A Framework for Automated Cyber Experimentation

Dhanish Mehta Thomas C. Eskridge Marco Carvalho

Harris Institute for Assured Information
Florida Institute of Technology
{dmehta,teskridge,mcarvalho}@fit.edu

Abstract

There are a variety of tools available to users that enable cyber experimentation, and many of them automate one or more stages of the experimentation lifecycle. Many of these tools are limited by the requirement to use one specific experimentation environment, or in facilitating only a part of the cyber experimentation lifecycle. To make cyber experimentation easier and more efficient, we introduce a new framework, called Hippolyte, that enables the design, creation, execution and analysis of cyber experiments and is capable of running in different underlying physical environments.

We discuss requirements for automated cyber experimentation and review previous work. We present a prototype implementation of Hippolyte and discuss the effectiveness of its use in a case study. We this discuss the benefits of automating the experimentation lifecycle and conclude with future work.

1 Introduction

The cyber experimentation lifecycle has a number of stages, and involves having to design the experiment, build the environment the experiment will run in, configure the environment, execute the experiment, collect data and interpret it. This cycle often repeats several times as new information is learned, enabling improvements in the results. The complexity and number of steps in an experiment entail having many points of failure, all of which must be monitored and managed to ensure consistent and correct execution.

Over the past ten years, there has been increased investment in research on cybersecurity by U.S. government agencies (including National Science Foundation (NSF), Defense Advanced Research Projects Agency (DARPA), and the Armed Forces and industry [1]. Security technologies sufficient to protect vital infrastructure are lacking large-scale deployment [12]. A key reason that this is not possible right now is due to the lack of scientific methodologies for developing and testing new cybersecurity technologies, as well as the lack of experimental infrastructure [12]. There is a steep learning curve associated with the set up and orchestration of experiments [12]. To improve the evaluation of network security mechanisms and make consistent advances in cyber defenses, researchers must focus on creating new experiment testbeds and accurate models of background and attack network traffic data [1] in order to adequately characterize the performance of new defense technologies.

Previous work has addressed different aspects of this issue [8, 13, 15, 19]. These solutions aid in accomplishing some stages of experimentation, such as provisioning a virtual machine, orchestration of services or collecting measurements, but none have addressed the complete experimentation lifecycle [17].

We focus on filling the gaps by creating Hippolyte, a framework that automates the process of experiment design, implementation, monitoring, and execution. Hippolyte provides users with a language to define the requirements for the experiment, and it will design the experiments, perform them, collect and then analyze the data. In order to do this effectively, the automated cyber experimentation framework must meet the following requirements [11, 17]:

1. **Expressivity** The framework must have a high level specification language using a common, extensible vocabulary to make the framework domain independent.

2. **Flexibility** The framework must be flexible enough to support numerous types of experiments.

3. **Portability** The framework must be agnostic to the tools used to implement the experiment. For example, it should be able to support multiple virtualisation providers as well as orchestration techniques.

4. **Repeatability** The framework must allowing others to replicate and verify experimental results.

2 Related Work

Previous research addresses the automation of cyber experimentation in different ways, and with different effectiveness. We review several research programs and address how they meet the automation experimentation framework requirements.

Security Experimentation Environment (SEER)[1] [15] is an experiment configuration tool that allows the user to configure experiments and provides monitoring support during execution. It facilitates repeated experiments with minimum effort, along with a controlled variation of experiment parameters. It also has a front-end graphical interface and scripts to configure and deploy experiments. There are also services provided to help users in creating experiments. SEER was specifically designed for use in DETERLab [2] and evolved from developing experiment methodology for Distributed Denial-of-Service (DDoS) defense systems and contains benign and malicious background traffic generation tools to aid in the experiment. SEER supports many traffic generators, and supports adding new attack tools or background traffic generators. SEER only supports DETERLab and is designed to create, execute and evaluate a DDoS experiment based on a given description and makes no indication of supporting automatic experimentation beyond parameter variation.

