls the VMManage module to trigger the backend hypervisor to execute its specific import/export functions. In the case of VirtualBox, open virtual appliances (OVAs) are created for virtual machines. In addition to this, any material files that are part of the experiment are included in the exported file. The result is a compressed file with the *RES* file extension. This module is also extensible; we provide the behavior described above as the fundamental and basic, however, developers are able to extend this module and provide additional functionality, e.g., to import and export remotely, implement authentication, secure with encryption, and others.

Behaviors related to experiments, such as cloning, starting, and stopping occur through the Experiment interface module. As with the other modules, experiment information is read from the Configuration and then batch operations are executed through this component. Specifically with our VirtualBox implementation, we use the guest control feature; which requires that the virtual machines are running the same version of guest additions as the host. We provided basic functionality, but as with the Package interface, developers may choose to extend functionality to include additional scripting, such as integration with other provisioning software including Vagrant and Ansible.

During the execution and setup of experiments, it's important to allow users to see the running systems. We developed the Connection interface module for this reason. Currently, it uses Apache Guacamole to broker remote desktop connections that are accessible using HTML5 with any modern web browser. User creation is also automated, using the guacapy REST API Python module.[3] Developers may write extensions to connect to virtual machines instead through VNC, add capabilities such as key-based authentication, or even swap out the guacamole module entirely with another remote desktop broker.

4 Case Study Design

To demonstrate and test the utility of RES, we developed a case study with the following process.

1. Recreate the execution environment for a network MTD algorithm and modify for use in a set of experiments (Sects. 4.1 and 4.2).
2. Construct and execute experiments several times in parallel using RES. Afterwards, the experiments and results are packaged into self-contained, archives that can be redistributed (Sects. 4.2 and 4.3).
3. Analyze the differences in the results when using RES versus manual execution. Characterize the scanning success rate when using different MTD shuffling time intervals (Sect. 5).

[3] Available at https://github.com/pschmitt/guacapy.

4.1 MTD Algorithm Implementation

We started by looking for free and open-source implementations of network MTD algorithms. We selected a Ryu controller-based SDN implementation [9] because it shared similarities with more sophisticated algorithms such as the Flexible Random Virtual IP Multiplexing [26]. The Ryu controller is written in the Python language, it provides access to its backend processes using an API, and it runs on Linux. Ryu is prominently used today and supports newer versions of the OpenFlow protocol. Rohitaksha and Rajendra [25] provide an in-depth study of the advantages and disadvantages of Ryu and other controllers such as Pox.

The Ryu implementation uses Python dictionaries to keep track of current mappings between real and virtual IPs and uses rules and flow tables to restrict communication to and from virtual IPs. The Ryu implementation was developed as a homework assignment for a course that teaches software-defined networking concepts. We made several modifications to the original code and introduced features that make it usable in our experiments. We refer to this modified implementation as mtd-ryu. Figure 3 shows the high-level processes that compose mtd-ryu.

Mtd-ryu inherits it's dependencies from the original implementation. It runs on an Ubuntu 16 VM, and it works with the latest version of Mininet (2.3.0d4). It requires Python2 (which is deprecated) and Ryu-manager version 4.3.2. The controller uses OpenFlow 1.3 to communicate with the switches, which run Open vSwitch version 2.5.5. Address resolutions are stored as flow entries on each switch and packet header field values are modified to use virtual IPs instead of real IPs.

Many MTD scenarios in the literature shuffle IP mappings in such a way that all established connections, even seemingly legitimate ones, are dropped indiscriminately. This seems impractical for a realistic scenario; important services would be interrupted and the stateful connections would constantly be restoring address resolutions and re-establishing connections after every shuffle. Non-stateful connections would at a minimum have to constantly resolve node addresses. These could cause significant delays in communications.

Rather than having flow tables simply cleared every specified number of seconds and dropping all connections, we used the concept of authorized and unauthorized entities. Connections that originate and are destined for authorized entities (based on IP Addresses) are considered legitimate connections. These are ignored during shuffling which eliminates disruptions. Connections that are not considered legitimate are dropped and must be re-established after every shuffle. In the code, we implement this behavior by having two separate flow tables on each switch; managed by the controller. The authorized flow tables contains legitimate connections and they are never cleared. Both tables use only virtual IP addresses. This does not affect the legitimate connections because ARP entries are persistent and domain name resolution only happens once; when the nodes communicate for the first time. Lastly, we added a throughput monitor that can be used in future work to base shuffles on anomalous traffic load.

