**REDCELL SIMULATOR – CASE STUDY REPORT**

### **1. Introduction**

The **RedCell Simulator** is an educational web-based malware simulation system designed to teach users how different types of malicious software—**Viruses**, **Worms**, and **Trojans**—spread and behave within computer networks.  
Developed as a web project and powered by **Python** and **JavaScript**, this simulator provides both theoretical and visual demonstrations of malware behavior in a safe, controlled environment.

The website is hosted at: [**https://shreyshah110.github.io/shreyshah2/**](https://shreyshah110.github.io/shreyshah2/?utm_source=chatgpt.com)

The main purpose of this project is to help students, cybersecurity beginners, and educators understand the mechanisms of malware infection and prevention using simple simulations.

### **2. Objectives**

The primary objectives of the RedCell Simulator are:

* To provide a visual and interactive understanding of how malware spreads.
* To demonstrate differences between Virus, Worm, and Trojan behaviors.
* To promote cybersecurity awareness and preventive strategies.
* To implement Python-based simulation logic that tracks infection metrics.
* To generate analytical results and allow users to interpret data visually.

### **3. Project Overview**

The project consists of an interactive dashboard that allows users to:

1. Select a malware type (Virus, Worm, or Trojan).
2. Configure simulation parameters such as number of nodes, infection rate, and protection rate.
3. Observe infection spread visually and numerically.
4. Generate and view analytical reports.

The simulator has multiple web pages:

* **Home Page:** Introduces the concept and simulator purpose.
* **Simulation Page:** Runs live infection models with animation.
* **Dashboard Page:** Shows metrics and infection statistics.
* **Report Page:** Provides detailed summaries and export options.

### **4. Tools and Technologies Used**

| **Category** | **Tools / Technologies** |
| --- | --- |
| **Frontend** | HTML, CSS, JavaScript |
| **Backend / Simulation Engine** | Python |
| **Libraries Used** | Matplotlib, Pandas, ReportLab (for reporting), Chart.js (for web analytics) |
| **Hosting Platform** | GitHub Pages |
| **IDE Used** | Visual Studio Code |
| **Version Control** | GitHub Repository |

### **5. Working Principle**

The simulator works on an **infection-propagation model**. Each node in the network represents a computer system.  
The simulation proceeds through discrete time steps (called “ticks”). During each tick, infection spreads from infected nodes to clean nodes depending on the malware type and infection probability.

The **Python engine** handles:

* Simulation of infection logic.
* Data logging of infection count per tick.
* Plot generation for visualization.
* Exporting results as JSON, CSV, or PDF.

The **Frontend (JavaScript)** visualizes these results dynamically in a web browser

### **6. Malware Types and Their Simulation Logic**

#### **6.1 Virus Simulation**

**Concept:**  
A virus attaches itself to legitimate files or programs. It requires user interaction—like executing an infected file—to spread.

**Python Logic (simplified):**

for node in nodes:

if random() < p\_file\_transfer:

target = choose\_random\_node()

if node.infected and random() < p\_execution:

target.infected = True

**Behavior:**

* Slow propagation depending on user actions.
* Can be controlled with antivirus or restricted sharing.
* Demonstrates real-world dependency on user execution.

#### **6.2 Worm Simulation**

**Concept:**  
A worm is a self-replicating malware that spreads automatically through network connections.  
It does not need user action.

**Python Logic (simplified):**

for node in nodes:

if node.infected:

for \_ in range(scan\_rate):

target = choose\_connected\_node(node)

if not target.infected and random() < p\_success:

target.infected = True

**Behavior:**

* Rapid exponential growth.
* Demonstrates how network worms spread in seconds.
* Best suited for teaching fast-propagating network threats.

#### **6.3 Trojan Simulation**

**Concept:**  
A Trojan disguises itself as a legitimate application. It relies on users to install it voluntarily.  
Once installed, it may open a backdoor for other malware.

**Python Logic (simplified):**

for node in nodes:

if not node.infected and random() < p\_installation:

node.infected = True

if node.infected and random() < p\_backdoor\_activation:

attempt\_infect\_neighbors(node)

**Behavior:**

* Slow initial infection.
* Secondary stage activates after installation (backdoor).
* Demonstrates deception-based infection mechanisms.

### **7. System Architecture**

**Frontend:**

* User Interface for selecting malware type and parameters.
* Visualization using HTML5, CSS3, and JavaScript animations.
* Charts and infection graphs powered by Chart.js.

**Backend (Python Engine):**

* Handles the main infection simulation logic.
* Generates time-series data for infections, protections, and clean nodes.
* Produces visual analytics using Matplotlib and exports results as .pdf reports.

**Hosting:**

* Entire project is hosted on GitHub Pages for free public access.

### **8. Data and Analytics**

The simulator tracks and displays:

* **Total Nodes**
* **Infected Nodes**
* **Protected Nodes**
* **Clean Nodes**
* **New Infections Per Tick**
* **Infection Rate Over Time**

These metrics help users visualize how malware evolves in different conditions.

### **9. Sample Results**

**Scenario A – Worm with High Scan Rate**

* Nodes: 1000
* Initial infected: 1
* Scan Rate: 20
* Result: Infected nodes double rapidly within few ticks.

**Scenario B – Virus with Low Execution Probability**

* Nodes: 500
* File Transfer Probability: 0.05
* Execution Probability: 0.02
* Result: Very slow infection growth, plateau reached early.

**Scenario C – Trojan with Backdoor Activation**

* Nodes: 200
* Installation Probability: 0.01
* Backdoor Activation: 0.8
* Result: Gradual infection rise, followed by sudden network compromise.

### **10. Screenshots Description**

1. **Home Page:** Displays project title, navigation links, and brief introduction.
2. **Dashboard:** Shows parameter configuration for each malware type.
3. **Simulation Page:** Displays live infection graph and counters.
4. **Report Page:** Summarizes simulation results with download options.

### **11. Security and Ethical Considerations**

* The simulator does **not execute any real malware code**; it purely models infection behavior.
* Used strictly for **educational and research purposes**.
* Promotes cybersecurity awareness and defensive strategies.
* Ensures that all code runs safely in isolated environments.

### **12. Conclusion**

The **RedCell Simulator** provides an engaging and educational experience to understand the fundamentals of malware propagation.  
It successfully demonstrates how different types of malware behave and spread, using both **Python-based simulations** and **interactive web animations**.

Through this project, students can:

* Observe malware growth in real-time.
* Analyze infection patterns.
* Learn about preventive cybersecurity measures.

This project can be further enhanced with:

* 3D visualization using Three.js.
* Additional malware categories like ransomware and spyware.
* Cloud-hosted database for tracking multiple simulation runs.

### **13. Future Enhancements**

* Implement Flask or FastAPI backend for live Python simulations.
* Add login and report saving features.
* Integrate AI-based anomaly detection for automated prevention analysis.
* Include a multiplayer learning mode for classroom activities.