Control and Management Framework (OMF) [14] was developed for the Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) testbed at Winlab, Rutgers University. Its architecture consists of three logical panes: Control, Measurement, and Management. It has an Experiment Controller, Resource Broker, and Aggregate Manager, and collection tools to record experiments. The framework provides the user with a domain-specific language named OMF Experiment Description Language (OEDL) [14] to describe an experiment. It includes the ability to create prototypes so application descriptions can be reused. The OEDL language features allow the creation of complex experiments, but execution is limited to ORBIT wireless testbeds. The OMF team has started discussions on describing hardware resources in an ontology to make experiment descriptions richer, and possibly transferable [14].

Network Experimentation Programming Interface (NEPI)[2] [13] is a Python-based library created to efficiently manage network experiments across a variety of environments (e.g. PlanetLab, OMF wireless networks) enabling design, deployment and control of complex experiments. The user sets up an experiment workflow by declaring the network components and the applications it needs to execute in the form of Python scripts or a graphical user interface. The goal of NEPI is to ease simplify the evaluation and reproduction of experiment scenarios. The user can describe arbitrarily complex experiments in a generic way. NEPI supports many virtualisation providers as it has a generic description of hardware resources, however, there is no description of any experiment execution parameters or how it executes artificial traffic. NEPI requires users to describe in detail their experimentation environment which requires the user to build and provision their experiment, requiring they know all the implementation details for the given experiment.

The Cyber Scientific Test Language (CSTL) [8] is an ontology-based experiment design language. The CSTL represents the entire experiment workflow which entails experiment planning, execution, and analysis cycle. It consists of a complex ontology which provides the building blocks for the cyber experiment [8]. Their goal was to abstract detail from the description of the experiment so the user could focus on the science of the experiment. CSTL descriptions allow it to infer values for resource execution, rather than relying on the experimenter to fully specify them.

Hippolyte shares many design goals with CSTL. CSTL supports transformations allowing it to work with other virtual providers, and includes the use of ontologies to enable high-level specification of the experiment stages. However, the closed nature of CSTL and its limited deployment encouraged us to continue with the development of Hippolyte.

3 Hippolyte

Hippolyte has been designed to overcome the issues found in previous work and at the same time meet the requirements for automated cyber experimentation. In this section, we explain how Hippolyte addresses the requirements for automated experimentation and additional features that increase efficiency.

Fig. 1 shows the main components of the Hippolyte framework. The Experiment Description and Experiment Goals are provided by the user through the Central Interface. The remaining components operationalize that information into a network topology, instantiation of services and traffic generators, and series of experiment trials to test the Experiment Goal hypothesis.

3.1 Expressivity

Hippolyte defines the context and objectives of the experiment in a high-level language that captures the goals and dependencies of experimental operations in a domain-independent manner. The framework translates these high-level concepts into instructions for the external environment and monitors its output. Hippolyte represents all aspects of the experiment and experimental process in the OWL ontology language, enabling semantically consistent and reusable definitions of goals, hypotheses, networks, network assets and experimental results [7]. This enables Hippolyte to satisfy requirement 1.

Because the Hippolyte high-level language includes a reasoning component, it does not require the user to understand all the various treatment variables necessary in building an

[1] http://seer.deterlab.net/
[2] http://nepi.inria.fr

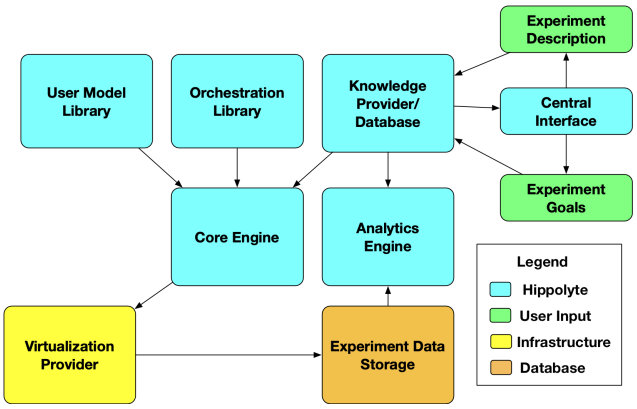

Figure 1: Overview of the various components of the framework.

