Author's copy of the paper published in Proceedings of the 27th IEEE International Conference on Industrial Technology (ICIT). Monterrey, Mexico, March 4 - 6, 2026.

SST Testbed: An Experimental Platform of Attacks and Defenses for Networked Embedded Systems

Carlos Beltran Quinonez[§]
Arizona State University
Tempe, AZ, USA
cjbeltr4@asu.edu

Dongha Kim[§]
Arizona State University
Tempe, AZ, USA
dongha@asu.edu

Hokeun Kim
Arizona State University
Tempe, AZ, USA
hokeun@asu.edu

***Abstract*—The growing prevalence of Internet of Things (IoT) devices has introduced significant security challenges, as their constant connectivity and limited resources make them vulnerable to network-level attacks. In this paper, we present the SST Testbed, an open-source, extensible platform for evaluating the security and resilience of embedded, resource-constrained systems. The testbed supports replay and denial-of-service (DoS) attacks, including distributed variants, through configurable input files and lightweight scripts that simplify setup and experimentation. By automatically generating topology and configuration files and enabling both simulation and deployment on embedded platforms, Secure Swarm Toolkit (SST) Testbed provides cost-effective, easy-to-use environments for simulating and deploying network attacks and defenses tailored to networked embedded systems. Our evaluation demonstrates the effectiveness of the SST Testbed under various deployment environments of network attacks from resource-constrained devices to more resourceful edge-computing devices.**

***Index Terms*—Embedded Systems, Network Security, Testbed**

I. Introduction

Networked embedded systems and the Internet of Things (IoT) have become deeply integrated into modern infrastructure and daily life as well as industry [1]. However, their immersive connectivity with limited security exposes them to a wide range of network threats [2], such as the Mirai botnet [3]. Despite the prevalence of network threats targeting embedded and IoT devices, there are few open-source security testbeds designed for these constrained environments. For example, to the best of our knowledge, there are no openly available testbeds for authentication and authorization of distributed systems, such as widely used Kerberos [4]. Consequently, developers lack accessible platforms to safely examine how embedded systems behave under real attack conditions.

To address this limitation, we propose **Secure Swarm Toolkit (SST) Testbed**, an open-source testbed tailored to networked embedded systems and the IoT. SST Testbed enables users to simulate attack scenarios, such as replay attacks or denial-of-service (DoS) attacks, within controlled, yet resource-constrained environments. By supporting customizable input files and lightweight deployment, SST Testbed provides both accessibility and practicality for developers seeking to evaluate device security prior to deployment.

[§]Both authors contributed equally to this research.

As research contributions, SST Testbed is characterized by the following capabilities and potential impacts on the development of secure networked embedded systems and IoT.

- SST Testbed supports a range of capabilities for common network attacks, such as replay, SYN-flood, as well as DoS, distributed DoS (DDoS) with excessive requests.
- Highly parameterized inputs of SST Testbed can specify diverse attack scenarios, supporting customized attack scenarios for domain or application-specific systems.
- SST Testbed is fully open-sourced, allowing researchers and developers to freely experiment with attacks and defenses.
- SST Testbed can be used as an educational tool for hands-on experience of real-world attacks and defenses for students studying embedded systems and the IoT.

Overall, by bridging the gap between theoretical research and practical experimentation, SST Testbed contributes to the advancement of more resilient and secure IoT ecosystems.

II. Related Work and Background

Prior work has proposed several testbeds to improve the security and resilience of IoT and IIoT (Industrial IoT) systems. Siboni *et al.* [5] developed a security testbed that evaluates IoT devices and communication channels such as Wi-Fi and ZigBee, and introduced resilience metrics to assess device behavior under attacks. Al-Hawawreh and Sitnikova [6] presented a testbed targeting industrial brownfield environments and intrusion detection experiments, but its complexity and hardware dependence limit lightweight experimentation. Gotham [7] provides a reproducible, container-based IoT environment designed for large-scale dataset generation and automated security experimentation, emphasizing reproducibility over hardware realism. In contrast, we focus on an open-source, customizable testbed tailored for embedded and resource-constrained IoT devices, with explicit support for replay and denial-of-service attack scenarios.

