# A Two-Constraint Approach to Risky Cybersecurity Experiment Management

John Wroclawski, Jelena Mirkovic, Ted Faber, and Stephen Schwab

*Abstract* — **Cybersecurity research demands extensive experimentation to be validated. This experimentation is inherently risky: it may involve any combination of live malicious code, disruptive actions and connectivity to the active Internet. Previously, risky experiments were heavily contained to reduce danger to the experimental infrastructure and the Internet. We argue that such an approach unnecessarily stifles research.**

**In this paper we explore a collaborative two-tiered risky experiment management model. It incorporates input from both experimenters and testbed operators; each party specifies constraints on their component's behavior. Experiment constraints limit behavior in ways that do not affect an experiment's validity, thus meeting usability goals. Testbed constraints build on experiment constraints to meet required safety goals. When combined, experiment and testbed constraints ensure that experiments are both useful to researchers and safe for the testbed and the Internet.**

*Index Terms*—**Computer Network Security, Computer Facilities, Cooperative Systems**

## I. Introduction

As computer networks and systems become ever more fundamental to modern society, concerns about cybersecurity become increasingly important. The challenge of securing modern cyber-systems is further increased by their rapidly growing complexity and scope of application. In response to these factors, the cybersecurity research community demands increasingly realistic and sophisticated experimental capabilities, tools and methodologies.

Such experimental cybersecurity research is often *inherently* risky. An experiment may involve releasing live malware, operating a botnet, or creating highly disruptive network conditions. These risks are fundamental to successful research; realism is required in replicating attacks so that proposed defenses can be thoroughly tested and future attacks anticipated.

The common response to this requirement is to implement strict containment or isolation capabilities within the experimental environment, in an attempt to ensure that no actual damage will be caused by an experiment. Such constrained environments are often provided by a *testbed,*[1] and we focus on testbed-based environments in our work. Depending on the testbed, containment mechanisms may include complete disconnection from the outside world, disk scrubbing before and after each experiment, prohibiting experiments that run live malware code, and the like.

But containment itself is highly limiting. A fully contained experiment is hard to observe, hard to establish, and hard to *control*, because it is so completely isolated from its environment. Similarly, it is hard to *create* with any assurance. Sneak paths, equipment failures and design mistakes can render containment ineffective in myriad unexpected ways. Most importantly, containment is not very *useful.* A far more interesting and powerful class of experiments are those that *do* interact with their larger environment, but only in carefully controlled and well-understood ways.

Our work aims to radically increase the scope and usefulness of testbed-based experimental cybersecurity research by accommodating this observation. We do this by moving from simple containment to risky experiment *management* as a strategy. The work is based on a simple line of reasoning:

- If the behavior of an experiment is completely *un*constrained, the behavior of the host testbed must be completely *constraining*, because it can assume nothing about the experiment.
- But, if the behavior of the experiment *is* constrained in some known and well-chosen way or ways, the behavior of the testbed can be less constraining, because the *combination* of experiment and testbed constraints together can provide the required overall assurance of good behavior.

This concept is illustrated in Figure 1. We call the first sort of constraints "experiment constraints" or "T1 constraints." We call the second class of constraints "testbed constraints" or "T2 constraints," and often refer to overall concept as the

This material is based upon work supported by the Department of Homeland Security and the Space and Naval Warfare Systems Center, San Diego, under Contract No. N66001-07-C-2001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Homeland Security or the Space and Naval Warfare Systems Center, San Diego.

Wroclawski, Mirkovic, and Faber are with the University of Southern California Information Sciences Institute. (e-mail: {jtw, mirkovic, faber}@isi.edu).

Schwab is with SPARTA Inc. (e-mail: Stephen.Schwab@sparta.com).

[1] Experimental capabilities, tools, and methodologies are often brought together to form *testbeds.* Testbeds [1][5][6][7] are becoming more prevalent in the larger computer systems research community because they allow researchers to share resource infrastructure, to access powerful tools, and act as a nucleation point for collaboration. These benefits combine to make testbeds a natural place for researchers in the same field to meet and conduct large-scale, complex, or cooperative experiments, and to build on others' work.

"T1/T2 model."

The remainder of the paper is devoted to exploring this idea further. We discuss the benefits of the approach in Section II, present a simple pedagogical example in Section III, consider some key aspects of the approach in Section IV and discuss a proposed initial implementation in Section V.