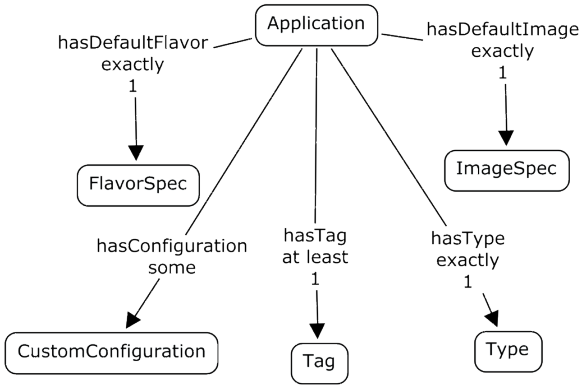

Figure 2: High level specification of an Application Test Scenario. The ontology figures in this paper were created using the COE ontology editor [5])

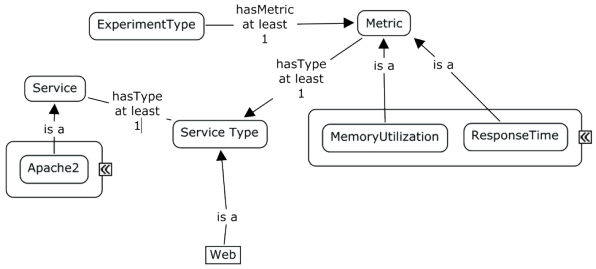

Figure 3: Experiment goal specification showing the connection between Metrics and Service Type

experiment. With the Hippolyte knowledge base, the user can choose to provide a few of the details of an experiment and the framework can still design, execute and analyse the experiment using the knowledge to fill out any information that is missing from the experiment description.

In the following subsections, we discuss two aspects of the language. For a more complete treatment, see [11].

3.1.1 Test Scenario

The Test Scenario is the application or network configuration that is to be tested. The Test Scenario is described using the Hippolyte language, a portion of which is shown in Fig. 2. Based on the knowledge of the object of the experiment, Hippolyte can fill in additional details necessary to complete the scenario and meet the experiment goals. Hippolyte has a library of application information including the supported operating systems, minimum hardware specifications, operating system service names, class of service, and configurable options. The user can define a new application if there is no existing definition for the application.

3.1.2 Experiment Goals

During the experiment description phase, a user has to choose the conditions under which an experiment is considered a success or a failure. The user can choose these conditions, which are one or more metrics and their success criterion. Hippolyte finds the metrics most closely related to the type of experiment and allows the user to choose from among them the most appropriate for the experiment.

Metrics can be collected locally or via remote sensors, depending on the metric. For instance, finding the delay in receiving successful web requests due to an attacker overloading the web server would entail having the sensors installed remotely. Hippolyte makes the decision of where to place sensors based on the experiment type and the information each metric provides.

3.2 Flexibility

Hippolyte's high-level experiment description language and reasoning capabilities enables considerable flexibility (requirement 2) in specifying and monitoring different types of experiments.

3.2.1 Topology Design and Generation

An experiment can result in having multiple testbeds due to different types of networks or factors such as hardware specifications. Hippolyte uses the definitions of various types of experiments and a library of network templates to design and configure the testbeds. These network templates also contain information to supply routing configuration used to configure the testbeds.

Designing the experiment is a two-step process. First, the framework establishes the number of different types of net-

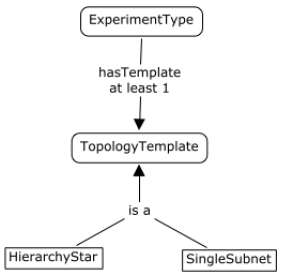

Figure 4: High level view of the relationships of Experiment Type to the Topology Design

works or scenarios that the experiment should operate on. Second, hardware factors like CPU, Memory and Disk that are not easily interchangeable require the creation of a new testbed for each combination. The total number of testbeds will be the product of the number of scenarios and the number of hardware combinations.

