Empirical Validation of System Dynamics Cyber Security Models

Uma Kannan
Department of Computer Information Systems
Alabama State University
Montgomery, AL, USA
ukannan@alasu.edu

Rajendran Swamidurai
Department of Mathematics and Computer Science
Alabama State University
Montgomery, AL, USA
rswamidurai@alasu.edu

***Abstract*— Model validation, though a process that's continuous and complex, establishes confidence in the soundness and usefulness of a model. Making sure that the model behaves similar to the modes of behavior seen in real systems, allows the builder of said model to assure accumulation of confidence in the model and thus validating the model. While doing this, the model builder is also required to build confidence from a target audience in the model through communicating to the bases. The basis of the system dynamics model validation, both in general and in the field of cyber security, relies on a casual loop diagram of the system being agreed upon by a group of experts. Model validation also uses formal quantitative and informal qualitative tools in addition to the validation techniques used by system dynamics. Amongst others, the usefulness of a model, in a user's eyes, is a valid standard by which we can evaluate them. To validate our system dynamics cyber security model, we used empirical structural and behavior tests. This paper describes tests of model structure and model behavior, which includes each test's purpose, the ways the tests were conducted, and empirical validation results using a proof-of-concept cyber security model.**

Keywords*— *Cyber security; cyber security modeling; system dynamics; continuous simulation; simulation and modeling; cyber-attacks/defenses; empirical validation.

I. Introduction

System dynamics (SD) [1] discerns how systems change over time, through a continuous-event simulation methodology. In SD, as defined, is a unified whole that comes from an interaction over time amongst a collection of elements. [2]

SD, developed by Massachusetts Institute of Technology's (MIT) Forrester in the early 1960's, is a modeling technique that solves persisting and continual dynamic industrial management problems [3]. The application of SD helps solve problems concerning both business policy and strategy today [4, 5, 6].

The totality of the relationships between the physical processes, information flows, and managerial policies, in SD, defines the system's "structure." The focus of SD hones in on the understanding of the dynamics of the variables of interest that is created through the interaction of these components. Operating over time, the "dynamic behavior patterns" of the "structure" of the system is generated. Thus here, it becomes essential that a valid description of the real processes be provided by the defined model structure. [6]

A prototypical SD study first understands how and why the dynamics of concern is generated, then looks for ways to further improve the system's performance through searching the upper management's policies, long-term, macro-level decision rules. [6]

Model validation establishes confidence in the soundness and usefulness of a model. Making sure that the model behaves similar to the modes of behavior seen in real systems, allows the builder of said model to assure accumulation of confidence in the model and thus validating the model. While doing this, the model builder is also required to build confidence from a target audience in the model through communicating to the bases. [7]

No strict standard of statistical predictive validity is used in SD models, this is because, the best way SD models are characterized are as a collective deduction of a group based on only the understanding of a system at a certain point in time. The validation of said model relies on a group of experts agreeing on a casual loop diagram of the system. The usefulness of a model through a user's eyes, remains as a valid standard by which we can evaluate them. [2, 5, 6, 8]

Though numerous formal quantitative and informal qualitative tools are used in model validation, both in general and specifically in the field of cyber security, system dynamics uses validation through a casual loop diagram of the system agreed upon by a group of experts. Amongst others, the usefulness of a model through a user's eyes, is a valid standard by which we can evaluate them. To validate our system dynamics cyber security model, we used empirical structural and behavior tests. This paper describes tests of model structure and model behavior, which includes each test's purpose, the ways the tests were conducted, and empirical validation results using a proof-of-concept cyber security model.

This paper present our experience in using cybersecurity testbed to empirically validate a cybersecurity system dynamics proof of concept model.

978-1-7281-0137-8/19/$31.00 ©2019 IEEE

II. System Dynamics Model Validation

The system dynamics model validation is a two-step process: First establish the validity of the structure of the model (structural testing), and then evaluate the accuracy of the model behavior's reproduction of real behavior (behavioral testing) [9].

