Journal of Cybersecurity Education, Research and Practice

Volume 2024 | Number 1 Article 12

12-11-2023

Turnstile File Transfer: A Unidirectional System for Medium-Security Isolated Clusters

Mark Monnin
mark.monnin@maine.edu

Lori L. Sussman
University of Southern Maine, lori.sussman@maine.edu

Recommended Citation

Monnin, Mark and Sussman, Lori L. (2023) "Turnstile File Transfer: A Unidirectional System for Medium-Security Isolated Clusters," *Journal of Cybersecurity Education, Research and Practice*: Vol. 2024: No. 1, Article 12.
DOI: 10.32727/8.2023.36
Available at: https://digitalcommons.kennesaw.edu/jcerp/vol2024/iss1/12

Turnstile File Transfer: A Unidirectional System for Medium-Security Isolated Clusters

Abstract

Data transfer between isolated clusters is imperative for cybersecurity education, research, and testing. Such techniques facilitate hands-on cybersecurity learning in isolated clusters, allow cybersecurity students to practice with various hacking tools, and develop professional cybersecurity technical skills. Educators often use these remote learning environments for research as well. Researchers and students use these isolated environments to test sophisticated hardware, software, and procedures using full-fledged operating systems, networks, and applications. Virus and malware researchers may wish to release suspected malicious software in a controlled environment to observe their behavior better or gain the information needed to assist their reverse engineering processes. The isolation prevents harm to networked systems. However, there are times when the data is required to move in such quantities or speeds that it makes downloading onto an intermediate device untenable. This study proposes a novel turnstile model, a mechanism for one-way file transfer from one enterprise system to another without allowing data leakage. This system protects data integrity and security by connecting the isolated environment to the external network via a locked-down interconnection. Using medium-security isolated clusters, the researchers successfully developed a unidirectional file transfer system that acts as a one-way “turnstile” for secure file transfer between systems not connected to the internet or other external networks. The Turnstile system (source code available at github.com/monnin/turnstile) provides unidirectional file transfer between two computer systems. The solution enabled data to be transferred from a source system to a destination system without allowing any data to be transferred back in the opposite direction. The researchers found an automated process of transferring external files to isolated clusters optimized the transfer speed of external files to isolated clusters using Linux distributions and commands.

Keywords

file transfer, asynchronous file transfer, transport processes, file servers, isolated clusters, file systems

Turnstile File Transfer: A Unidirectional System for Medium-Security Isolated Clusters

Mark Monnin
Department of Technology
University of Southern Maine
Portland, Maine
https://orcid.org/0009-0003-7365-8882

Lori L. Sussman
Department of Technology
University of Southern Maine
Portland, Maine
https://orcid.org/0000-0003-3667-0340

Abstract
Data transfer between isolated clusters is imperative for cybersecurity education, research, and testing. Such techniques facilitate hands-on cybersecurity learning in isolated clusters, allow cybersecurity students to practice with various hacking tools, and develop professional cybersecurity technical skills. Educators often use these remote learning environments for research as well. Researchers and students use these isolated environments to test sophisticated hardware, software, and procedures using full-fledged operating systems, networks, and applications. Virus and malware researchers may wish to release suspected malicious software in a controlled environment to observe their behavior better or gain the information needed to assist their reverse engineering processes. The isolation prevents harm to networked systems. However, there are times when the data is required to move in such quantities or speeds that it makes downloading onto an intermediate device untenable. This study proposes a novel turnstile model, a mechanism for one-way file transfer from one enterprise system to another without allowing data leakage. This system protects data integrity and security by connecting the isolated environment to the external network via a locked-down interconnection. Using medium-security isolated clusters, the researchers successfully developed a unidirectional file transfer system that acts as a one-way "turnstile" for secure file transfer between systems not connected to the internet or other external networks. The Turnstile system (source code available at github.com/monnin/turnstile) provides unidirectional file transfer between two computer systems. The solution enabled data to be transferred from a source system to a destination system without allowing any data to be transferred back in the opposite direction. The researchers found an automated process of transferring external files to isolated clusters optimized the transfer speed of external files to isolated clusters using Linux distributions and commands.

Keywords ***Secure file transfer, secure asynchronous file transfer, secure transport processes, secure file servers, isolated clusters, securing file systems***

I. Introduction

Cybersecurity educators and researchers urgently require advanced isolated clusters for essential lab work and critical investigations, where students can safely develop vital technical skills using various hacking tools. These isolated environments are crucial for probing high-risk malware and testing potentially dangerous hardware and software, simulating realistic network systems without external risks. However, the pressing challenge arises in transferring large volumes of data or at high speeds, where traditional methods falter, underscoring an immediate need for innovative solutions to maintain the continuity and effectiveness of these vital cybersecurity operations.

Such is the situation when performing cluster maintenance. Practitioners create these environments using hypervisor/virtual machine solutions such as VMware vSphere, Linux KVM, and Xen [1]. These clusters contain multiple virtual systems within a virtualization platform and may have dozens to thousands of interconnected virtual systems. By design, these isolated clusters do not have direct access to any outside network, including the commodity Internet. Additionally, networks within these systems only communicate with other networks within the environment and do not extend outside the domain. This isolation aims to prevent any activity within the environment from affecting systems outside the cluster. For example, malware released within the isolated cluster cannot infect computer systems outside.

Transferring external files into these groups is necessary for the environment's users and the overall cluster's administration. While some of these isolated clusters have internal, complex simulations of the Internet, by design, this does not provide a method to move data from Internet sites into the isolated environment. This study proposes a novel turnstile model, a mechanism for one-way file transfer from one enterprise system to another. This system protects data integrity and security by connecting the isolated environment to the external network via a locked-down interconnection. This research shows a more reliable file transfer approach that eliminates system states with mixed versions.