To do this, we build upon the Secure Swarm Toolkit (SST) [8] using the C API [9], which is open-source and can be readily adopted in diverse applications [10]–[13]. SST introduces a local authorization entity, *Auth* [14], that provides authentication and authorization services for connected devices, and works as a Key Distribution Center (KDC). By allowing devices to authenticate and obtain session keys

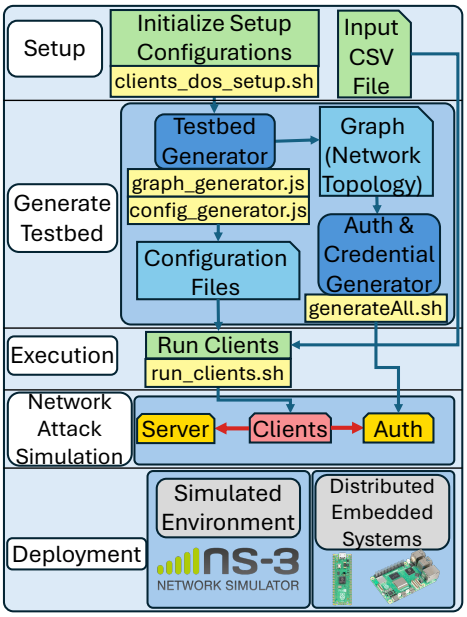


Fig. 1: Workflow of the testbed for simulating DoS attacks, from setup and testbed generation to execution on clients, server, and Auth within a simulated environment.

locally, SST enables more resilient communication [15]. This locally centralized, but globally decentralized design forms the basis for our testbed, which extends SST with configurable attack simulations.

We especially target two important classes of network attacks. Replay attacks [16], [17] occur when adversaries intercept legitimate messages and resend them to bypass authentication or repeat actions unexpectedly. Denial-of-Service (DoS) attacks [18], [19] aim to exhaust critical resources of a server or authentication service by flooding them with excessive requests or connections, with distributed variants (DDoS) [20], [21], amplifying the disruption. Our testbed enables safe, repeatable experimentation with both attack types in the context of embedded and IoT systems.

III. Proposed Approach: SST Testbed

This section presents the overall architecture and workflow of the SST Testbed. We begin by describing the system workflow, which is organized into five stages. We then detail the supported attack models and their usage within the framework. Finally, we highlight the novel contributions of our work.

A. Description of the Tool

The SST Testbed workflow is organized into five stages, as shown in Fig. 1: ***Setup***, ***Generate Testbed***, ***Execution***, ***Network Attack Simulation***, and ***Deployment***. Each stage plays a distinct role in preparing, configuring, executing, and evaluating security experiments. These stages provide a modular and repeatable process that allows researchers and developers to design and launch customized attack scenarios against embedded and IoT systems.

1) ***Setup****:* The workflow begins with the setup stage, where the user specifies the desired number of clients and the type of attack to be performed. This is done by editing the input CSV file, which defines parameters such as attack category (e.g., replay, DoS, or DDoS) and attack intensity. Once the CSV is prepared, the `clients_dos_setup.sh` script is executed to initialize the environment, creating the necessary files and directories for later stages. Through this simple configuration process, users can easily tailor the experiment without modifying the underlying code, ensuring both flexibility and reproducibility.

2) ***Generate Testbed****:* After completing the setup and executing `clients_dos_setup.sh`, the Testbed Generator is invoked to construct the necessary artifacts for the experiment. This process internally runs two scripts: `graph_generator.js` and `config_generator.js`.

The `graph_generator.js` script produces the *Graph file*, which defines the network topology of the testbed. In this graph, each entity, including clients and the server, is registered with the Auth/KDC. Using this information, the Auth service can generate the credentials for all registered entities.