To measure the quality of our model (or to ensure that the model was successfully completed), we ran the following structural and behavioral tests on model-generated values against hypothetical or real system parameters/values:

- *Tests of Model Structure:* Structure verification test, parameter verification test, extreme conditions test, and dimensional consistency test.
- *Test of Model Behavior:* Behavior reproduction test and behavior anomaly test.

A. Tests of Model Structure

To test model's structure direct structure tests are used. There are two types of direct structure tests: empirical and theoretical. In the empirical structural tests each model equations or relationships are compared with the real system's quantitative or qualitative information; whereas, in the theoretical structure tests the model equations or relationships are compared with the generalized knowledge available in the literature about the system. [9]

- *Structure Verification Test:* The structure verification test asks whether the equations of the model with the relationships is consistent with the knowledge of the real system relevant to the purpose. [5, 7, 9, 10]
- *Parameter Verification Test:* The parameter verification test is a two stage process, first identifying the model parameters that correspond to the real system and then numerically evaluating each parameter for accuracy. [5, 7, 9]
- *Extreme Conditions Test:* The extreme conditions test make sure that each equation is valid even when it input parameters receives extreme values and checks whether the model respond is plausible similar to the real system when subjected to extreme policies, shocks, and parameters [5, 7, 9].
- *Dimensional Consistency Test:* The dimensional consistency test ensures that the units of measure are consistent in all model/mathematical equations. [7, 9]

B. Tests of Model Behavior

To test the model's behavior, structure-oriented behavior tests (also known as indirect structure tests) are used. While direct structural tests (or simply structural tests) do not involve any simulation, these structure-oriented behavioral tests involve simulation to uncover structural flaws that might hide in the model. These structure-oriented behavior tests can be applied to both the whole as well as sub-models. Unlike direct structure tests, these indirect structure tests enable us to conduct quantitative evaluations on the model. [9]

- *Behavior Reproduction Test:* The behavior reproduction test evaluates the correctness of the model-generated behavior by comparing it to the real system's observed behavior [7].
- *Behavior Anomaly Test:* The behavior anomaly test verifies whether model exhibits an anomalous behaviors when assumptions of the model are changed or deleted [5, 7].

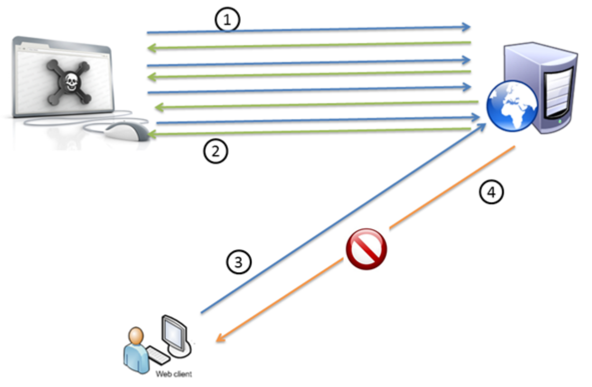
III. PROOF-OF-CONCEPT (PoC) MODEL

To convince ourselves of the feasibility of the overall research objectives, we constructed a Proof-of-Concept (PoC). The PoC simulated an HTTP Slow Read Attack on the Webserver-Clients Interface Sub-Model of the proposed IT Node.

A. Concept Design

1) The Problem

By design the HTTP protocol requires that when a client wants to transmit or receive data to/from a server, then the client must send its full data to the server before it can process it. If the data from the client is not complete or the data transfer rate is too slow, the server keeps that connection active and keeps its resources busy waiting for the client to complete its request. The Slow HTTP DoS attacks (known variously as Slowloris, Slow HTTP POST, and Slow HTTP GET) uses this fact about the HTTP protocol and try to keep as many server connections open as possible. If the server keeps too many resources busy, this creates a denial of service (Figure 1).