This research focused on solutions for medium-security clusters. High-security clusters like air-gapped systems would only permit users to transfer files with a highly controlled, human-centric, manual process. At the other end of the spectrum are low-security systems. A simple firewall with restrictive traffic rules in these systems might provide a sufficient balance between protection and access. However, many medium-security systems exist where any external

network-oriented connection is not warranted, but users desire an automated approach to file transfer [2].

II. Best Practices for Transferring Large External Files

Transferring data and software between your workstation and a remote computer is standard for scientific workflows. Sometimes, these files are enormous but still should be transmitted securely. Data transfers between cloud storage and computing facilities are becoming increasingly common. There are several utilities available to help accomplish these essential tasks.

According to reference [3], transferring external files into isolated clusters generally involves:

1) Identify the location of the external files that you want to transfer. Destination examples include files stored on a local machine, a network share, a cloud storage service, or another location.

2) Determine the method of transfer. Depending on the source and destination of the files, you may need to use different transfer methods. For example, if transferring files from a local machine to an isolated cluster, you could use a tool like *rsync*, a command-line tool for copying files and directories between local and remote systems, or Secure Copy Protocol (SCP). On the other hand, if transferring files from a cloud storage service, you might use a cloud-based transfer tool or Application Programming Interface (API).

3) Set up authentication and access controls. One must authenticate with the cluster to transfer files into an isolated cluster and provide the appropriate access controls. This control could involve creating and managing user accounts, setting up SSH keys, or using other forms of authentication [4].

4) Initiate the transfer. Once you have identified the location of the files, determined the transfer method, and set up the appropriate authentication and access controls, you can initiate the transfer. Depending on the size and complexity of the files, this process could take a few minutes or several hours.

5) Verify the transfer. After the user transfers files into the isolated cluster, one should verify that they moved the files correctly and these files are accessible within the cluster by running a checksum on the files or testing their functionality within the cluster environment.

The choice of data transfer utility depends on how much data the user wants to transfer, preferences for transfer, and priorities. Reference [5] defined transfer speed, ease of use, security, and validation priorities. This project offers a new transfer process for isolated clusters where large amounts of data must be transferred from a sandbox or dirty cluster into a production or clean cluster environment using the reference [5] framework for measures of success.

Transferring external files into isolated clusters can be a complex process that requires careful planning and attention to security and access controls. Therefore, it is essential to follow best practices for secure file transfer. That way, the administrator can ensure that the users carry out the process safely and efficiently.

III. Current Techniques for Transferring External Files

Several techniques exist to transfer external files that can be adapted to work with semi-isolated clusters, depending on your specific needs and available tools. Here are some common methods:

1) SCP is a secure file transfer protocol that uses the SSH protocol for authentication and encryption. It allows you to copy files from a remote server to your isolated cluster or vice versa. Unix-based operating system developers build SCP into varieties available via third-party clients for Windows systems. It is one of the most straightforward transfer protocols suitable for smaller transfer sizes (<= 100 GB). However, its slow speed for larger amounts of data will lead to long transfer times and increase the likelihood of failures. In addition, SCP uses Secure Shell (SSH) to transmit encrypted versions of your credentials and data, increasing security [4].

2) Secure File Transfer Protocol (SFTP) is like SCP but provides a more comprehensive set of file transfer capabilities. Developers also build SFTP into most Unix-based systems and are available as third-party clients for Windows systems. It is another suitable protocol for transferring smaller amounts of data (<= 100 GB). It provides the interactive functionality of FTP, but like SCP, it uses SSH to communicate with remote systems [4]. This secure remote capability gives the user powerful functionality and securely transmits your encoded credentials and data. However, the challenge of using SFTP is that it transfers data slower than SCP and FTP.

3) File Transfer Protocol Secure (FTPS) is an extension of the FTP protocol that uses Secure Sockets Layer (SSL)/ Transport Layer Security (TLS) encryption for secure file transfer [6]. It lets you transfer files between your isolated cluster and an external server over an encrypted channel. FTP is a venerable protocol for transferring files. An FTP transfer requires establishing a connection between a client on the local machine and an FTP server on a remote machine. Once the user establishes the connection, one should transfer files between the local and remote systems. At the same time, the administrator can perform some file management operations on the remote system.

4) FTP was once the most popular way to transfer files between systems, but this is no longer true. Newer protocols and utilities have resolved the primary limitations of FTP, which are:

a) In default FTP, the user's login and password are sent unencrypted, posing security risks.

b) Administrators often run FTP servers in "passive" mode, which opens more ports to traffic and poses a security risk to avoid problems with client firewalls.

c) Like all TCP applications, FTP can suffer significant slowdown over bad connections.

d) FTP does not support partial file transfers, making it impossible to recover from interruptions [6].

5) Hyper Text Transfer Protocol Secure (HTTPS) is a secure web communication protocol that transfers files between your isolated cluster and an external server [7]. Practitioners can transfer files using a web-based client or by writing custom HTTPS code. If you need to transfer files over the internet, you can use HTTP or HTTPS to transfer files securely. HTTPS is the more secure option, as it encrypts the connection between the two machines [7].

6) Network File System (NFS): a distributed file system that allows you to share files between machines. It is commonly used in clustered environments, making it a good option for transferring files to an isolated cluster [8].

7) Cloud-based file transfer services like Dropbox, Google Drive, and Amazon S3 can provide secure file transfer capabilities [9]. These services can transfer files between your isolated cluster and the cloud-based service, and users can access them from anywhere with an internet connection.

When selecting a technique for transferring files, it is crucial to consider factors such as security, ease of use, and compatibility with your existing infrastructure [10]. Additionally, you should always verify that the file transfer is successful and that the files are accessible and complete. Finally, regardless of the chosen technique, securing the connection is vital to prevent unauthorized access or data breaches during transfer [11].