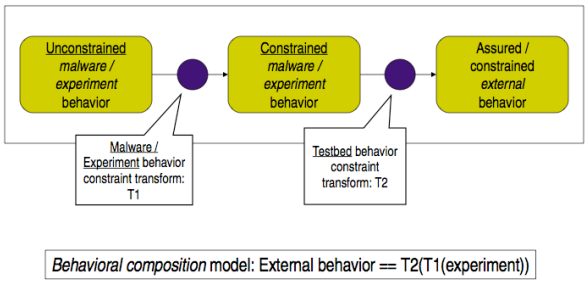


***Figure 1: Composition of T1 and T2 Constraints***

## II. Benefits

The separate expression of experiment and testbed constraints in our model represents a *separation of concerns*. This separation of concerns is extremely powerful, because it allows experiment constraints and testbed constraints to be framed and expressed independently, in terms directly meaningful to their respective audience of experimenters and testbed designers. Because experiment (T1) constraints are explicit and presented in the language of the experimenter, he can begin to reason directly about which constraints he might accept without affecting the validity of his experiment, and without concern about how the testbed (T2) constraints are implemented. Similarly, the testbed designer and operator can reason about testbed constraints in terms directly related to facility function and implementation – the area they are best equipped to understand.

The mechanism of composing experiment and testbed constraints to obtain a desired overall assurance is interesting and rich. As suggested below, it is possible to obtain useful risk mitigation behaviors through the composition of a broad range of T1 and T2 constraints, tuned to the needs of different experiment classes and acceptable risk levels. This richness is critical to our goal of supporting a diverse and sophisticated range of experiments.

The factoring of constraints into separate experiment and testbed constraints simplifies risky experiment design and increases reusability. The experimenter presented with a selection of predefined T1 constraints can quickly choose an appropriate set – one that does not affect the validity of his experiment, but will still support known external properties. Similarly, the testbed designer can begin to reason about how to offer different "standard" testbed constraint environments as well known, robust, and documented services, rather than having to separately consider and review his operating procedure for each new experiment.

Finally, the model we present leads to significantly increased verifiability of safe overall behavior. Two reasons are relevant. First, because experiment and testbed constraints are factored, the mechanisms implementing each can be simpler, extensively tested, and then reused across many experiments. Second, the assurance provided by the composition of these simpler constraints C = T2(T1()) may be subject to formal analysis.

## III. Example

As a pedagogical example, imagine an experiment designed to study worm propagation. The behavior of this experiment could be constrained as follows:

- The worm code could be designed to commit suicide if it does not receive a "heartbeat" message from a specific source – perhaps digitally signed - every 30 seconds. This behavior is a constraint on the experiment behavior, or T1 constraint.
- The source node generating the heartbeat message could be located within a testbed, and the message blocked from propagating outside the testbed through a variety of mechanisms. This is a testbed, or T2 constraint.

When these constraints are combined, the overall effect is clear: the worm dies if it leaves the testbed.

The astute reader may now notice a problem: A fast-acting worm might spread very far in the 30 seconds allowed by the above constraints. To address this, one could define an additional constraint – that the worm is only allowed to send messages at the rate of one message after each heartbeat. We now have *two* T1 constraints: one related to message generation rate, and one related to requirement for a heartbeat message. It should be apparent that with these and the T2 heartbeat constraint, the resulting overall behavior is that the worm can only propagate one generation outside the testbed environment.

This example illustrates a key benefit of the T1/T2 model: its ability to make explicit its impact on experiments. To illustrate, we observe that our second experiment above is valid only if limiting the message rate does not affect whatever property the experiment is designed to observe. This may or may not be true. If the experimenter is interested in how the worm chooses its next target, a limited message rate may be perfectly acceptable. If the experimenter is interested in timing the worm's spread, it clearly is not. But, because the constraints on experiment behavior are explicit and defined, the experimenter can reason about the question and its answer.

A further benefit is also apparent: constraints may be explicitly selected to match the experiment. If the message rate limiting constraint is unacceptable, it may be possible to devise an entirely different set of T1 and T2 constraints – ones that would be acceptable for a rate-of-propagation experiment, but may be unworkable for a choice-of-target experiment. For example, the experimenter could limit destination addresses in scans to a small, non-routable range. This ensures that scans will never leave the testbed, but prohibits studies of worm target selection at larger scale. More generally, we note that entirely different constraints, suited to entirely different experiments, may produce the same overall limits on

undesirable behavior. By choosing T1 and T2 constraints appropriate to the circumstances, our model allows a wide variety of research to be carried out, while still providing well-defined, verifiable limitations on risk.

## IV. Design Issues

This section considers some key design issues related to the use of the T1/T2 model to manage risky experiments.