(1) HTTP GET requests from attacker
(2) Attacker Reads the HTTP GET responses from server as slow as possible to keep the connections active for a longer period of time.
(3) HTTP GET request from client
(4) Server busy/unavailable message

Fig.1. PoC Architecture and Data Flow

2) Scenarios

a) Normal Scenario [11]

Read a file of size 1 MB (1048576 bytes) from the HTTP Server.

1. Establish a connection to the server
2. Download the file (meaning, receive the response) through 1448-byte TCP packets, the maximum segment size that the underlying communication channel supports.

3. Assume the download speed is 14480 bytes/sec. The file will take 72.5 seconds (1048576/14480=72.5) to download resulting in the client receiving a TCP packet with FIN (Finish) flag, indicating no more data from sender/server.

b) Attack Scenario [11]

Read a file of size 1 MB (1048576 bytes) from the HTTP Server. Send legitimate HTTP requests and slowly read responses with the intent of keeping as many connections as possible in a active state.

1. Request a file which is larger than the server's send buffer.
2. Request a large amount of connections, say 1000 connections in total at a rate of 200 connections per second.
3. Let each client connection read the file at a rate of 500 bytes per second.
4. Steps 1 through 3 will keep many TCP connections active for a prolonged period of time that puts the HTTP server under DoS attack.

B. Model

Figure 2 shows the stack and flow diagram for the model and the SD model equations are shown in Figure 3.

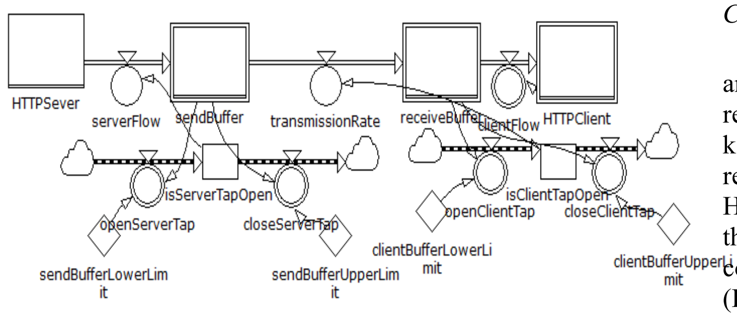


Fig.2. SD Model for HTTP Slow Read DoS Attack

1. $HTTPServer(t) = 1048576 - \int_0^t (serverFlow(t))dt$
2. $sendBuffer(t) = 0 + \int_0^t (serverFlow(t) - transmissionRate(t))dt$
3. $receiveBuffer(t) = 0 + \int_0^t (transmissionRate(t) - clientFlow(t))dt$
4. $HTTPClient(t) = 0 + \int_0^t (clientFlow(t))dt$
5. *IF (sendBuffer<=sendBufferLowerLimit) Then serverFlow.openServerTap=True*
6. *IF (sendBuffer>sendBufferUpperLimit) Then serverFlow.closeServerTap=True*
7. *IF(receiveBuffer<=receiveBufferLowerLimit)Then transmissionRate.openClientTap=True*
8. *IF(receiveBuffer>=receiveBufferUpperLimit)Then transmissionRate.closeClientTap=True*

Fig.3. Model Equations for HTTP Slow Read DoS Attack

IV. VALIDATION RESULTS FOR PoC MODEL

A. Structure Verification Test

The HTTP slow read DoS attack model equations (shown in Figure 3) were verified with the Webserver-Clients Interface Module (Figure 2) and Apache Webserver [12] default parameters available in the literature.

B. Parameter Verification Test

The values assigned to the parameters of the simulation were sourced from the existing knowledge and numerical data from Apache webserver data [13]. For illustration purposes, Table 1 lists some of the parameters and their values.