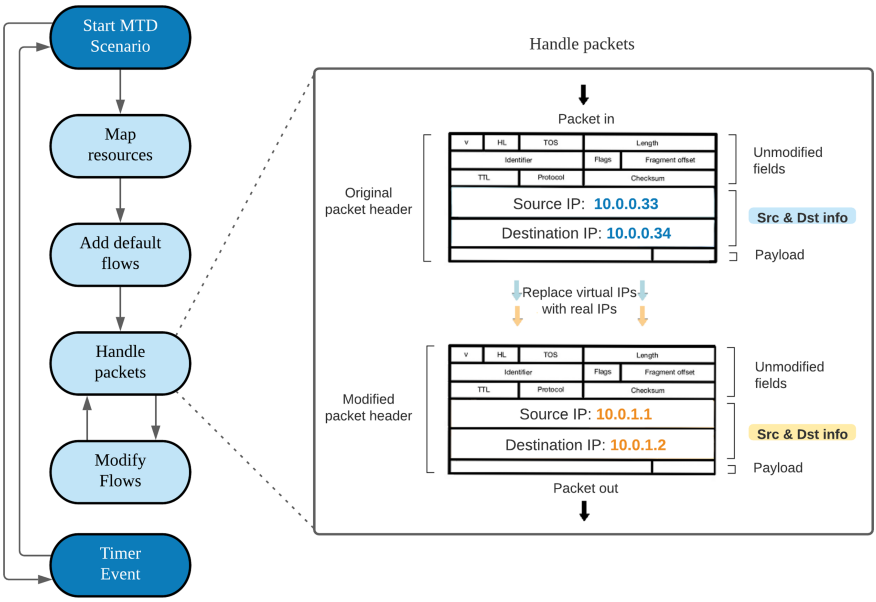

Fig. 3. Mtd-ryu high-level process

4.2 Scenario

The following is the scenario that we constructed for the case study. The network consists of two switches, four host nodes and one controller (as shown in Fig. 4.)

Each switch is running Open vSwitch and gets flow updates through the controller. The four nodes are segregated into two subnets (h1 and h2; h3 and h4). Nodes can communicate across the subnetworks through the switch nodes. Subnet s1 and s2 are using a moving target defense algorithm for all traffic; nodes that reside within the s1 subnetwork are considered trusted entities and all others are untrusted. The moving target defense algorithm is configured to change all node virtual IP addresses shuffle at some given interval (ranging from 20 s to 230 s). The algorithm uses a pre-defined set of virtual IP addresses, all within a /24 subnetwork, (resulting in 255 possible IP addresses) for assignment. It is possible that a node receives a previously used virtual IP multiple times during execution. Any traffic across trusted entities is never interrupted by shuffles.

At the start of the scenario, h1 runs a script that opens 20 disperse ports from 22–9999 using the netcat software. Nodes h3 and h2 both run an Nmap scan against node h1, particularly looking for any open ports in the range 0–9999. Nodes use Nmap's template level 4, which ranges from 0–5, where 0 is the slowest and 5 is the fastest. This template is suggested for use on ethernet-connected networks [19]. All nodes write observed traffic to a network packet capture file (pcap) and scanning nodes write nmap results to text files; h2 and h3 write unauth_scan_output and auth_scan_output respectively.

