JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

11100 Johns Hopkins Road • Laurel, Maryland 20723-6099

AOS Report No. AOS-18-0389

June 2018

VIRTUE TEST PLAN

Version 1.2

Prepared for: Intelligence Advanced Research Projects Activity (IARPA)

Prepared by: VirtUE Team

Task No.: VJY26

Contract No.: 2017-16110100002S

This Page Intentionally Left Blank

CONTENTS

TABLES

1. INTRODUCTION

JHU/APL is providing support to the Virtuous User Environment (VirtUE) program sponsored by the Intelligence Advanced Research Projects Activity (IARPA). JHU/APL's main efforts are focused along three technical areas: systems engineering, software engineering, and test and evaluation (T&E). As part of JHU/APL's T&E support, JHU/APL is responsible for developing a testbed to support the test and evaluation of performer solutions. This document describes JHU/APL's T&E methodology, including discussions on program requirements, test cases, test metrics, and scoring.

In months 9 and 18, performers shall travel to the government-selected T&E facilities at JHU/APL to undergo examinations that shall provide a midterm and end-of-phase assessment of their solution's performance. For planning purposes, proposers should allow three days for interactive testing at JHU/APL for each examination. Performers are expected to deliver all software and documentation required for JHU/APL to adequately assess their offering to T&E team members prior to examinations. T&E team members are responsible for instrumenting and creating a testing environment within Amazon Web Services (AWS) to assess performers' solutions against the requirements of the program.[1]

1.1 Program Background

As stated in the VirtUE Broad Area Announcement (BAA), "VirtUE seeks to leverage the federal government's impending migration to commercial cloud-based Information Technology (IT) infrastructure and the current explosion of new virtualization and operating system (OS) concepts to create and demonstrate a more secure interactive user compute environment (UCE) than is currently available for government use or is expected to be available in the near future."[1]

- Phase 1 of VirtUE focuses on providing a UCE designed to be more secure than existing VDI systems and capable of functioning as a sensor and defender in a commercial cloud environment. Functionality and performance characteristics of the Phase 1 solution should be comparable to current government UCEs.
- In Phase 2, performers take the technologies and concepts developed in Phase 1 and create novel external analytics and security controls that leverage them. The purpose of this effort is to create dynamic detection and protection capabilities that make the VirtUE user environment more resistant to attacks expected in the commercial cloud while minimizing the costs associated with these capabilities.

In support of Phase 1 of the VirtUE program, JHU/APL is serving as the T&E partner for IARPA and is responsible for conducting rigorous testing of solutions developed by Phase 1 performers. JHU/APL will conduct tests at the midpoint of Phase 1 of the program (month 9) and at the completion of Phase 1 of the program (month 18). Information presented in this document is specific to Phase 1 of the VirtUE program.

[1] Program requirement are enumerated in the VirtUE BAA: https://www.iarpa.gov/index.php/research-programs/virtue/virtue-baa.

1.2 Purpose

This document describes JHU/APL's planned T&E methods for the performers' Virtue system. The test plan covers test cases, test execution, and testing methodology; it also includes details on usability testing of performer solutions and planned testing artifacts, such as preservation of performer logs and data captured from the system under test.

It is important to emphasize that this document addresses a blend of testing and evaluation activities, in spite of the fact that terms like "evaluation" or "benchmarking" are not mentioned explicitly in the document title. Some of the requirements address functional aspects of a Virtue that must be executed correctly to ensure the overall success of a Virtue—for example, the Virtue's ability to function in a standard AWS environment. Other requirements, however, address properties or attributes of a Virtue that are desirable to improve overall performance, such as minimizing the use of memory and other computational resources. The metrics data that is collected will be combined with the collected test data and used to assist the evaluation of the various proposers' solutions. This document is not intended to serve as a final acceptance document.

1.3 Referenced Documents

The following table lists VirtUE related documentation and references that should be examined for additional context on the program and testbed capabilities and operation.

Document	Date	Description
A Method for Calculation of the Resilience of a Space System	November 2013	Paper by Boeing Space and Intelligence Systems defining resilience and introducing a mathematical model used to calculate a system's resilience
An Approach to Measuring A Systems Attack Surface	August 2007	Paper by Carnegie Mellon University describing attack surface measurement techniques
DoD Defense Science Board Task Force Report: Resilient Military Systems and the Advanced Cyber Threat	January 2013	Report identifies and describes 6 threat tiers of cyber actors (https://nsarchive2.gwu.edu/NSAEBB/NSAEBB424/docs/Cyber-081.pdf)
NSA Threat Framework Version 2.0	March 2018	Document identifies a threat coverage matrix defining specific attacks actions mapped to phases of the attack lifecycle (https://www.iad.gov/iad/library/reports/nsa-css-technical-cyber-threat-framework-v1.cfm)
VirtUE Reference Implementation (RI) and Testbed Integration Overview	March 31, 2017	Describes approach to integrating JHU/APL's reference implementation with the VirtUE testbed
VirtUE Testbed Algorithms and Tools	March 31, 2017	Description of algorithms and tools used to support VirtUE test and evaluation
VirtUE Testbed Software Application Programming	March 31, 2017	VirtUE API used to enable interaction between VirtUE testbed and performer solutions

Interface (API)		
Virtuous User Environment (VirtUE) Phase 1 IARPA-BAA-16-12	October 18, 2016	VirtUE Broad Area Announcement

Table 1: Referenced Documents

1.4 Document Structure

This document is organized around four main threads: (1) requirements; (2) design and test metrics; (3) test case generation and test cases; and (4) scoring.

The first thread involves identifying program level requirements that must be satisfied by performer solutions. The second thread introduces and defines metrics that are used to assess performer solutions. The third thread describes the test case generation methodology and introduces a set of functionality, security, and performance test cases. The test cases have been written in a standard and generic manner to facilitate reuse and automation to the greatest extent possible. The fourth thread defines the scoring approach.

2. PROGRAM REQUIREMENTS

The VirtUE Program has established eleven explicit program requirements to guide the design and implementation of proposer solutions. The solutions are evaluated on the extent to which they address these requirements and optimize functionality, performance, and security. Each requirement is important to the success of Phase 1 of the program, and each performer should attempt to satisfy each requirement. The requirements are listed below in Table 2.

ID	Requirement
1	Proposers shall devise a repeatable, automated, secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure.
2	Virtues shall present themselves as atomic, largely immutable entities to other Virtues and external processes. They shall be simpler and more modular than current VDI solutions with a minimized attack surface.
3	Virtues shall be role-based.
4	Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud.
5	Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats.
6	Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior.
7	Virtues shall be capable of invoking and interacting with several legacy applications (including several Windows apps).
8	Virtues shall be capable of securely saving, passing, and retrieving basic state information (hyperlinks, file locations, individual files, default printers, etc.) with each other (Virtue-to-Virtue) and also with a secure Analytics/Control Layer (Virtue-to-Control Layer).
9	Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device.
10	Virtues shall be capable of interacting with their environment in a performant manner.
11	Proposers shall provide methodology and utilities to define, construct, assure, and transport Virtue images.

Table 2: VirtUE Program Requirements

3. TEST EVENTS

The VirtUE program established two formal test events to assess VirtUE performer solutions. The first test event, referred to as the Midterm Test, is scheduled 9 months following program kick-off. The second test event, referred to as the Final Test, is scheduled 18 months following program kick-off. The objective of the test event is to conduct a series of functionality, performance, and security tests against performer solutions to assess their ability to satisfy VirtUE program requirements. Test events are conducted at JHU/APL using a combination of JHU/APL-managed systems and networks and AWS-hosted systems and networks.

3.1 Test Event Schedule

Performer teams have been broken into two groups for midterm testing. The testing event for group 1, consisting of Siege Technologies and Raytheon/BBN, is scheduled for April 30, 2018 through May 25, 2018. The testing event for group 2, consisting of Next Century Corporation and Star Lab, is scheduled for June 4, 2018 through June 29, 2018.

3.2 Pre-Test Event Activities

A test readiness review will be held prior to midterm and final testing. The test readiness review, a three-day event hosted by JHU/APL, is one week prior to testing. During this time, performer teams and T&E team members will review Virtue deployment configurations to ensure that all systems are correctly deployed and operating as expected. The T&E team will also execute a set of functionality, performance, and security tests to verify that tests can be properly executed and that all required data is being collected as expected. At the conclusion of the test readiness review, the VirtUE program manager decides whether to proceed with formal testing.

4. TEST CASE OVERVIEW

Test cases span functionality, security, and performance test categories. Functionality test cases evaluate a system's ability to support a variety of management tasks and user tasks. Security test cases for midterm are concentrated in two areas – attack surface enumeration and malicious insider attacks. Performance test cases concentrate on analyzing two specific Virtue workflows. While there are only two performance test cases for midterm testing, performance metrics will be gathered while executing both functionality and security test cases. The table below describes the number of test cases for each test category.

Test Category	Number of Test Cases
Functionality	35
Security	18
Performance	2

Table 3: Number of test cases per test category

4.1 Functionality Test Cases

This test category focuses on determining if a Virtue provides all required capabilities correctly. These capabilities include the creation and execution of Virtues, the execution of various applications, and the ability to interact with other Virtues and resources.

VirtUE functionality tests are designed to gather data required to evaluate a solution's ability to satisfy the functional aspects of VirtUE requirements. Functionality tests are conducted using two separate methods: automated functional testing through the use of an automated test suite; and hands-on testing performed by the VirtUE T&E team. Functionality tests are conducted to gather metrics on the ability of users to create Virtues per specifications, and then use these Virtues to execute various Windows and Linux applications in support of business operations. The table below lists functionality-oriented tests that are targeted for mid-term testing.

ID	Test Case	Notes
1.1	System Installation	
1.2	System Start	
2.1	Create User	
2.2	Update User Settings	
2.3	Delete User	
2.4	Create Role	
2.5	Assign Role to User	
2.6	Remove Role from User	
2.7	Import Virtue Image	
2.8	Export Virtue Image	
2.9	Verify Storage and Retrieval of User-Persistent Settings	
2.10	Verify Virtue Immutability by Updating Application	
2.11	Log in to Virtue System (not a specific Virtue)	
2.12	Log Off Virtue System (not a specific Virtue)	
2.13	Verify release of Virtue resources	
3.1	Start Virtue	
3.2	Stop Virtue	
3.3	Transfer File to Another Virtue	
3.4	Open URL embedded in Email Virtue in a Web Browser Virtue	
3.5.1	Interact with Windows MS Word	
3.5.2	Interact with Windows MS Excel	
3.5.3	Interact with Windows MS Edge	
3.5.4	Interact with Windows MS Outlook	
3.5.5	Interact with Windows Google Chrome	
3.5.6	Interact with Windows MS Skype	Not tested at midterm
3.5.7	Interact with Windows Command Terminal	
3.5.8	Interact with Windows PowerShell	
3.6.1	Interact with Linux Mozilla Firefox	
3.6.2	Interact with Linux Google Chrome	
3.6.3	Interact with Linux Mozilla Thunderbird	
3.6.4	Interact with Linux Command Terminal	
3.7	Virtue UI enables integration with multiple Virtues	
3.8	Print to networked attached printer	Not tested at midterm
3.9	Retrieve file from shared directory	
3.10	Verify Virtue sensing and protection configuration does not change over time	Not tested at midterm

Table 4: Functionality Test Cases

4.2 Performance Test Cases

This test category focuses on determining elapsed time and computational resources that a solution consumes in order to support its functionality and security requirements. BAA Metric 10.a states the goal for Virtues to minimize the cost of consumed cloud resources; however, the program as a whole also requires solutions to provide "performance characteristics comparable to the current government UCE." This test plan employs techniques to measure the time and resource consumption characteristics of proposer's solutions in various modes: (1) a baseline "Virtue only" mode, where the Virtue is running with only minimal (if any) applications being executed; (2) one or more Virtues executing a sequence of applications and data transfers; and, (3) Virtues being subjected to one or more attacks as mentioned in the security test category. Also, since an operator's experience is a critical aspect of overall Virtue performance, the T&E team will address usability concerns in this test category.

VirtUE performance tests are designed to gather data required to evaluate a solution's responsiveness, resource utilization, scalability, and operational cost. Performance tests are conducted using three separate methods: automated performance testing through the use of an automated test suite, hands-on testing performed by the VirtUE T&E team; and usability performance related testing performed by a group of individuals operating as VirtUE users and/or administrators. It should be noted that the majority of performance data will be gathered while conducting functionality and security tests. The table below identifies specific performance test cases that will be conducted to gather performance data that cannot be collected when executing functionality and security tests. It should also be noted that usability testing requiring a group of individuals operating as VirtUE users will not be conducted at midterm but will be deferred to the final test event.

ID	Test Case	Notes
5.1	Complete Workflow to Define and Construct a New Virtue Image	
5.2	Complete Workflow of Starting a Virtue, Accessing an Application, and Terminating a Virtue	

Table 5: Performance Test Cases

4.3 Security Test Cases

This test category focuses on a Virtue's ability to maintain its integrity and confidentiality in the face of adversarial attacks. The VirtUE program focuses on four primary attack vectors: (1) conventional external attacks, (2) malicious insider attacks, (3) peer guest attacks, and (4) cloud provider memory snooping/alteration-based attacks. This test plan employs a variety of attacks that seek to exploit Virtues, including attempts to exfiltrate information, cause system failures, access otherwise unauthorized resources, and degrade system responsiveness.

Security tests are conducted using two separate methods: automated security testing through the use of an automated test suite and hands-on testing performed by the VirtUE T&E team. The table below lists security-oriented tests that are targeted for mid-term testing.

ID	Test Case	
4.1	Brute-force login attempt	
4.2	Malicious Insider Data Reconnaissance (Tier 1 | Linux)	
4.3	Malicious Insider Data Reconnaissance (Tier 1 | Windows)	
4.4	Malicious Insider Data Reconnaissance (Tier 2 | Linux)	
4.5	Malicious Insider Data Reconnaissance (Tier 2 | Windows)	
4.6	Malicious Insider Data Reconnaissance (Tier 3 | Linux)	
4.7	Malicious Insider Data Reconnaissance (Tier 3 | Windows)	
4.8	Malicious Insider Data Exfiltration (Tier 1 Linux)	
4.9	Malicious Insider Data Exfiltration (Tier 1 Windows)	
4.10	Malicious Insider Data Exfiltration (Tier 2 Linux)	
4.11	Malicious Insider Data Exfiltration (Tier 2 Windows)	
4.12	Malicious Insider Data Exfiltration (Tier 3 Linux)	
4.13	Malicious Insider Data Exfiltration (Tier 3 Windows)	
4.14	External Attack Surface Enumeration	
4.15	Internal Attack Surface Enumeration	
4.16	Verify Virtue image at startup	
4.17	Verify Virtue peer-to-peer communications are encrypted	
4.18	Verify Virtue to Virtue Eco-Systems communications are encrypted	

Table 6: Security Test Cases

5. DESIGN METRICS

Design metrics are those measurements and data for comparisons against the baseline that are assessed against non-runtime aspects of the system. These are metrics taken against the design, stated capabilities, and as-built architecture and may include things like the number of unique indicators capable of being sensed. Sometimes, these measures can be subjective, and in those cases, the subjectivity is governed by a set of discrete rules. We lay out the rules in the sections dealing with these specific metrics.