### A. Top-Level Goals

The ultimate use of the two-constraint model is to implement a well-defined set of goals for each potentially risky experiment. These goals may be further subdivided as follows:

1. Researcher's *experiment* requirements. These goals define experiment categorization or behavior required to produce a valid research result. One subclass of experiment goals identifies explicitly risky behaviors that the user desires in the experiment, such as: self-propagating malware, high-volume traffic, etc. Another subclass identifies required behaviors that are not themselves risky, but that are important to experiment validity. These might be blocked by certain testbed mechanisms if not explicitly identified.
2. Researcher's *privacy* goals. It is likely that some monitoring of user actions and traffic by the testbed will be necessary for several purposes, including: (1) to support overall testbed performance monitoring and management, (2) to ensure that user constraints are audited, as described in Section IV.C, (3) to reduce testbed liability in case of malicious incidents. Current academic testbeds assume one liberal set of user privacy goals, but this is not realistic since, e.g., commercial users have very different privacy expectations than academic users.
3. User and Testbed safety goals. While the safety goals of each researcher and testbed will have a common flavor of "no harm to other users, the testbed or the Internet," each testbed may define notions of harm in different ways. Unacceptable behaviors differ, depending on the testbed's mission and the policies at the hosting institutions. Testbed goals must thus be explicitly defined in detail.

Within this space, a first design task is to identify specific goals that are useful to researchers. While the list of goals is potentially infinite, observation of current research testbed use patterns suggests that a small subset list may address the needs of many common experiments, providing a useful starting point for both further work and experimental validation of the T1/T2 model's usefulness.

### B. Constraint Sets

Our ultimate research goal is the development of a fine-grain framework for T1 and T2 constraints and a formal structure to reason about their composition to meet top-level goals. However, as a more immediate, practical step towards deploying useful risky experiment management capabilities and towards assessing the validity and value of the T1/T2 model, we introduce the concept of the T1/T2 *constraint set.*

A T1/T2 constraint set is a pre-established set of complementary T1 and T2 constraints that, when met, allow useful, potentially risky experiments to be run with well-understood limits on external behavior. To be useful, a T1/T2 constraint set must exhibit all of the following properties:

- The semantic combination of the chosen T1 and T2 constraints must produce a desirable risk management result.
- The T1 constraints must be useful to the experimenter: some set of interesting experiments must execute correctly under the given constraints.
- The T2 constraints must be implementable and verifiable within a particular testbed environment.

It is useful to define and implement a number of T1/T2 constraint sets within a given experimental environment. Figure 2 shows the motivation. In the figure, we assume that each of the represented constraint sets provides the same risky behavior management semantics. But we see that different T1 and T2 constraints produce different results along another axis. The top set, with very weak T1 constraints, allows an essentially unconstrained experiment to proceed within the testbed, but in return requires strict constraints on testbed behavior and perhaps on experimental procedure. In contrast, if the behavior of the experiment itself is known to be strongly constrained, the behavior of the testbed can be less constraining, leading to a simpler experiment that is more able to interact with its environment. Because T1 and T2 constraints are explicitly separated within the set, we are able to reason about these tradeoffs. Because several complementary constraint sets can be made available, the researcher is able to act on that reasoning to select the most suitable one.

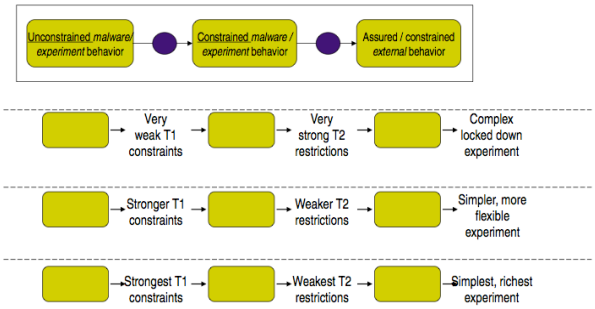


***Figure 2: Alternate T1/T2 Constraint Sets***

While we have introduced constraint sets as a simplifying abstraction, we note that their ultimate value may be more fundamental. A constraint set is an example of a "design pattern" for risky experiments, and the existence of proven, well validated and documented constraint sets may serve to simplify the job of the experimenter in several dimensions. We thus expect to maintain the constraint set abstraction even as our work advances towards a more fine-grain constraint manipulation capability.

### C. Assuring T1 Constraints

The usefulness of the T1/T2 model depends fundamentally on the idea that it is possible to ensure that the behavior of an experiment is constrained in some dimension. At first glance, this is a challenging assumption. In fact, a variety of approaches to this problem are possible, each with its own strengths and weaknesses.