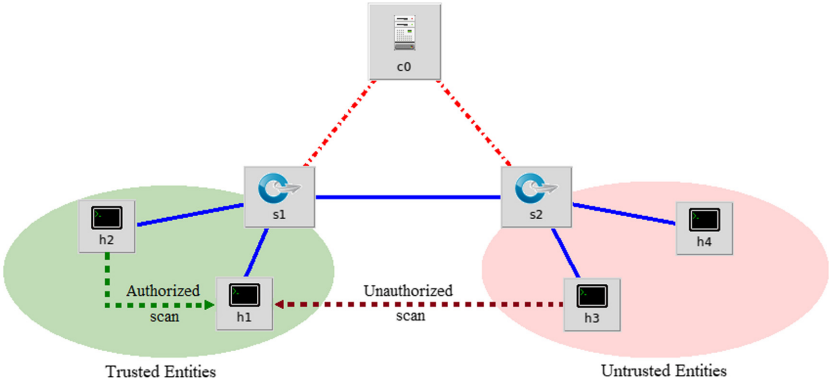

Fig. 4. Experiment scenario

To run mtd-ryu, we used Mininet [17], a virtual networking Linux-based emulator that uses process isolation to enable analysts to create multi-node scenarios on a single system. Mininet scales very well (demonstrating more than 4096 individual nodes on a single Linux Kernel) and its main purpose is to create OpenFlow applications [17]. While not perfect (e.g., in practice, it can only be used on systems running Linux) it is a useful tool for testing and experimentation of SDNs, such as those used for MTD. Mininet also allows analysts to specify technologies to use for switches and controllers. We used Miniedit, Mininet's graphical user interface, to construct the scenario as shown in Fig. 4.

We set up the entire scenario in a Virtual Machine and then automated the execution and data transfer with RES.

4.3 Experiment Construction with RES

We used the RES graphical interface to create a new experiment and added the Ubuntu VM along and a README file that documents the steps required to run the experiment manually. We configured the VM to allow remote connections and specified the localhost as the address. We configured the system to create 10 clones and to use the link-cloned option for speed.

In order to fully automate the experiment, we iteratively added 6 commands to be executed 60 s after the machines are instantiated. This process required some fine-tuning through trial during which time we used VBoxManage. The commands are as follows.

1. Start the controller
2. Sleep for 10 s to allow time for the controller to start and the virtual IPs to be generated and mapped
3. Start the topology; including all scripts for packet capture, opening ports, and nmap scans
4. Sleep for 3600 s to allow the scans to complete

5. Stop the scenario by killing all associated processes (python2 in this case)
6. Copy all data collected from the scenario to a unique path on the host.

A special construct, {{RES_CloneNumber}}, is used to specify a unique number associated with experiment instances. This is useful especially when specifying the directory where data should be copied from the instances to the host (step 6 above). For example specifying ryu-out30s_{{RES_CloneNumber}} denotes the directory ryu-out30s_1 for the first instance.

At first, a Dell 7700 Laptop with 8 processors running Windows 10 was used as our testing platform, but this was not well-suited for running the 10 simultaneous experiments. The CPU utilization consistently stayed at 100% throughout the executions and after a short while, the system was unusable. To alleviate this issue, we created a RES file using the export feature and then imported and re-ran the experiments on a Rack Mounted Desktop Server with two 1.80 GHz Eight Core Intel Xeon Silver 4108 Processors - 11 MB Cache processors and 128 GB RAM. This system was running Linux. Performance is shown later in Fig. 5 and Fig. 6. VirtualBox and RES were installed on the remote system; no modifications to the experiment were needed. Using the remote display features allowed us to use a Remote Desktop Client to occasionally and remotely connect to the virtual machines.

5 Ryu Experiment Analysis

5.1 Impact of RES Parallel Execution

We first wanted to test if executing several instances of virtual machines, running the individual scenarios, had any impact on the results. We proceeded with two paths; first we executed the network scans without shuffling manually and sequentially 10 times. Afterwards, we used RES to run a 10-set simultaneous execution. We recorded the times required to complete the scans of all 20 open ports. When using no shuffling, in every case, all 20 ports were identified. Table 1 shows the results.

Overall, the timings observed between the manual and automated scans are relatively close, considering the inherent variability associated with networking scanning in general. It is worth noting, that while the results vary across the runs, they consistently fall within a general range. The average, min, and max values between the manual and automated runs are all close, within 5.25 s. The behavior can, therefore, be characterized and used when making decisions about defenses. Running these in sequence is time consuming, but the parallelization of RES alleviates this issue.

