

USB Drop Attack Simulation Project Documentation

Barbara Andino, Amelia DiGennaro, Damian Figueroa-Gonzalez, Moath Hussein

Florida International University

CIS 4951 - Capstone II

Professor Seyedmasoud Sadjadi

December 3rd, 2025

(1) Executive Summary

This capstone project focuses on the simulation of a USB drop attack to educate users about the risks associated with unknown USB devices. USB drop attacks are a social engineering technique where malicious USB devices are intentionally left in public spaces to exploit human curiosity, and once they are plugged in, trigger unauthorized system access to the device. This attack is highly effective, as unsuspecting users often insert unknown USB devices without considering the potential security consequences.

The primary beneficiaries of this project includes students, faculty, and general computer users that might encounter an unknown USB device in the real world. Educational institutions and IT departments also benefit from this project, as it provides a practical training resource for cybersecurity awareness initiatives. By demonstrating how easily a USB drop attack can unfold and providing mitigation strategies, this project raises awareness and reinforces safe behavior when handling unfamiliar USBs.

To address this security risk, we developed a solution in a controlled and ethical way. Our solution is a self-contained simulation that activates when a user opens an executable stored on a USB device. The program imitates the behavior of a malicious payload through elements that mirror a real attack. This includes terminal-style warnings, fake system scans, and a progress loading bar, ensuring no actual harm or data access occurs. The executable file was developed using Python and packaged as a standalone .exe to maximize portability across Windows systems and not require Python to be installed. The file was given the name “CNT4403_AnswerKey” to simulate a tempting label an attacker might use to persuade the user to click the file. Our implementation focuses on visual, behavioral, and psychological cues that are commonly associated in real USB-based attacks. This allows users to experience the

consequences of plugging in an unknown USB drive in a safe training environment. The simulation concludes with a message-up containing an educational message that explains the attack, discusses the risk, and provides steps for safe cybersecurity practices.

Key results include increased user awareness of USB-related risks, successful demonstration of the attack flow stages, and the creation of clear educational content that explains how to recognize and respond to suspicious USB devices. Overall, the project highlights how seemingly small and innocent actions can lead to serious security breaches and system takeovers, emphasizing the importance of proactive user actions and mitigations.

(2) Problem & Requirements

Context

USB Drop Attacks continue to be highly effective social engineering techniques due to the tendency of users to trust physical devices and underestimate the risks of inserting unfamiliar USBs. By planting these malicious USB drives in accessible locations, attackers lure users to plug them in, creating an entrypoint into the system to perform malicious activity like launching harmful payloads and scripts. This product addresses these security problems by creating a controlled, ethical simulation that demonstrates the consequences of plugging in an unknown USB device. Instead of deploying real malware, the USB contains a self-contained Python based executable (.exe) that activates when opened. It mimics malicious behaviors through fake scans, terminal-style warnings, progress bars, and threat messages before ending with an educational explanation. This experience shows users how fast an attack can occur, helping them to recognize warning signs and develop safer habits.

Stakeholders

Primary Users

- Students
- Faculty Members
- General Computer Users

Educational & Organizational Stakeholders:

- Cybersecurity Instructors
- University IT Departments
- Awareness and Training Program Coordinators

Project Team

- Barbara Andino
- Amelia DiGennaro
- Damian Figueroa-Gonzalez
- Moath Hussein

Personas

Persona 1: The Curious Student

- This student found a USB drive labelled “Test Answer Key” left behind at the library and plugged it into their computer out of curiosity, without considering the potential risks. They have a low technical background and are primarily motivated by curiosity and temptation than by security awareness. They benefit from experiencing first-hand how fast plugging in an unknown USB can lead to suspicious activity.

Persona 2: The Busy Employee

- This employee frequently inserts USB drives into their device quickly at their shared workspace, without verifying their source, because they are busy and focused on completing their assigned tasks efficiently. They usually assume that the IT department handles all security threats, therefore in need of educational feedback to understand the dangers of unknown USB devices.

Persona 3: The Security-Aware Beginner

- This user makes an effort to follow cybersecurity best practices, but does not fully understand how USB-based threats work. A friend hands them a USB and says “Can you check what’s on this?”, and they plug it in because they want to be helpful and assume it is safe. They benefit most from hands-on demonstration rather than text instructions, as interactive simulations help reinforce safe decision making.

Functional & Non-Functional Requirements

Functional Requirements

1. **Executability:** The USB must contain a standalone executable (exe.) that runs without requiring Python installed on the system.
2. **Simulation Behavior:** When opened, the program must simulate attacker behaviors with fake download, scans, and warning pop-ups.
3. **Realism:** Display a fake loading sequence, progress bar, and terminal-style alerts and pop-ups that mimics a suspicious malware execution and download activity.
4. **Education:** Conclude with an educational message that explains the attack was a simulation and outlines safe practices to follow.

5. **Safety:** Ensure the simulation performs no real malicious actions, system changes, or data access.
6. **Reproducibility:** Run consistently across Windows systems commonly used for demonstrations and allow the simulation to be rerun for training sessions.

Non-Functional Requirements

1. **Safety:** The simulation must be fully non-malicious and compliant with all academic and ethical standards.
2. **Portability:** The executable must function on standard Windows systems without any installations.
3. **Performance:** Messages, visuals, and pop-up warnings must all be clear and understandable to non-technical users.
4. **Usability:** The simulation must run smoothly and trigger all visual elements without delays.
5. **Realism:** Alerts, scans, and loading bars must all be convincing and mirror a real-life malicious behavior attack.
6. **Reproducibility:** The code should be simple to update for future improvements or new educational and training contexts.

Constraints

- Upon USB insertion, the user must open the executable file manually because autorun limitations on modern systems prevent automatic execution.
- The project scope must remain within the semester and course timeline limits, using only available hardware, and student skill sets.

- The simulation is packaged as a Windows executable and therefore it cannot run directly on macOS or Linux without additional tools or modifications.

Success Criteria

- The executable runs reliably and smoothly across the demonstration systems.
- Visual elements like scans, warnings, and progress bars convincingly simulate malicious activity.
- Educational messaging clearly explains the attack simulation and promotes safe user prevention behaviors.
- The users report increased awareness of USB-related security risks after completing the simulation.
- The product integrates risk analysis, mitigation strategies, and awareness materials into an overall cohesive training simulation and tool.