5.1 Security

Some of the descriptions that follow refer to the "user-accessible Virtue," which is the portion of the system with which the user interacts (e.g., by running an application).

5.1.1 # of sensors

Total number of sensors used to monitor a Virtue. A Virtue *sensor* is a software component that captures events occurring within the Virtue environment. The number of sensors is the total number of software components used to monitor a Virtue.

5.1.2 # of sensors outside the user-accessible Virtue

Total number of sensors outside the user-accessible Virtue. Such a sensor operates "out-of-band" and cannot be modified or compromised by a user or malicious application (including the operating system). An example is a sensor integrated with the hypervisor.

5.1.3 Degree of sensor configurability

Level of configurability of the sensor system

Value	Score	Description
Low	0	Sensors cannot be enabled / disabled
Medium	1	Sensors can be enabled / disabled in groups, but not individually
High	2	Sensors can be enabled / disabled and controlled individually

5.1.4 # of protections

Total number of protections used to defend a Virtue. A Virtue *protection* is a software component that defends against or responds to actions related to an attack. The number of protections is the total number of software components used to defend a Virtue.

5.1.5 # of protections outside the user-accessible Virtue

The total number of protections outside the user-accessible Virtue. Such a protection operates "out-of-band" and cannot be modified or compromised by a user or malicious application (including the operating system). An example is a protection integrated with the hypervisor.

5.1.6 Degree of protection configurability

Level of configurability of the protection system.

Value	Score	Description
Low	0	Protections cannot be enabled / disabled
Medium	1	Protections can be enabled / disabled in groups, but not individually
High	2	Protections can be enabled / disabled and controlled individually

5.1.7 # of unique infrastructure sensors

Total number of sensors in the Virtue Infrastructure level. This metric subsumes "# of unique hypervisor logging options" in the VirtUE BAA. The number of unique hypervisor logging options presumes the use of nested virtualization, which is not a program requirement.

5.1.8 # of unique network sensors

Total number of sensors at the network level.

This metric subsumes "# of unique network logging options" in the VirtUE BAA.

5.1.9 # of unique internal sensors

Internal logging refers to sensing within the user-accessible Virtue.

This metric subsumes "# of unique internal logging options" in the VirtUE BAA.

5.1.10 # of unique protections against AWS hypervisor attacks

As described in the VirtUE BAA, this metric is the number of unique protections against hypervisor-based attacks. A hypervisor attack is an attack originating from the commercial cloud infrastructure.

5.1.11 # of unique protections against network attacks

"The number of unique protections against network […] attacks provided by a Virtue" (VirtUE BAA). A network attack is an attack originating from a malicious user/system using a network path to target a Virtue.

5.1.12 # of unique insider threat protections

"The number of unique insider threat protections provided by a Virtue" (VirtUE BAA). An insider threat protection is a design feature or function that inhibits the actions of a malicious insider.

5.1.13 Communication paths that traverse a trust boundary

TBD (experimental)

6. TEST METRICS

Test categories, test threads, and test assessment criteria aid in the identification, description, and organization of metrics used to characterize and evaluate performer solutions. They also assist the development of the performer-tailored tests that are used to generate and collect data and analyze results for scoring and evaluation. For example, some tests require manual analysis of each performer's system documentation or source code to determine if the system provides a specific capability and, if the system does provide the capability, to determine how to

reconfigure the test environment to assess the capability and to collect test results. The metrics generated by other tests are created via automated means: either by executing pre-configured test scripts and test cases or by manually identifying and executing selected test cases based on the design characteristics of the performer's solution. Later sections of this test plan address those test cases in more detail; this section presents an overall description of Virtue test metrics.

The diversity of the metrics being measured means that the results data is represented in a variety of formats. A brief overview of different result data formats is presented here along with references to program requirements when applicable.

These metrics are those whose data is gathered as a result of running a live test against the system. Metrics such as number of active attacks thwarted are measured as a result of the test case.

6.1 Functional

The functional tests are those that implement an operation required by the user to accomplish enterprise work; for example, the ability to run a word processing application to create intelligence reports.

Such tests result in two metrics collected: a *test result*—a measure of success at implementing the function (e.g., has-word-processor: yes/no)—and *elapsed time*—the time required to complete the designated phase of the function (e.g., time to start the word processor, from initial directive to launch until the time at which the word processor interface is presented to the user).

6.1.1 Test Result (Pass | Fail | Skip / Error*)*

Test results indicate whether the desired function has been executed successfully. There are four values:

- Pass: The performer has implemented the function and the function works as expected
- Fail: The performer has implemented the function, but the function does not work as expected
- Skip: The performer has not implemented the function (as reported by the performer to the test and evaluation team), or the test was not executed due to prior failures.
- Error: The test aborted due to a problem with the test framework.

6.1.2 Elapsed Time

Elapsed time is the time between initiation of some function and a desired, confirmed result. We illustrate this through a few examples of anticipated measures of elapsed time:

- The time to launch an application: the time from initiating a request to start some application to the appearance of the application graphical window on the user's GUI desktop; measurable via a windowing interface monitor registering the mapping of the GUI window
- The time to save a file: the time of a file save being initiated to the time that a file handle close event is detected on the file server

Note that these are only examples; the test team may measure time using other methods or guidance, but it will use consistent mechanisms like the examples above.

The test team plans to measure elapsed time via an automated test script; in certain cases, if the functional test is not scriptable, test team members may measure elapsed time manually. Any such manual measurement will be defined carefully, to make sure the testers know consistently when to start and stop the timers.

The test team will attempt to test every performer and the baseline the same way; therefore, if a coarse-grained estimation of time is made for the performers and also for the baseline, then the measurements will be equally comparable.

6.2 Performance

Certain tasks are measured in terms of the various resources that are consumed by the Virtue, including (but not limited to) computer processing cycles, random access memory, persistent storage, and network bandwidth. In most cases, lower consumption rates will represent more efficient operation, which is the desired goal.

The following descriptions are taken from the AWS CloudWatch documentation.[2] In the descriptions, *instance* refers to a virtual machine being managed by AWS EC2.

6.2.1 CPU Utilization

The percentage of allocated EC2 compute units that are currently in use on the instance. This metric identifies the processing power required to run an application upon a selected instance.

6.2.2 Disk Read Operations

Completed read operations from all instance store volumes available to the instance in a specified period of time.

6.2.3 Disk Read # of Bytes

Bytes read from all instance store volumes available to the instance. This metric is used to determine the volume of the data the application reads from the hard disk of the instance.

6.2.4 Disk Write Operations

Completed write operations to all instance store volumes available to the instance in a specified period of time.

6.2.5 Disk Write # of Bytes

Bytes written to all instance store volumes available to the instance. This metric is used to determine the volume of the data the application writes onto the hard disk of the instance.

[2] https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ec2-metricscollected.html

6.2.6 Network Bytes In

The number of bytes received on all network interfaces by the instance. This metric identifies the volume of incoming network traffic to a single instance.

6.2.7 Network Bytes Out

The number of bytes sent out on all network interfaces by the instance. This metric identifies the volume of outgoing network traffic from a single instance.

6.2.8 AWS Cost

Instance cost (as computed by AWS).

6.3 Security Metrics

The following metrics pertain to the security posture of a Virtue. Metrics 6.3.1 – 6.3.19 may be collected for each test case as an estimate of the Virtue's attack surface. Metrics 6.3.20 – 6.3.26 pertain specifically to security test cases – e.g., the time elapsed from the initiation of an attack to the first sensor event relevant to that attack.

6.3.1 Externally-accessible ports

Externally-accessible ports are those TCP and UDP network ports accessible from a reference point external to the system. Open ports are enumerated using a port-scan all IP addresses associated with the exposed portion of the system.

6.3.2 Internally-accessible ports

Internally accessible ports are those TCP and UDP network ports that are accessible by software components running inside the Virtue. In Linux, this includes any sockets bound to and listening to requests from localhost and any locally-assigned IP address. In Windows, this includes ports opened on localhost and any locally-assigned IP addresses.

6.3.3 Internal communication paths

Internal communications paths are those accessible via inter-process communication (IPC) channels. In Linux, internal communications paths are discovered by executing the ipcs command. In Microsoft Windows, named pipes are a type of IPC.

6.3.4 Exposed system calls

Exposed system calls are the number of system calls available to userspace processes running within the virtual container. The definition of the *exposure* of a system call is a work in progress and will not be ready for midterm testing.

6.3.5 Running processes

This metric is the count of processes running on the system at a given point in time.

6.3.6 Running privileged processes

This metric is the count of processes running on the system at a given point in time, possessing the effective permission set of a super user (root, administrator, or equivalent). Permissions will be assessed at a point in time, through "`ps aux`" or other trusted means.

6.3.7 Kernel modules loaded

This metric is the number of kernel modules currently loaded in the running Linux kernel. This metric applies only to Linux.

6.3.8 Running threads

This is the number of process threads currently running on the system.

6.3.9 Available Services

This is the number of background OS services, called "systems services" in Windows and background processes or "daemons" in Linux, running in the virtual instance.
Although the available services might overlap with the number of privileged processes, a service might run in userspace – e.g., an application that opens a socket to send / receive data.

6.3.10 Installed applications / executables

The number of files on the system either defined as an application to the system or a file with execute permissions enabled.

6.3.11 Resources that can be a target (e.g., files)

The number of files on the system.

6.3.12 Open files

On Linux, the number of allocated file descriptors. On Windows, the number of handles.

6.3.13 Files with SUID (set owner user id)

Executable files that temporarily elevate privilege by running under the file owner's user id. (Applies only to Linux.)

6.3.14 Files with SGID (set group id)

Executable files that temporarily elevate privilege by running under the file owner's group id. (Applies only to Linux.)

6.3.15 User accounts

The number of local user accounts on the system.

6.3.16 User accounts with shell

The number of local user accounts that can log into the system using either a terminal or GUI. (Applies only to Linux.)

6.3.17 Users that can elevate their privileges

On Linux, the number of users with sudo privileges. On Windows, the number of users in the admin group.

6.3.18 Groups that can elevate their privileges

On Linux, the number of groups with sudo privileges.

6.3.19 Credentials in the exposed portion of a Virtue

The number of user credentials (username and password, Kerberos token, etc.) that are stored within the portion of a Virtue that the user interacts with and are considered most susceptible to attack.

6.3.20 # of Attack Phases Detected

The number of attack phases sensed by Virtue sensors. Attack phases and associated attack actions are based on the NSA Threat Framework Version 2.0. Table 7 lists the attack phases identified in the NSA Threat Framework. Note that Phase 0 is not applicable to Virtue testing.

Table 7: NSA Threat Framework (version 2.0)

Phase	Description
0	Administer (Intent/Resource Development)
1	Prepare (Reconnaissance/Staging and Weaponization)
2	Engage (Delivery and Initial Compromise/Exploitation)
3	Propagate/Presence (Persistence, Privilege Escalation, Defense Evasion, and Credential Access)
4	Effect (Monitor/Observation/Exfiltration and Alter/Deceive)

6.3.21 # of Attack Phases Protected

The number of attack phases inhibited by Virtue protections. Attack phases and associated attack actions are based on the NSA Threat Framework Version 2.0. Table 7 lists the attack phases identified in the NSA Threat Framework. Note that Phase 0 is not applicable to Virtue testing.

6.3.22 Time elapsed (attack to first sensed event)

Time elapsed from first attack step to first sensed event collected by a Virtue sensor.

6.3.23 # of unique sensor classes logging the event

Sensor classes include network, infrastructure, and internal.

6.3.24 Relevance to attack action(s)

Ratio of logged entries relevant to attack.

6.3.25 Protection by Attack Phase

The number of attack phases prevented by Virtue protections. Attack phases and associated attack actions are based on the NSA Threat Framework version 2.0 (see Table 7).

6.3.26 Time elapsed (attack to first protect event)

Time elapsed from first attack step to first protection action executed by a Virtue.

7. DATA COLLECTION METHODS

This section details the data collection methods that will be used during testing. There will be multiple sources of data that must be collected for each test. Data collection is necessary to assess performer solutions, verify integrity of performer solutions, and provide evidence that the testbed is performing as expected. The data will be archived for auditing purposes in case of a dispute about a test result.

7.1 Network Ground Truth

A minimum of two packet capture devices will be active within the test framework. The near-side router (packet capture device operating within JHU/APL's Virtue test network) captures all network traffic leaving the test framework at JHU/APL, and the far-side router (operating within AWS test environment) captures all network traffic leaving the performer's VPC. The packet captures provide ground truth as test events occur. For example, they provide evidence that one resource contacted or attempted to contact another resource. This ground truth is critical for validating that network attacks are occurring, in addition to many other types of events such as encryption of Virtue to Virtue communications, encryption of Virtue to Virtue Eco-System Services, and monitoring of Virtue traffic that egresses AWS VPC boundary.

Additional packet capture devices may be deployed depending upon the performer's network architecture. For instance, a packet capture device may need to be deployed to capture network traffic between two Virtues to validate that a peer attack is occurring.

The packet capture (PCAP) logs will be stored as artifacts for each test case that requires network data collection.

7.2 Amazon CloudWatch

Amazon CloudWatch is a monitoring service for Amazon Web Services (AWS). CloudWatch stores metrics for different resources in AWS. Example CloudWatch metrics include network flow rates, disk read and write rates, and CPU utilization. CloudWatch data will be collected for all tests. The collected data may be used to verify activity as well as to monitor resource usage for performance analysis.

7.3 Virtue Sensor Data

Per Virtue design requirements, Virtue solutions are expected to gather various types of sensor data. Virtue sensors are expected to collect information that can be used to identify malicious behavior, compromised systems, and other evidence that can be used to assess the security state of a Virtue. Sensor data is critical for Phase II of the Virtue program, as this information will be used to support advanced detection analytics. At the conclusion of each security test, performer's sensor data will be collected and analyzed to determine the effectiveness of a performer's sensing apparatus.

8. TEST READINESS

The testbed and system under test (SUT) present a complex test and evaluation environment with many interdependencies that must be validated prior to each test event. To ensure accurate test results, each test event must be performed only after the testbed has been calibrated and found ready to perform the test. This section identifies processes and procedures employed by the T&E team in order confirm test execution readiness.