Table 1. Manual vs. automated timings without shuffles (in seconds)

	Intra-Subnet		Inter-Subnet	
Run	Manual	Auto	Manual	Auto
1	29.34	29.5	204.81	196.31
2	56.5	18.25	131.33	105.56
3	49.43	30.72	198.7	126.13
4	21.16	27.78	198.89	208.03
5	19.58	16.51	150.86	100.16
6	75.21	62.52	136.63	216.65
7	69.59	40.01	201.62	138.41
8	35.04	35.97	102.62	202.70
9	30.34	22.91	159.08	123.85
10	27.15	76.61	119.4	225.90
Avg	41.33	36.08	160.39	164.37
Std. Dev.	19.05	18.36	36.25	47.20
Min	19.58	16.51	102.62	100.16
Max	75.21	76.61	204.81	225.9

The somewhat high standard deviation is due to some of Nmap's non-deterministic scanning behavior as well as complexities in the underlying network stack. This exemplifies the need for experimentation that emphasizes characterization of behaviors across several executions versus assumptions based on limited-samples.

Nmap completion time was on average four times greater when the scans originated from a remote node (inter-subnet scans) than from an intra-subnet node. This is due to the additional network device between the nodes: the switch. The switch may also cause delays e.g., when it queries the controller for a flow table addition, removal, or modification.

With respect to execution performance (as shown in Fig. 5), the total CPU load was stable at 4–6% at rest. In the case of a single experiment (with a single virtual machine), load spiked to roughly 12%; when the virtual machines were first instantiated by RES. This subsided when the VMs finish booting (when the VMs reached their login screens). When the scenarios and scans started, the load stayed between 10 and 16% (averaging 13%) and when scans complete, at roughly 240 s, the load decreased until the data transfer (pulling the results from the VMs to the host) and then VM shutdown. Very similar behavior was observed during the parallel executions, except at a higher magnitude. Memory utilization was stable at 8 GB for the single run and 26 GB for the parallel run throughout execution; this is because each VM was allocated 2 GB of memory and the host used roughly 6 GB.

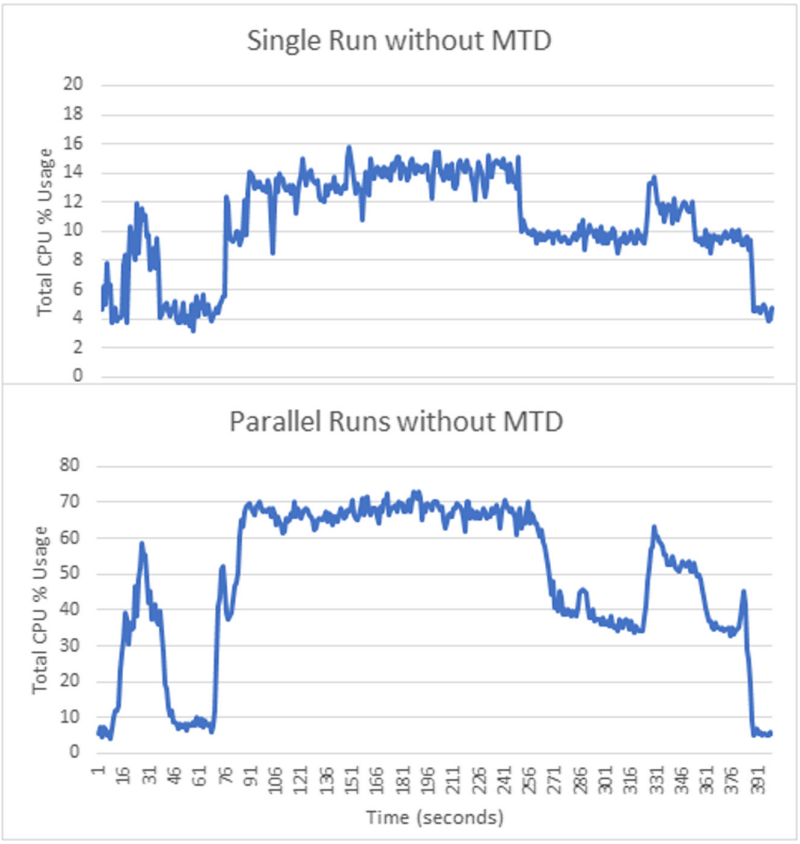

Fig. 5. CPU load on host without MTD

5.2 MTD Characterization

To demonstrate the utility of RES in characterizing behaviors of scanning and defense technologies, we tested the impact of the Ryu-MTD algorithm against Nmap completion times and accuracy in the inter-subnet case. More specifically, we observed the time it would take for a scan to complete as well as the number of ports correctly identified as open when IPs are shuffled at different time intervals. The algorithm is not suited for the intra-subnet case due to the short completion times (as shown in Table 1). Every scan was executed using RES, configured to run 10 instances simultaneously. We report the statistics for these runs in Table 2.