The `config_generator.js` script creates a *Configuration file* for each entity in the testbed. Each configuration file contains essential details such as credential paths, the Auth server's IP address and port, the Auth ID, and the entity's identifier (e.g., client name or server name). These configuration files ensure that each client and the server can authenticate with the Auth/KDC and establish secure communication channels.

3) ***Execution****:* The execution stage runs the generated testbed. Using `run_clients.sh`, the system launches the specified number of clients and instantiates the server, each component loading its corresponding configuration file. During this process, the server prepares to accept incoming connections, while clients initiate communication and, if necessary, request session keys from the Auth/KDC. This stage effectively transitions the system from a static configuration into a dynamic environment capable of simulating live traffic and attack behavior.

4) ***Network Attack Simulation****:* At this stage, the testbed carries out the configured attack scenarios as defined in the input CSV. Clients may attempt actions such as replaying messages, issuing excessive session key requests, repeatedly connecting to the server, or sending large volumes of messages. These behaviors can be scaled from a single client to many clients, thereby representing both DoS and DDoS attacks. This flexible design allows researchers to simulate a wide range of attack patterns and evaluate their impact on system stability and availability, offering a safe yet realistic means of stress-testing embedded and IoT systems.

5) ***Deployment****:* The final stage is deployment, where the generated artifacts and attack scenarios can be executed in different environments. For simulation purposes, the SST Testbed can be integrated with ns-3 [22], allowing users to observe attack behaviors in a virtualized network setting that approximates real-world conditions. Alternatively, the same

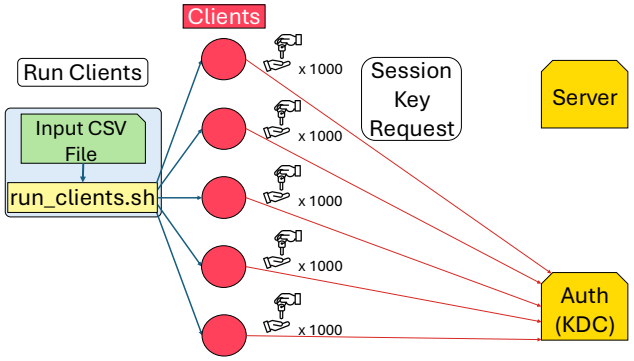


Fig. 2: Clients launching excessive session key requests toward the Auth/KDC, exhausting authentication resources.

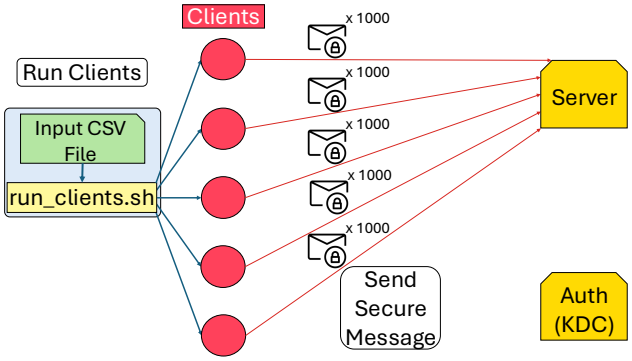


Fig. 3: Clients overwhelming the server by sending a large volume of secure messages, leading to message-handling exhaustion.

configuration can be deployed to actual distributed embedded devices, such as Raspberry Pi boards, without modifying the attack logic. This decoupling between configuration (graph and config files) and the simulation is a key feature of the testbed. We ensure that experiments are reproducible and adaptable to resource-constrained embedded platforms. As a result, the deployment stage bridges the gap between theoretical simulation and practical validation, making the testbed valuable for both academic research and applied security evaluation.

B. Supported Attacks in Detail

Our testbed currently supports two categories of network threats: replay attacks and denial-of-service attacks.

1) ***Replay Attack****:* The replay attack is supported by the ability given to the user to change the sequence number. The sequence number attaches a count to each message sent by the client to the server to identify each message. Attempting to manipulate this number would be a replay attack, since the user could try to use the identity of a previous message to confuse or manipulate the server.