Risks

- Technical & Hardware Risks:
 - Some systems may restrict or block unsigned executables, preventing the simulation from adequately launching.
 - Limited compatibility testing across different Windows versions could produce inconsistent or varied displays.
 - Differences in USB port types (USB-A or USB-C) may require adapters to plug in and limit compatibility.
- User Interpretation Risks
 - Users may mistake the simulation for real malware and panic to shut down or close the computer abruptly, interrupting the demonstration.

- Environment Risks:
 - Institutional or lab computers may block executables from USB devices due to strict security restrictions.

Prioritized Backlog:

Top 10 Items:

1. **US-18 - Build Harmless USB Payload:** As a team member, I want to build a harmless USB payload that simulates the risk without actual harm.
2. **US-19 - Document Attack Flow:** As a team member, I want to document the attack flow (from the USB drop to potential breach).
3. **US-20 - Demo Quality Review:** As a team member, I want to test and review the simulation/demonstration to ensure its clear.
4. **US-17 - Diagram for USB Attack Scenario:** As a team member, I want to diagram a theoretical USB attack scenario so that the risk is easier to visualize.
5. **US-22 - Educational Tips Sheet Writing:** As a team member, I want to write short, clear tips for users to avoid USB-related threats.
6. **US-21- Create Awareness Flyer:** As a team member, I want to create a flyer to warn users about USB drop attacks.
7. **US-23- Slideshow Presentation:** As a team member, I want to create a presentation for a 3 minute awareness talk session.
8. **US-25 - Final Report Editing:** As a team member, I want to review and polish the full Risk Assessment for spelling, formatting, and flow.
9. **US-27 - Compile and Submit Deliverables:** As a team member, I want to compile all final project files into one submission folder.

10. **US-28 - Presentation Rehearsal:** As a team member, I want to prepare and rehearse my portion of the final presentation to deliver it confidently.

Link to the full Product Backlog:

<https://drive.google.com/file/d/1gZa6pS5IQ592Mz-KuRvsZWctfKFN0vDL/view?usp=sharing>

(3) System Overview & Architecture

System Purpose & Design

When the user plugs in the USB and opens the disguised executable file, the system begins a staged “attack simulation”. The program displays fake loading bars, terminal-like output, and an imitated “file-scanning” that appears to be threatening to a non-technical user. Once the simulation is complete, a message will appear informing the user the attack was fake and teaches about the real risks of USB drop attacks. This tool was created with the intent to be used by students, teachers, cybersecurity clubs, training programs, and workplaces as an awareness resource.

The architecture is intentionally simple and self-contained to ensure compatibility across Windows environments. The goal of design is to be harmless, realistic, portable, maintainable, and educational. All logic and interface components run locally on the machine without requiring internet access or other external dependencies. The product is built as an executable to simulate a USB drop attack in a controlled and educational environment. The simulation is free of networking, data collection, and destructive behavior. It is written in Python, packaged as a standalone Windows executable with PyInstaller, and designed to run on any Windows machine. It does not require Python to be installed.

Product Flow

The behavior of our product is as follows:

1. USB is inserted into a personal computer.
2. The user double-clicks the executable.
3. Payload simulation begins automatically, depicting:
 - Fake system scan
 - Fake loading instructions
 - Terminal-style warnings
 - Undismissable progress window
4. Simulation concludes with an educational message:

 SECURITY WARNING

You attempted to open a file from an unknown USB device.

Malicious USB files can:

- Deploy malware automatically
- Steal or corrupt your data
- Execute unauthorized commands
- Spread infections across networks

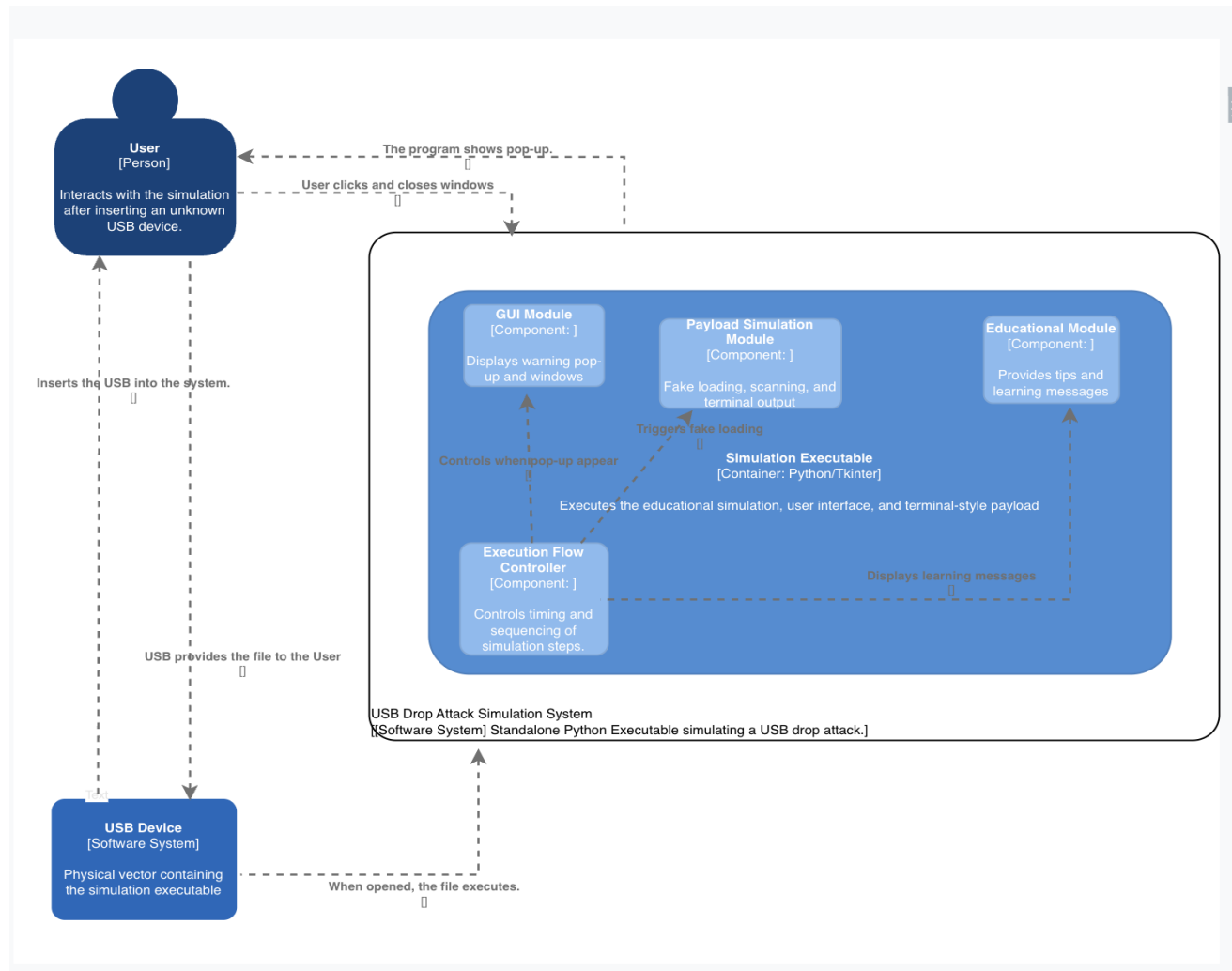
How to stay safe:

- Never plug in found USB drives
- Verify devices before connecting them
- Report suspicious devices to IT immediately

This was a harmless educational simulation.

5. Training and awareness tips are displayed.
6. The user clicks OK or Enter to close the message displayed.
7. The program exits and returns the user to their desktop.

Architecture Diagram



Key Technologies & Framework

The product uses a set of portable technologies that allow the simulation to run on a broad range of systems:

Component	Technology
Core Language	Python 3.12
Packaging	Pyinstaller
UI	Tkinter & Console output
OS Target	Windows 10/11
IDE	VS Code
Physical	USB stick

Services

This product consists of a small set of core components packaged into a single executable.

- GUI Component: Handles pop-up message and warnings
- Payload Simulation: Produced terminal-style output, fake scanning messages, and timed sequences
- Execution Flow Controller: Manages the order of events to create realistic attack
- User Educational Component: Displays best security practices

(4) Discipline-Specific Depth

Threat Model & Risk Analysis

Our system is an educational USB drop attack simulator: a Windows executable placed on a USB drive that, when opened, shows a fake “analysis/loading” sequence in the console and then displays a security warning popup. It intentionally does not modify files, access the network, or request elevated privileges. The key assets are student/lab devices, user accounts, and user privacy, along with the educational value of the exercise. The reference adversary is a real-world attacker who plants USBs in public areas to gain access.

The main attack surfaces we model are the physical USB port, the removable drive’s contents, and the social-engineering vector. Under STRIDE, our simulator primarily illustrates Spoofing and social engineering, while explicitly avoiding Tampering, Information Disclosure, Denial of Service, or Elevation of Privilege. Technical risk from the executable itself is intentionally low, but the exercise highlights the real-world risk: if this were a real payload, the same user behavior could lead to data theft or ransomware.

Risk Matrix

Scenario	Likelihood	Impact	Overall Risk	Notes
Real attacker plants malicious USBs in campus areas	Medium	High	High	Same initial behavior we simulate, but with real malware.
Students plug in simulator USBs in controlled lab setting	High	Low	Medium	Intended behavior; technical impact is low, educational impact is high.
Students generalize “plugging random USBs is okay” after lab	Low–Med	Medium	Medium	Mitigated by clear framing and a strong debrief.
Simulator binary behaves unexpectedly	Low	Low	Low	Reduced by code review, no temp files, and standard-library-only design.

Controls & Assurance

The simulator is designed with least privilege and safety in mind. The executable uses only Python standard libraries (time, os, Tkinter), runs entirely in user space, and does not read or write user files, edit the registry, or open network connections. All text shown to the user is static and deterministic; there is no external input that could be abused for injection or code

execution. Earlier versions that wrote temporary files (and triggered antivirus) were refactored so that all behavior is in-memory and non-persistent.

Procedurally, the exercise is further controlled by limiting distribution of USBs to approved areas and time windows, communicating the educational purpose, and conducting a post-exercise debrief that ties the simulation back to organizational policies (“do not plug in unknown media”). Conceptually, the project supports NIST-style controls such as AT-2 (security awareness training) and MP-7 (media use restrictions), and follows secure design principles: minimize privileges, avoid unnecessary functionality, and make security tradeoffs explicit.

Security Testing

Although the application is small, basic security testing was performed. A manual static review (SAST) of the Python source confirmed that `os.system` is used only to clear the console with fixed arguments, and that there are no file I/O, registry edits, socket calls, or dynamic code execution. The code is organized into a few simple functions (payload simulation, show warning popup), which makes it easy to review and reason about behavior.

Dynamic testing (DAST-style) was done by running the compiled executable in a Windows VM and observing its behavior: no unexpected network traffic, no persistent files, no registry changes, and clean termination after the popup closes. The dependency/SBOM surface is minimal because only standard library modules are used; there are no third-party packages. When antivirus flagged an earlier build that used temp files, that feature was removed and the program retested, resulting in no alerts. No vulnerabilities such as data leakage, persistence, or privilege escalation were identified; the only “risk” left is intentional and educational, demonstrating how easy it is to execute unknown code from a USB device.

(5) Implementation Notes

Repository Structure

The project repository is organized with a `src/` folder containing the main Python script for the simulation program. The `README.txt` provides instructions for setup, running the program, and an overview of the project. A `requirements.txt` file lists the necessary Python packages, and the `dist/` folder contains the compiled executable `.exe` from the Python script.

Code/

```
|— README.txt
|— src/
|   |— CNT4403_AnswerKey.py
|— dist/
|   |— CNT4403_AnswerKey.exe
|— build/
|— (PyInstaller build artifacts)
```

Coding Conventions

The Python code follows PEP 8 style guidelines for readability. Descriptive variable and function names are used clearly to indicate their purpose, and comments are included to explain key logic for GUI creation and the timed terminal output. For full code structure, see Appendix C.

Functions:

- `payload_simulation()` : console-based fake loading and “analysis” stages.

- `show_warning_popup()` : Tkinter warning dialog with educational content.
- `main()` : orchestrates the flow (run simulation, show popup).
- `if __name__ == "__main__": main()` : standard Python entry point.

Notable Patterns

The project uses a modular design where core functionalities like the GUI creation, timed terminal output, and the message popup, are organized into separate functions to maximize clarity. The GUI and Tkinter manage the user interaction and trigger the educational pop-up.