Table 2. Scan times (t) and port identification accuracy (p) with varying shuffle times

Shuffle	Avg		Std. Dev.		Min		Max	
Time(s)	t	p	t	p	t	p	t	p
20	1459.49	10	583.52	4.69	450.63	4	2200.48	20
50	1676.81	12.5	794.10	5.12	98.64	4	2464.63	20
80	1553.80	12.1	809.00	2.70	687.95	9	3244.14	18
110	762.067	15.6	328.51	2.11	250.43	13	1249.41	20
140	777.35	15	597.84	2.24	227.36	12	2242.52	20
170	556.182	17	243.14	1.90	214.25	14	951.03	20
200	353.29	18.9	320.07	1.64	155.65	15	1050.85	20
230	164.37	20	47.20	0	100.16	20	225.9	20

The 230 s shuffle time is a duplicate of the case with no shuffling (the last column in Table 1), since all scans were completed before the shuffle occurred. All other scan time averages are at least twice as long, and as much as 10 times as long (50 s shuffle), compared to when no shuffling is used. The trend is for time to complete the shuffle to decrease and for the accuracy of port identification to rise as shuffle times increase. Additionally, the standard deviation for completion times and port identifications decrease as the shuffle times increase. An interesting outcome of these results is that in all cases except 1 (80 s) the maximum number of correctly identified ports found is 20 of the possible 20. This means that there is always a risk, even when using the mtd-ryu implementation that a port scan will succeed in identifying all open ports correctly. However, the time taken to do so will very likely be higher, giving a defender more time to detect and react, than without MTD.

The CPU execution load during the shuffle experiments is shown in Fig. 6. As with the observations without MTD, the load times during the single and parallel runs are very similar, except at different scales. The experiments were run for one hour using 20 s shuffles. The same spikes were observed during boot. During the single run, the CPU load averaged 7%; this is less than the case with no MTD. The reason is because Nmap scans are slowed due to the shuffles. When a host is identified, a probe is sent and a reply is awaited; this delay causes a decreased load on the system. Memory usage did not change: 8 GB for the single run and 26 GB for the parallel run.

As mentioned previously, the experiment file containing all assets required to recreate the experiment executions is available for download.