2) ***Denial of Service (DoS) Attack****:* DoS attacks in our testbed are designed to exhaust system resources and disrupt services. We model attacks against both the authentication service (KDC/Auth) through excessive session key requests and the application server through repeated connection attempts, message flooding, or SYN packet flooding. Furthermore, the testbed supports scaling to multiple malicious clients, enabling simulations of distributed denial-of-service (DDoS) scenarios.

- **DoS & DDoS attack via Session Key Request** The denial-of-service attack is implemented by allowing the user to modify the number of times the client will request a session key from Auth. If the client makes an excessive amount of requests, it will exhaust Auth's resources and force Auth to be unable to respond to other legitimate clients requesting a session key. As shown in Fig. 2, the DDoS attack can also be performed by creating an enormous number of clients, each of which requests a session key.
- **DoS & DDoS attack via Sending Messages** When a session is already established between a server and a client, messages between them can be used maliciously to overload the server with messages. If a client sends the server a massive amount of messages, the server's resources will be exhausted, rendering the server unable to receive incoming messages from other clients. Moreover, as illustrated in Fig. 3, if a large number of malicious clients attempt to send messages at the same time, the server will be overwhelmed and will not be able to receive and respond to each message in a timely manner. Benign clients will be left to wait a long time before receiving a response from the server.

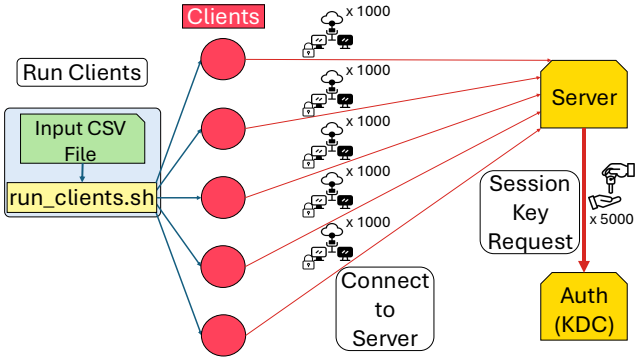


Fig. 4: Clients repeatedly connecting to the server, which indirectly triggers excessive session key requests to the Auth/KDC.

- **DoS & DDoS attack via Server Connect** In this attack, the client repeatedly attempts to connect to the server a user-defined number of times. This exhausts the server's available ports with illegitimate connections, preventing legitimate clients from accessing the service. Moreover, each connection request also triggers the server to obtain a session key from the Auth/KDC, meaning that an attack directed at the server indirectly generates excessive load on the authentication service as well.
  Similarly to other attacks, as depicted in Fig. 4, when many malicious clients are launched simultaneously, the server is more flooded with connection requests, occupying all available ports. This DDoS attack also amplifies the indirect load on the Auth, further disrupting both the server and the Auth. As a result, legitimate clients are not able to request session keys, and not able to connect to the server entity.

- **DoS & DDoS attack via SYN flooding** Unlike the attacks above, which assume attackers are authenticated, a SYN flooding attack can be executed by any unauthenticated host that knows the victim's IP address. In a SYN flood [23], [24] the attacker sends a large number of TCP SYN packets to the target but never completes the final ACK, leaving the server with many half-open connections that occupy connection table entries and backlog slots. An attacker-controlled botnet can amplify this into a DDoS by distributing the SYN streams across many sources. The result is that the server's connection queue and resources such as memory and CPU are exhausted, preventing legitimate clients from establishing new connections.

C. Usage

To use the testbed, the user must first fill out the CSV with the required information to launch each attack. Once done, the user must launch the client using the CSV file they created.

```
DELAY_TIME,MESSAGE,ATTACK_TYPE,ATTACK_PARAMETER
-----------------------------------------------
1000,Hello,DOSK,12345
```

Above is an example of what a general input file will look like. The example shows the format the file should be in and an example of someone launching a Denial-of-Service attack against the Auth.