8.1 Security and Safety of Test Network

Maintaining the security and integrity of the test environment and network is of critical importance. Network isolation of the testbed network is provided through a combination of local virtual network isolation and VPN connectivity to the AWS VPC hosting the SUT. It is imperative that the test network remain isolated from external networks to protect external networks from potentially harmful effects of security testing. The procedures to ensure the testbed network has been properly isolated are as follows:

- **DNS is local only:** Verify that all testbed systems' DNS is configured to query the testbed DNS
- **Local Routing:** Verify that testbed traffic is routed within the VPN/VPC connection only, and remove default routes as needed
  Traffic is local to VPN: Verify that traffic is local to the VPN by running tcpdump on testbed hypervisor's management interface. No VPN traffic should be detected.

8.2 Timing

Precise timing is critical to the successful execution of tests and test data collection. All testbed components and the SUT must be synchronized to the same clock source to ensure that data artifacts from a test event can be correlated for an accurate history of the test event. The

procedures used to verify accurate time coordination of both T&E and SUT components are as follows:

- **Linux Testbed Systems:** Run ntpq -p. Verify the server selected with a "*" is the testbed NTP server
- **Windows Testbed Systems:** Run w32tm /monitor to verify testbed NTP server is listed

8.3 Testbed Readiness

The testbed will be validated using the following pre-test checklist:

- Verify sufficient free disk space on C2 and scoring DB to record test event
- Verify time is synchronized across all testbed and SUT components
- Verify testbed to SUT connectivity
- Verify that network traffic capture instrumentation can see traffic at collection points
- Execute dry-run of test event to validate that testbed components are functioning properly
- Verify that testbed network is isolated from external access

9. TEST CASES

This section provides a detailed description of each test case, including its preconditions, execution steps, and post-conditions.

The approach to test case generation focused on identifying generic tests cases that would be applicable to all performer solutions. With the objective of consistency and repeatability with tests across performer solutions, it is imperative to define test cases that require minimal customization or tailoring in order to support a specific performer solution. However, due to the unique approaches selected by performers, some test cases require modification to the testbed to properly exercise a solution's capabilities and gather the data required for evaluation.

Test cases define a series of specific actions required to assess a solution's capabilities against defined Virtue metrics. These actions identify the required interactions that need to be supported in order to execute the test case. This allows a consistent set of test cases and specific test instructions to be executed against each performer's solution. Other advantages of this approach include repeatability and test automation. Some of the test instructions associated with test cases can be encoded into an executable form that can be invoked by the testbed's test automation tools. This automation allows for a level of precision in performing the exact system interactions when repeating each test case, allowing each test case to be executed multiple times and the data from each run to be collected and analyzed.

Unless otherwise noted, the follow metrics are collected for every test case:

- Test Result
    - Pass: Test case executes without error and all post-conditions satisfied
    - Fail: Test case does not satisfy post-conditions following execution
    - Skip: Functionality is not (currently) available in the performer's solution or is not believed to work correctly due to prior test failures

    - Error: Test was not completed successfully (e.g., due to an error in the test framework)
- Elapsed time for test to complete (captured as wall clock time)
- Resource Utilization includes data regarding resources consumed by an environment operating on AWS using the available AWS resource metrics:
    - CPU utilization
    - Disk read operations
    - Disk write operations
    - Disk read bytes
    - Disk write bytes
    - Network bytes in
    - Network bytes out
- AWS cost

If a test case has a unique interpretation of these metrics, then that information is provided as part of its description. In addition, any metrics that are unique to a particular test case are noted in the test case descriptions that follow.

9.1 System Install and Start

System install and start test cases are used to verify the installation, configuration, and operation of the Virtue-supporting infrastructure on AWS. The Virtue-support infrastructure consists of all services and components that must be operational to support the deployment of Virtues. Examples of Virtue-supporting infrastructure include authentication services, Virtue life-cycle services, and Virtue administration services. Once installation and operation are confirmed, the Virtue system is in a state capable of supporting deployment and interaction with Virtues by users.

9.1.1 System Installation

Test case verifies that the Virtue system has been properly installed, including confirmation that all supporting services have been installed, AWS networking has been properly configured, and the Virtue system is ready to be activated or started.

Test Case Id: 1.1

Test Category: Functional

BAA Requirements Addressed

- Requirement 1: Proposers shall devise a repeatable, automated secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure.

Preconditions

- Performer account on AWS
- Logged in to performer system as Virtue System Owner

Steps

1. With performer guidance, T&E team examines system deployed in AWS to verify all components have been installed and configured correctly

Postconditions
- Network connectivity to T&E testbed confirmed
- Network connectivity to public Internet (if needed) confirmed

9.1.2 System Start

Test case verifies that the Virtue system and all supporting services have been started and the system is operational and ready for use, including all Virtue-related services operating on AWS as well as Virtue components operating on the user-accessible Virtue endpoint.

Test Case Id: 1.2

Test Category: Functional

BAA Requirements Addressed
- Requirement 1: Proposers shall devise a repeatable, automated secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure.

Preconditions
- Logged in to performer system as Virtue System Owner

Steps
1. With performer guidance, T&E team performs actions to start all system services and confirms that the system is operating correctly

Postconditions
- The Virtue system is running and ready for users to log in.

9.2 Virtue, User, and Role Creation

9.2.1 Create user

Test case verifies the ability to interact with the Virtue administrative interface to create a new Virtue user account with default privileges.

Test Case Id: 2.1

Test Category: Functional

BAA Requirements Addressed
- Requirement 3: Virtues shall be role-based.

Preconditions
- An administrative user has been created

Test Steps

1. Log in to the Virtue system as an administrative user
2. Create a new user account with default privileges
3. Log out of the administrative user account
4. Log in to the Virtue system as the newly created user
5. Verify new user account has default privileges
6. Log out of the user account

Postconditions

- Virtue system contains the newly created user account with default privileges

9.2.2 Update user settings

Test case verifies the ability of a Virtue administrator to alter user-specific settings such as permissions and the roles assigned to the user.

Test Case Id: 2.2

Test Category: Functional

BAA Requirements Addressed

- Requirement 3: Virtues shall be role-based

Preconditions

- An administrative user has been created
- A test user has been created
- A test role has been created
- The test user does not have access to the test role
- The test role contains at least one test application that is not in any role currently assigned to the test user

Steps

1. Log in to the Virtue system with the test user account
2. Verify the test user does not have access to the test role and cannot start the test application
3. Log out of the test user account
4. Log in to the Virtue system administrative interface as an administrative user
5. Assign the test role to the test user
6. Log out of the administrative user account
7. Log in to the Virtue system with the test user account
8. Verify the test user has access to the test role and can start the test application
9. Log out of the test user account

Postconditions

- The test user now has access to the test role and test application

9.2.3 Delete user

Verify a Virtue administrator's ability to delete a user account from the system.

Test Case Id: 2.3

Test Category: Functional

BAA Requirements Addressed
- Requirement 3: Virtues shall be role-based

Preconditions
- An administrative user has been created
- A test account is created and maintained within the Virtue System user store

Steps
1. Log in to the Virtue system using the test user account
2. Log out of the test user account
3. Log in to the Virtue system as an administrative user
4. Delete the test user account
5. Log out of the administrative user account
6. Attempt to log in to the Virtue system using the previously deleted user
7. Verify test user account cannot log in

Postconditions
- The deleted test user account should no longer be accessible for users to use.

9.2.4 Create role

Test case verifies the Virtue administrator interface can be used to create a new Virtue role and that the new role can be assigned to users.

Test Case Id: 2.4

Test Category: Functional

BAA Requirements Addressed
- Requirement 3: Virtues shall be role-based

Preconditions
- An administrative user has been created
- A test account has been created
- A list of applications is available to be compiled into a role in the Virtue system

Steps
1. Authenticate to the Virtue system as an administrative user
2. Create a new role consisting of a subset of the available applications

3. List available roles to assign to users
4. Verify the newly created test role is in the previous list
5. Log out of the administrative user account

Postconditions

- The newly created test role exists and is available to assign to users

9.2.5 Assign role to user

Test case verifies a Virtue administrator's ability to assign a new role to a user

Test Case Id: 2.5

Test Category: Functional

BAA Requirements Addressed

- Requirement 3: Virtues shall be role-based

Preconditions

- An administrative user has been created
- A test account has been created
- A test Virtue role has been created

Steps

1. Authenticate to the Virtue system as an administrative user
2. Create a new role consisting of a subset of the available applications
3. Log out of the administrative user account
4. Log in to the Virtue system as a test user
5. Confirm that the newly created role is not available
6. Log out the Virtue test user
7. Authenticate to the Virtue system as an administrative user
8. Edit user settings for the test user to assign the newly created role to the user
9. Log out of the administrative user account
10. Log in to the Virtue system as the test user
11. Confirm that the new created role is available to the user
12. Log out of the test user account

9.2.6 Remove role from user

Test case verifies a Virtue administrator's ability to remove a role from a user.

Test Case Id: 2.6

Test Category: Functional

BAA Requirements Addressed

- Requirement 3: Virtues shall be role-based

Preconditions

- An administrative user has been created
- A test account has been created
- Test Virtue roles have been created and assigned to the test user

Steps

1. Log in to the Virtue system using the test account
2. Confirm that the test Virtue role is available
3. Log out the Virtue test user
4. Authenticate to the Virtue system as an administrative user
5. Remove the test role from the user's list of available roles
6. Log out the Virtue administrator
7. Log in to the Virtue system as the test user
8. Confirm that the test role is no longer available to the user
9. Log out of the test user account

9.2.7 Import image

Test case verifies the import of a published Virtue image into a Virtue environment.

Test Case Id: 2.7

Test Category: Functional

BAA Requirements Addressed

- Requirement 11: Proposers shall provide methodology and utilities to define, construct, assure, and transport Virtue images

Preconditions

- An administrative user has been created
- Availability of a Virtue image

Steps

1. Authenticate to the Virtue system as an administrative user
2. Upload a Virtue image to the Virtue image data store/repository
3. Log out of the administrative user account

Postconditions

- A Virtue image was uploaded into the Virtue environment

9.2.8 Export image

Test case verifies the export of a Virtue image by downloading to a user defined location (e.g., file store or removable media)

Test Case Id: 2.8

Test Category: Functional

BAA Requirements Addressed

- Requirement 11: Proposers shall provide methodology and utilities to define, construct, assure, and transport Virtue images

Preconditions

- An administrative user (with access to the Virue image management capability) has been created

Steps

1. Authenticate to the Virtue system as an administrative user
2. Exercise the export function by referencing the target Virtue image
3. Log out of the administrative user account

Postconditions

- The Virtue image has been downloaded to the specified location
- The Virtue image uses the specified package format
- The Virtue image includes a version number
- The Virtue image has an associated hash digest
- The Virtue image metadata describes the associated role, sensors, and protections

9.2.9 Verify storage and retrieval of user-persistent settings

Test verifies a user's ability to create persistent settings in a manner that does not conflict with the immutability property of a Virtue.

Test Case Id: 2.9

Test Category: Functional

BAA Requirements Addressed

- Requirement 8: Virtues shall be capable of securely saving, passing, and retrieving basic state information (hyperlinks, file locations, individual files, default printers, etc.) with each other (Virtue-to-Virtue) and also with a secure Analytics/Control Layer (Virtue-to-Control Layer)

Preconditions

- A test user account has been created
- The test user account has access to a web browser Virtue containing Google Chrome within the presentation interface

Steps

1. Authenticate to the Virtue system using the test user account
2. Launch the web browser Virtue containing Google Chrome
3. In the address bar, input the URL https://www.iarpa.gov and navigate to that site
4. Select the **Bookmarks** Icon in the Address Bar and
    a. In the **Name** Field input: *virtue_test*

   b. In the **Folder** Field input: *Bookmarks Bar*
5. Select the **Done** button
6. Close the Virtue web browser session
7. Log out of the test user account
8. Authenticate to the Virtue system using the test user account
9. Using the presentation interface, launch the web browser Virtue containing Google Chrome
10. Select the *virtue_test* bookmark from the bookmarks bar
11. Verify that this user setting was retrieved and navigated to https://www.iarpa.gov
12. Log out of the test user account

Postconditions

- A user-persistent bookmark setting is stored while preserving the Virtue's immutability

9.2.10 Verify immutability by updating application

Test case verifies that the immutability property of the Virtue is maintained by preventing updates to applications from surviving a reboot of the Virtue. A Virtue user will initiate an application update and verify the update was successful. Next, the user will terminate the Virtue session and then re-launch the Virtue to confirm that the software update was not persisted.

Test Case Id: 2.10

Test Category: Functional

BAA Requirements Addressed

- Requirement 2: Virtues shall present themselves as atomic, largely immutable entities to other Virtues and external processes. They shall be simpler and more modular than current VDI solutions with a minimized attack surface

Preconditions

- A test user account has been created
- The test user account has access to a web browser Virtue containing an outdated version of Firefox

Steps

1. Authenticate to the Virtue system as the test user
2. Launch the web browser Virtue with the outdated version of Firefox
3. From the Firefox Menu Bar, Select Help > About Firefox
4. Note the current version of Firefox
5. Select the Restart to update Firefox button
6. Verify that the Firefox application was updated
7. Log out of the test user account and terminate the Virtue session
8. As the test user account, authenticate to the Virtue system
9. Launch the web browser Virtue that supports Firefox
10. From the Firefox Menu Bar, Select Help > About Firefox
11. Confirm that the version of Firefox matches the original, outdated version number
12. Log out of the test user account

Postconditions

- The version of Firefox is the same as the version reported by the system prior to the application update

9.2.11 Log in to system (not a specific Virtue)

Test case verifies a user's ability to log in to the Virtue System using the Virtue endpoint application. Logging in to the Virtue System does not imply launching an actual Virtue. Instead, logging to the Virtue System allows the user to access (or browse) Virtues that are available to them.

Test Case Id: 2.11

Test Category: Functional

BAA Requirements Addressed

- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device

Preconditions

- A test user account has been created

Steps

1. An end user launches the presentation interface from the Virtue endpoint
2. An end user submits to the presentation interface the correct set of identity credentials for the test user account

Postconditions

- The test account is granted access to the Virtue system

9.2.12 Log off of system (not a specific Virtue)

Test case verifies that a Virtue user is able to log off of the Virtue System.

Test Case Id: 2.12

Test Category: Functional

BAA Requirements Addressed

- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device

Preconditions

- A test user account has been created
- The test user account has an active, authenticated session with the Virtue System

Steps

1. The end user invokes the logout capability from within the Virtue endpoint presentation interface
2. The end user invokes another instance of the presentation interface from the Virtue endpoint
3. Verify that no session state information carried over from the initial authenticated session (from Step 1) and into the newly launched presentation interface instance (from Step 2)

Postconditions:

- The test account is able to terminate their current session, which results in the need for the user re-authenticate to obtain authorized resource access

9.2.13 Verify release of Virtue resources

Test case verifies that when a user logs out of a Virtue and terminates a running Virtue, AWS resources associated with the Virtue are released and no additional costs for supporting the operation of this Virtue are incurred.