3.2.2 Virtual Testbed Generation

After the design phase, Hippolyte instantiates the topology in the selected virtualization provider, including configuration options such as the memory and disk sizes of the machines, their operating systems, etc.

Fig. 5 is a representation of the experiment in the ontology with some of the relationships needed to establish the base steps of an experiment.

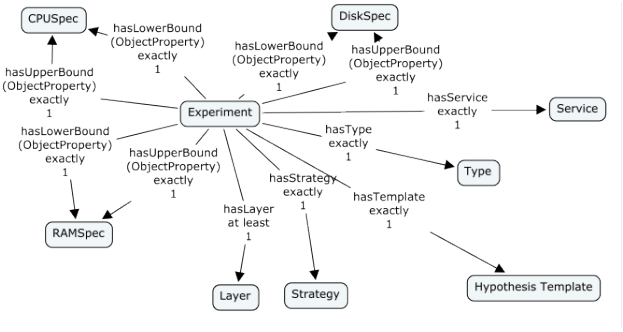

Figure 5: High level view of the Experiment in the ontology

The representation contains the hardware configurations, the connection configuration to each testbed (*Layer*), the *Strategy* to choose from various independent variables, etc. It links to the Experiment Goals as well.

While in the design phase, the framework tags each virtual machine to be created with identifiers that are used later while orchestrating the deployment and configuration of applications. For example, if we need a web server for an experiment, one virtual machine would be tagged with *web* and *nginx* to act as orchestration goals.

Hippolyte communicates with the running experiment trial over a service link to enable data collection and virtual machine management tasks (see Fig. 6). Communication controlling the execution of the experiment takes place over private links that are created during initialization. This isolates experiment data collection from contamination with experimental control messages. The framework can distinguish between these links and directs traffic accordingly.

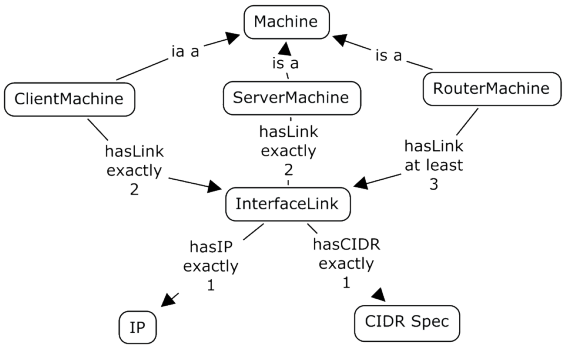

Figure 6: High level view of a host having multiple links for communication, which isolates the experiment

3.2.3 Metric Tracking

Every experiment trial is executed with multiple independent variable combinations that need to be tracked. The data generated from these trials also need to be tracked (see Figure 3. Hippolyte uses the virtual testbed tags generated during design to identify each trial and initializes the scenario application, services, and user models to label any data that is sent for analysis. The process for this is described in Section 4.

3.3 Portability

To advance the science of cybersecurity, it is critical that experiments be portable (requirement 3) to new testbed infrastructures and network tools. A key part of this portability is the ability to use multiple network infrastructures and tools as part of the experiment. The Hippolyte high-level language creates a level of abstraction where these variants can be addressed consistently.

3.3.1 Virtualization Provider

The virtualization provider is the interface between the framework and the virtual machines that are used in experimentation. This enables the framework to be agnostic of the un-

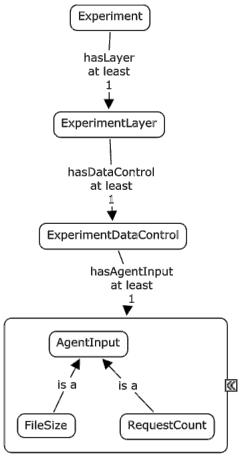

Figure 7: High level view of Metric Tracking

derlying virtualization tools. It expects the provider to create a communication link with each virtual machine along with a set of required interfaces necessary to fulfil various functions such as instantiating new virtual hosts, and adding internet addresses. Virtual Infrastructure for Network Emulation (VINE) [4] is an in-house developed virtualization provider that provides us with the necessary features and also supports multiple infrastructure technologies including OpenStack, VMWare, and VirtualBox, while using a unified interface [17].