IV. Limitations of Existing File Transfer Techniques

Transferring files onto an isolated cluster is a bit different. Practitioners often use isolated clusters to protect sensitive data, applications, or workloads from internal or external threats. In many cases, the administrator will disconnect isolated clusters from the external network or only have minimal connectivity to reduce the risk of unauthorized outgoing data transfers or other data breaches. However, even in this case, the practitioner needs techniques to transfer external files to isolated clusters to move data or files between an external network and an isolated or secured network.

Therefore, there is a compelling need to safely import data to isolated clusters when there is a requirement to protect sensitive data, applications, or workloads from cyber threats and when compliance with data protection and privacy regulations is required. These researchers developed a unidirectional file transfer system for medium-security isolated clusters designed to allow the secure transfer of files from one system to another in a controlled and one-way manner. This research aims to develop a system to transfer sensitive information between systems while minimizing the risk of information leakage or intrusion.

Looking at medium-security isolated clusters, the researchers developed a unidirectional file transfer system that acts as a one-way "turnstile" for secure file transfer between systems not connected to the internet or other external networks. Unidirectional file transfer systems solve this problem by enabling data to be transferred from a source system to a destination system without allowing any data transfer back in the opposite direction. This type of system can be advantageous in heavily regulated environments where there is a need to transfer sensitive data securely between different systems. Organizations can use a unidirectional file transfer system to reduce the risk of data breaches, unauthorized access, and other security threats. During the setup and design process, the investigators attempted to address the following research questions:

1. How can we automate the process of transferring external files to isolated clusters that still optimizes the transfer speed of external files to isolated clusters?
2. What security measures can protect data already transferred from being retransmitted to the non-isolated ("outside") environment?
3. How can we ensure the integrity and compatibility of transferred external files to isolated clusters?

The Turnstile File Transfer (TFT) approach builds on techniques to send files between systems without creating a potential backchannel. The researchers intend to use it in distributed computing environments where data transfers between clusters or systems that are isolated from each other. TFT ensures that data is transferred securely and without any loss or corruption.

The rest of the paper provides some related file transfer techniques. A robust discussion of Turnstile File Transfer (TFT) concepts follows the technique section. The researchers use these concepts to describe a prototype based on the TFT model with TFT implementation details. Examples and evaluations demonstrate how to deploy services and scale the solution rapidly. Finally, the researchers present limitations, conclusions, and recommendations.

V. Related Work

A. Firewall-based Solutions

A simple solution is connecting the isolated environment to the external network via a locked-down firewall. Unfortunately, this approach requires application whitelisting enforced at the device level, and many potential attacks are possible. As such, administrators create scripts that perform functions such as implementing firewall policies. These scripts can configure clients and servers, including protocol, port, IP, and application, to help reduce attack vulnerabilities [12]. Administrators can populate scripts with the host names and/or internet protocol (IP) addresses of your domain resources, creating precise firewall rules for your domain. While appropriate in some circumstances, this would likely not provide sufficient security

for anything beyond a low-security cluster due to its susceptibility to sophisticated attacks.

B. Block-Based File Systems

One solution would involve having two computers share a common drive, physically or logically, at a low-level "block-by-block" method. It is possible to use this option in a virtual environment by creating a read/write partition available via iSCSI to the external cluster and then extending the same partition to the isolated cluster in a read-only manner. While this method allows data transfer, most modern file systems do not expect this dual-homed configuration. As a result, file system caching becomes problematic and can lead to data corruption issues. Administrators design distributed file systems for dual-homed (or multi-homed) scenarios. This design has the clusters rely on a shared network for cache control and file or block-locking messages. Systems in the isolated cluster would not be able to be active participants in these distributed solutions.

A workaround for this would be to have the two clusters, the isolated cluster, and the external cluster, alternate between which of the two systems has the file system mounted. A method to synchronize the time between the two clusters would be necessary. However, it could still lead to race conditions and only permit periodic data transfer, not on-demand, continual data transfer.

C. Media Transport Protocol (MTP) Device Access

A second approach would be to leverage existing file-based USB transfer, mimicking data transfer from USB smartphones to larger desktop systems. The Media Transport Protocol (MTP) is a common protocol well-suited for this application. This approach uses a shared file system extended to isolated clusters using the same methods that some smartphones and digital cameras transfer files to desktop computers.

Instruments can implement MTP as an additional "logical" USB interface. In many cases, instruments can add an MTP interface with firmware modifications, thus avoiding needing a new physical connector and hardware changes [13]. When sharing files with MTP, the instrument retains control of the shared files and moves them in one way only. This on-demand feature works but cannot handle large numbers of files easily, nor can it take symbolic links.

D. Serial Communication Protocols

Administrators use serial data communication strategies and standards when they can secure limited lines (channels) for communication [14]. These strategies and standards are the primary modes of transfer in long-distance communication. Also, administrators use this protocol when embedded systems with various subsystems share the communication channel, and the speed is not a critical issue (Dawoud & Dawoud, 2020). The slow speeds, bi-directionality, and vulnerability to serial line attacks by hackers using network links using PPP or other L2 network protocols make this technique unsuitable.

VI. Methodology

The study design used two virtual system clusters (one isolated, the other Internet-connected) to demonstrate a unique unidirectional system for medium-security isolated clusters. The researchers explored ways to automate the process of transferring external files to isolated clusters that optimized the transfer speed of external files to isolated clusters. However, equally important were the security measures used to protect external files during transfer to isolated clusters while minimizing the risk of data corruption. The investigators used integrity and compatibility of transferred external files to isolated clusters as success measures.