Tradeoffs

Using a single script simplifies deployment, however, it does limit scalability for future feature expansion. Conversion to a .exe file allows the program to run easily on Windows systems but can trigger security warnings from some antivirus programs. There are minimal external dependencies that reduce installation complexity, yet they limit the ability to use more advanced GUI or animation features.

(6) Testing & Evaluation

Test Strategy (Unit / Integration / End-to-End)

The USB drop attack simulator is intentionally small, so the testing strategy focused on a mix of lightweight unit checks, integration tests on the packaged executable, and full end-to-end scenarios that mirror how a real user would interact with the system.

- **Unit-level testing.**

At the code level, the core functions (payload simulation, and show warning popup) were repeatedly executed from the Python source to confirm that:

- The loading sequence runs through all stages without throwing exceptions.

- The progress bar updates correctly from 0–100%.
- The Tkinter popup is created, shows the full warning text, and closes cleanly when the user acknowledges it.

These checks ensured that each logical component behaved as expected before packaging.

- **Integration testing.**

After building the Windows executable, integration tests verified that:

- The console window appears, displays the fake analysis/loading messages, and hands off correctly to the GUI popup.
- No additional windows or error dialogs are spawned.
- The program exits cleanly after the user dismisses the popup, leaving no persistent files or processes.

- **End-to-end (E2E) testing.**

E2E tests were performed using the realistic scenario: a USB drive is inserted, the executable is opened from the drive, and the user experiences the full flow. The E2E checks focused on:

- Confirming that the experience feels “realistic” enough to simulate a USB drop attack (curiosity → execution → surprise/warning).
- Ensuring that after the exercise, the system is in the same state (no files added, no configuration changes, no instability).

Coverage

Given the limited codebase, functional coverage is effectively close to 100%: all branches of the loading loop and popup flow were exercised. Special cases tested included:

- Closing the popup immediately vs. leaving it open for several seconds.
- Running the simulator multiple times in a row from the same USB.
- Launching the executable on different Windows machines to verify consistent behavior.

Performance

Performance expectations were modest but important for user perception. The key checks were:

- The console “loading” sequence runs smoothly without noticeable stutters, with each stage progressing at a consistent pace.
- The total time from double-clicking the executable to the appearance of the warning popup is short enough to keep the user engaged, but long enough to feel like a believable “file analysis.”

During tests, CPU and memory usage stayed low, and there were no observable slowdowns on typical lab-grade hardware.

Reliability

Reliability was assessed by:

- Repeatedly executing the program across multiple sessions without system crashes, hangs, or residual processes.
- Verifying that the application handles normal user actions (closing the popup, re-running the file, closing the console) gracefully.

No crashes, freezes, or unrecoverable states were observed, and the program behavior remained deterministic across runs.

KPIs and Metrics

Because this is an educational security-awareness tool, the key performance indicators (KPIs) emphasize user behavior and experience rather than traditional throughput metrics. The main KPIs defined for the project were:

- **Execution success rate:**

Percentage of test machines on which the executable ran end-to-end without being blocked or quarantined by antivirus or endpoint controls.

- **Flow completion rate:**

Percentage of users who see the complete flow (console loading + popup message) versus those who close the program early.

- **Time to warning:**

Approximate time from opening the file to the moment the warning is displayed. This helps balance realism (it feels like “something is happening”) against user patience.

- **Awareness uplift (qualitative):**

Informal feedback from participants after the debrief, focusing on whether the exercise made them more likely to avoid plugging in unknown USB devices in the future.

Defect Summary and Remediation

A small number of defects were identified and resolved during development and testing. The most relevant ones are summarized below:

ID	Defect Description	Impact	Resolution
D-1	The loading bar did not correctly reach 100% or appeared “stuck.”	Reduced realism; confusing UI.	Adjusted the progress loop and print formatting so the bar updates correctly on each iteration.
D-2	Early version wrote a temporary file to disk, which triggered antivirus as suspicious behavior.	Risk of AV blocking/quarantining the demo; undermined trust.	Removed all temp file creation and refactored the simulation to operate entirely in memory (console + popup only).
D-3	Multiple popups could be spawned if the function was accidentally triggered more than once.	Annoying user experience; cluttered screen.	Ensured (show warning popup) is called only once in the main flow and the program exits after the user dismisses it.

After these fixes, no remaining defects were observed that affected security, reliability, or the educational goal of the project. The simulator now provides a stable, low-risk way to demonstrate the danger of executing unknown code from dropped USB devices.

(7) Security, Privacy, Accessibility & Compliance

The USB Drop Attack simulation was intentionally designed to be safe, ethical, and compliant with cybersecurity best practices all while serving as an educational tool. Given that the demonstration involves mimicking the behavior of a malicious USB device, strong attention was given to ensuring that the simulation does not violate user privacy, compromise systems, or create any unintended harm.

Ethical Impacts

The simulation was developed with a strong emphasis on ethical responsibility. Although the project demonstrates how a malicious USB device can behave, the simulation remains strictly non-destructive and transparent to the user. All of the actions that are displayed such as the fake downloads and warning pop-ups are entirely simulated, ensuring no harm to the user or system. The final educational message makes it clear that the demonstration is not real malware, preventing any fear-based manipulation, and instead promoting responsibility, education, and cybersecurity awareness.

Legal Impacts

The product complies with institutional and legal expectations for cybersecurity education. It does not bypass any security controls, install software, access system files, or engage in any behaviors that violate acceptable-use policies. Since the executable does not contain malicious code or perform any unauthorized system changes, it aligns with academic integrity guidelines. The product remains suitable for use in classroom or training environments because it contains compliance with academic cybersecurity guidelines around the safe handling of removable media.

Societal Impacts

USB drop attacks succeed because they exploit the human tendencies of curiosity, temptation, and misunderstanding. This simulation addresses these societal vulnerabilities by giving users a safe way to actually experience what could happen when plugging in an unknown USB device. With the use of realistic visuals and an educational explanation, the project encourages safer digital habits and increases overall cybersecurity awareness. It serves as a practical educational tool to help reduce human-factor risks to cybersecurity in society.

Security Considerations

The simulation was designed in order to avoid performing any real malicious actions. The executable does not modify any system settings, collect data, or install any files. All of the behaviors such as the fake scans and warnings are only visual, giving users a sense of realism without actually exposing them to any risks. The program is run locally from a USB and can be safely rerun across all standard Windows systems.