In the input file, each field has a significant meaning. The first field DELAY_TIME refers to how long the client will wait before sending the message. The second field, MESSAGE, is the aforementioned message. Once the client has waited for the delay time to end, it will send the text in this field to the server for it to print out. The third field is ATTACK_TYPE, where the user will specify what kind of attack they want to test. There is currently a replay attack method and three different denial-of-service attack methods implemented for the user to test against. The fourth field is ATTACK_PARAMETER. All attacks use a parameter defined by the user to specify how to the attack should be conducted. For the replay attack, the user must choose how the sequence number will be changed. For the denial-of-service attacks, the user specifies the number of times the corresponding function is called to attempt to overwhelm the server or the Auth.

If the user decides to do a DoS attack using many clients, the user must run the `client_dos_setup.sh` script to create a new graph file, which is the network topology, and new configuration files so that Auth recognizes all the different clients trying to connect. The script requires a command-line parameter: the number of clients the user would like to have. This needs to be specified by the user so that Auth is aware of all the clients and the user be able to customize the intensity of the attack. When the script starts, the user will have to enter a password upon request from the command-line to regenerate the Auth and to launch the Auth server.

Once that is finished, the user should run `run_clients.sh` to start the testing. This script requires two parameters: the first is the number of clients that should be launched, and the second is the input file to be used.

TABLE I: Device configurations for each entity (Auth, Server, Benign client, or Attacker) used in the experiments.

Configurations	Config #1	Config #2	Config #3
Auth	Workstation 1	Workstation 1	Workstation 1
Server	Workstation 1	RPI 4B #1	RPI 4B #1
Benign client	Workstation 2	RPI 4B #2	RPI 4B #2
Attacker	MacBook Pro	RPI 4B #3	RPI 4B #3, MacBook Pro

Setting these as user-defined parameters gives the user the power to choose the intensity of the attack by selecting the number of clients, and also conveniently allows the user to switch attack types easily by simply choosing a different input file upon running the script.

D. Discussion and Contributions

The SST Testbed introduces several contributions that strengthen both research and education in embedded and IoT security. First, it supports a broad range of common network attacks, including replay, SYN-flood, and denial-of-service (DoS/DDoS) using excessive requests. These capabilities allow researchers to evaluate the resilience of networked embedded systems under realistic attack conditions.

Second, the platform employs highly parameterized input files that enable flexible configuration of attack parameters. This design allows users to construct customized scenarios for testing diverse attack behaviors and mitigation strategies.

Third, SST Testbed is fully open-source[1], ensuring accessibility, transparency, and community-driven extension. Researchers and developers can freely modify the framework, experiment with new attack or defense modules, and contribute to its continuous evolution.

Finally, SST Testbed serves as a hands-on educational tool, providing students and practitioners with direct exposure to real-world IoT attacks and defenses. Through this role, it helps lower the barrier to cybersecurity experimentation and training.

Overall, by bridging the gap between theoretical research and practical experimentation, SST Testbed contributes to the advancement of secure and resilient IoT ecosystems.

IV. Evaluation

We evaluate the effectiveness and usability of the SST Testbed through a case study of attacking networked embedded systems. Our goals are twofold: (1) show that the testbed can deploy attacks on resource-constrained embedded devices, (2) demonstrate how the attack intensity (number of malicious clients) affects benign client performance in terms of latency and throughput.

A. Experimental Setup

We create a experimental environment, simulating real world networked embedded systems. Fig. 5, and TABLE I shows the topology and the hardware used in our experiments. The Auth service runs on a workstation (Intel i9-13900 CPU, 128 GB RAM).