A. Turnstile Design Goals

The research aim was to create an automated tool to integrate into an academic organization's Security by Design (SbD) controls. The turnstile approach allowed an administrator another means to layer the formalized infrastructure design and automate security controls. In practice, this approach added to defense in depth (DiD), allowing educational and research systems to build security into every part of their IT management process but allowing student flexibility. As such, the Turnstile used standardized coding and repeatable, automated architectures and remained consistent across multiple environments for audit standards. As such, the goals for this prototype included:

1) Avoiding network and telecommunication links;

2) Permitting multiple, concurrent access;

3) Providing a self-serve kiosk for individuals to push/pull files;

4) Handling mirroring of large repositories (such as Linux distribution repositories);

5) Providing sufficient bandwidth for multiple users;

6) Using existing common off-the-shelf (COTS) hardware components;

7) Using existing software libraries as much as possible;

8) And limit the additional security risks to the isolated environment.

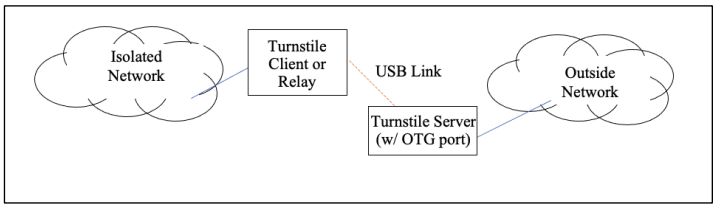
These goals provided a strategy for using multiple layers of security to protect the organization's assets and delay attackers' advances.

B. Physical Connection

Turnstile used a pair of Linux systems connected via a single USB cable: one acting as the server, while the other acts as the client (or optionally the relay). The server needed to support USB device mode (sometimes called USB On-The-Go [OTG]). The prototype used a Raspberry Pi 4 as the server system since this system permits the administrator to configure one of its USB ports in device mode. The other system, the client or relay system, did not need to support USB device mode. For this reason, it could be either a physical or virtual system with a USB port attached. For example, one could use the hypervisor's USB pass-through capabilities.

The researchers connected each system to their respective networks: the isolated network for the client system and an Internet-connected network, or other less restrictive network, for the server. Typically, this is via a physical or virtualized Ethernet connection. In the prototype case, the administrator configured both links for IP networking. Since the USB link does not use IP (or any network protocol), the two networks (or clusters) continued to be separate (Figure 1).

FIGURE 1. TURNSTILE ETHERNET CONNECTION



Note. The network connections can be physical or virtualized.

These configurations were for the automated process of transferring external files to isolated clusters and allowed for an optimized transfer speed of external files to isolated clusters.

C. Low-Level Transfer using USB

The turnstile transfer protocol provided low-level USB bulk transfer operations and Linux's GadgetFS and ConfigFS/FunctionFS API systems on the server. This type of transfer avoids introducing any logical networked or high-level data communication interfaces on either system and limits the attack surface from the client system. Once the administrator started the turnstile server, the connection appeared as a new USB device with a custom vendor ID and product ID on the client system. It presented a single pair of bulk transfer USB endpoints on the client system as if the administrator plugged a USB device into it, known as a hotplug.

The Turnstile system aimed to mitigate apparent attack vectors for data exfiltration. The goal was to continue safe operation and to ensure both the server and the client remained uncompromised. Using only low-level USB API routines reduced the attack surface from the client to the server. No additional endpoints were available on the client system, and if the turnstile daemon on the server exited for any reason, the USB endpoints were closed. In this way, the turnstile added security measures that limited connection hijacking attacks and protected data already transferred from being retransmitted back to the non-isolated or other "outside" environments.

D. Turnstile Packets over USB

Turnstile used single USB bulk transfer packets to communicate between the two systems. Physically, the USB bulk transfer packets were limited to 64, 512, or 1024 bytes, depending on the USB revision negotiated. However, Linux will invisibly fragment and recombine a single transactional packet into and out of these lower-level packets and automatically compute and verify data integrity using a CRC checksum. Turnstile used this larger bulk transfer fragmentation technique to transmit significant amounts of data as a single operation (transferring up to 32KB by default). For the remainder of this paper, the term **TPacket** will represent data transmitted using a single Linux Bulk Transfer request.

All communication to the server was via a simple request-response system. Only the client system was permitted to initiate a request. All requests and all responses only use one TPacket. Larger data transfers required the client to send multiple "continue" requests to receive all of the data completely. The first byte of the TPacket represented the request command (if the client) or the response status code (if the server). Since Turnstile used a single byte as the request or response, a maximum of 256 different requests and 256 different responses were possible. However, the prototype did not define all possible requests/responses. As currently implemented, there were eleven requests [all encoded as capital letters] and four responses [all encoded as lowercase letters].

A sample request/response event was:

C: <request [1-byte]> <optional argument [0+ bytes]>

S: <response [1-byte]> <transaction-id [1-byte]> <response data [0+ bytes]>

If the requested data fitted into a TPacket, the system returned a response code of "l" (indicating the last packet), the transaction ID set to 0, and the system appended the requested data. If the requested data could not fit into a single transaction, then the system responded with "d" (indicating a non-last packet) along with a randomly selected transaction ID and returned the first segment of the data. The client could then request subsequent segments of data using the "continue" request and the corresponding transaction ID. A sample file transfer ("get"):

C: G <path>
S: d <trans-id> <first segment>

C: C <trans-id>
S: d <trans-id> <next segment>

C: C <trans-id>
S: d <trans-id> <next segment>

C: C <trans-id>
S: l <trans-id> <last segment>

Internally, the Turnstile system used the transaction ID to deploy a relay between the clients and servers, allowing multiple requests to occur concurrently. This technique assured the integrity and compatibility of transferred external files to isolated clusters while permitting concurrent access.