3.3.2 Orchestration

After the testbed/s are instantiated on the virtual provider and the necessary installation parameters are decided, the framework begins orchestrating the installation and configuration of these goals. Hippolyte generates all of the information necessary to orchestrate, install and configure the testbed using an orchestration tool, such as Ansible or Chef. It gathers this information from the experiment ontology, the testbed and tags it has assigned to each host on the testbed (Section 3.2.2).

3.3.3 User Models

To ensure cyber experiments realistically emulate operational scenarios requires user models that can accurately emulate human tasks. These models should be able to adapt to changes in the environment. Hippolyte can make use of a variety of user modeling tools. One tool that integrates tightly with Hippolyte is called Yoshka, an adaptive user modeling tool that uses conditional behavior trees to emulate network traffic [10].

Finding the correct user model for an experiment is achieved by finding the type of the experiment and the type of service that Test Scenario uses. There can be multiple user models from multiple sources used in a single experiment.

Fig. 8 shows the central concepts in the ontology that establish these relationships.

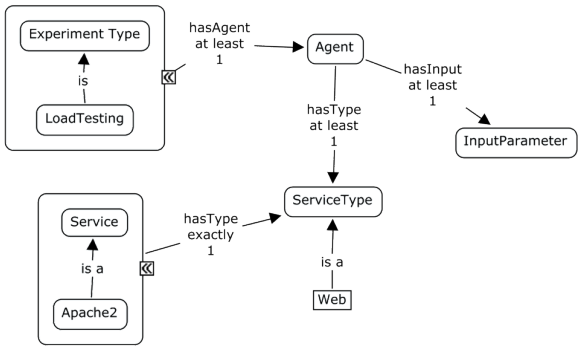

Figure 8: High level view of User Model Agent and the connection between service types

3.4 Repeatability

The high-level specification of experiments and experiment testbed requirements enables Hippolyte to quickly spin up testbeds that have all of the information needed to replicate a given experiment. This provides experimenters with the ability to easily conduct multiple trials of an experiment under widely varying conditions. When taken in combination with the measures in Hippolyte for meeting the portability requirement, test scenario, topology, application, service, and metric information can be provided to other researchers for independent validation of the experimental findings, thus satisfying requirement 4.

3.5 Additional Features

Hippolyte has two additional features that reduce experimenter workload and makes experimentation more efficient.

- **User Model Library:** Hippolyte comes with various user models to support different kinds of experiments. These models are built to work with Yoshka and additional models can be added over time to increase support.
- **Orchestration Library:** Hippolyte comes with in-built orchestration tools to assist in deploying applications and services as well as configuring them. Additional definitions can be added over time to support deploying more applications.

4 Illustrative Scenario

In this section, we give a very brief overview of an experiment using the Hippolyte prototype.

In this example, the user wants to test a hypothesis that an *Apache2* web server utilizes only up to 95 percent of the host memory for the following controlled input range shown in Table 1.

Control Variable	Lower Bound	Upper Bound	Step
Request Count	2000	3000	1000
Request Rate /s	1000	2000	500
File Size	10MB	20MB	5MB

Table 1: Controlled Input Range

The Test Scenario includes this information plus the hardware configuration for the web server of 1 CPU, 1024MB of RAM and 10GB of disk space. The user inputs this description on the central web interface and specifies *Load Testing* as Experiment Goal. The specification of this goal instantiates a number of other network entities, such as network clients to communicate with the server (see Fig 10).

Hippolyte calculates the combinations of the independent input variables by performing an exhaustive search, an example of which is shown in Fig. 9.