[1]Available at our repository here: https://github.com/iotauth/iotauth

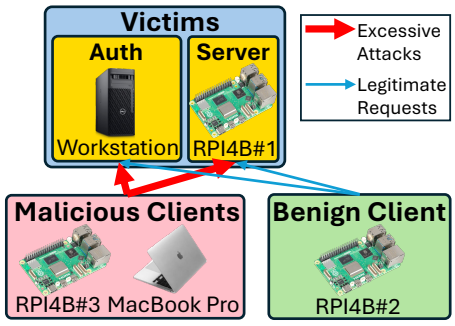
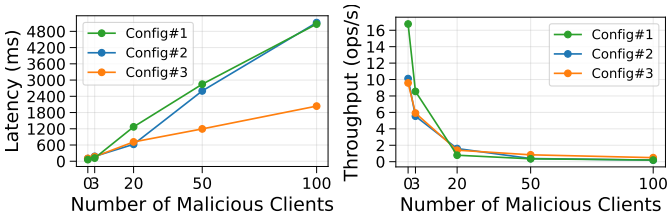


Fig. 5: Illustration of the Config #3 deployment setting.



(a) Latency degradation (b) Throughput degradation

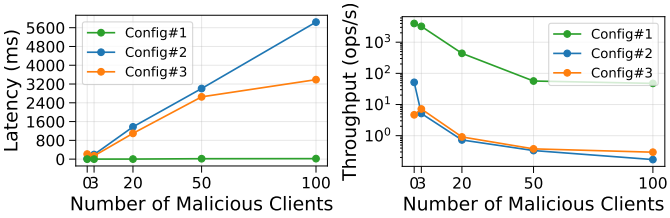
Fig. 6: DDoS Key Attack Latency and Throughput

We evaluate three deployment settings. Config #1 is a workstation based setting, which the server entity co-locates with the Auth on the same workstation. Attacks are launched from a MacBook Pro M3 Max (36 GB RAM) and the benign client runs on a second workstation. Config #2 is an embedded setting, the server entity and benign client run on Raspberry Pi (RPI) 4B devices (4 GB), and attackers execute from a separate RPI 4B (8 GB). Config #3 is a mixed setting combining the two above: the server runs on an RPI 4B, while attackers include both an RPI 4B and the MacBook Pro.

For each experiment we vary the number of malicious clients over 0, 3, 20, 50, and 100, as zero representing no attacks. Malicious clients execute the attack workloads provided by the SST Testbed against either the Auth, the server entity, or both. Simultaneously, a single benign client performs the same operation (e.g., session key request, message exchange, or connection attempt) and we measure its performance. Each reported data point is the average of 100 trials: latency is the average round-trip time for a successful operation (milliseconds), and throughput is the rate of completed operations per second. When an operation does not complete, it is recorded as a failure. All devices were executed in the same network for safe experiments.

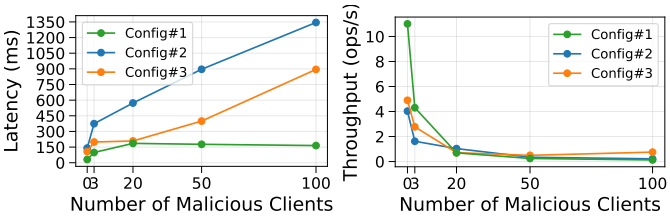
B. Experimental Results

*1) **DDoS Key Attack**:* We first evaluate an attack that targets the authentication service (Auth) by each malicious client executing a large number of session-key requests. The benign client simultaneously requests session keys. Fig. 6 shows that as the number of malicious clients increases, the benign client's average latency grows linearly up to 85x, and throughput drops up to 98.9%. The attack was not able to fail the benign client's session key request and did not cause



(a) Latency degradation (b) Throughput (log scale)

Fig. 7: DDoS Message Attack Latency and Throughput



(a) Latency degradation (b) Throughput degradation

Fig. 8: DDoS Connect Attack Latency and Throughput

Auth to crash. However, it drove Auth to its maximum key-provisioning capacity, after which the service was unable to issue additional session keys.