E. Turnstile Requests

The client system initiated all data transfers using single-byte commands. The researchers modeled after the FTP system but made it simpler and more limited. For instance, while FTP supports sending and receiving files, the Turnstile system only supports receiving files. Implementing this technique explicitly limited exfiltration-style attacks, although some side-channel attacks might still be possible.

The researchers used four primary requests for data transfer. The Get ("G") command requests the server to transmit a file to the client. The List ("L") command requests a list of files and subdirectories within a specified directory. The Stat ("S") command returns metadata (e.g., file creation date, permissions, and file type) about a single remote file. Finally, the Symlink ("K") command returns the destination path of a remote name. This approach was for simplicity.

The administrator gave the system process paths to the server through a mapping function. This approach precluded the clients from seeing the actual file system structure, but the server administrator could specify access to specific subdirectories for the clients. The researchers also placed further restrictions on these commands to minimize data leakage. All commands did not provide information on files that were neither files nor directories (e.g., block files) and would respond as if the file did not exist. Likewise, the system checked all requested symbolic links on the server beforehand and ignored any pointing to files or directories in non-approved directories.

Lastly, the stat command returned limited information. It returned only the file size, the creation and modification times, the file permissions, and three flags indicating if the item is a file, a directory, and a symbolic link. Turnstile does not return User ID, Group ID, and inode and link count information. The client system could use the stat command to synchronize the permissions of files and directories. Synchronizing permissions and modification dates were essential when creating local mirrors of distributions and other data sources, as some systems used this information to update files.

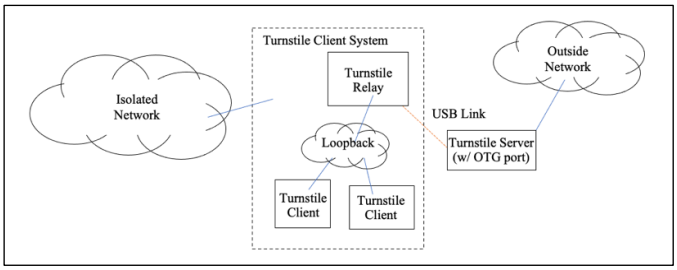
F. Turnstile Relay and Turnstile over UDP

Turnstile used a singular USB endpoint from the server. This approach could require multiple clients to coordinate access to the server by connecting to the server one at a time. The Turnstile relay addressed this need and allowed multiple clients to issue request data concurrently. The clients then talked to the relay using the same Turnstile protocol, wrapped within a User Datagram Protocol (UDP) packet. These UDP packets ***never*** left the isolated cluster; thus, the system maintained the "no networking" requirement between the two clusters. By default, the turnstile relay only accepted UDP packets from the system it was on ("localhost"), and clients could not use the service from other virtual/physical systems within the isolated cluster (Figure 2).

Figure 2

Two clusters with a relay placed between the clients and the server

FIGURE 2. CLUSTERS WITH RELAY BETWEEN CLIENT & SERVER



Note. UDP transmits packets (datagrams) directly to the target device without setting a connection, specifying packets' order, or examining how they are delivered.

The relay implemented a prioritized queuing system. Each command had a different priority, allowing directory-oriented commands to take priority over file transfer commands. Additionally, the clients could raise or lower their base priority to allow interactive uses of the connection to take precedence over background/batch uses, such as mirroring a distribution directory tree. This dynamic prioritization could lead to starvation (DoS) attacks on Turnstile within the isolated cluster. Still, since the Turnstile maintained the isolated nature of the cluster, these attacks were considered out-of-scope in trying to mitigate.

G. Turnstile Command-Line Tools

Turnstile provided two methods for isolated cluster uses to transfer data users: via command line tools and a web interface. For the command line tool method, Turnstile delivered a small set of Linux programs (Table 1).

TABLE 1 TOOLS MODELED AFTER STANDARD LINUX **rcp** & **scp** UTILITIES

Command	Description
t-cat	View a remote file similar to "cat."
t-ls	View a remote directory, similar to "ls."
t-mirror	Copy files/directories as necessary, similar to "rsync"
t-rcp	Copy remote files, supports Linux system wildcards and recursive requests
t-status	Determine if the server and/or relay is working, similar to "ping'
t-sum	Display the SHA512 hash for a local and/or remote file

Note. The functionality of the Linux SCP command is analogous to the older RCP command. As with RCP, the SCP syntax on the command line follows the CP command used to copy the files on the local system.

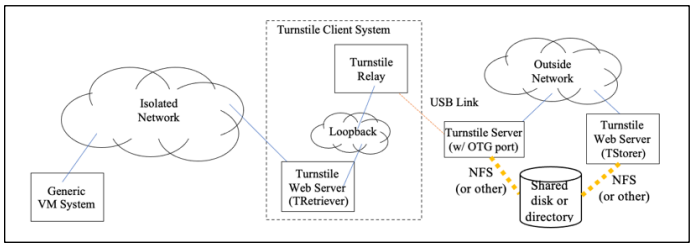
The commands cannot access any files that do not already exist on the server system. The administrator designed command line utilities to assist the network (or cluster) administrators. Non-administrative users of the isolated environment typically would use the web interface.

The commands could directly talk to the Turnstile relay or the Turnstile server and use the relay by default. The *t-mirror* command permitted upgrading Linux-like systems in the isolated cluster from a local mirrored copy of the distribution as if they were on a commodity Internet. This approach allowed an otherwise air-gapped system to pull packages locally and copy requisite patches to the physical media for an update (see Appendix A).

H. Turnstile Push/Pull Web Interface

Once the researchers deployed Turnstile, users could transfer files to the isolated cluster via a web interface without needing intervention from an administrator. The system used two separate web servers: one on the isolated cluster and the other on the outside cluster or network (Figure 3).