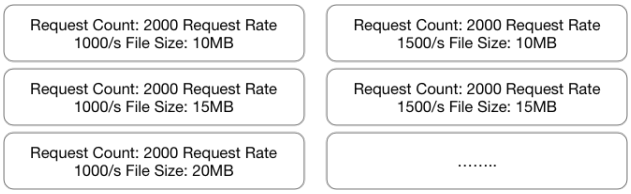

Figure 9: Subset of the combinations of the controlled parameters

Hippolyte updates the user on the progress of the experiment through the central interface. The initialization steps are listed below.

1. Generates the combinations of experiment variables.
2. Designs the topology and notifies the virtual infrastructure provider to generate the testbed.
3. Once generated, orchestrate the deployment and configuration of services and user agents.
4. User agents query the Test Scenario specification for their operating parameters and begin execution.
5. Experiment trials begin.

The process of user agents querying is depicted in Fig. 10. The user can also view a dashboard showing collected metrics in real time as the experiment progresses to see the various effects on the testbed as a whole (Fig. 11).

As each combination is being executed, the results are tagged and stored in the database as seen in Fig 10. After all

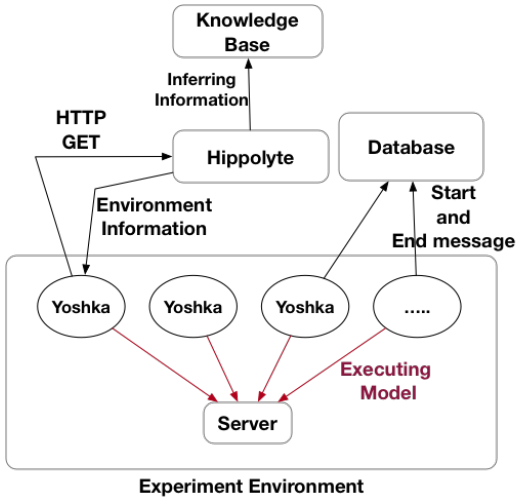

Figure 10: User Model Agent communications within the experiment testbed

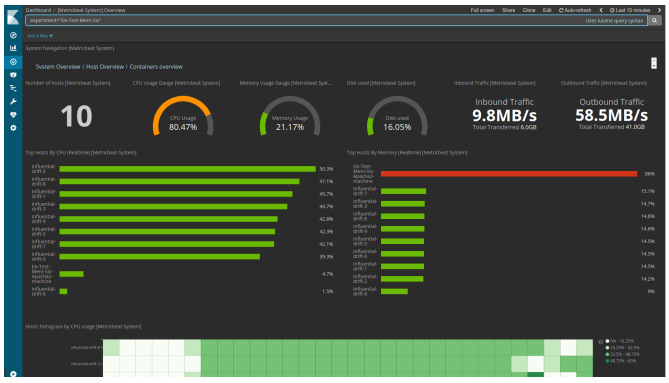

Figure 11: Metric dashboard to view metrics in real time.

executions are complete, the user has the option to tell the framework to began analyzing the data and test for the hypothesis. It displays the results on the web interface in charts where each chart fixes the control value of all independent variables except one. An example graph is shown in Fig. 12.

5 Advantages of an automated framework

While automated experimentation for cyber security is a difficult task, there are a number of benefits that result from it, some of which are outlined below.

5.1 Scale

Automated experimentation frameworks are capable of adjusting the scale of an experiment with minimal user input. Such frameworks manage experiment scale through topological