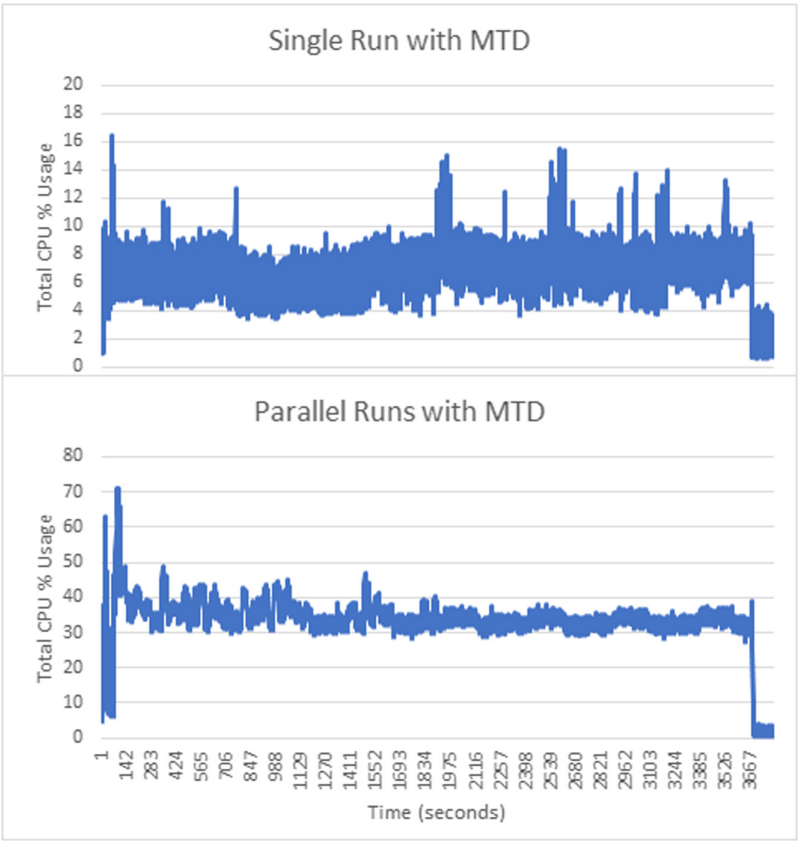

Fig. 6. CPU load on host with MTD

6 Future Work

We plan to make incremental improvements to RES based on community feedback. More importantly, we plan to use this tool to conduct comparative analysis on different defense techniques. The data generated from the executions of different defense mechanisms will become inputs to an autonomous decision support system that will provide insights into which mechanisms may work better under different conditions.

We built RES and we provide it to the community with hope that it will encourage distribution of, not only of written scientific results, but of entire studies; including materials, results, and mechanisms required to reconstruct and improve on the research.

References

1. Ahrenholz, J., Danilov, C., Henderson, T.R., Kim, J.H.: CORE: a real-time network emulator. In: MILCOM 2008–2008 IEEE Military Communications Conference, pp. 1–7 (2008)
2. Al-Shaer, E., Duan, Q., Jafarian, J.H.: Random host mutation for moving target defense. In: Keromytis, A.D., Di Pietro, R. (eds.) SecureComm 2012. LNICST, vol. 106, pp. 310–327. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36883-7_19
3. Carroll, T.E., Crouse, M., Fulp, E.W., Berenhaut, K.S.: Analysis of network address shuffling as a moving target defense. In: 2014 IEEE International Conference on Communications (ICC), pp. 701–706 (2014)
4. Carter, K.M., Riordan, J.F., Okhravi, H.: A game theoretic approach to strategy determination for dynamic platform defenses. In: Proceedings of the First ACM Workshop on Moving Target Defense, MTD 2014, pp. 21–30 (2014). https://doi.org/10.1145/2663474.2663478
5. Cho, J.H., et al.: Toward proactive, adaptive defense: a survey on moving target defense. arXiv preprint arXiv:1909.08092 (2019)
6. Clark, A., Sun, K., Poovendran, R.: Effectiveness of IP address randomization in decoy-based moving target defense. In: 52nd IEEE Conference on Decision and Control, pp. 678–685 (2013)
7. Crouse, M., Prosser, B., Fulp, E.W.: Probabilistic performance analysis of moving target and deception reconnaissance defenses. In: Proceedings of the Second ACM Workshop on Moving Target Defense, MTD 2015, pp. 21–29 (2015). https://doi.org/10.1145/2808475.2808480
8. Dishington, C., Sharma, D.P., Kim, D.S., Cho, J., Moore, T.J., Nelson, F.F.: Security and performance assessment of IP multiplexing moving target defence in software defined networks. In: 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 288–295 (2019)
9. Gangappa, G.S.: Moving target defense RHM using SDN (2018). https://github.com/girishsg24/Moving-Target-Defense-RHM-using-SDN. Accessed 20 Feb 2021
10. Apache guacamole. https://guacamole.apache.org, https://guacamole.apache.org. Accessed 20 Feb 2021
11. Hamlet, J.R., Lamb, C.C.: Dependency graph analysis and moving target defense selection. In: Proceedings of the 2016 ACM Workshop on Moving Target Defense, MTD 2016, pp. 105–116 (2016). https://doi.org/10.1145/2995272.2995277
12. Vagrant by HashiCorp. https://www.vagrantup.com, https://www.vagrantup.com. Accessed 20 Feb 2021
13. Hong, J.B., Kim, D.S.: Scalable security models for assessing effectiveness of moving target defenses. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 515–526 (2014)
14. Huang, C., Zhu, S., Yang, Y.: An evaluation framework for moving target defense based on analytic hierarchy process. EAI Endorsed Trans. Secur. Saf. **4**, e4 (2018)
15. Huang, Y., Ghosh, A.K.: Introducing diversity and uncertainty to create moving attack surfaces for web services. In: Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S. (eds.) Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats, pp. 131–151 (2011). https://doi.org/10.1007/978-1-4614-0977-9_8