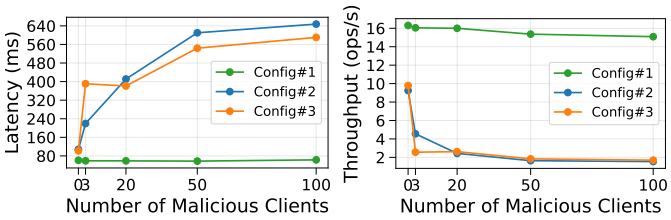
*2) **DDoS Message Attack**:* Next, we evaluate the message-flooding attack that targets the server entity, assuming clients have successfully established sessions with the server. In this scenario, multiple malicious clients send excessive messages to the server, overwhelming its message-handling loop and network buffers. A benign client, already connected to the same server, attempts to exchange messages concurrently, and we observe the impact on response latency and throughput.

Unlike the key request attack, which involves a three-way handshake with the Auth service, the message flooding attack stresses the *server's* message processing path. We define a *failure* as a case where the benign client does not receive a server reply within the timeout window. For each configuration, we measure the latency and throughput of the message exchange and record any failures.

As illustrated in Fig. 7, increasing the number of malicious clients increases the latency up to 301x and decreases the throughput up to 99.99%. However, no complete message failures were observed under the tested loads, indicating that the server was properly operating but with degraded responsiveness.

*3) **DDoS Connect Attack**:* We evaluate the connection-flooding attack, which targets the server entity and indirectly stresses the Auth. In this attack, malicious clients repeatedly establish sessions with the server, consuming the server's CPU time and memory resources for session management. Each connection attempt also triggers a session key exchange with the Auth, amplifying the overall load on the server and Auth.

As shown in Fig. 8, latency increases up to 9.6x and throughput decreases up to 98.8% as the number of malicious clients grows. Unlike previous attacks, as the number of attackers increased from 0 to 100, the benign client's session



(a) Latency degradation (b) Throughput degradation

Fig. 9: DDoS Syn Flood Latency and Throughput

failures rose sharply, 0, 16, 68, 88, and 90 for 0, 3, 20, 50, and 100 attackers, respectively. These failures occurred when the server's active session pool were filled of malicious clients' connections, preventing new legitimate sessions from being established.

*4) **DDoS SYN Flood Attack**:* Unlike the other attacks, which require authenticated access, a SYN flood can be launched by any host that knows the target IP address. In this experiment we targeted the Auth host with flooding TCP SYN packets while a benign client attempted a session key request. The attack did not crash the Auth or cause any key request failures, but latency increased up to 6.0x and throughput decreased up to 82.7% as shown in Fig. 9.

C. Summary of Evaluation and Discussion

Our evaluation demonstrates that the SST Testbed can effectively launch a variety of attack scenarios, from simple replay attacks to advanced DDoS attacks, on networked embedded environments. Our testbed enables the execution of large numbers of malicious clients, even on resource-constrained devices such as Raspberry Pi.

For all experiments, running on different configurations showed simliar tendencies, increasing latency, and decreasing the throughput. Furthermore, our evaluation results show that, with our testbed, we can induce drastic performance degradation on victims, up to 301x increased latency, even with a small number of devices in a cost-effective manner.

V. Conclusion

In this paper, we present the SST Testbed, an open-source and extensible platform for simulating network attacks on embedded and IoT systems. The testbed enables controlled experimentation with replay and denial-of-service scenarios, providing researchers and developers with a practical environment to evaluate system resilience prior to deployment. As future work, we plan to extend the testbed by supporting robust defenses to be used against the attacks.

Acknowledgment

This work was partially supported by grants from the National Science Foundation (NSF-SFS-1663651 and NSF-POSE-2449200).

References

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] W. H. Hassan and et al., "Current research on Internet of Things security: A survey," *Computer Networks*, vol. 148, pp. 283–294, 2019.

[3] M. Antonakakis and et al., "Understanding the Mirai botnet," in *Proc. 26th USENIX Security Symposium*, 2017, pp. 1093–1110.

[4] B. C. Neuman and T. Ts'o, "Kerberos: An authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, 1994.

[5] S. Siboni and et al., "Security testbed for Internet-of-Things devices," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 23–44, 2018.