Parameters in the Model	**Assigned Valve**	**Assumed Valve**
Number of connections		10
Read rate from receive buffer (Normal Scenario)		1448 bytes/sec
Read rate from receive buffer (Attack Scenario)		500 bytes/sec
Wait Period (Amount of time the server will wait for certain events before failing a request)	60 sec	
Target Test Duration	240 sec	

Table.1. Model parameters and their values [11, 12]

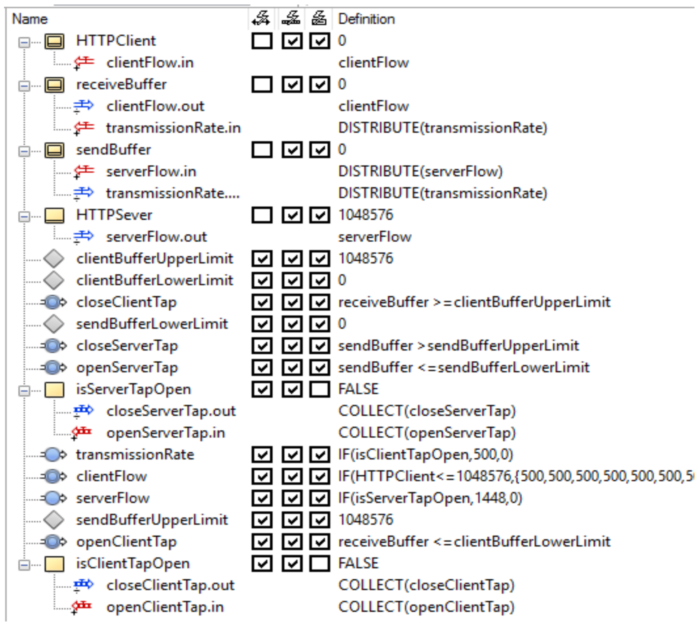
C. Extreme Conditions Test

This was verified using the attack scenario (See Figure 5 and Figure 6). Once the HTTP server received a request for a resource that did not fit into the server's socket send buffer, it kept the connection active until the client received the entire requested file/resource. Sending a large number of legitimate HTTP requests that were slowly acted on by the client caused the system to keep connections in an active state until the connections were available. This created a Denial-of-Service (DoS) when all the available connections were occupied by the attacker clients.

Attack Scenario (Read a file of size 1 MB (1048576 bytes) from the HTTP Server.)

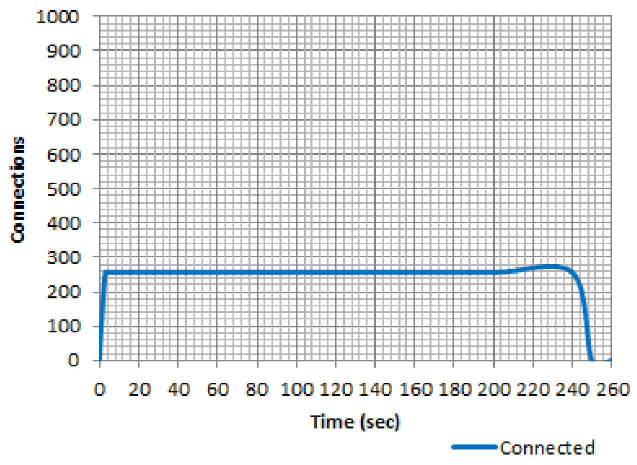
1. Establish a connection to the server.
2. Download the file (or receive the response) through several TCP packets sized 500 bytes, the default MinRate allowed by Apache server.
3. Set the download speed to 500 bytes/sec, the default MinRate allowed by Apache server.
4. As shown in Figure 6, the HTTP server is under DoS attack – the file is never downloaded by the clients and all the available connections are occupied by the attacker clients.

The attack scenario parameters are shown in Figure 4, the simulation results are shown in Figure 5, and the actual attack results on the testbed are shown in Figure 6 and Figure 7.