Privacy Considerations

The executable does not access or tamper with any information from the user's computer, maintaining full user privacy. No system files or personal data are interacted with during the demonstration, and there is no access to or storing of any information from the user's computer. This makes sure that the users remain anonymous and their devices remain completely unaffected beyond the visual simulation.

Accessibility Considerations

The educational content and visual components of the simulation were created to be clear and easy to understand for users of all levels of technical knowledge. The high contrast pop-ups, simple language, and readable format all help to ensure accessibility across a diverse audience.

The awareness materials, tips sheet, and presentation slides all follow accessible formatting practices, including consistent headings and concise language for easy comprehension.

(8) Deployment & Operations

The USB drop attack simulator is deployed as a standalone Windows executable placed on removable USB drives. The executable is built from the Python source code (using only standard-library modules such as time, os, and Tkinter) and packaged into a single .exe file so that no Python installation is required on lab or student machines. There is no installer and no persistent service; users simply double-click the executable from the USB drive to run the simulation. During operation, the simulator runs entirely in user space and terminates cleanly after the warning popup is dismissed. No background processes are left running, no files are written to disk, and no network connections are opened. Because the application is non-persistent, updates are handled by rebuilding the executable, verifying it in a test environment, and then copying the new version to the USB drives before the next exercise. For full installation steps, see Appendix A.

(9) Limitations & Future Work

Limitations

The payload simulation uses simplified behaviors and does not mimic real malware complexity. The program relies on a single Python script, therefore limits modularity and maintainability. Upon trying to email the .exe file to a group member for a test on their computer, the email provider flagged and blocked the .exe, restricting testing. Also, the simulation currently only runs on Windows and is not cross platform. We attempted to open the file on a macOS

device and it failed to open. Lastly, the user interactions are not recorded, including click behavior or engagement tracking.

Shortcuts

The GUI layout and timing logic are hard-coded instead of using reusable components. This means that the window size, message placement, and pop-up behavior are defined directly in the script. Like each delay (`time.sleep()` values), it is manually set. This limits flexibility because changes require editing the script instead of using configurable components or a modular layout system.

Future Work

Prioritized roadmap items for future iterations include adding more realistic behavior scenarios, improving modular design by splitting components into separate scripts, and expanding compatibility for macOS and Linux. Additions may look like face encryption or keylogging demo.

(10) References

- Chandrashekar, N. D. (2024). *Understanding User Behavior for Enhancing Cybersecurity Training with Immersive Gamified Platforms*. MDPI. DOI: <https://doi.org/10.3390/info15120814>
- Chung, M.H., et al. (2023). *Enhancing cybersecurity situation awareness through visualization: A USB data exfiltration case study*. National Library of Medicine. DOI: <https://doi.org/10.1016/j.heliyon.2023.e13025>
- Kaabouch, N., & Salahdine, F. (2019). *Social Engineering Attacks: A Survey*. MDPI. DOI:10.3390/fi11040089.
- Koffi, K. A. (2024). *To (US)Be or Not to (US)Be: Discovering Malicious USB Peripherals through Neural Network-Driven Power Analysis*. MDPI. DOI: <https://doi.org/10.3390/electronics13112117>
- Srivastava, S., & Verma, H. C. (2024). *The Role of Human Factors in Cybersecurity Vulnerabilities*. ResearchGate. DOI: 10.4018/979-8-3693-5.ch001
- Tischer, M., et al. (2016). *Users Really Do Plug in USB Drives They Find*. IEEE Symposium on Security and Privacy. DOI: 10.1109/SP.2016.26.

Appendix A.

Installation Guide (step-by-step)

USB Drop Attack Simulation Installation Guide

This installation guide describes the installation of the USB Drop Attack simulation, explaining how to install the required tools and build the executable (.exe) from the Python script.

1. Install Python

Download and install Python 3.14.0 or latest version from the official Python website. During installation, make sure that the checkbox “Add Python to PATH” is selected. Complete the setup using the default installation settings.



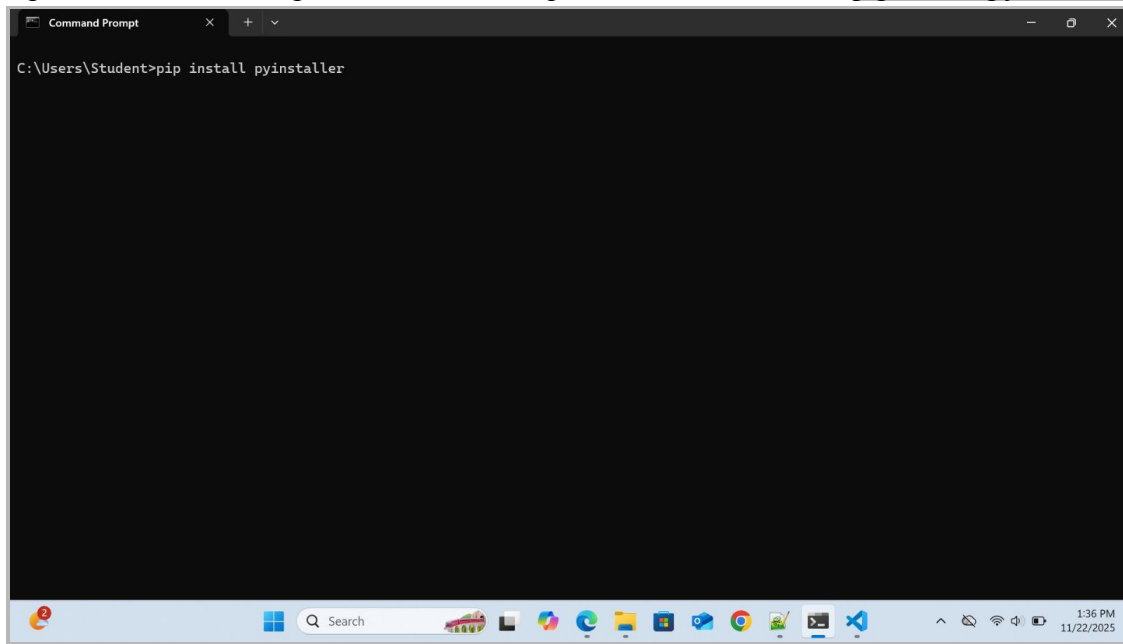
During installation, make sure that the checkbox “Add Python to PATH” is selected. Complete the setup using the default installation settings.