Test Case Id: 2.13

Test Category: Functional

BAA Requirements Addressed

- Requirement 10: Virtues shall be capable of interacting with their environment in a performant manner

Preconditions

- A test user account has been created

Steps

1. Authenticate to the Virtue system using the test user account
2. Launch the web browser Virtue
3. Log out of the test user account and terminate the Virtue session
4. Access the appropriate AWS or Virtue management service to confirm the Virtue is no longer active and no additional costs for supporting the operation of this Virtue will be incurred

Postconditions

- Virtue has been successfully terminated and no additional charges for this Virtue are incurred

9.3 Virtue Interaction

9.3.1 Start a Virtue

Test case verifies a user's ability to start a Virtue from the Virtue endpoint interface and access applications residing within Virtue.

Test Case Id: 3.1

Test Category: Functional

BAA Requirements Addressed:

- Requirement 1: Proposers shall devise a repeatable, automated secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure
- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device

Preconditions

- The test user account has an active, authenticated session with the Virtue system
- Role-based Virtue(s) are authorized for use to the test account

Steps

1. Verify that the end user is only provided the set of Virtues that he or she is authorized to start/launch from within the presentation interface
2. The end user invokes the trigger(s) from within the presentation interface to start/activate an authorized Virtue
3. Verify that the end user can interact with the launched Virtue by conducting a basic function using an application hosted from within the Virtue

Postconditions

- The test user account is able to launch an authorized Virtue and is able to interact with it using the presentation interface.

9.3.2 Stop Virtue

Test case verifies that Virtue lifecycle operations can be carried out by the Virtue user interface via the Virtue endpoint

Test Case Id: 3.2

Test Category: Functional

BAA Requirements Addressed

- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device
- Requirement 10: Virtues shall be capable of interacting with their environment in a performant manner

Preconditions

- The test user account has an active, authenticated session with the Virtue System
- Role-based Virtue(s) are authorized for use to the test account
- A user's role-based Virtue has been started and is in an active state

Steps

1. Verify that the end user is only provided the set of Virtues that he or she is authorized to stop/terminate from within the presentation interface
2. The end user invokes the trigger(s) from within the presentation interface to stop/terminate an authorized Virtue
3. Verify that the target Virtue selected transitioned from an Active and into a Terminated state

Postconditions

- User's running Virtue has been shutdown and terminated

9.3.3 Transfer file to another Virtue

Test case verifies that authorized Virtues can transfer files between Virtues

Test Case Id: 3.3

Test Category: Functional

BAA Requirements Addressed

- Requirement 8: Virtues shall be capable of securely saving, passing, and retrieving basic state information (hyperlinks, file locations, individual files, default printers, etc.) with each other (Virtue-to-Virtue) and also with a secure Analytics/Control Layer (Virtue-to-Control Layer)

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System
- The test user account has access to two separate Virtues that are authorized to transfer a file
- A file (which will be transferred) exists within the designated source Virtue

Steps

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the two Virtues that are authorized for file transfer
3. Reference the target file to be transferred from the source Virtue using the transfer mechanism
4. Select the destination Virtue using the transfer mechanism
5. Execute the transfer using a secure protocol
6. Verify the target file reached the destination Virtue and can be accessed

Postconditions

- The target file is either transferred (or shared) between the two authorized Virtues using a secure transport protocol.
- Examine Virtue sensor data to confirm that file transfer action from one Virtue to another was captured by Virtue sensor(s)

9.3.4 Open URL embedded in email in browser

Test case verifies that a URL received within a message in an Email Virtue can be passed, opened, and rendered within a separate but authorized Web Browser Virtue.

Test Case Id: 3.4

Test Category: Functional

BAA Requirements Addressed

- Requirement 8: Virtues shall be capable of securely saving, passing, and retrieving basic state information (hyperlinks, file locations, individual files, default printers, etc.) with each other (Virtue-to-Virtue) and also with a secure Analytics/Control Layer (Virtue-to-Control Layer)

Preconditions

- A test user account is created & maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System
- The test user account has access to two separate Virtues, 1 for Email and 1 for Web Browsing
- A message exists with a URL within the Email Virtue

Steps

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the Email Virtue and the Web Browser Virtue
3. Within the Email Virtue, open the target message that contains the URL
4. Within the transfer mechanism, reference the email URL and select the Web Browser Virtue as the destination Virtue
5. Within the Web Browser Virtue, navigate to the resource specified by the transferred URL

Postconditions

- The transferred URL from the Email Virtue is accessed from within the Web Browser Virtue
- Examine Virtue sensor data to confirm that file transfer action from one Virtue to another was captured by Virtue sensor(s)

9.3.5 Interact with Windows applications (e.g., Microsoft Word, Microsoft Excel, Microsoft Edge, Google Chrome, Microsoft Outlook, Skype, command terminal, and PowerShell)

Test case verifies that a user can interact with a Virtue and perform work functions for the following Windows applications as required by the VirtUE program. The required applications include: Microsoft Word, Microsoft Excel, Microsoft Edge, Microsoft Outlook, Skype, Command Terminal, and PowerShell.

Note: Skype will not be tested during midterm T&E.

Test Case Id: 3.5

Test Category: Functional

BAA Requirements Addressed

- Requirement 1: Proposers shall devise a repeatable, automated secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure
- Requirement 7: Virtues shall be capable of invoking and interacting with several legacy applications (including several Windows apps)

Preconditions

- A test user account is created & maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System
- The test user account has authorized access to Virtues that contain the necessary Windows applications
- Recipient test accounts are generated for Microsoft Outlook email messages and Skype calls

Steps

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the Virtue containing the target Windows application
3. Conduct a work function that is appropriate for each Windows application Virtue
4. See the Test Customizations Section for work function details by application
5. Using the presentation interface, stop the current Windows application Virtue
6. As the test user account, log off from the Virtue system

Test Customizations

Tailor the work functions relating to Step 3 for the following Windows applications:

- Microsoft Word – Create, Update with Text, and Save a Test Word Document File
- Microsoft Excel – Create, Update with Pivot Table, and Save a Test Excel Spreadsheet
- Microsoft Edge – Navigate to an Internet Website
- Microsoft Outlook – Create an email message and send it to the recipient test account
- Skype – Generate a Skype Call with the recipient test account
- Command Terminal – List the files and subdirectories under the C:\ drive
- PowerShell – Create and execute a “Hello World” script

Postconditions

1. The test user is able to successfully accomplish the work function designated for each Windows Application Virtue

9.3.6 Interact with Linux applications (e.g., Mozilla Firefox, Google Chrome, Mozilla Thunderbird, and command terminal)

Test case verifies that a user can interact with a Virtue and perform work functions for the following Linux applications as required by the VirtUE program. The required applications include: Mozilla Firefox, Google Chrome, Mozilla Thunderbird, and a Command Terminal.

Test Case Id: 3.6

Test Category: Functional

BAA Requirements Addressed

- Requirement 1: Proposers shall devise a repeatable, automated secure means of storing, launching, and interacting with their Virtues within the AWS infrastructure
- Requirement 7: Virtues shall be capable of invoking and interacting with several legacy applications (including several Windows apps)

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System
- The test user account has authorized access to Virtues that contain the necessary Windows applications
- Recipient test account is generated for Mozilla Thunderbird

Steps

For each of the Linux Application Virtues, conduct the following:

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the Virtue containing the target Linux application
3. Conduct a work function that is appropriate for each Linux application Virtue
4. See the Test Customizations Section for work function details by application
5. Using the presentation interface, stop the current Linux application Virtue
6. As the test user account, log off from the Virtue system

Test Customizations

Tailor the work functions relating to Step 3 for the following Windows applications:

- Mozilla Firefox – Navigate to an Internet Website
- Google Chrome – Navigate to an Internet Website
- Mozilla Thunderbird - Create an email message and send it to the recipient test account
- Command Terminal – List the files and subdirectories under root (e.g., /)

Postconditions

- The test user is able to successfully accomplish the work function designated for each Linux Application Virtue

9.3.7 Virtue UI enables integration with multiple Virtues

Test case verifies that the Virtue user interface allows a user to interact with 6 different Virtues.

Test Case Id: 3.7

Test Category: Functional

BAA Requirements Addressed

- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System
- The test user account has authorized access to a minimum of (6) unique Virtues

Steps

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the first of 6 Virtues
3. Continue launching Virtues until all 6 are started and active
4. Toggle between each active Virtue and conduct a basic work function that corresponds to the current Virtue
5. Using the presentation interface, stop each of the 6 active virtues
6. As the test user account, log off from the Virtue system

Postconditions

- The test user was able to successfully authenticate into the Virtue system and launch, interact, and stop a minimum of 6 authorized Virtues.

9.3.8 Print to Networked Attached Printer

Test case verifies that a Virtue operating in AWS is able to print documents to an on-premise corporate printer.

Note: Test case not executed during midterm T&E.

Test Case Id: 3.8

Test Category: Functional

BAA Requirements Addressed

- Requirement 7: Virtues shall be capable of invoking and interacting with several legacy applications (including several Windows apps).
- Requirement 10: Virtues shall be capable of interacting with their environment in a performant manner.

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user has launched a Virtue that contains MS Word

- A networked attached printer is accessible from the user's Virtue

Steps

1. As the test user account, authenticate to the Virtue system
2. Using the presentation interface, launch the MS Word application
3. Enter text into the blank word document – "Test 1, 2, 3."
4. Select print document from the Word application window
5. As the test user account, log off from the Virtue system

Postconditions

- The word document was successfully printed to the networked printer.

9.3.9 Retrieve File from Shared Directory

Test case verifies a Virtue's ability to access and retrieve a file residing on a shared directory. This shared directory could be located on an on-premise corporate file share.

Test Case Id: 3.9

Test Category: Functional

BAA Requirements Addressed

- Requirement 7: Virtues shall be capable of invoking and interacting with several legacy applications (including several Windows apps).
- Requirement 10 - Virtues shall be capable of interacting with their environment in a performant manner.

Preconditions

- A test user account is created & maintained within the Virtue System user store
- A Virtue with access to a networked shared folder exists.
- Networked shared folder has read-only access

Steps

1. As the test user account, authenticate to the Virtue system
2. Launch a Virtue that has access to a networked shared folder
3. Navigate to the networked shared folder and download the available file
4. Open the downloaded file and verify the contents of the file
5. Write to file and save file locally within Virtue (not in shared folder)
6. As the test user account, log off from the Virtue system

Test Customizations

1. As the test user account, authenticate to the Virtue system
2. Launch a Virtue that has access to a networked shared folder
3. Navigate to the networked shared folder and open the available file
4. Attempts to write to file and save file
5. As the test user account, log off from the Virtue system

Postconditions

- The downloaded file now exists in the user's Virtue
- Virtue sensor(s) captured user-initiated action to retrieve file from external file share
- Test Customization Only: User attempt to write to file stored on shared folder are denied due to read-only permissions when executing test customization
- Test Customization Only: Virtue sensor(s) captured user-initiated action to write to file in shared folder

9.3.10 Verify Virtue Sensing and Protection Configuration does not Change Over Time

Test case verifies a Virtue's sensing and protection configuration remains intact during use of the Virtue.

Test Case Id: 3.10

Test Category: Functional

BAA Requirements Addressed

- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats.
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior.

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System

Steps

1. Verify the Virtue's sensing and protection configuration
2. Shutdown the Virtue
3. Restart the Virtue
4. Verify the Virtue's sensing and protection configuration matches the configuration captured during step 1

Test Customizations

1. Verify the Virtue's sensing and protection configuration after x minutes of uptime. Instead of shutting down and restarting the Virtue, leave the Virtue running for x minutes. Check the sensing and protection configuration at the start of the test and again after x minutes.

Postconditions: N/A

9.4 Security

In addition to the metrics previously noted, the security tests cases also collect the following information:

- Attack Vector
  - External
  - Internal
  - Peer-to-Peer
  - Infrastructure
- Detection by Attack Phase
- Protection by Attack Phase
- *Number of attack steps sensed (experimental for midterm)*
- *Number of attack steps prevented due to protections (experimental for midterm)*
- Threat Tier
- Relevance
- Time Elapsed – Attack to first sensed event
- Time Elapsed – Attack to first protect event
- Number of unique sensor classes logging event

If a test case has a unique interpretation of these metrics, then that information will be provided as part of its description. In addition, any metrics that are unique to a particular test case will be noted in the descriptions that follow.

Unless indicated in the test case, the following definitions are used to determine the result of each security test case:

- Pass: Collection of at least one event by Virtue sensor pertaining to attack action(s) or at least one protection inhibiting an attack action
- Fail: No events pertaining to attack action(s) collected and no attack actions inhibited
- Error: Test was not completed successfully
- Skip: Functionality is not available

9.4.1 Brute-force login attempt

Test case simulates actions performed by a malicious external attacker to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue and assess the external attack surface of the Virtue. This test case is performed on Virtues that are either Windows or Linux and are connected to the network. The SSH aspects of this test will likely not apply to Windows Virtues. It should be noted that the intent of the test cases is to evaluate a Virtue sensor's ability to detect a brute-force login attempt, not actually succeed with a brute-force attack.

Test Case Id: 4.1

Test Type: Security

BAA Requirements Addressed:

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions

- A Virtue has been created that has network access and an IP address that is reachable from the T&E system.

Steps

1. Ground truth collection is started
2. nmap port scan is performed against the Virtue's IP address
3. Attempt to enumerate users on an OpenSSH server (CVE-2006-5229)
4. Brute-force SSH login attack is executed, using a list of common usernames and passwords
5. Disconnect from target system

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.2 Malicious Insider Data Reconnaissance (Tier 1 | Linux)

Test case simulates actions performed by a malicious insider with authorized access to the Virtue to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), examine available resources (e.g., accessible files) and to test network connectivity to a remote host. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 1 adversary.

Test Case Id: 4.2

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud.
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats.
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior.

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.3 Malicious Insider Data Reconnaissance (Tier 1 | Windows)

Test case simulates actions carried out by malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Windows operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), and examine available resources (e.g., accessible files). Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 1 adversary.

Test Case Id: 4.3

Test Type: Security

BAA Requirements Addressed

- Requirement 4 - Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud

- Requirement 5 - Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6 - Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions

Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.4 Malicious Insider Data Reconnaissance (Tier 2 | Linux)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), examine available resources (e.g., accessible files) and to test network connectivity to a remote host. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 2 adversary.