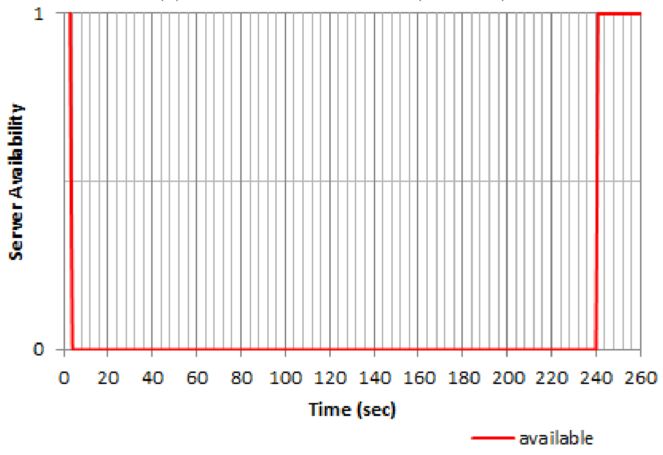


Name				Definition
HTTPClient	☐	☑	☑	0
clientFlow.in				clientFlow
receiveBuffer	☐	☑	☑	0
clientFlow.out				clientFlow
transmissionRate.in				DISTRIBUTE(transmissionRate)
sendBuffer	☐	☑	☑	0
serverFlow.in				DISTRIBUTE(serverFlow)
transmissionRate....				DISTRIBUTE(transmissionRate)
HTTPSever	☐	☑	☑	1048576
serverFlow.out				serverFlow
clientBufferUpperLimit	☑	☑	☑	1048576
clientBufferLowerLimit	☑	☑	☑	0
closeClientTap	☑	☑	☑	receiveBuffer >=clientBufferUpperLimit
sendBufferLowerLimit	☑	☑	☑	0
closeServerTap	☑	☑	☑	sendBuffer >sendBufferUpperLimit
openServerTap	☑	☑	☑	sendBuffer <=sendBufferLowerLimit
isServerTapOpen	☑	☑	☐	FALSE
closeServerTap.out				COLLECT(closeServerTap)
openServerTap.in				COLLECT(openServerTap)
transmissionRate	☑	☑	☑	IF(isClientTapOpen,500,0)
clientFlow	☑	☑	☑	IF(HTTPClient<=1048576,{500,500,500,500,500,500,5
serverFlow	☑	☑	☑	IF(isServerTapOpen,1448,0)
sendBufferUpperLimit	☑	☑	☑	1048576
openClientTap	☑	☑	☑	receiveBuffer <=clientBufferLowerLimit
isClientTapOpen	☑	☑	☐	FALSE
closeClientTap.out				COLLECT(closeClientTap)
openClientTap.in				COLLECT(openClientTap)

Fig.4. Attack Scenario Parameters Settings



(a) HTTP Server Status (Service)

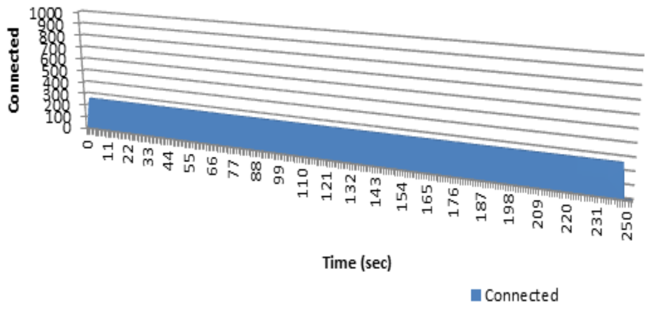


(b) HTTP Server Status (Availability)

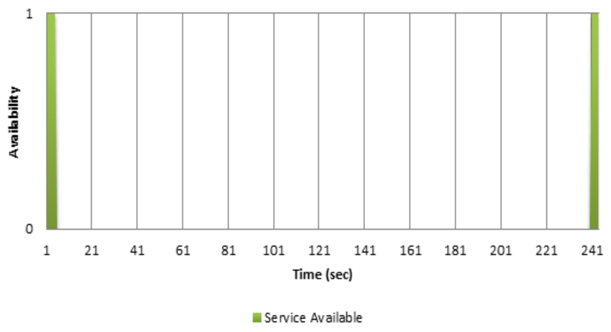
Fig.5. Attack Scenario Simulation Result