FIGURE 3 TURNSTILE SYSTEM USING TWO WEB SERVERS



Note. In this configuration, only the webserver must be public-facing, and the turnstile server itself can be protected behind a firewall if desired.

Users started the transfer process from the outside network and on a web server called the **TStorer** system, which had access to the Internet or internal corporate network. This system should be separate from the turnstile server to avoid attacks on the web server that could affect the turnstile server. Docker or other container systems could provide this separation. The **TStorer** webserver and the turnstile server shared a common directory through an NFS-mounted directory.

The **TStorer** system used Linux's built-in authentication system, PAM, to require a valid user before initiating a file transfer. Since the system used PAM, an external authenticator like Microsoft's Active Directory or any corporate LDAP-based authentication system, it could respond to authentication requests. A user could upload a file, enter text (via an HTML textbox), or download a file from an external URL on the user's behalf. The web server then provided the user with a short URL similar in format to bit.ly or another URL shortener. The user then entered that URL into the isolated cluster for retrieval. On the isolated cluster, one could use any virtual or physical system web browser, and this approach did not require authentication to retrieve the file. However, the storing web server and the retrieving webserver logged all transfers so administrators could audit to ensure integrity.

I. Protecting Data Integrity

In this application, the investigators used two techniques to protect data integrity. The USB protocol itself provided a certain level of data integrity. The second protection mechanism was from the Turnstile. USB Bulk Transfers packets at the lower level included a CRC16 field [15]. These two protection techniques avoided nearly all physical/logical layer data corruption issues and provided equivalent protection to what an Ethernet packet would provide [16].

Beyond this protection, the Turnstile protocol included a **TPacket** command for a client to request the server return a SHA512 hash of a remote file. Both the copy command (*t-rcp*) and the mirror command (*t-mirror*) included an option to require all file transfers to be verified by comparing the local and remote hash. Additionally, one could use a separate command (*t-sum*) to request the SHA512 hash of a remote file and have identical output to an existing Linux command (*sha512sum*).

The designers used the hashing functionality as part of the development process. They performed a system test using files containing random data cache anytime they made substantial changes to the underlying code. Test files sweep in size from 1 byte to 65,600 bytes. The researchers used this test to verify that the system had no edge-case issues with packet sizes. Additionally, they used larger files (1 MB to 8 MB) to look for other problems, such as buffering and performance.

It is important to note that currently, Turnstile does not use any encryption between the client and server. As such, protecting the physical access of the server, the client, and the USB connection was still essential to avoid man-in-the-middle style confidentiality/integrity attacks [17]. Files transferred using Turnstile may still be encrypted before transfer to provide another layer of defense.

J. Bootstrapping the Client

Since one expects placing the client system on an isolated network, the researchers designed Turnstile to need very few additional packages beyond what a Linux operating system typically installs, even in a "lite" installation. The researchers wrote Turnstile in Python (v3), and the client only needed one additional distribution package (typically named "python3-usb1") before enabling the Turnstile command line tools (for example, *t-rcp*) that could transfer other desired files. On Ubuntu and Raspberry Pi OS, this package depends on two packages, python3 and libusb1, and both vendors typically install these packages with the initial operating system. The web retriever requires additional packages, including Python's flask package (python3-flask), but one can transfer those other packages using the included *t-rcp* command. As such, the Turnstile prototype was compatible with various standard environments.

K. Testing Assessment Approach

The researchers measured performance between a Turnstile server and client testing a Raspberry Pi 4 and a Raspberry Pi with a virtualized Intel-based Linux system as servers and clients. The researchers did not include the Turnstile relay in the performance testing environment for consistency. The hardware and software attributes of the testing environment were extensive and specific (Appendix B).

L. Performance Testing Procedure

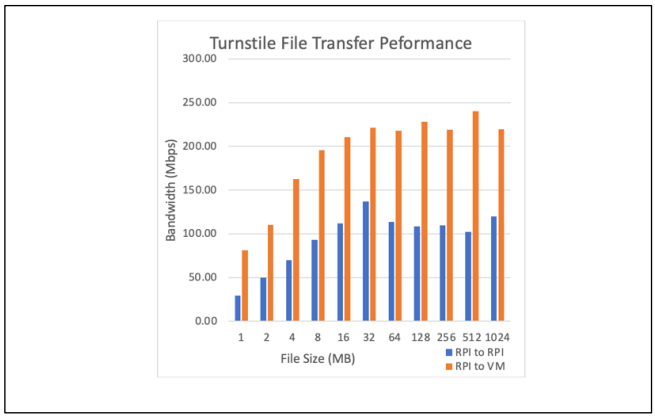
Researchers created test files using random data (initially created via the Linux *dd* command using /dev/random as the source). The pilot used random data to avoid any unintentional

compression or caching interactions. Prototype files were transferred to and from a RAM disk file system (a.k.a. **tmpfs** mounted on /tmp) to limit any effect of local file systems on the transfer speed. File transfer duration was measured on the client system using the Linux **time** command. The pilot used the overall (sometimes called "real" or "wall clock") time for the measurement. The pilot tested a variety of file sizes, as well as an assortment of USB transaction packet (**TPacket**) sizes.

VII. Results

The Turnstile automated process for transferring external files to isolated clusters optimized the transfer speed of external files to isolated clusters. Turnstile had an adequate bandwidth of approximately 110 Mbps when the client and server systems were Raspberry Pi 4 computers. Replacing the Raspberry Pi client with an Intel-based virtual computer client effectively doubled file transfer performance to approximately 220 Mbps (Figure 4).