Test Case Id: 4.4

Test Type: Security

BAA Requirements Addressed:

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats

- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metric:

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
  6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application and standard Linux tools. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performer's Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

1. T&E team launches malicious insider Linux reconnaissance threat tier 2 test
   a. Ground truth collection is started
   b. T&E Attack Asset (e.g. Metasploit) is executed and creates a reverse shell handler and subsequent connection to target system running shell code
2. T&E team copies the following files to the target system via Attack Asset connection
   a. Documents: Principia, Calculus Made Easy, General and Special Relativity, Thermo Dynamics, and Dracula
   b. Script: Linux Tier 2 Reconnaissance
3. Execute Linux Tier 2 Reconnaissance script that was uploaded to the target system in step 2
   a. Execute "id" command
   b. Execute "groups" command
   c. Execute "w" command
   d. Execute "ps" command
   e. Execute "lsmod" command
   f. Execute "mount" command
   g. Execute "ls" command against user's home
   h. Execute "find /home/<user> -name *.pdf"
   i. Execute "find /home/<user> -name *.doc"
   j. Execute "find / -perm /u=s"
   k. Execute "find / -perm /g=s"
   l. Execute "netstat" and look for LISTEN (open ports)
   m. Execute "nc –vz 8.8.8.8 53 –w 1 || ping -c 1 8.8.8.8"
   n. Execute "nc –vz <attack host> 22 –w 1 || ping -c 1 <attack host>"
   o. Clear the user's ~/.bash_history file
4. Files that were placed on target system are removed
5. Attack Asset disconnects from target system

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.5 Malicious Insider Data Reconnaissance (Tier 2 | Windows)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Windows operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), examine available resources (e.g., accessible files) and to test network connectivity to a remote host. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 2 adversary.

Test Case Id: 4.5

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.6 Malicious Insider Data Reconnaissance (Tier 3 | Linux)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), examine available resources (e.g., accessible files) and to test network connectivity to a remote host. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 3 adversary.

Test Case Id: 4.6

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application and standard Linux tools. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performer's Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.7 Malicious Insider Data Reconnaissance (Tier 3 | Windows)

Test case simulates actions carried out by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on

Virtues that support Windows operating systems. Malicious insider performs actions within a Virtue to obtain system level information (e.g., groups, users, uptime), examine available resources (e.g., accessible files) and to test network connectivity to a remote host. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 3 adversary.

Test Case Id: 4.7

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system

Steps

1. T&E team launches malicious insider Windows reconnaissance threat tier 3 test
   a. Ground truth collection is executed
   b. Metasploit is executed and creates a reverse shell handler and subsequent connection to target system running shell code
2. T&E team copies the following files to the target system via Metasploit connection
   a. Documents: Principia, Calculus Made Easy, General and Special Relativity, Thermo Dynamics, and Dracula
   b. Script: Tier 3 Reconnaissance
3. T&E team executes Tier 3 Windows Reconnaissance Metasploit meterpreter script, which performs the following actions on the target system.
   a. Run post/multi/gather/env
   b. Run post/windows/gather/checkvm
   c. Run killav

d. Execute “getuid” command
e. Execute “getsid” command
f. Execute “ls” command
g. Execute “ipconfig” command
h. Run post/windows/gather/forensics/enum_drives
i. Run post/windows/gather/enum_shares
j. Run post/windows/gather/enum_applications
k. Run post/windows/manage/migrate
l. Run post/windows/gather/dumplinks
m. Run getsystem
n. Run post/gather/win_privs
o. Execute “getuid” command
p. Execute use kiwi
q. Execute creds_all
r. Execute ls_dump_sam
s. Execute “search -f *.pdf” command
t. Execute “search -f *.doc” command

4. Execute “clearev” command to clear the Windows Event Log

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.8 Malicious Insider Data Exfiltration (Tier 1 | Linux)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 1 adversary. Malicious insider performs actions within a Virtue to exfiltrate local user documents to a remote server.

Test Case Id: 4.8

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application and standard Linux tools. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performer's Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.9 Malicious Insider Data Exfiltration (Tier 1 | Windows)

Test case simulates actions carried out by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Windows operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 1 adversary. Malicious insider performs actions within a Virtue to exfiltrate information (e.g., documents, system information) identified by earlier reconnaissance test(s).

Test Case Id: 4.9

Test Type: Security

Virtue Target Operating System: Windows

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors

- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system. Documents to be targeted for exfiltration will be pre-placed within the Virtue as well as a corporate file share. The following documents will be used for testing: Principia, Calculus Made Easy, General and Special Relativity, Thermo Dynamics, and Dracula.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.10 Malicious Insider Data Exfiltration (Tier 2 | Linux)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 2 adversary. Malicious insider performs actions within a Virtue to exfiltrate local user documents to a remote server.

Test Case Id: 4.10

Test Type: Security

BAA Requirements Addressed:

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metric:

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors

- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application and standard Linux tools. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performer's Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

1. T&E team launches malicious insider Linux exfiltration threat tier 2 test
    a. Ground truth collection is started
    b. T&E Attack Asset (e.g. Metasploit) is executed and creates a reverse shell handler and subsequent connection to target system running shell code
2. T&E team copies the following files to the target system via Attack Asset connection
    a. Documents: Principia, Calculus Made Easy, General and Special Relativity, Thermo Dynamics, and Dracula
    b. Script: Linux Tier 2 Exfiltration
3. Execute Linux Tier 2 Exfiltration script that was uploaded to the target system in step 2
    a. Create a gzipped tar file of /home/<user>/Documents/*
    b. SCP the gzipped tar file to the remote host (i.e., testbed attack platform) using an SSH key
    c. Remove the gzipped tar file
    d. Clear the user's ~/.bash_history file
4. Files that were placed on target system are removed
5. Attack Asset disconnects from target system

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.11 Malicious Insider Data Exfiltration (Tier 2 | Windows)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Windows operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 2 adversary. Malicious insider performs actions within a Virtue to exfiltrate information (e.g., documents, system information) identified by earlier reconnaissance test(s). Actions include steps taken to erase evidence of activity.

Test Case Id: 4.11

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud

- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.12 Malicious Insider Data Exfiltration (Tier 3 | Linux)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Linux operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 3 adversary. Malicious insider performs actions within a Virtue to exfiltrate local user documents to a remote server.

Test Case Id: 4.12

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically

by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue router admin role exists and router admin Virtue has been created. Router admin Virtue is equipped with a terminal application and standard Linux tools. The router admin Virtue is running and has network access. The T&E system is connected to the router admin Virtue via the performer's Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on router admin Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

Postconditions: Ground truth data collected. Testbed test result files collected. Performer logs collected.

9.4.13 Malicious Insider Data Exfiltration (Tier 3 | Windows)

Test case simulates actions performed by a malicious insider to stimulate the Virtue system in order to exercise Virtue sensors used to monitor the Virtue. This test case is performed on Virtues that support Windows operating systems. Malicious insider actions performed in this test case are aligned with behaviors of a threat Tier 3 adversary. Malicious insider performs actions within a Virtue to exfiltrate information (e.g., documents, system information) identified by earlier reconnaissance test(s). Actions include steps taken to erase evidence of activity.

Test Case Id: 4.13

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud
- Requirement 5: Virtues shall offer numerous logging capabilities and be capable of dynamically adjusting their sensoring when commanded by the Analytics/Control Layer to address newly suspected threats
- Requirement 6: Virtues shall offer numerous protection options tailored to the expected risks of the environment. They shall demonstrate the capability to respond dynamically by invoking various protections once security analytics determine the possibility of suspicious behavior

BAA Metrics

- 4.a - The success rate and performance cost of a Virtue with default protections in deterring attacks to each of the four vectors
- 6.c - The success rate and performance cost of a Virtue with all pertinent protections deployed in deterring attacks to each of the four vectors

Preconditions: Virtue Windows Power User role exists and Power User Virtue has been created. Windows Virtue is equipped with the ability to execute cmd.exe and putty.exe. The Power User Virtue is running and has network access. The T&E system is connected to the Power User Virtue via the performers Virtue endpoint. Shell code to be executed to simulate malicious insider actions is placed within the Virtue. T&E team executes shell code on Power User Virtue via Virtue endpoint to create reverse shell connection to T&E system.

Steps

- Redacted from version delivered to performer teams

9.4.14 External Attack Surface Enumeration

Test case performs network scans to collect data on the Virtue's external attack surface.

Test Case Id: 4.14

Test Type: Security

BAA Requirements Addressed
Virtues shall present themselves as atomic, largely immutable entities to other Virtues and external processes. They shall be simpler and more modular than current VDI solutions with a minimized attack surface

BAA Metrics

- 2.a - The number of exposed system calls in the exposed portion of a Virute
- 2.b – The number of running processes in the exposed portion of a Virtue
- 2.c – The number of active/available services in the exposed portion of a Virtue
- 2.d – The number of communication paths that traverse a Virtue trust boundary
- 2.e - The number of credentials in the exposed portion of a Virtue

Metrics Collected:

- # of external accessible ports

Preconditions: A Virtue has been created that has network access and an IP address that is reachable from the T&E system.

Steps

1. An nmap service/version scan is performed against the Virtue's IP address from the testbed

Postconditions:

- List of services including port numbers and version information is gathered

9.4.15 Internal Attack Surface Enumeration

Test case performs various actions to gather data to characterize a Virtue's internal attack surface.

Test Case Id: 4.15

Test Type: Security

BAA Requirements Addressed
Virtues shall present themselves as atomic, largely immutable entities to other Virtues and external processes. They shall be simpler and more modular than current VDI solutions with a minimized attack surface.

BAA Metrics:
- 2.a - The number of exposed system calls in the exposed portion of a Virtue
- 2.b – The number of running processes in the exposed portion of a Virtue
- 2.c – The number of active/available services in the exposed portion of a Virtue
- 2.d – The number of communication paths that traverse a Virtue trust boundary
- 2.e - The number of credentials in the exposed portion of a Virtue

Metrics Collected:
- # of internal accessible ports
- # of internal communication paths (IPC)
- # of running processes
- # of running privileged processes
- # of loaded kernel modules
- # of running threads
- # of available services
- # of installed applications
- # of resources to target (e.g., files)
- # of open files
- # of files with SUID and SGID
- # of user accounts
- # of users that can elevate privilege
- # of groups with SUDO

Note: Number of exposed system calls, number of communication paths that traverse a Virtue trust boundary, and number of credentials in the exposed portion of a Virtue will not be collected at midterm.

Preconditions: A Virtue has been created that has network access and an IP address that is reachable from the T&E system.

Steps (Linux)
1. Enumerate the number of loaded kernel modules

2. Enumerate the number of running processes
3. Enumerate the number of privileged processes (i.e., processes running as root)
4. Enumerate the number of running threads
5. Enumerate the number of files on the system
6. Enumerate the number of open file handles
7. Enumerate the number of executable files
8. Enumerate the number of SETUID files
9. Enumerate the number of SETGUID files
10. Enumerate the number of users
11. Enumerate the number of users with a shell
12. Enumerate the number of SUDO user accounts
13. Enumerate the number of SUDO groups
14. Enumerate the number of TCP listeners
15. Enumerate the number of UDP listeners
16. Enumerate the number of IPC channels

Test Customization: (Windows)

1. Enumerate the number of running processes
2. Enumerate the number of running threads
3. Enumerate the number of running services
4. Enumerate the number of files on the C drive
5. Enumerate the number of open file handles
6. Enumerate the number of pipes
7. Enumerate the number of dlls
8. Enumerate the number of users
9. Enumerate the number of user with administrator privileges
10. Enumerate the number of TCP listeners
11. Enumerate the number of UDP listeners
12. Enumerate the number of IPC channels

Postconditions:

- Testbed test result files collected.

9.4.16 Verify Virtue Image at Startup

Test case verifies that a Virtue has a mechanism to verify the Virtue image before launching the Virtue.

Test Case Id: 4.16

Test Type: Security

BAA Requirements Addressed

- Requirement 4 - Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud

Preconditions

- A test user account is created and maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System

Steps

1. With performer guidance, T&E team inspects Virtue launch process to verify that a Virtue image is verified before Virtue launch

9.4.17 Verify Virtue Peer-to-Peer Communications are Encrypted

Test case verifies that communications between Virtues are encrypted

Test Case Id: 4.17

Test Type: Security

BAA Requirements Addressed:

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud

Preconditions

- A test user account is created & maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System

Steps

1. With performer guidance, T&E team inspects Virtue to Virtue communications to verify that communications are encrypted

9.4.18 Verify Virtue to Virtue Eco-System Communications are Encrypted

Test case verifies that communications between Virtues and the Virtue Eco-System are encrypted.

Test Case Id: 4.18

Test Type: Security

BAA Requirements Addressed

- Requirement 4: Virtues shall be built on a more defensible virtualized construct than the current general purpose VDI VM to address threats expected on a public cloud

Preconditions

- A test user account is created & maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System

Steps

1. With performer guidance, T&E team inspects Virtue to Virtue communications to verify that communications are encrypted

9.5 Performance

9.5.1 Complete workflow to define and construct new Virtue

Test case captures time elapsed to complete the workflow of defining and constructing a new Virtue. Virtue construction encompasses time required to package (and to prepare) the new Virtue for use. This may include installation of required applications or containers within the Virtue. Virtue construction does not include time required to start a newly constructed Virtue.

Test Case Id: 5.1

Test Type: Performance

BAA Requirements Addressed

- Requirement 11: Proposers shall provide methodology and utilities to define, construct, assure, and transport Virtue images

BAA Metrics:

- 11.a - An end-to-end execution of the methodology to define and construct Virtue images shall be completed within x

Preconditions

- A Virtue administrator account is created & maintained within the Virtue System
- The Virtue administration GUI/portal is accessible

Steps

1. Log in to the Virtue administration GUI/portal as a Virtue administrator
2. Using the Virtue administration GUI/portal define and construct a new Virtue
3. Log out as the Virtue administrator

Preconditions

- Verify the newly created Virtue exists and contains the correct applications and sensing and protection configurations.

9.5.2 Complete workflow of starting a Virtue, accessing an application, and terminating a Virtue

Test case captures time elapsed to complete the workflow of starting a Virtue, accessing an application, interacting with the application, and terminating the Virtue.

Test Case Id: 5.2

Test Type: Performance

BAA Requirements Addressed

- Requirement 9: Proposers shall provide an intuitive presentation interface allowing access to numerous authorized Virtues to provide all the functionality required by a user delivered securely to an end device.

BAA Metrics:

- 9.b - A user shall successfully complete a functional use case that involves starting, accessing, and terminating a Virtue within x minutes.