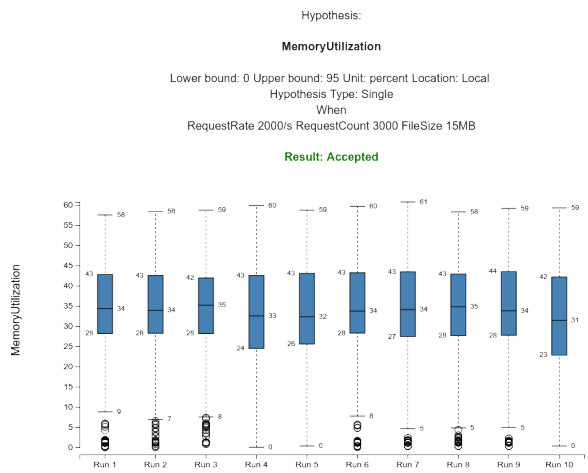

Figure 12: Variance of Memory Utilization for background traffic of 3000 requests sent at 2000 requests per second while downloading a 15MB file executed 10 times. The Null-hypothesis set forth for the *Apache2* test scenario could not be rejected.

definition and provisioning of testbed infrastructure. Naturally, these capabilities greatly reduce the start-up workload on the experimenter.

5.2 Reducing the number of experiments

Automated experimentation frameworks that define experiment variables semantically, like Hippolyte, can be used to detect causal relations. Identifying causal relations between variables limits experiments to those variables and their combinations preventing an exhaustive search [3]. This capability could help in reducing the number of trials required, which could reduce the total experiment time even more. More recent work is investigating how the ordering of experiment trials can maximize the amount of information gained about the Experiment Goals, which may lead to an early stopping criteria rather than executing all possible variable combinations [6].

5.3 Closing the Experimentation Loop

Performing experiments requires completing all the steps to design, create, analyse data and evaluate the experiment. Closing the loop requires that the framework also understand the analysed data and determine the next hypothesis to investigate. One of the advantages of having an automated system to provide cyber experimentation is that the system can be built to generate evaluation criteria based on the results of the previous experiment. With this, the system can drill down on the most causality relevant variables without a human in a loop. There has been previous work in other domains with the goal of closing-the-loop [18]. Waltz [18] sums this up nicely, '*Computers with intelligence can design and run experiments, but learning from the results to generate subsequent experiments requires even more intelligence.*'

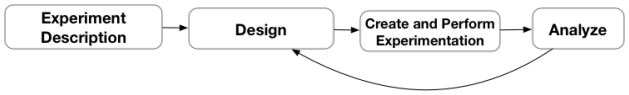

Figure 13: Experimentation Process

To achieve this, automated experimentation frameworks need the ability to form new hypotheses by collecting evidence about observations and using abductive reasoning to generate new hypothesis [9, 16]. There are many references to hypothesis formation performed in other domains such as physics and geology [16]. This capability would amplify the experimenter's efforts by automatically creating, running, and analyzing new experiments based on the science goals specified by the user.

5.4 Informed Decisions

One of the main advantages for this framework, other then designing and creating experiments, is in the analysis of data: helping the user make decisions about their application. Automated experimental frameworks can show the statistical difference between trials, enabling users to observe the causality of variables affecting their application. Automatically calculating experiment significance can also save time and effort.

6 Conclusion

In this paper, we presented Hippolyte, a new framework for automated cyber experimentation. We surveyed existing tools and previous work that focused on various parts of the cyber experimentation lifecycle. Lessons learned from this previous work enabled the generation of a new set of requirements for automated cyber experimentation. We presented a case study using Hippolyte to generate an experiment with a series of trials to test the capabilities of a web server and to present the analyzed data. While not complete, Hippolyte demonstrates the ability to perform all aspects of the cyber experimentation lifecycle except for automatic hypothesis generation, which would completely close the experimentation loop.

The benefits of automated cyber experimentation make the development of automated experimentation frameworks like Hippolyte not only appropriate, but inevitable. To advance the science of cybersecurity at a rate comparable to the rate of advancement in cyber attacks, more research and development will need to address user and application libraries, network topology templates, and experimental templates. The ability to experiment quickly, easily, continuously and cheaply will

allow cyber researchers and practitioners both to better understand their systems and take the advantage away from cyber criminals.