2. Install Required Python Module PyInstaller

PyInstaller is used to convert the Python script into a Windows executable. Install it using:

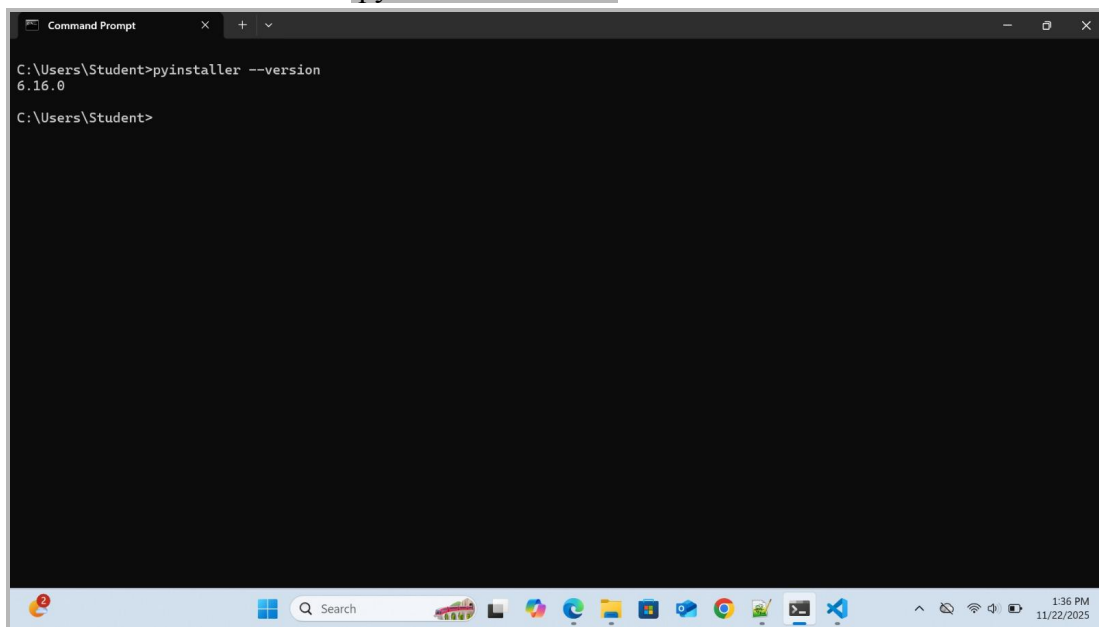
```
pip install pyinstaller
```

Open Command Prompt and install the required external module: `pip install pyinstaller`



```
Command Prompt
C:\Users\Student>pip install pyinstaller
```

Confirm it is installed with `pyinstaller --version`



```
Command Prompt
C:\Users\Student>pyinstaller --version
6.16.0
C:\Users\Student>
```

Step 3: Before converting the script, run it in a Python environment to confirm it executes correctly without errors in Visual Studio Code.

You should see the fake terminal output beginning and the Tkinter popup appearing.

```

1 import time
2 import os
3 import tkinter as tk
4 from tkinter import messagebox
5
6 # =====
7 # Fake USB Payload Simulation and Educational Warning.
8 # This script simulates the execution of a malicious USB payload.
9 # It displays a terminal animation followed by a security warning pop-up.
10 # Created for educational purposes only.
11 # =====
12
13 # --Function purpose: Simulate a fake USB payload execution in the terminal.--
14 def payload_simulation():
15
16     # Clear the screen for cleaner look and feel
17     os.system('cls' if os.name == 'nt' else 'clear')
18
19     print("\n...USB Device Detected...")
20     time.sleep(1)
21
22     print("\n...Initializing payload...")
23     time.sleep(1.2)
24
25     # Fake 'payload stages'
26     stages = [
27         "Initializing payload",
28         "Injecting runtime processes: [#####] 100%",
29         "Retrieving stored credentials: [#####] 100%",
30         "Deploying background services: [#####] 100%",
31         "Triggering final execution phase: [#####] 100%",
32         "...Suspicious activity detected.",
33         "...Displaying educational safety message..."
34     ]
35
36     for stage in stages:
37         print(stage)
38         time.sleep(0.5)
39
40     # Display a security warning message box
41     messagebox.showwarning("Security Warning", "A simulated USB payload was detected. This is a security demonstration. Please do not execute unknown files from USB drives.")
42
43 # Run the simulation
44 payload_simulation()

```

Step 4: Convert PythonScript to Executable (.exe)

Navigate to the directory where your Python script is stored and run the following command:

```
pyinstaller --onefile --windowed CNT4403_AnswerKey.py
```

--onfile bundles everything into a single executable

--windowed prevents a secondary console window if not wanted

Pyinstaller will then generate two folders: build/ and dist/.

The final executable will be located inside the dist/ folder.

```

Command Prompt
\site-packages\PyInstaller\hooks\__hooks__
12673 INFO: Creating base library.zip...
12724 INFO: Looking for dynamic libraries
13024 INFO: Extra DLL search directories (AddDllDirectory): []
13024 INFO: Extra DLL search directories (PATH): []
13306 INFO: Warnings written to C:\Users\Student\build\CNT4403_AnswerKey\warn-CNT4403_AnswerKey.txt
13399 INFO: Graph cross-reference written to C:\Users\Student\build\CNT4403_AnswerKey\xref-CNT4403_AnswerKey.html
13555 INFO: checking PYZ
13556 INFO: Building PYZ because PYZ-00.toc is non existent
13744 INFO: Building PYZ (ZlibArchive) C:\Users\Student\build\CNT4403_AnswerKey\PYZ-00.pyz
13796 INFO: Building PYZ (ZlibArchive) C:\Users\Student\build\CNT4403_AnswerKey\PYZ-00.pyz completed successfully.
13796 INFO: checking PKG
13796 INFO: Building PKG because PKG-00.toc is non existent
13796 INFO: Building PKG (CArchive) CNT4403_AnswerKey.pkg
16352 INFO: Building PKG (CArchive) CNT4403_AnswerKey.pkg completed successfully.
16403 INFO: Bootloader C:\Users\Student\AppData\Local\Programs\Python\Python314\Lib\site-packages\PyInstaller\bootloader\Windows-64bit-intel\run.exe
16403 INFO: checking EXE
16404 INFO: Building EXE because EXE-00.toc is non existent
16404 INFO: Building EXE from EXE-00.toc
16404 INFO: Copying bootloader EXE to C:\Users\Student\dist\CNT4403_AnswerKey.exe
16452 INFO: Copying icon to EXE
16477 INFO: Copying 0 resources to EXE
16477 INFO: Embedding manifest in EXE
16506 INFO: Appending PKG archive to EXE
16617 INFO: Fixing EXE headers
17294 INFO: Building EXE from EXE-00.toc completed successfully.
17342 INFO: Build complete! The results are available in: C:\Users\Student\dist
C:\Users\Student>