[6] M. Al-Hawawreh and E. Sitnikova, "Developing a security testbed for Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5558–5573, 2020.

[7] X. Sáez-de Cámara, J. L. Flores, C. Arellano, A. Urbieta, and U. Zurutuza, "Gotham testbed: A reproducible IoT testbed for security experiments and dataset generation," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 1, pp. 186–203, 2023.

[8] H. Kim, E. Kang, E. A. Lee, and D. Broman, "A toolkit for construction of authorization service infrastructure for the Internet of Things," in *Proc. 2nd Int. Conf. Internet-of-Things Design and Implementation*, 2017, pp. 147–158.

[9] D. Kim, Y. Jo, T. Kim, and H. Kim, "SST v1.0.0 with C API: Pluggable security solution for the Internet of Things," *SoftwareX*, vol. 22, p. Art. no. 101390, 2023.

[10] D. Kim, C. Lee, and H. Kim, "A case study of API design for interoperability and security of the internet of things," in *Proc. Int. Conf. Security and Privacy in Cyber-Physical Systems and Smart Vehicles*, 2024, pp. 135–157.

[11] V. Thakkar, D. Kim, Y. Lai, H. Kim, and Z. Cao, "SHIELD: Encrypting persistent data of LSM-KVS from monolithic to disaggregated storage," *Proceedings of the ACM on Management of Data*, vol. 3, no. 3, pp. 1–28, 2025.

[12] Y. Jo, Y. Cho, and H. Kim, "Secure and lightweight access control for highly decentralized and distributed file systems," in *Proc. 1st Int. Workshop on Middleware for the Computing Continuum*, 2023, pp. 1–6.

[13] D. Kim and H. Kim, "Securing edge-based real-time IoT systems," in *Proc. 21st ACM Conf. Embedded Networked Sensor Systems*, 2023, pp. 544–545.

[14] H. Kim, A. Wasicek, B. Mehne, and E. A. Lee, "A secure network architecture for the Internet of Things based on local authorization entities," in *Proc. IEEE 4th Int. Conf. Future Internet of Things and Cloud (FiCloud)*, 2016, pp. 114–122.

[15] H. Kim, E. Kang, D. Broman, and E. A. Lee, "Resilient authentication and authorization for the Internet of Things using edge computing," *ACM Transactions on Internet of Things*, vol. 1, no. 1, pp. 1–27, 2020.

[16] Y. Feng, W. Wang, Y. Weng, and H. Zhang, "A replay-attack resistant authentication scheme for the Internet of Things," in *Proc. IEEE Int. Conf. Computational Science and Engineering (CSE) and IEEE Int. Conf. Embedded and Ubiquitous Computing (EUC)*, vol. 1, 2017, pp. 541–547.

[17] Y. Mo and B. Sinopoli, "Secure control against replay attacks," in *Proc. 47th Annual Allerton Conf. Communication, Control, and Computing*, 2009, pp. 911–918.

[18] E. Džaferović, A. Sokol, A. Almisreb, and S. Mohd Norzeli, "DoS and DDoS vulnerability of IoT: A review," *Sustainable Engineering and Innovation*, vol. 1, no. 1, pp. 43–48, 2019.

[19] K. Pelechrinis, M. Iliofotou, and S. V. Krishnamurthy, "Denial of service attacks in wireless networks: The case of jammers," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 2, pp. 245–257, 2010.

[20] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[21] T. Mahjabin, Y. Xiao, G. Sun, and W. Jiang, "A survey of distributed denial-of-service attack, prevention, and mitigation techniques," *International Journal of Distributed Sensor Networks*, vol. 13, no. 12, p. Art. no. 1550147717741463, 2017.

[22] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 15–34.

[23] M. Bogdanoski, T. Suminoski, and A. Risteski, "Analysis of the SYN flood DoS attack," *International Journal of Computer Network and Information Security*, vol. 5, no. 8, pp. 1–11, 2013.

[24] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proc. IEEE INFOCOM*, vol. 3, 2002, pp. 1530–1539.