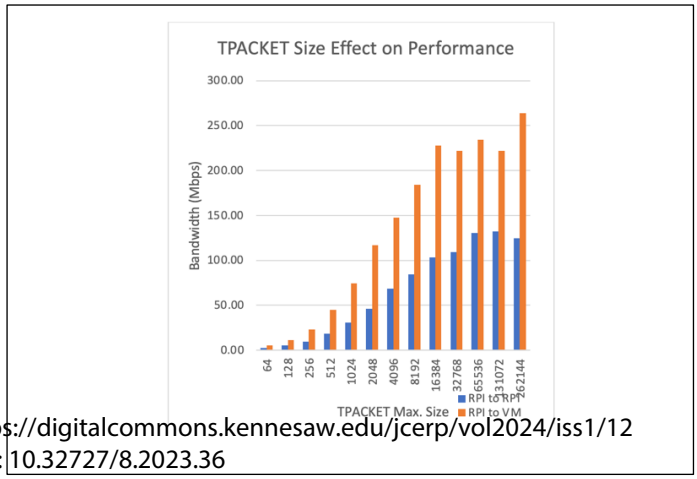
FIGURE 4 RESULTANT SPEED BETWEEN FAST ETHERNET & GIGABIT ETHERNET CONNECTION



One parameter controlling the overall transfer speed was the USB transaction packet size, also known as the size of the **TPacket**. Since Turnstile is a simple request/response system, the larger the packet, the more efficient the overall transfer speed. However, a large packet size will affect the wait time for queued requests if multiple clients use the relay. Regardless, this automated transferring of external files to isolated clusters optimized the transfer speed of external files to isolated clusters.

File transfer speeds improved with the increase of **TPacket** size, but the improvement was less pronounced beyond a packet size of 64KB (Figure 5).

FIGURE 5 TPACKET SIZE EFFECT ON PERFORMANCE



Researchers chose a **TPacket** size of 32KB as the default value for the server as a compromise between file transfer performance and latency issues when connecting multiple clients to a single server via a relay. The network administrator can specify a different maximum **TPacket** size when starting the server, and all client programs will query the server for this value before file transfers.

VIII. Conclusion

One of the primary goals of Turnstile is to prevent protected data in the isolated cluster from being transferred to the non-isolated ("outside") environment. The Turnstile system (source code available at github.com/monnin/turnstile) provides unidirectional file transfer between two computer systems. As discussed earlier, an administrator has no protocol command to transfer a file in this direction. One can only use a "GET" operation, not any "PUT" operation. An attacker could modify the code to add such a command. Still, defense in depth makes this more challenging to achieve, as an attacker would need to have two separate successful attacks (one against the client system and another against the server system) and modify both the client and server software.

The Turnstile also includes mechanisms to ensure the integrity of the files transferred through low-level packet checks (via CRC16 verifications) and a file-level check (via SHA512 hash verification). The two systems work in concert and ensure that any alteration of the data during transmission is detected. Turnstile also tries to keep a low administrative overhead for the system's support staff. The client system's command line utilities only require one additional Python module beyond what distributions provide in a typical Python installation package. Thus, the command line utilities permit Turnstile users to transfer any other files necessary, such as those needed to deploy the web interface. Additionally, all command line utilities are modeled after existing Linux commands, so a seasoned Linux administrator will learn the commands quickly.

Using medium-security isolated clusters, the researchers successfully developed a unidirectional file transfer system that acts as a one-way "turnstile" for secure file transfer between systems not connected to the internet or other external networks. The Turnstile provided unidirectional file transfer systems. The solution enabled data to be transferred from a source system to a destination system without allowing any data to be transferred back in the opposite direction. The researchers found an automated process of transferring external files to isolated clusters that still optimized the transfer speed of external files to isolated clusters using Linux distributions and commands. The USB and turnstile architecture protected data transferring and prevented that data from being retransmitted back to the non-isolated ("outside") environment. Using physical security and architectural design, the Turnstile assured data integrity and compatibility of transferred external files to isolated clusters.

The researchers built the Turnstile File Transfer (TFT) approach using techniques to send files between systems without creating a potential backchannel. Administrators can use it in distributed computing environments where data transfers between clusters or systems that are isolated from each

other. TFT ensures that data is transferred securely and without any loss or corruption. This TFT system can be advantageous in heavily regulated environments where there is a need to transfer sensitive data securely between different systems. Organizations can use a unidirectional file transfer system to reduce the risk of data breaches, unauthorized access, and other security threats.

REFERENCES

[1] A. Blenk, A. Basta, M. Reisslein & W. Kellerer, *Survey On Network Virtualization Hypervisors For Software Defined Networking*. (Seventh Ed.). Ithaca: Cornell University Library, arXiv.org, 2016.

[2] A. R. Mamidala, S. Narravula, A. Vishnu, G. Santhanaraman, & D. K. Panda, D. K. On using connection-oriented vs. connection-less transport for performance and scalability of collective and one-sided operations: Trade-offs and impact. Paper presented at the 46-54. https://doi.org/10.1145/1229428.1229437, 2007.

[3] J. L. Hennessy, and D. A. Patterson, *Computer Architecture: Quantitative Approach* (5th ed.). Morgan Kaufmann/Elsevier, 2012.

[4] D. J. Barrett, R. G. Byrnes, & R. E. Silverman, *SSH, The Secure Shell: The Definitive Guide* (2nd ed.). O'Reilly, 2005

[5] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan, & A. Nataraj. Benchmarking The Effects Of Operating System Interference On Extreme-Scale Parallel Machines. *Cluster Computing*, *11*(1), 3-16. https://doi.org/10.1007/s10586-007-0047-2 , 2008.

[6] C. Fan, A. Karati, & P. Yang. Reliable File Transfer Protocol with Producer Anonymity for Named Data Networking. *Journal of Information Security and Applications*, *59*, 102851. https://doi.org/10.1016/j.jisa.2021.102851, 2021.