```

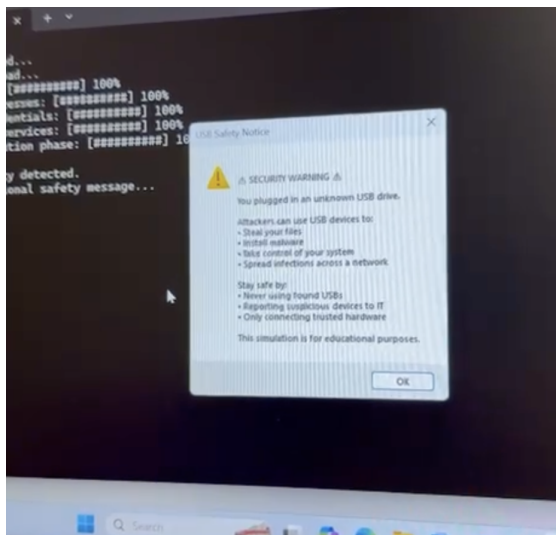
Terminal will show “Build complete!” when the executable is ready.

Step 5: Test the Executable

After the PyInstaller finishes, open the dist/ folder and double -click the generated .exe to test the application.

Verify

- Fake terminal simulation works
- GUI warning pop-up displays
- Program closes cleanly



Step 6: Distribute the Executable

The .exe file can now be dragged into a USB drive and shared with users.

For any questions please contact the Product Manager.

Appendix B.

User Manual

Overview and Roles

The USB Drop Attack Simulator is a standalone Windows executable placed on a USB drive.

When a user double-clicks the file, it:

1. Opens a console window with a fake “loading” sequence.
2. Displays a warning popup explaining the risks of plugging in unknown USB devices.
3. Exits cleanly after the user closes the popup.

There are two main user roles:

- **Instructor** – prepares USB drives, deploys them, and runs the exercise.
- **Participant (Student / User)** – encounters the USB and runs the simulator.

This manual is organized by tasks those roles need to perform.

Task 1 – Preparing a USB Drive (Instructor)

Goal: Get one or more USB drives ready with the latest simulator executable.

Prerequisites

- Windows computer with USB ports.
- The latest version of the simulator executable.
- One or more USB flash drives.

Steps

1. Insert the USB drive into your computer.
2. Copy the simulator executable:
 - Locate the .exe file on your computer.
 - Drag and drop or copy/paste it onto the USB drive.
3. (Optional) Rename the executable to match the exercise theme
4. (Optional) Physically label the USB drive to increase realism (e.g., “CNT4403_Fall2025”).
5. Safely eject the USB drive:
 - In the taskbar, click Safely Remove Hardware → select the USB drive → wait for confirmation → unplug it.

Your USB is now ready for the simulation.

Task 2 – Running the Simulator (Participant)

Goal: Experience what happens when the USB simulator is executed.

Prerequisites

- A Windows machine.
- A prepared USB drive containing the simulator.

Steps

1. Plug the USB drive into a free USB port.
2. Open File Explorer, then click on the USB drive in the left sidebar.

3. Locate the simulator executable.
4. Double-click the file “CNT4403_AnswerKey”:
 - A console window should open and start showing messages such as “Opening file...”, “Analyzing embedded data...”
 - A “loading bar” will progress as the simulation runs.
5. After the loading sequence, a warning popup will appear:
 - Read the message describing the risks of plugging in unknown USB devices.
 - Click OK to dismiss the warning.
6. The program will exit automatically:
 - The console window closes.
 - No further action is required.

Afterwards:

You can safely remove the USB drive. The simulator does not install anything, does not save files, and does not change system settings.

Appendix C.

Full Code Structure

```
import time
import os
import tkinter as tk
from tkinter import messagebox

# =====
# Fake USB Payload Simulation and Educational Warning.
# This script simulates the execution of a malicious USB payload.
# It displays a terminal animation followed by a security warning pop-up.
# Created for educational purposes only.
# =====

# --Function purpose: Simulate a fake USB payload execution in the terminal.--
def payload_simulation():

    # Clear the screen for cleaner look and feel
    os.system('cls' if os.name == 'nt' else 'clear')

    print("\n...USB Device Detected...")
    time.sleep(1)

    print("...Initializing payload...")
    time.sleep(1.2)

    # Fake 'payload stages'
    stages = [
        "Initializing payload",
        "Injecting runtime processes",
        "Retrieving stored credentials",
        "Deploying background services",
        "Triggering final execution phase",
    ]

    # Displays a loading bar for each stage
    for stage in stages:
        for i in range(1, 11):
            bar = "#" * i + "-" * (10 - i)
```

```

        print(f"\r{stage}: [{bar}] {i*10}%", end="")
        time.sleep(0.15)
    print() # New line after each stage is printed

    # Final 2 alert message in terminal before pop-up
    time.sleep(0.7)
    print("\n...Suspicious activity detected.")
    time.sleep(0.7)

    print("...Displaying educational safety message...")
    time.sleep(1.5)

# --Function purpose: Show a warning pop-up about USB security risks.--
def show_warning_popup():
    #Creates multi-line warning message
    message = """
    ⚠ SECURITY WARNING ⚠

    You plugged in an unknown USB drive.

    Attackers can use USB devices to:
    • Steal your files
    • Install malware
    • Take control of your system
    • Spread infections across a network

    Stay safe by:
    • Never using found USBs
    • Reporting suspicious devices to IT
    • Only connecting trusted hardware

    This simulation is for educational purposes.
    """

    # Initialize Tkinter root and show warning message box
    root = tk.Tk()
    root.withdraw()

    # Show warning message box
    messagebox.showwarning("USB Safety Notice", message)

```

```
# --Function purpose: Main function to run the payload simulation and show the warning
pop-up.--
def main():
    payload_simulation()
    show_warning_popup()

# Script entry point
if __name__ == "__main__":
    main()

#
=====
=
# End of CNT4403_Answerkey.py
#
=====
=
```

Appendix E.