Preconditions

- A test user account is created & maintained within the Virtue System user store
- The test user account has an active, authenticated session with the Virtue System

Steps

1. As the test user, use the Virtue interface to navigate to available Virtues and start the Document Editor Virtue.
2. Within the Document Editor Virtue, start the MS Word application (if application is not already started).
3. Enter “Hello World” into the MS Word application
4. Save the MS Word application
5. Shutdown/terminate the Document Editor Virtue

Preconditions

- Verify the Document Editor Virtue for the test user has been terminated

10. SCORING

This section describes the approach to scoring performer solutions. The scoring methodology provides a general framework to combine the results from individual test cases into a single “roll-up” score addressing functionality, performance, and security. This section also outlines where each test case fits into the aforementioned roll-up score.

10.1 Methodology

The wide variety of test metrics (see Section 5 and 6) poses a challenge to scoring performer solutions and to comparing the diverse approaches that each performer solution includes. In part, this challenge arises because many of the metrics are not directly comparable: for example, elapsed time and resources consumed (e.g., storage) have different units (e.g., seconds vs. GBs) that preclude a direct comparison. One approach to reconcile these differences is cost because both processing time and storage consumption incur charges on AWS. However, this approach suffers two drawbacks. First, it tightly couples the evaluation with a particular cloud provider (e.g., AWS), which is undesirable because other cloud providers might offer similar services at different costs. Second, more abstract metrics such as security lack a simple means to derive the costs of various design alternatives—i.e., the actuarial science of cybersecurity is not yet sufficiently mature to assess many risks. Hence, a more general approach to score solutions is desirable.

Measurements of both static and dynamic metrics create a vector of scores (of length *n* where *n* is the number of metrics) that characterize the system as a whole. Given two performer solutions *A* and *B*, *A* is said to dominate *B* if *A* scores better on at least one dimension and at least as well as *B* on all other dimensions. Stated differently, there is no characteristic of the system where *B* outperforms *A*. This problem is known as the *maximum vector problem*,[3] but it is rare for a total ordering to exist among all the alternatives. In most cases, solution *A* performs better than *B* along some dimensions (e.g., cost) but fares worse than solution *B* along other dimensions (e.g., security). Because some users emphasize cost effectiveness whereas others demand better security, it is not possible—in the general case—to provide a single ranking of the solutions; to do so requires assigning a weight (i.e., importance) to each dimension so that a final score can be computed. Recognizing that lack of consensus regarding the "ideal" weighting, a pragmatic approach is simply to report each solution's score for each metric and allow users to determine which solution is best for their particular requirements (cost, security, etc.).

Unfortunately, the aforementioned approach does not address the short-term need for a succinct summary of the performance and effectiveness of each performer's solution. Ideally, solutions should have a small number of scores that permit the direct comparison of the advantages and disadvantages of various approaches. Thus, we combine scores to create a "roll-up" that characterizes broad aspects of the system (such as performance and security). It is important to note that these combined scores elide information and are not a substitute for the aforementioned approach that requires user-specific weights to determine the ideal solution for a particular set of requirements.

Normalization is an essential first step toward the direct comparison of metrics. Normalization essentially allows a relative comparison among approaches where the absolute values differ widely. Consider, for example, the internal attack surface metrics of the number of resources that could be a target (e.g., the number of files on the system) and the number of user accounts. One clearly expects a difference of several orders of magnitude between these quantities. Any scheme to combine these quantities is frustrated by the strong influence by "outliers" that skew simple statistics. One way to address such outliers is to bin results – e.g.,

$$normalize(x) = \log(x + 1) + 1$$

where the logarithm dramatically compresses the range in observed values even when those values span several orders of magnitude. For example, assume that the baseline VDI system contains 125,000 files and a performer solution contains 250,000 files. The baseline VDI system score ≈5.097 whereas the performer solution scores ≈5.398 – a relatively small absolute difference.

Normalization is an essential step that allows the combination of scores. While there are many ways to combine scores, an intuitive approach is to sum the normalized values. Thus, a performer solution with twice as many files and twice as many user accounts as the baseline system has twice the attack surface. While one can certainly argue that a simple sum is too simplistic, anything more complicated requires general agreement regarding the weight of each metric toward the combination, and expecting such universal agreement is impractical. Thus, the

[3] H. T. Kung, F. Luccio, and F. P. Preparata, "On Finding the Maxima of a Set of Vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, October 1975. Online: http://dx.doi.org/10.1145/321906.321910

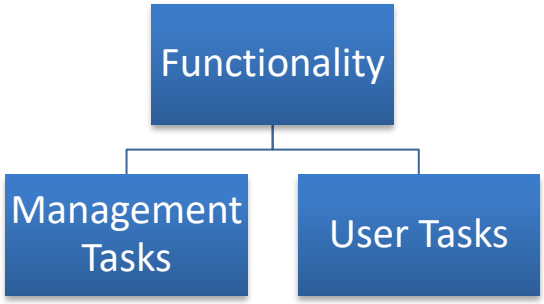
pragmatic approach is to prefer a simple combination technique with its known limitations. Reporting the "raw" scores for each metric in addition to their (simple) combination allows others to provide alternative weights should they so choose.

The combination of scores is an essential step toward providing a succinct summary of a solution in comparison to the baseline. The static and dynamic metrics conceptually form a tree where the internal nodes summarize the scores at lower levels of the tree. For example, the internal attack surface "channels" node summarizes the communication paths that traverse a trust boundary, the internal communication paths (e.g., IPC), and the internally-accessible ports. The internal attack surface is then combined with the external attack surface to characterize the solution's entire attack surface. This process is repeated until reaching the root of the tree (e.g., security, functionality, or performance). These scores succinctly compare the solution to the baseline. It is important to reiterate, however, that this summary should not be used as a total ordering among performer solutions because the abstraction elides information that differentiates the solutions at a more granular level.

10.2 Combining Scores

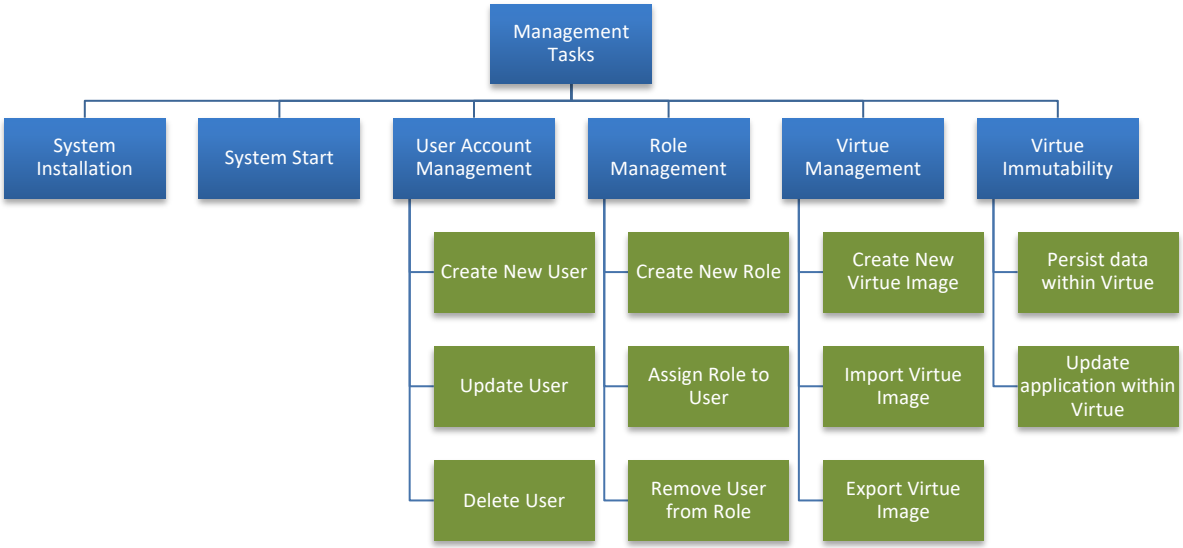
Virtue performers are assessed against a set of functionality, security, and performance evaluation criteria. These criteria are guided by the requirements outlined in the BAA and describe the characteristics of solutions that are broadly acceptable to the VirtUE program team and its customers. This section describes how results from individual test cases are combined to provide a succinct summary of a performer's solution.

10.2.1 Functionality



The functionality criteria cover the basic operational features of a performer system. These consist of management tasks that involve managing Virtues, user roles, and related tasks, and user tasks that concern operational workflows.

10.2.1.1 Management Tasks

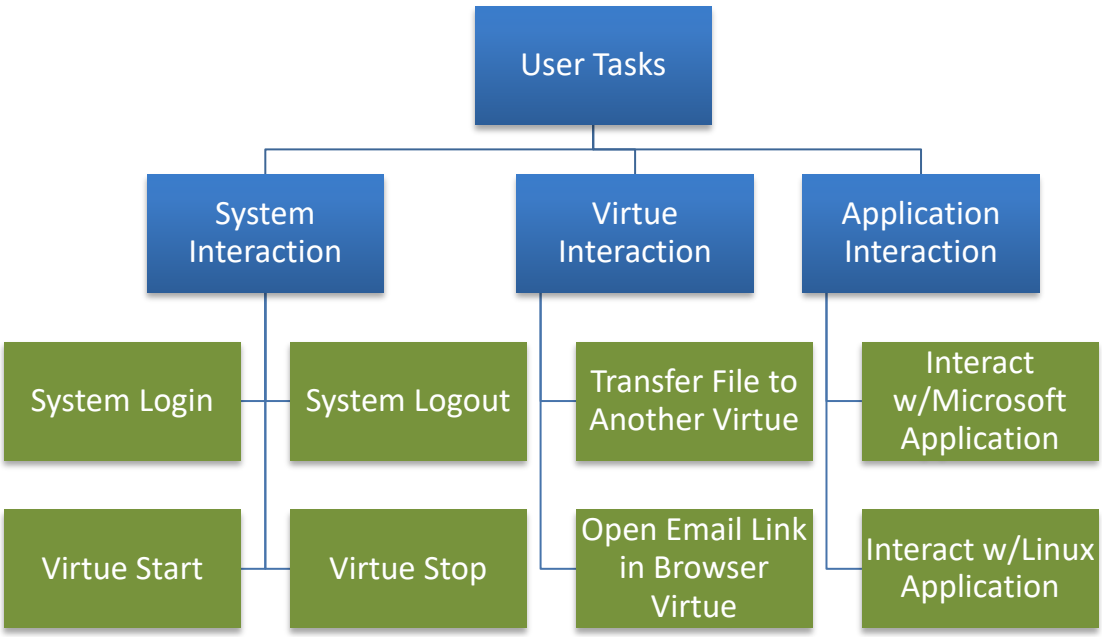


Management tasks consist of Virtue system installation, including establishing the AWS infrastructure and installing the performer software; system start, including starting all the performer components necessary to achieve a working Virtue system ready for users to log in; and management of user accounts, roles, and Virtues.

Further, data persistence and immutability are program requirements, and will be tested as part of the Management Tasks suite. This criterion is to test that an administrator can make authorized changes to Virtues, and store enterprise-wide data in Virtues (e.g., host lists, certificate stores, email configurations and the like).

Test cases 2.1 – 2.10 cover management tasks.

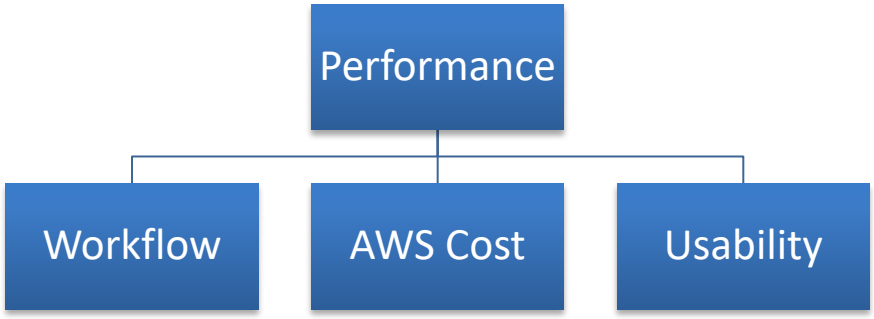
10.2.1.2 User Tasks



User tasks describe the expected interactions that users will have with the Virtue system.[4] Virtues interact with other Virtues under user control, including transferring files and performing convenience features such as opening a link in an email and having an Internet Virtue start automatically to handle the request.

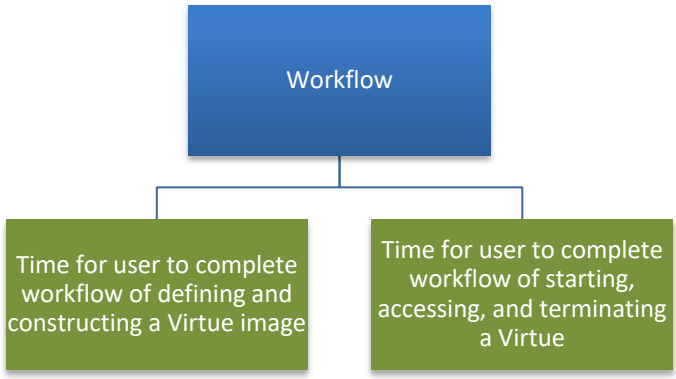
Test cases 2.11 – 2.12 and 3.1 – 3.9 cover user tasks.

10.2.2 Performance



Performance is assessed across a range of dimensions. The performance category also includes evaluation of the usability of a performer's solution. Most of the performance metrics in this section are collected during the evaluations of the functional and security capabilities.

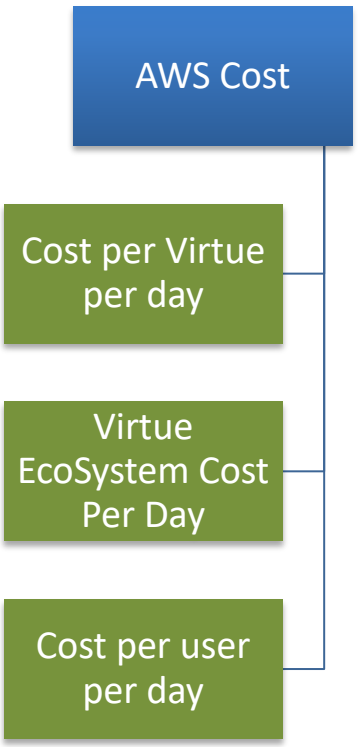
10.2.2.1 Workflow



A *workflow* is a complete sequence of events that a Virtue enterprise user might perform to complete a mission task. For instance, "writing a document using information sources obtained from Internet" may constitute a workflow. For Phase 1, the workflows will be relatively simple, testing only the ability of an administrator to create a Virtue image and a non-administrator user to activate, briefly use, and terminate a Virtue image.