References

[1] R. Bajcsy, T. Benzel, M. Bishop, B. Braden, C. Brodley, S. Fahmy, S. Floyd, W. Hardaker, A. Joseph, G. Kesidis, K. Levitt, B. Lindell, P. Liu, D. Miller, R. Mundy, C. Neuman, R. Ostrenga, V. Paxson, P. Porras, C. Rosenberg, J. D. Tygar, S. Sastry, D. Sterne, and S. F. Wu. Cyber defense technology networking and evaluation. *Commun. ACM*, 47(3):58–61, March 2004.

[2] Terry Benzel. The science of cyber-security experimentation: The DETER project. In *In Proceedings of the Annual Computer Security Applications Conference (ACSAC '11)*, 2011.

[3] Frederick Eberhardt, Clark Glymour, and Richard Scheines. N-1 experiments suffice to determine the causal relations among n variables. In *Innovations in machine learning*, pages 97–112. Springer, 2006.

[4] Thomas C. Eskridge, Marco M. Carvalho, Evan Stoner, Troy Toggweiler, and Adrian Granados. VINE: A cyber emulation environment for MTD experimentation. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, MTD '15, pages 43–47, New York, NY, USA, 2015. ACM.

[5] Thomas C. Eskridge, Pat Hayes, Robert R. Hoffman, and Margaret Warren. Formalizing the informal: Building a bridge between concept maps and the semantic web. In *Proc. of the Second Int. Conference on Concept Mapping*, pages 247–254, San José, Costa Rica, 2006.

[6] Thomas C Eskridge, Dhanish Mehta, and Marco Carvalho. Using bayes networks to predict information gain on experiment trials, in preparation. August 2019.

[7] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[8] Peter Haglich, Robert Grimshaw, Steven Wilder, Marian Nodine, and Bryan Lyles. Cyber scientific test language. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2011*, volume 7032 of *Lecture Notes in Computer Science*, pages 97–111. Springer Berlin Heidelberg, 2011.

[9] Peter D. Karp. Hypothesis formation as design. In Pat Langley, editor, *Computational Models of Scientific Discovery and Theory Formation*. Morgan Kaufmann, Burlington, 1990.

[10] Samir Mammadov, Dhanish Mehta, Evan Stoner, and Marco M Carvalho. High fidelity adaptive cyber emulation. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on*, pages 1–8. IEEE, 2017.

[11] Dhanish Mehta. On the automation of cyber experimentation. Master's thesis, Florida Institute of Technology, 2018.

[12] J. Mirkovic, T. V. Benzel, T. Faber, R. Braden, J. T. Wroclawski, and S. Schwab. The deter project: Advancing the science of cyber security experimentation and test. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 1–7, Nov 2010.

[13] Alina Quereilhac, Mathieu Lacage, Claudio Freire, Thierry Turletti, and Walid Dabbous. NEPI: An integration framework for network experimentation. *19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5, 2011.

[14] Thierry Rakotoarivelo, Maximilian Ott, Guillaume Jourjon, and Ivan Seskar. OMF: A control and management framework for networking testbeds. *SIGOPS Oper. Syst. Rev.*, 43(4):54–59, January 2010.

[15] Stephen Schwab, Brett Wilson, Calvin Ko, and Alefiya Hussain. Seer: A security experimentation environment for deter. In *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test on DETER Community Workshop on Cyber Security Experimentation and Test 2007*, pages 2–2. USENIX Association, 2007.

[16] Jeff Shrager and Pat Langley. Computational models of scientific discovery and theory formation. In *Symposium on Computational Models of Scientific Discovery and Theory Formation (1989: Stanford University)*. Morgan Kaufmann Publishers, 1990.

[17] Evan Lawrence Stoner. A foundation for cyber experimentation. Master's thesis, Florida Institute of Technology, 2015.

[18] David Waltz and Bruce G. Buchanan. Automating science. *Science*, 324(5923):43–44, 2009.

[19] Kara Zaffarano, Joshua Taylor, and Samuel Hamilton. A quantitative framework for moving target defense effectiveness evaluation. In *Proceedings of the Second ACM Workshop on Moving Target Defense*, pages 3–10, 2015.