## Project: Ransonware

**Group 3:** Ahmed Hussain Sameed & Fnu Abdullah

## Actions:

Language: Python  
Cryptographic library: pyca/cryptography

Cipher: AES-256

Key Management Scheme: Passwords are generated server-side by a Supabase Edge Function and stored in the Supabase database. Supabase service credentials remain hidden and never reach the client. The client only communicates with the Edge Function using a short-lived shared key, exported and unset immediately on the local machine. All direct anon access to the database is disabled via RLS, so reads and writes are strictly controlled by the server logic. Temporary setup files are removed after execution to minimize exposure.

## Infection:

Rubber Ducky: We will be using a webserver to host our script; github will suffice. We will write a DuckyScript to deliver the payload. Convert the DuckyScript into an inject.bin file and put it on the RubberDucky SD card.  
(What if the host does not have a Python interpreter?)

## Monitoring:

Elastic Auditbeat: It communicates with the Linux Audit framework and collects audit events and shows them on a Dashboard- Kibana, like logs. Here, we can also implement our own rules.

We can define the path to the sensitive file   
 rules: |

-w /home/$USER/Desktop/personal\_Fa0337/secrets.txt -p r -k sensitive\_watch

## Detection:

Scope: /Desktop/csce5550/

Legitimate READ access

* Who: the directory owner (local, non-admin session).
* Where: local console on the host (no SSH/RDP/containers)
* Tools: /usr/bin/vim, /usr/bin/nano, /usr/bin/less

Legitimate WRITE access

* Who: directory owner (local, non-admin session).
* Where: local console only.

Everything else is unauthorized:

* Any access by other users or by root outside windows.
* Any access from remote sessions or containers.
* Any process not in the tools list

Advantage:

* Blocks automated abuse paths. By not allowing automation (scripts, agents) , we are preventing bulk reads/exfiltration. Access to the directory must be manual and deliberate.
* Enforcing owner only access improves traceability of an event.
* Mitigates HID attacks (e.g., Rubber Ducky): According to the policy, only a human at the local console may read the files in the directory. Keystroke-injection devices and scripted inputs fall outside the scope and are treated as unauthorized, which will throw alerts.
* Only the owner, from the console, can modify the directory, greatly reducing risk of tampering.

Disadvantages:

* Remote-assist tools often present as a local session, so local console only rules can be bypassed unless we explicitly detect those processes.
* If an attacker planted a symlink inside csce5550/ before we locked it down, then any allowed reader (e.g., less, vim, nano) that opens that entry will follow the link and read the target file.
* Network egress: Even if we tightly control who can open the file, once it’s open, nothing stops that program (or a plugin/child process) from sending the contents out.

## Mitigation:

Using the trigger from our detector, we can get the PID and freeze the process to capture the evidence using sudo kill -STOP <PID>. TO log the attacking event,we can use the following commands:  
sudo sh -c "ps -p $PID -o pid,ppid,uid,gid,cmd > $DIR/ps.txt". This will capture who/what the process is- PID, parent PID, user IDs, and the full command line.  
We can also use other commands in conjunction with the above to get more information about the attack.

To log the attacks on our Elastic Dashboard, we can use Filebeat, which forwards our operator messages (e.g., “froze and killed PID 5636; more info here”) from the system log to Elasticsearch.

To classify the attack, we can:

* Store identity in form of absolute path + SHA.
* Origin: parent process (ppid), terminal/TTY vs remote, container cgroup, working directory.
* Configure policy breach reason: not on tools list, remote session, root outside window, from /tmp, followed symlink, etc. and classify each attack as such based on origin data.

**Backups:**

For backups, we can follow the 3-2-1 principle. Keep three copies of the /csce5550/ data, on two different kinds of storage, with at least one copy offline. Use a tool such as restic to write to an external disk, encrypt the backups client-side and keep the key-encryption key somewhere secure and recoverable, and scope the job to include the directory.

**Recovery:**

Restore the needed files from the last known-good backup to a clean staging path, so we can verify it. We can validate the staged files by file counts and sizes and check if they match the backup manifest. We can also keep checksums and verify them. We can then reinstate the files to the original location and enable monitoring.