Figure 5 shows the SD simulation results of the slow read attack. The X-axis indicates the time (in seconds) taken to download the file and the Y-axis indicates the number of active connections maintained by the attacker at any particular time. As shown in Figure 5(a), attacker clients were able to hold their TCP connections by slowly reading the data from the server for a very long time. Until the entire file is read (1048576 bytes), the established connections were active. Figure 5(b) shows that there were no available connections for the new (legitimate) users during the time of attack – all the available connections were occupied by the attacker. This indicates that the server was under DoS attack.

For the PoC model validation we developed a cybersecurity testbed which consisted of a wireless LAN. The testbed consisted of Apache Webserver, a botnet consisted of three laptop computers with Kali Linux 64-bit Operating System running in a Virtual Machine environment installed on MacBook Pro with 2.7GHz Intel Core i5 processor and Mac OS Sierra version 10.12 Operating System, and the workstations consists of two Mac Book Pro laptop computers with Mac OS Sierra version 10.12 and 2.7GHz Intel Core i5 processor.



(a) HTTP Server Status (Service)



(b) HTTP Server Status (Availability)

Fig.6. Attack Scenario Actual Result on Testbed HTTP File Server

Figure 6 shows the HTTP (testbed) server status under an HTTP Slow Read attack. The X-axis indicates the time (in seconds) taken to download the file and the Y-axis indicates the number of active connections maintained by the attacker at any particular time. Figure 6(a) shows that the attacker clients keep all the 256 available connections, Apache Webserver default value, busy for the entire attack duration (240 seconds). Figure 6(b) shows the server availability. As the

Figure 6(b) indicates, the server was available only for the first 5 seconds and once the attacker clients occupied all the available connections the server was not available for the legitimate user until the attack was over (240 seconds).

D. Dimensional Consistency Test

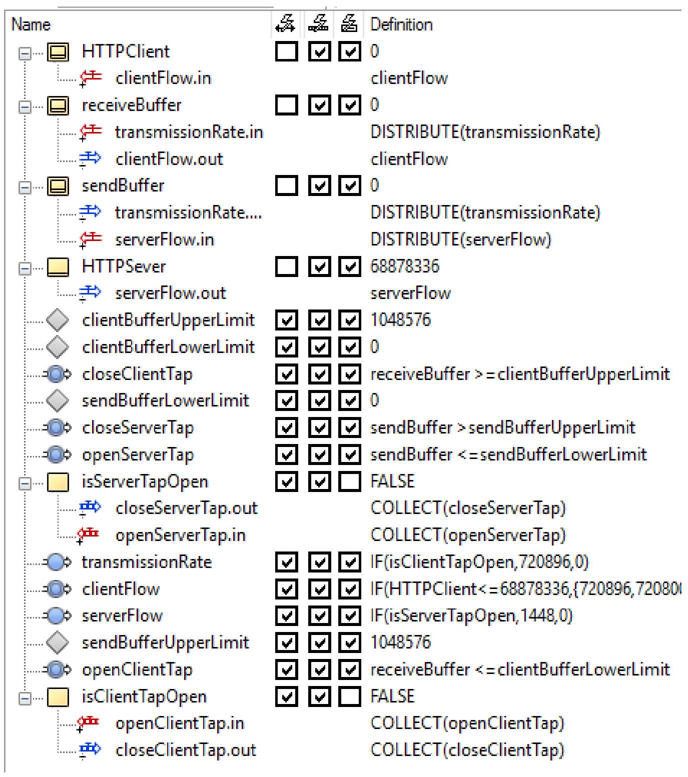
We ensured our units of measure were consistent with all mathematical equations. Specifically, times were in seconds and all data sizes were in bytes.