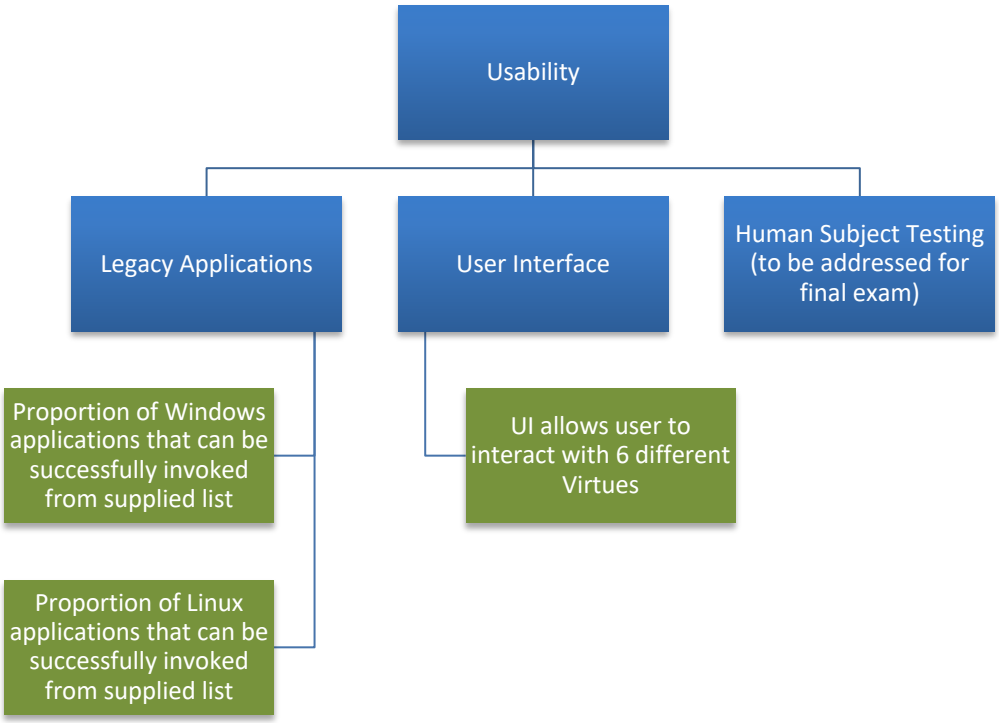
Test cases for Virtue workflows are described in Sections 9.5.1 and 9.5.2.

10.2.2.2 AWS Cost



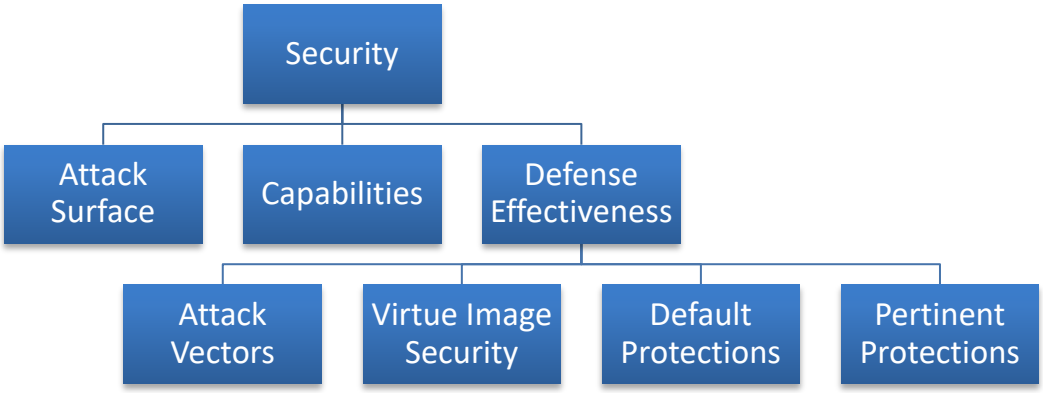
Cost data will be collected during evaluations of other capabilities; these will be aggregated and scored as a rollup to assess the overall performer solution.

10.2.2.3 Usability



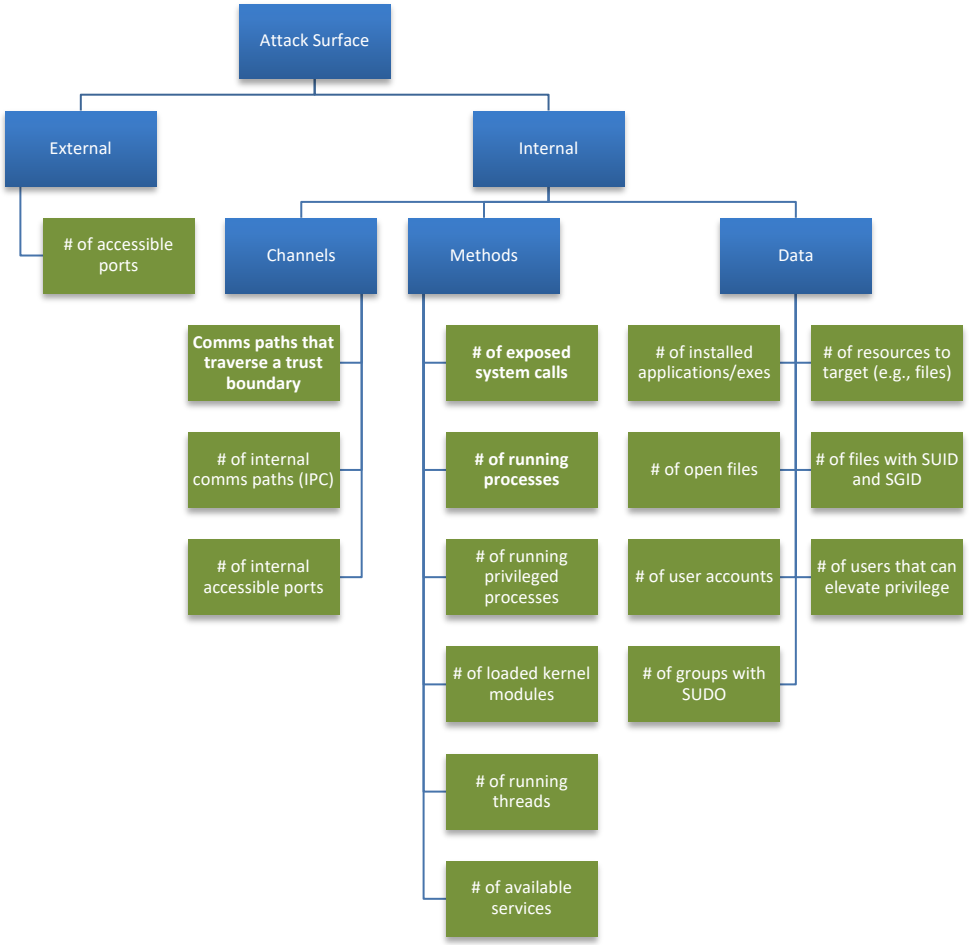
Usability covers the effectiveness and efficiency of the performer's Virtue solution, interfaces, and components as a whole. For midterm testing, legacy application usability will be simple tests to verify the number of Windows and Linux applications from a specified list that are supported. The user interface evaluation will also determine whether the BAA requirement of a user interacting with at least 6 different Virtues can be satisfied.

10.2.3 Security



Security is an important concept to the VirtUE program. The following sections depict each aspect of security in depth.

10.2.3.1 Attack Surface



Virtues are expected to have a small attack surface compared to a baseline virtualization implementation. An attack surface can be defined as the set of ways an adversary can enter a system and potentially cause damage using channels (e.g., sockets) to connect to a system, invoking methods (e.g., API) to carry out particular functions, and sending data (e.g., input strings) or receiving data from the system.[5] The attack surface is divided into two parts, *external* and *internal*.

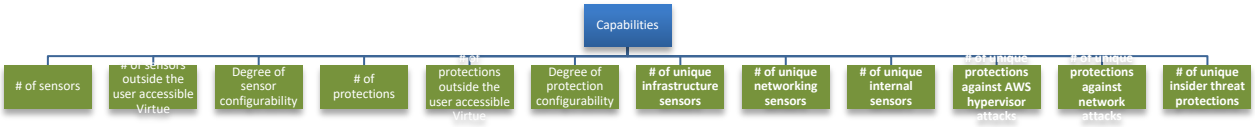
The intent is to use tools and techniques similar to the Attack Surface Analyzer for Windows-based operating systems to measure attributes of the Virtue's operational environment over time. These tests support the evaluation of Requirement 2 specifically, and many of the other requirements in general. The following metrics will measure a Virtue's user exposed attack surface for a series of prescribed roles.

The *external attack surface* describes features of the Virtue accessible by entities operating outside of the Virtue. The external attack surface is measured as the enumeration of network communication paths and accessible endpoints from a reference point external to the system, e.g. from another host on the same or different network segment.

[5] http://www.cs.cmu.edu/~wing/publications/CMU-CS-07-146.pdf

The *internal attack surface* is only visible to entities operating inside the Virtue. This attack surface encompasses internal communications paths accessible from a reference point internal to the system. These channels include interprocess communications channels; exposed methods such as system calls available to both privileged and non-privileged programs; kernel modules that may be loaded or unloaded; threads that may be attached to; running processes with privilege that may be interacted with. Internal attack surface also consists of available data, such as knowledge of memory allocation, number of open files, details about administrative users, and cached authentication credentials.

10.2.3.2 Capabilities



The T&E team will assess the capabilities noted above in a static sense; that is, the test team will conduct an as-designed review of the security architecture and assess the solutions based on claimed capabilities.

Sensors are supposed to identify security relevant events that may indicate attacks. Sensors may be placed at a variety of locations, but the Virtue evaluation team will focus on those present in the *infrastructure, network,* and *internal (within Virtue)* layers in the architecture.

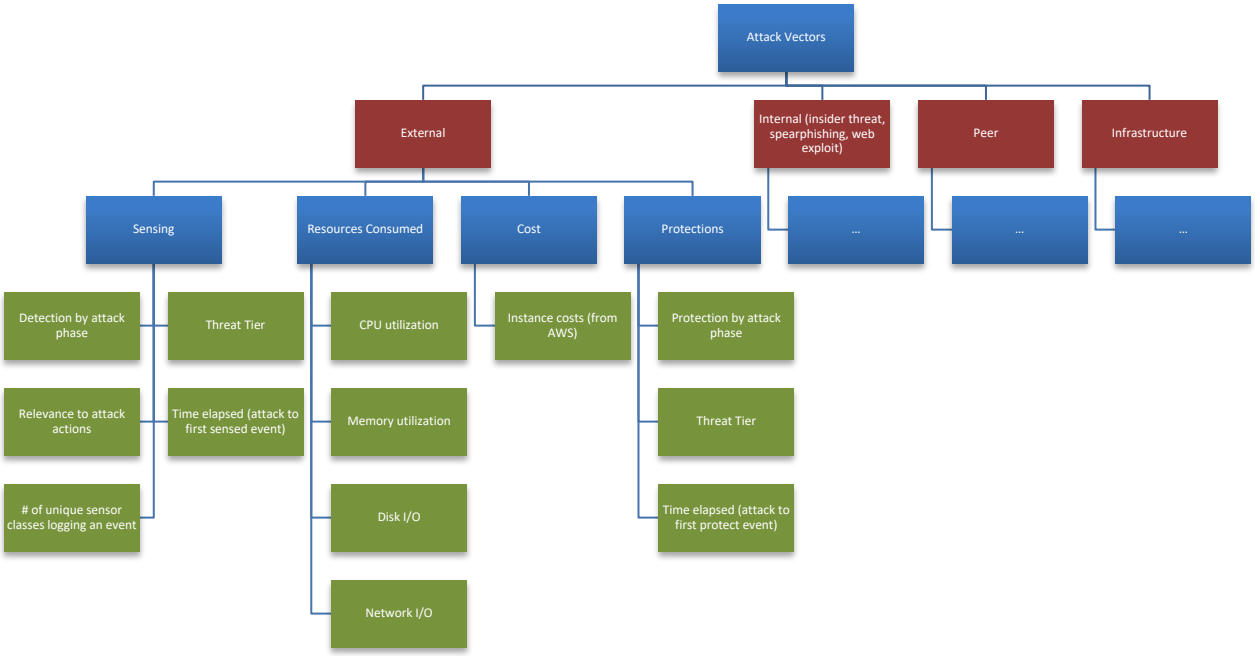
Protection capabilities are those that defend against attacks, either reactively (such as intrusion prevention systems, malware quarantines) or as always-on capabilities (such as firewalls). Although there are many possibilities, the Virtue evaluation team expects at a minimum that performers will include firewalls, malware/antivirus, encryption capabilities for sensitive data, application controls such as application whitelisting, and access controls such as user authentication components.

Together, sensors and protection capabilities should mitigate many concerns of information security. To this end, the Virtue evaluation team will attempt to map the sensors and protections against general security threats toward this type of virtual enterprise, and will assess the architecture accordingly. Coverage includes the relative amount of attack surface that is sensed; the numbers of sensors and protections; and, the number of unique sensed events, and their utility in identifying different types of attacks.

10.2.3.3 Defense Effectiveness

The performer's security architecture will be evaluated for its ability to defend against various threats. Complementing the capabilities' static "paper study" evaluation of the security architecture, the effectiveness tests will determine dynamically how well the performer's solution works against real attacks.

10.2.3.3.1 Attack Vectors



Four major attack vectors will be attempted against performer solutions in the evaluation lab environment: external, internal, peer, and infrastructure attacks. Within each of these vectors, the team will evaluate the performance of the sensing and protections; determine the resources consumed to defend against the attacks; and identify the added AWS monetary costs of operating the defenses compared to a steady-state, non-attack cost. Each of these characteristics will be repeated under the other types of attacks (although the diagram omits them for brevity).

10.2.3.3.2 Virtue Image Security

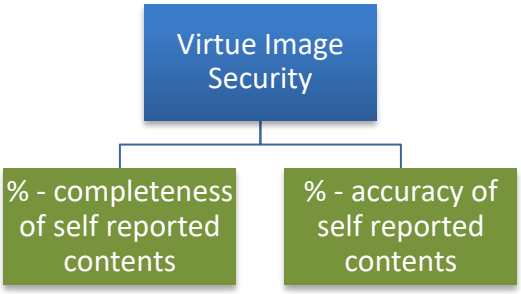


Image security includes integrity of the image and ability to provide evidence that something has been changed without authorization. Tests will attempt to change contents of Virtues, and the system is expected to report such discrepancies.

Note: Virtue image security will not be assessed as part of midterm testing.