[7] E. Rescorla. RFC 2818, HTTP Over TLS. Internet Engineering Task Force. Retrieved from https://www.rfc-editor.org/rfc/rfc2818, May 2000.

[8] B. Callaghan. *NFS illustrated*. Addison-Wesley, 2000.

[9] J. Taheri, A. Y. Zomaya, H. J. Siegel, & Z. Tari. Pareto Frontier for Job Execution and Data Transfer Time in Hybrid Clouds. *Future Generation Computer Systems*, *37*, 321-334. https://doi.org/10.1016/j.future.2013.12.020, 2014.

[10] R. U. Rasool, H. F. Ahmad, W. Rafique, A. Qayyum, & J. Qadir. Security and Privacy of Internet of Medical Things: A Contemporary Review in the Age of Surveillance, Botnets, and Adversarial ML. *Journal of Network and Computer Applications*, *201*, 103332. https://doi.org/10.1016/j.jnca.2022.103332, 2022.

[11] L. P. Davis, C. J. Henry, R. L. Campbell Jr., & W. A. Ward Jr. High-performance computing acquisitions based on the factors that matter. *Computing in Science & Engineering*, *9*(6), 35-44. https://doi.org/10.1109/MCSE.2007.115, 2007.

[12] Steve_N. Windows Defender Firewall, PowerShell Scripts/GUI Tools, GitHub. Retrieved on April 21, 2023, at https://github.com/SteveUnderScoreN/WindowsFirewall, Nov 21, 2019.

[13] A. Purcell. Get Instrument Screen Captures and Data Files Quickly Using USB: Implementing the USB Media Transport Protocol (MTP) Helps Facilitate File Sharing. *Electronic Design*, *62*(11), 36, 2014.

[14] D. S. Dawoud, & P. Dawoud. *Serial Communication Protocols and Standards: RS232/485, UART/USART, SPI, USB, INSTEON, Wi-Fi and WiMAX*. River Publishers. https://doi.org/10.1201/9781003339496, 2020.

[15] Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V. Universal Serial Bus, Revision 2.0., USB Implementers Forum. Retrieved July 26, 2023, at https://www.usb.org/document-library/usb-20-specification, April 27, 2000.

[16] IEEE Xplore. IEEE Standard for Ethernet, in *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, vol., no., pp.1-4017, doi: 10.1109/IEEESTD.2016.7428776, March 4, 2016.

[17] J. Axelson. *USB complete: The developer's guide* (Fifth ed.). Lakeview Research, 2015.

Appendix A: Packet Format for Requests and Responses

Requests (Client to Server):

Command	Name	Description	Comments
C<trans-id>	Continue	Get the next packet in a larger response	The only command that sends a transaction ID.
G<path>	Get	Get a file (can use multiple "P" packets prior)	Returns an error code if the path is not a [normal] file
H<path>	Hash	Return a SHA512 hash for a filename	Returns value as a hexadecimal string and not as binary bytes
K<path>	Symlink	Return the destination if the path is a symbolic link	Returns an empty packet if valid, but not a symlink)
L<path>	List / ls	Performs a simplified "ls" on a directory	If the path is a directory: Returns the name (but not the full path) of all files/dirs in that dir (non-recursively) (\0 between items) If a path is a file: Returns two \0\0 if the file exists If a path is neither a file nor a directory: Returns an error code
M	Max Packet	Get MaxPacket	Set by the server or relay. Returns the maximum packet size as a 32-bit number
N<data>	No Op	It does nothing on the relay/server but has a valid response packet.	Relay special case: If data is a NUL (or is not present), then the relay answers the NoOp; otherwise, it forwards the request to the server (that way, a client can check both entities)
P<ppath>	Prefix Path	Allows for paths > 511 bytes long	Since the path might take multiple packets, one can send multiple "prefix" packets ahead of the actual command packet. (e.g., P<path1>, P<path2>, G<path3> would request a "get" operation with a longer file path). The server responds with a continue packet
Q<prio>	Queue Priority	Set the priority of the connection	Used by the relay, ignored by the server
S<path>	Unix-like stat(2) command	Return (limited/filtered) metadata about a file/directory	Returns flags, mode, size, mtime, and ctime. Mode does NOT include file type. Returns an error code if the path is not a file or directory. Flags: LSB: 0=file, 1=dir MSB: 1=symlink, 0 otherwise If a directory: a returns a single item, not the contents of the directory.
Z	Reset	Clears/deletes all partial transfers	(aka resets incomplete transactions)

Responses (Server to Client):

Response	Name	Description	Comments
c	Continue	Continue with command	Only used with Path (“P”) requests
d<id><data>	Data	First/next Data Packet	Data from the server with more unsent data
l<id>[<data>]	Last	The last/only data packet	Completes the transaction. An empty data section is permitted.
z<id>	Error	An error response to a request	

Appendix B: Performance Testing Environment

	Turnstile Server	Turnstile Client A	Turnstile Client B
H/W	Raspberry Pi 4 Model B	Raspberry Pi 4 Model B	Virtualized Linux system
O/S	Raspberry Pi OS; Debian Version 11	Raspberry Pi OS; Debian Version 11	Ubuntu 20.04 LTS
CPU	Broadcom BCM2711	Broadcom BCM2711	(Virtualized) Intel 12 core vCPU
RAM	8 GB	8 GB	(Virtualized) 12 GB

Virtualization Environment for Turnstile Client B

Motherboard	Asus ESC8000 G4
CPU	[Dual] Intel Xeon Gold 6225R (@ 2.90 Ghz)
RAM	768 GB
Hypervisor	VMWare ESXi 7.0u3

DOI: 10.32727/8.2023.36