E. Behavior Reproduction Test

The simulation outputs for a normal scenario (Figure 7) verified the model-generated behavior (Figure 8) similar to observed behavior of the real system using real hardware (Figure 9)

Normal Scenario (Read a file of size 67,264 KB (68,878,336 bytes) from the HTTP Server.)

1. Establish a connection to the server.
2. Download the file (meaning, receive the response) through 1448-byte TCP packets, the maximum segment size that the underlying communication channel supports.
3. The download speed of the Internet connection is 5.5 Mbps = 720,896 bytes/sec, then after 96 seconds later, the client receive a TCP packet with FIN flag, indicating no more data from sender/server (that is, the file is downloaded).



Name				Definition
HTTPClient	☐	☑	☑	0
clientFlow.in				clientFlow
receiveBuffer	☐	☑	☑	0
transmissionRate.in				DISTRIBUTE(transmissionRate)
clientFlow.out				clientFlow
sendBuffer	☐	☑	☑	0
transmissionRate....				DISTRIBUTE(transmissionRate)
serverFlow.in				DISTRIBUTE(serverFlow)
HTTPSever	☐	☑	☑	68878336
serverFlow.out				serverFlow
clientBufferUpperLimit	☑	☑	☑	1048576
clientBufferLowerLimit	☑	☑	☑	0
closeClientTap	☑	☑	☑	receiveBuffer >=clientBufferUpperLimit
sendBufferLowerLimit	☑	☑	☑	0
closeServerTap	☑	☑	☑	sendBuffer >sendBufferUpperLimit
openServerTap	☑	☑	☑	sendBuffer <=sendBufferLowerLimit
isServerTapOpen	☑	☑	☐	FALSE
closeServerTap.out				COLLECT(closeServerTap)
openServerTap.in				COLLECT(openServerTap)
transmissionRate	☑	☑	☑	IF(isClientTapOpen,720896,0)
clientFlow	☑	☑	☑	IF(HTTPClient<=68878336,{720896,72080
serverFlow	☑	☑	☑	IF(isServerTapOpen,1448,0)
sendBufferUpperLimit	☑	☑	☑	1048576
openClientTap	☑	☑	☑	receiveBuffer <=clientBufferLowerLimit
isClientTapOpen	☑	☑	☐	FALSE
openClientTap.in				COLLECT(openClientTap)
closeClientTap.out				COLLECT(closeClientTap)