The *basis* of a T1 constraint is the premise or principle on which it is asserted. A number of bases are possible. Among these are:

**Belief.** The claim of constrained behavior on the part of an experiment can be asserted because the experimenter believes it to be true, based on understanding of the experiment and/or historical observation. Belief is a very weak basis, and only in very low-risk cases would it be appropriate to rely on belief alone. However, belief becomes a more legitimate basis for asserting a T1 constraint in conjunction with *monitoring* by the testbed to ensure that the asserted constraint is continuously valid.

**Correctness by construction.** A T1 constraint may be asserted on the basis that it is implemented and assured by automatically generated code or a similar mechanism. As an example, we are exploring modifications to Metasploit [8] that enable it to generate exploits with known T1 constraints.

**Verification.** A T1 constraint may be asserted on the basis that the code implementing the experiment has been *verified* externally to exhibit or enforce the constraint. Verification may range from human auditing of the code to the use automatic program verification tools. Here the explicit and narrowly scoped nature of T1 constraints may be particularly valuable in limiting the difficulty of the program verification problem.

Each of the methods above asserts the T1 constraint on a basis that is intrinsic to the experiment itself. A fourth option differs in that an external agent is used to ensure that the constraint holds. This is:

**Enforcement.** A T1 constraint may be asserted on the basis that it is enforced on the experiment by an external agent or action of the experimenter, independently of whether the experiment would exhibit the constraint on its own. Enforcement-based constraints are valuable because they are potentially very strongly assured, and because they provide a means to artificially constrain the behavior of an experiment within the T1/T2 framework.

A variety of enforcement mechanisms may be useful to ensure experiment T1 constraints in different circumstances. Among these:

**Wrapping** is the addition of a "wrapper" around existing code to limit its externally visible behavior along some dimension.

**System call interposition** is similar to wrapping in concept, but imposes the behavior constraint by restricting the experiment code's interaction with the host operating system.

**Code modification** is the external action of modifying the experiment code itself to impose a constraint on behavior.

**Emulation** is the execution of experiment code in an emulated environment that constrains behavior, and/or execution of an emulation of the risky code itself, such as the use of a synthetically generated "worm" rather than the actual malware.

Each of these bases for ensuring that T1 constraints hold will be appropriate in different circumstances. Combinations will likely be used when several constraints are to be ensured. Central questions to be resolved by ongoing work are the practicality of enforcing useful T1 constraints by these methods, and the best ways to provide these constraint management tools to the experimenter.

## V. Proposed Initial Implementation

We are presently developing an initial implementation of the ideas presented here, within USC/ISI's DETER [1] testbed. DETER is a large facility located at USC/ISI and UC Berkeley, targeted to support cybersecurity research. The testbed consists of some 370 experimental nodes that can be reserved for exclusive use in creating experiments, together with additional hardware such links and switches that are shared by all experiments. DETER's control software is derived from Emulab [5]. An experimenter reserves nodes by "creating an experiment" with a desired topology and characteristics, using a Web interface offered by the testbed. The testbed then allocates resources according to experimenter's specification. Our work builds on several existing aspects of the DETER facility.

We assume that each experiment is potentially risky until proven otherwise. We initially address three types of risky behavior: (1) running malware code, (2) exhibiting disruptive behavior such as denial of service attacks, and (3) requiring connectivity with the outside Internet. The first two behaviors are intentionally risky, whereas the third behavior may be risky by accident, if experimental traffic with the outside is misconfigured and overloads resources, provokes an external attack on the testbed, or creates liability.

We identify and address the following specific experimental risks: (1) malware traffic may infect testbed infrastructure needed for correct operation such as the testbed's control software or switches, (2) experimental traffic of any sort may overload the testbed's control plane or shared hardware, (3) disruptive actions may affect the control plane or shared hardware (4) in experiments with outside connectivity, experimental traffic sent to remote machines may infect, overload or disrupt these machines and remote networks, (5) in experiments with outside connectivity, experimental traffic may provoke retribution toward the testbed (e.g., from the Storm Network [2]) or create liability problems for the testbed, (6) malware may stay resident on machines after they are reclaimed by the testbed and may affect future experiments by other users. We expect this list of risks to grow as we proceed with our work.

The above risks are contained via experiment and testbed constraints. Constraints are associated with each running experiment at its creation time, and are continuously active. Because a given experimental topology can be used for multiple different purposes over time, constraints would ideally be generated and applied dynamically. This however

increases the burden for users whose different runs exhibit the same risky behavior, and is also challenging because the testbed lacks means to detect different "runs" or changes in use within the same experiment. Instead, we associate constraints with experiments, but will provide mechanisms for users to modify these constraints while the experiment is active.