10.2.3.3.3 Default and Pertinent Protections

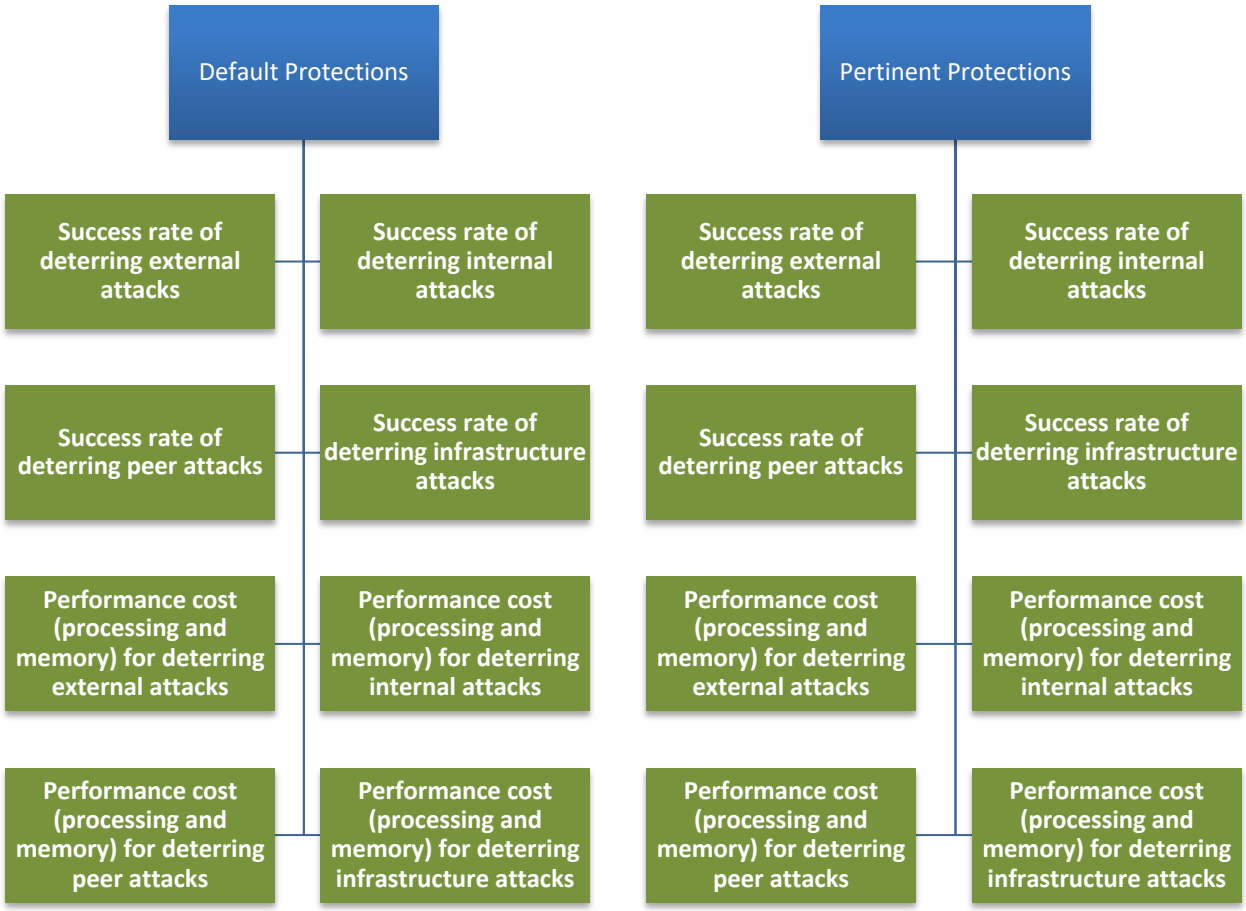
Protections will be assessed as an aggregation of data collected during live attacks, to include success rates as well as performance costs.

There are two types of protections. *Default protections* are those protections assumed to be always active by default, regardless of any attack that is occurring or that is expected to occur imminently. For instance, a firewall blocking inbound traffic on port 23 is considered a default protection.

Pertinent protections are those that are activated in response to an anticipated attack. In VirtUE Phase 2, performers are expected to provide automated attack response mechanisms that include activating special protections normally off by default, as well as adjusting the configuration certain default protections. For instance, if a worm is detected spreading among networked Virtues on a TCP port 1234, a *pertinent protection* might be to command all firewalls to block port 1234. In phase 1, this automated capability is not expected to be present, and therefore the Virtue evaluation team will activate targeted protections manually prior to launching an attack to simulate an automated response mechanism.

Performers will be assessed in Phase 1 on the performance of the default protections, and also on the range and ability to deploy effective pertinent protections against attacks.

Note: Default versus pertinent protections will not be assessed as part of midterm testing.



11.RISK MITIGATION

Designing and developing a testbed with limited knowledge of performer solutions and limited access to a functional prototype or system presents a variety of challenges and introduces risk with successfully conducting T&E in the allotted timeframe. In addition, designing and developing a testbed that is able to accommodate different solutions provided by different performers also presents a set of unique technical challenges. This section identifies activities intended to reduce risk associated with integrating various solutions into the testbed in order to carry out T&E activities per the program schedule.

11.1 Technical Discussions with Performers

In coordination with the VirtUE program office, the T&E team supports an individual one-hour conference call with each performer team every other Tuesday. Between program kick-off in September of 2017 and the first midterm T&E event held in May of 2018, 12 technical status discussions were supported [9/26/2017, 10/10/2017, 10/24/2017, 11/07/2017, 11/28/2017, 12/12/2017, 01/09/2018, 01/23/2018, 02/06/2018, 02/20/2018, 03/06/2018, 04/17/2018].

Technical discussions with performers provide the T&E team an excellent opportunity to gain insight into emerging designs and implementation decisions being made by each performer team.

Close and continuing interaction with performer teams allows the T&E team to stay abreast of critical design changes and technical challenges each performer team is facing. As a result of ongoing technical discussions, critical integration issues will surface early allowing the T&E team time to explore various networking and testbed configuration/integration alternatives well in advance of an official test event.

11.2 Site Visits

Select members from the T&E team accompanied the VirtUE program office on visits to each performer's work site to support face-to-face discussions. Each visit included meetings held over the span of one or two days to focus on deep dives of performer's Virtue architectures, designs, and demonstrations of rough prototypes. The following site visits were supported by the T&E team.

Date(s)	Performer	Location
4–5 December 2017	Siege Technologies[6]	Manchester, NH
6 December 2017	Raytheon[7]	Cambridge, MA
10 January 2018	Next Century[8]	Annapolis Junction, MD
18–19 January 2018	Star Lab[9]	San Antonio, TX

11.3 Test and Evaluation Integration Points

To reduce the risk of integration challenges from interfering with test execution, the T&E team provided performers with a testbed configuration guide[10] in December of 2017 and also defined a testbed integration timeline outlining key integration phases to enable a gradual integration of performer solutions into the testbed.

The Virtue testbed configuration guide describes the T&E environment, its networking architecture, integration points between a performer's Virtue and testbed components, and the set of enterprise services to be provided by the T&E environment and available for use by performer systems. The intent of the testbed configuration guide is to establish a common vision of the T&E environment, provide context to ensure performer design choices consider potential integration impacts, and to assist performers with identifying potential integration conflicts so they can be addressed as early as possible.

An integration plan, including key milestones and gradual integration phases, was provided to performers via email on February 12, 2018. The integration plan defines a series of activities to be conducted in the months leading up to the midterm test event. The following bullets summarize key integration activities.

---

[6] Trip Report: Siege Technologies (AOS Report No. AOS-17-1468)
[7] Trip Report: Raytheon (AOS Report No. AOS-17-1486)
[8] Trip Report: Next Century (AOS Report No. AOS-18-0135)
[9] Trip Report: Star Lab (AOS Report No. AOS-18-0136)
[10] Virtue Performer Testbed Configuration Guide (JHU/APL Report No. AOS-17-1339)

- February 2018 – Provide performer teams access to APL-managed AWS accounts to allow teams to begin migrating Virtue system assets (e.g., Amazon Machine Images) from performer AWS accounts to the APL-managed account to begin testing connectivity to the APL on premise testbed.
- March 2018 – Integrate T&E functional, security, and performance testing tools with each performer's Virtue system. The March integration effort allows the T&E team to understand where T&E tools need to be tailored to support testing of each performer's system.
- April 2018 –Verify data collection and analysis capabilities provided by the testbed against performer systems. Hold a T&E dry run and a Test Readiness Review to assess readiness to begin midterm testing.

11.4 Test Execution Mitigation Plans

T&E mitigation plans will guide T&E efforts in the event that unforeseen complications are encountered during testing. T&E mitigation plans include the following: supplemental testing window and interaction with performer teams during testing to troubleshoot/resolve issues. A four-week time period has been allotted for midterm testing. While T&E is expected to complete at the close of week two, a two-week supplemental testing window is included in the schedule. The T&E team will interact with performer teams to obtain guidance during midterm testing if serious complications are encountered.

12.SUMMARY

The VirtUE Test Plan presents JHU/APL's strategy for conducting T&E against performer solutions. The document introduces VirtUE program requirements, test metrics, test cases, and scoring strategy. The team's methodology for test case generation is discussed and specific test cases are introduced. The strategy presented is intended to maximize test coverage of proposers' solutions with respect to functionality, security and performance requirements, while emphasizing accurate and efficient measurement and collection of the results data as much as possible.

The VirtUE Test Plan is under active development and future releases of this document are planned.

GLOSSARY

This section provides definitions for some of the key terms used in this document. Most of these terms have been drawn or adapted directly from the IEEE Standard for Software and System Test Documentation (IEEE Std 829-2008, IEEE Computer Society).

test (A) A set of one or more test cases. (B) A set of one or more test procedures. (C) A set of one or more test cases and procedures. (adopted from IEEE Std 610.12-1990 [B3] (D) The activity of executing (A), (B), and/or (C).

test approach A particular method that will be employed to pick the particular test case values. This may vary in specificity from very general (e.g., black box or white box) to very specific (e.g., minimum and maximum boundary values).

test case (A) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (B) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item. (adopted from IEEE Std 610.12-1990 [B2])

test class A designated grouping of test cases.

test design Documentation specifying the details of the test approach for a software feature or combination of software features and identifying the associated tests (commonly including the organization of the tests into groups). (adopted from IEEE Std 610.12-1990 [B2])

test effort The activity of performing one or more testing tasks.

test level A separate test effort that has its own documentation and resources (e.g., component, component integration, system, and acceptance).

testing (A) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component. (B) To conduct an activity as in (A).

test item A software or system item that is an object of testing.

test module A designated grouping of tests in the context of the VirtUE test plan, similar to the concept of a test class. The test module is expressed as a set of related test cases that share significant portions of one or more test procedures.

test plan (A) A document describing the scope, approach, resources, and schedule of intended test activities. It identifies test items, the features to be tested, the testing tasks, who will do each task, and any risks requiring contingency planning. (B) A document that describes the technical and management approach to be followed for testing a system or component. Typical contents identify the items to be tested, tasks to be performed, responsibilities, schedules, and required resources for the testing activity. (adopted from IEEE Std 610.12-1990 [B2]) The document may be a Master Test Plan or a Level Test Plan.

test procedure (A) Detailed instructions for the setup, execution, and evaluation of results for a given test case. (B) A document containing a set of associated instructions as in (A). (C) Documentation that specifies a sequence of actions for the execution of a test. (adopted from IEEE Std 982.1TM-2005 [B7])

testware All products produced by the testing effort, e.g., documentation and data.

threat tier A Defense Science Board (DSB) threat hierarchy that describes the capabilities of potential attackers based on skill level and available resources. Descriptions of each threat tier is as follows (as taken from the DSB report[11]):

- **I** Practitioners who rely on others to develop the malicious code, delivery mechanisms, and execution strategy (use known exploits).
- **II** Practitioners with a greater depth of experience, with the ability to develop their own tools (from publically known vulnerabilities).
- **III** Practitioners who focus on the discovery and use of unknown malicious code, are adept at installing user and kernel mode root kits10, frequently use data mining tools, target corporate executives and key users (government and industry) for the purpose of stealing personal and corporate data with the expressed purpose of selling the information to other criminal elements.
- **IV** Criminal or state actors who are organized, highly technical, proficient, well-funded professionals working in teams to discover new vulnerabilities and develop exploits.
- **V** State actors who create vulnerabilities through an active program to "influence" commercial products and services during design, development or manufacturing, or with the ability to impact products while in the supply chain to enable exploitation of networks and systems of interest.
- **VI** States with the ability to successfully execute full spectrum (cyber capabilities in combination with all of their military and intelligence capabilities) operations to achieve a specific outcome in political, military, economic, etc. domains and apply at scale.

VirtUE (Virtuous User Environment) A reference to the program attempting to create both new user computing environments and security analytic methods

Virtue A reference to a single, role-specific user environment that is part of the larger system.

[11] DoD Defense Science Board Task Force Report: Resilient Military Systems and the Advanced Cyber Threat

APPENDIX A.KEY STAKEHOLDERS

This section identifies participants in VirtUE T&E activities and describes the role and responsibilities of each participant.

A.1 VirtUE Program Team

The VirtUE Program Team consists of the VirtUE IARPA Program Manager and supporting Systems Engineering Technical Assistance (SETA) members. This team oversees VirtUE program execution and is responsible for establishing program requirements and metrics and is the final authority in all program and T&E matters.

A.2 VirtUE Test and Evaluation (T&E) Team

The VirtUE T&E Team consists of staff from JHU/APL supporting the VirtUE Program Office with all T&E activities. This team serves the role of independent evaluators of VirtUE performer solutions and is responsible for developing the T&E methodology, constructing the VirtUE testbed, and executing all tests against VirtUE performer solutions.

A.3 VirtUE Performers

VirtUE performers consist of organizations selected by the VirtUE Program Office to develop fully functioning Virtues. Performer teams are responsible for design, implementation, integration, and preliminary testing of their respective solutions. Solutions developed by VirtUE performers are expected, to the greatest extent possible, satisfy VirtUE program requirements. In support of VirtUE T&E activities, each performer will provide the following documentation: the system architecture, software design, and integration and operation manuals; a deployable package consisting of their VirtUE environment and any dependent executables required to deploy and operate their system in AWS; and all source code developed in support of the program.

Performers for Phase 1 of the VirtUE program include:

Organization(s)	**Principal Investigator (PI)**
Next Century Corporation	Dr. Chris Long
Raytheon / BBN	Dr. Brian Krisler
Siege Technologies	Martin Osterloh
Star Lab	Adam Fraser

APPENDIX B.TEST CASE TEMPLATE

A description of the test case in one or two sentences.

Test Case Id: Unique identifier used to identify the test case

BAA Metrics:

Test Category: Functional/Security/Performance

BAA Requirements Addressed: Program requirements that are addressed by the test case

BAA Metrics: Metrics identified in the VirtUE Broad Area Announcement that are applicable to the test case

Preconditions: Expected state or configuration before test execution.

Steps

1. Enumerated, templated steps defining actions for the tester to perform

Test Customizations

- Describes performer specific modifications to the test case (if needed) to enable successful test case execution

Notes and Assumptions: Any required notes or assumptions. Usually blank.

Postconditions: Expected state after completing the test.

Metrics Collected: T&E metrics that have been collected and evaluated