Fig.7. Normal Scenario Parameters Settings

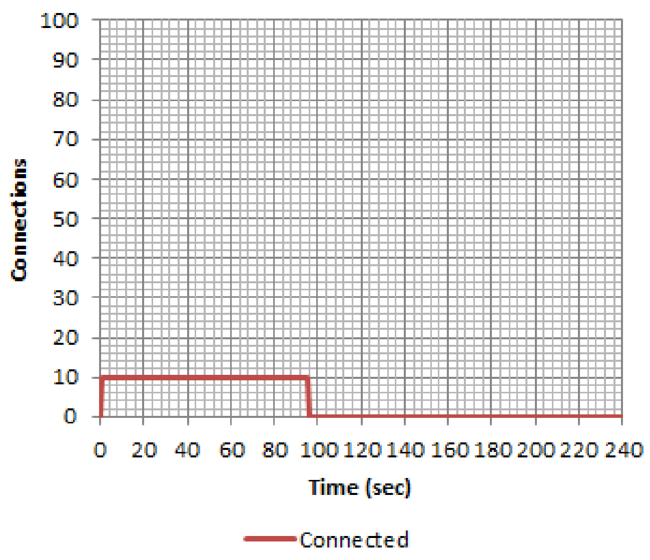


Fig.8. Normal Scenario Simulation Result

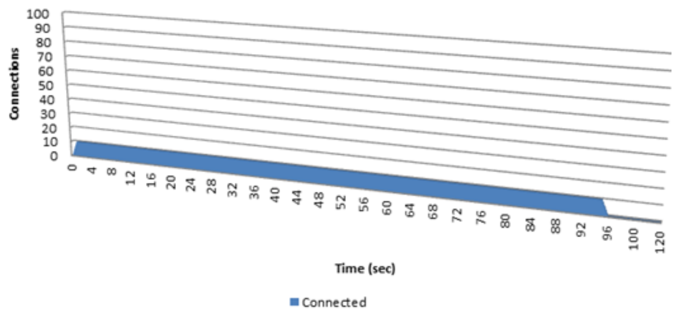


Fig.9. Normal Scenario Actual Result on Testbed

F. Behavior Anomaly Test

The model behaved like the real system under study and we did not discover any anomalous features of model behavior, which sharply conflict with behavior of the real system.

V. Summary and Conclusion

Though various formal quantitative and informal qualitative tools are used in model validation, the system dynamics model validation in general and in the field of cyber security in particular is done by getting a group of experts to agree on a causal loop diagram of the system. Usefulness in the user's eyes is the appropriate standard by which to evaluate these models. To validate a system dynamics cyber security model, we developed a cybersecurity testbed, conducted structural and behavior tests on the cybersecurity model, and finally compared the simulation results of the proof-of-concept cyber security model with the actual testbed results.

VI. References

[1] Forrester JW, "Industrial dynamics," Cambridge, MA: MIT Press, 1961.

[2] Albert Sweetser, "A comparison of system dynamics (SD) and discrete event simulation (DES)," 17th International Conference of the System Dynamics Society, 1999.

[3] Yaman Barlas, "System dynamics: systemic feedback modeling for policy analysis in knowledge for sustainable development—an insight into the encyclopedia of life support systems," Paris, France, Oxford, UK: UNESCO Publishing—Eolss Publishers, 2002.

[4] R.G. Coyle, "System dynamics modelling: a practical approach," London: Chapman & Hall, 1996.

[5] John D. Sterman, "Business Dynamics: Systems Thinking and Modeling for a Complex World," Irwin McGraw-Hill, McGraw-Hill Higher Education, 2000, ISBN 0-07-231135-5.

[6] Dimitrios Vlachos, Patroklos Georgiadis, and Eleftherios Iakovou, "A system dynamics model for dynamic capacity planning of remanufacturing in closed-loop supply chains," Computers & Operations Research 34 (2007) 367–394.

[7] J.W. Forrester and P.M. Senge, "Tests for building confidence in system dynamics models," TIMS Studies in the Management Sciences 1980, 14:209–28.

[8] Thiago Barros Brito, Edson Felipe Capovilla Trevisan, and Rui Carlos Botter, "A Conceptual Comparison between Discrete and Continuous Simulation to Motivate the Hybrid Simulation Methodology," Proceedings of the 2011 Winter Simulation Conference

[9] Yaman Barlas, "Formal aspects of model validity and validation in system dynamics," System Dynamics Review 2000, 12(3):183–210.

[10] Osman Balci, "Validation, verification, and testing techniques throughout the life cycle of a simulation study," Annals of Operations Research, Baltzer Science Publishers, Baarn/Kluwer Academic Publishers, December 1994, Volume 53, Issue 1, pp 121–173, DOI: https://doi.org/10.1007/BF02136828

[11] Sergey Shekyan, "Are you ready for slow reading?," Security Labs, January 5, 2012, https://blog.qualys.com/securitylabs/2012/01/05/slow-read

[12] Apache HTTP Server Version 2.4 Documentation, https://httpd.apache.org/docs/2.4/, Last Accessed: 2/17/2019, 8:47 PM

[13] Directive Quick Reference, https://httpd.apache.org/docs/2.4/mod/quickreference.html, Last Accessed: 2/17/2019, 8:48 PM