Our initial list of experiment (T1) constraints contains the following actions: (1) users limit scanning behavior (rate and/or target selection) of self-propagating malware, (2) users limit targets of disruptive actions, such as denial-of-service, to non-routable addresses within the experimental network, (3) users limit experimental connectivity with the outside world to a set of machines under their control, and to specific protocols, (4) users limit the rate of traffic in their experiments, (5) users limit all experimental traffic to the experimental network, (6) users implement signatures or self-terminating behavior in malware they plan to use. We expect to grow this list as our work progresses.

Section IV.C discusses different methods for ensuring the validity of T1 constraints. As an initial position, we adopt a verification approach. We require that all experiment constraints are auditable by the testbed. We utilize the testbed's experiment management infrastructure to implement monitoring tools: these tools verify that each constraint associated with an experiment holds throughout the experiment's lifetime. In case experiment constraints are violated, the management infrastructure will take corrective actions that may range from emailing the user and testbed operators to terminating the experiment.

Our initial list of testbed (T2) constraints and corresponding actions to be implemented includes: (1) isolation of experiments on the control plane using a separate virtual LAN for each experiment, (2) experimental traffic filtering and rate-limiting on the control plane using hardware-specific filters at switches to prevent disruption and overload of shared infrastructure, (3) allowing outside connectivity via specialized machines (tunnel nodes) that connect the experimental network to the outside, (4) controlling experimental traffic contents and rate with the outside Internet via firewall rules and the Bro intrusion detection system [3] for deep packet inspection, both installed on tunnel nodes, (5) recording of traffic on tunnel nodes, recording of login activity on experimental nodes, and association of traffic, logged in users, and experiment names for potential liability reasons. Again, this list of constraints and actions will grow during the course of our work.

To ensure that T2 constraints are continuously met, we again augment the testbed's management infrastructure to detect violations of these constraints caused by hardware or software failures. Because these T2 constraint failures should never occur, our initial response will be to log debugging information and terminate testbed operation.

To capture risk management parameters related to experiments, we are developing a domain-specific language known as REALM for specification of top-level goals and T1/T2 constraints. REALM specifications will be standardized, so users and testbed operators will have a limited choice of goal and constraint sets. The standardization enables us to automate specification processing and determiniation of testbed constraints, given top-level goals and experiment constraints. The REALM language will also be extensible, so novel goals and constraints can be added as our ability to support and reason about them develops.

Although it is possible for users to write REALM specifications directly, our intent is that REALM be the output and interchange language for a variety of tools that capture and manipulate risky experiment management information. As an example, we will explore strategies for high-level specification of goals and constraints within DETER's existing experiment management tool SEER [4]. User input will be recorded and automatically translated into REALM specifications, and the resulting constraints will be associated with the experiment.

Using this system, researchers will be able to specify experiment categorization, privacy goals and appropriate experiment constraints at the time of experiment creation. Testbed constraints will be generated based both on the REALM specification input by the user, and the testbed safety specification (also in REALM) defined once by the testbed operators. Mechanism is then executed to allow researcher and testbed operator to agree on particular available constraint sets, as described in Section IV.B. The selected constraints will then be put into place by the testbed and monitored by management infrastructure as described previously, to ensure that they are continuously enforced.

## ACKNOWLEDGMENT

The authors thank John Hickey, Alefiya Hussain, Calvin Ko, Kevin Lahey, Keith Sklower and Brett Wilson for ongoing discussions and advice during the course of this work.

## REFERENCES

[1] T. Benzel, R. Braden, D. Kim, B. C. Neuman, A. Joseph, K. Sklower, R. Ostrenga and S. Schwab, "Experience with DETER: A Testbed for Security Research," In Proceedings of Tridentcom (International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities), March 2006.

[2] J. Stewart, "Storm Worm DDoS Attack," SecureWorks Research, available at http://www.secureworks.com/research/threats/storm-worm

[3] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," Computer Networks, 31(23-24), pp. 2435-2463, December 1999.

[4] S. Schwab, B. Wilson, C. Ko, and A. Hussain, "SEER: A Security Experimentation EnviRonment for DETER," In Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test, August 2007.

[5] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, USENIX, Boston, MA, 2002.

[6] WAIL Network Testbed, http://www.schooner.wail.wisc.edu/

[7] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," Proceedings of Operating Systems Design and Implementation Symposium (OSDI '06), November 2006.

[8] J. C. Foster, "Metasploit Toolkit for Penetration Testing, Exploit Development, and Vulnerability Research," Syngress Publishing, 2007.