Poster, Slides, Video Links

Poster



Knight Foundation School of Computing and Information Sciences

Fall 2025 Senior Design Project

USB Drop Attack Simulation

Students: Barbara Andino, Amelia DiGennaro, Damian Figueroa-Gonzalez, Moath Hussein
Instructor: Dr. Masoud Sajjadi, Florida International University

PROBLEM & MOTIVATION

USB drop attacks remain one of the most effective social engineering methods. Curious users often plug unknown USB drives into work or personal computers without understanding the risks. Traditional awareness training is passive, text-heavy, and often ignored. Organizations need an interactive and realistic, yet also safe, way to demonstrate how quickly malware can execute once a user opens a suspicious file.



Figure 1: USB dropped in a public space, waiting to be plugged in to victim's personal device.

OUR MISSION

Our goal is to create a harmless, educational USB-drop simulation that mimics the behavior of a malicious payload. When executed, it simulates malware-like activity and then educates the user on safe USB practices. This interactive experience raises awareness, reduces risky behavior, and supports cybersecurity training.

Category	Drives Opened	F
Drive Type		
Flash Drive	26784 (100%)	0.72
Hard Drive	10000 (100%)	0.72
Removable	10000 (100%)	0.47
Other	17192 (100%)	0.36
Total	53976 (100%)	-
Location Type		
Academic Room	25768 (100%)	0.38
Common Room	26400 (100%)	0.36
Hallway	24736 (100%)	0.23
Library	25000 (100%)	0.76
Parking Lot	25000 (100%)	0.76
Location Geography		
North	46976 (100%)	0.76
South	46976 (100%)	0.36
East	46976 (100%)	0.76
West	46976 (100%)	0.76
Time of Day		
Morning	71192 (100%)	0.76
Afternoon	64736 (100%)	0.72
Day of Week		
Monday	58147 (100%)	0.88
Tuesday	4192 (100%)	0.72
Wednesday	71192 (100%)	0.76

Note: Data from "Users Really Do Plug in USB Drives They Find," by Bursstein, E., et al. (2016)

SYSTEM OVERVIEW

The system guides the user through a safe and step-by-step demonstration of what a real USB attack could look like.

- When the user opens the file on the USB drive, a simulated malware sequence begins.
- Fake system scans and warning messages appear.
- A terminal-like window displays scripted activity to mimic an infection.
- The simulation ends with an educational message explaining the risk and providing best practices for USB handling.



SYSTEM DESIGN

The simulation is built as a standalone executable that runs locally and does not require an internet connection.

Components:

- Simulation Engine:** Manages the scripted sequence of events.
- User Interface Layer:** Built using Python's GUI and terminal-emulation libraries to generate pop-ups, warnings, and terminal-style animations.
- Executable Packaging Layer:** The single Python script is converted into a Windows-compatible standalone executable using PyInstaller, enabling one-click execution without requiring Python installed on the host machine.

This modular design within a single script ensures a simple, portable solution while allowing future enhancements to be integrated easily.

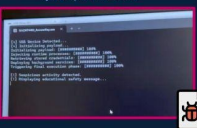


Figure 3: "Physical" screen in the simulation.

PAYLOAD SIMULATION

The payload replicates the "look and feel" of common malware behaviors without performing any harmful actions.

Our features include:

- Fake system scan
- Artificial loading sequences
- Terminal-style logs
- Warning pop-up
- Progress window
- Final awareness message explaining the risks

Together, this creates a realistic but controlled learning experience.

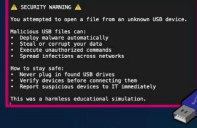


Figure 4: User interface showing the warning message.

IMPLEMENTATION

The executable was developed in Python as a single script, using modules to manage the user interface and pop-up messages.

Key elements of development included:

- GUI creation using Tkinter
- Timed terminal-style output
- Conversion of our single Python script into a standalone .exe file using PyInstaller
- Organized development practices, including daily sprints and sprint-based workflows

This setup ensures the simulation program runs smoothly, providing users with a clear and educational experience.



Figure 5: Python code snippet controlling the educational pop-up simulation.

TESTING & EVALUATION

Our testing focused on reliability, user experience, and safety.

To implement this we included:

- Functional Testing:** Verified each simulation step executes correctly.
- Safety Testing:** Ensured no real system changes occur.
- User Testing:** Observed participants' reactions and gathered feedback.

We concluded that the payload effectively conveys the risks associated with unknown USB drives, and users found the simulation realistic and informative.

SUMMARY

- This project provides a hands-on, engaging method to teach users about USB-related risks.
- It is interactive, realistic, and prioritizes a user-focused solution.
- It provides a clean and engaging experience for individuals who may unknowingly plug in unknown USB devices.
- The harmless simulation encourages individuals to think critically before interacting.
- This structure allows for future enhancements to be incorporated and ensures the tool remains easy to maintain.

REFERENCES

- Bursstein, E., et al. (2016). *Users Really Do Plug in USB Drives They Find*. IEEE Symposium on Security and Privacy. DOI: 10.1109/SP.2016.26.
- Cento, G., et al. (2023). Enhancing cybersecurity situation awareness through visualization: A USB data exfiltration case study. National Library of Medicine. DOI: 10.1016/j.heliyon.2023.e13925.
- Khaloufi, N., & Salahine, F. (2019). *Social Engineering Attacks: A Survey*. MDPI. DOI:10.3390/11040089.
- Shrivastava, S., & Verma, H. C. (2024). *The Role of Human Factors in Cybersecurity Vulnerabilities*. ResearchGate. DOI: 10.4018/979-8-3693-5.ch001






ACKNOWLEDGEMENT

We thank Dr. Masoud Sajjadi for his assistance and mentorship that we received throughout the senior design project.

FIU
FLORIDA INTERNATIONAL UNIVERSITY

Slide Links

[PresentationSlides](#)

Video Links

[IntroVideo](#)

[DemoVideo](#)

Appendix E.

Scrum Evidence Index

▣ USB Drop Attack - Scrum Evidence Index