16. Jafarian, J.H., Al-Shaer, E., Duan, Q.: OpenFlow random host mutation: transparent moving target defense using software defined networking. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN 2012, pp. 127–132 (2012). https://doi.org/10.1145/2342441.2342467
17. Kaur, K., Singh, J., Ghumman, N.S.: Mininet as software defined networking testing platform. In: International Conference on Communication, Computing & Systems (ICCCS), pp. 139–42 (2014)
18. Kyi Oo, W.K., Koide, H., Vasconcellos Vargas, D., Sakurai, K.: A new design for evaluating moving target defense system. In: 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), pp. 561–563 (2018)
19. Lyon, G.F.: Nmap network scanning: the official Nmap project guide to network discovery and security scanning. Insecure. Com LLC (US) (2008)
20. MacFarland, D.C., Shue, C.A.: The SDN shuffle: creating a moving-target defense using host-based software-defined networking. In: Proceedings of the Second ACM Workshop on Moving Target Defense, MTD 2015, pp. 37–41 (2015). https://doi.org/10.1145/2808475.2808485
21. Manadhata, P.K.: Game theoretic approaches to attack surface shifting. In: Jajodia, S., Ghosh, A.K., Subrahmanian, V., Swarup, V., Wang, C., Wang, X.S. (eds.) Moving Target Defense II, pp. 1–13 (2013). https://doi.org/10.1007/978-1-4614-5416-8_1
22. MITRE ATT&CK (2019). https://attack.mitre.org/. Accessed 20 Feb 2021
23. Paulos, A., Pal, P., Schantz, R., Benyo, B.: Moving target defense (MTD) in an adaptive execution environment. In: Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop, CSIIRW 2013 (2013). https://doi.org/10.1145/2459976.2460045
24. Duan, Q., Al-Shaer, E., Jafarian, H.: Efficient random route mutation considering flow and network constraints. In: 2013 IEEE Conference on Communications and Network Security (CNS), pp. 260–268 (2013)
25. Rohitaksha, K., Rajendra, A.B.: Analysis of POX and Ryu controllers using topology based hybrid software defined networks. In: Karrupusamy, P., Chen, J., Shi, Y. (eds.) ICSCN 2019. LNDECT, vol. 39, pp. 49–56. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-34515-0_6
26. Sharma, D.P., Kim, D.S., Yoon, S., Lim, H., Cho, J.H., Moore, T.J.: FRVM: flexible random virtual IP multiplexing in software-defined networks. In: 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), pp. 579–587 (2018)
27. Strom, B.E., Applebaum, A., Miller, D.P., Nickels, K.C., Pennington, A.G., Thomas, C.B.: MITRE ATT&CK: design and philosophy. MITRE Product MP, pp. 18–0944 (2018)
28. Vadlamud, S., et al.: Moving target defense for web applications using Bayesian stackelberg games. In: AAMAS 2016 - Proceedings of the 2016 International Conference on Autonomous Agents and Multiagent Systems, pp. 1377–1378 (2016)
29. Van Leeuwen, B.P., Stout, W.M.S., Urias, V.E.: Empirical assessment of network-based moving target defense approaches. In: MILCOM 2016–2016 IEEE Military Communications Conference, pp. 764–769 (2016)
30. VirtualBox. https://www.virtualbox.org/, https://www.virtualbox.org/. Accessed 20 Feb 2021

31. Xu, J., Guo, P., Zhao, M., Erbacher, R.F., Zhu, M., Liu, P.: Comparing different moving target defense techniques. In: Proceedings of the First ACM Workshop on Moving Target Defense, MTD 2014, pp. 97–107 (2014). https://doi.org/10.1145/2663474.2663486
32. Yoon, S., Cho, J.H., Kim, D.S., Moore, T.J., Free-Nelson, F., Lim, H.: Attack graph-based moving target defense in software-defined networks. IEEE Trans. Netw. Serv. Manage. **17**(3), 1653–1668 (2020)
33. Zhuang, R., DeLoach, S.A., Ou, X.: Towards a theory of moving target defense. In: Proceedings of the First ACM Workshop on Moving Target Defense, pp. 31–40 (2014). https://doi.org/10.1145/2663474.2663479