Project: Remote Anonymized Security Audit Tool (RASAT)

1. Executive Summary

This Bash-based security framework is designed to perform automated, anonymous, and remote network reconnaissance. The tool allows a local auditor to connect to a remote server via SSH, utilize that server as a pivot point to scan targets (Single IP or entire Subnets), and securely retrieve the data.

Crucially, the tool enforces operational security (OpSec) by routing local traffic through the Tor network using **NIPE**, ensuring the auditor's location remains obfuscated during the setup and control phases.

2. Key Features & Implementation

🛡️ Anonymity & OpSec (NIPE Integration)

The script routes traffic through the Tor network and includes a "Geo-Location Kill Switch." It verifies the Tor exit node's location and refuses to proceed if the exit IP is located in the auditor's home country (specifically checking against "IL"), preventing accidental exposure.

Code Snippet:

```
# Verify anonymity: Ensure exit node is NOT Israel (IL)
RESULT=$(curl -s --max-time 8 https://wtfismyip.com/json)
```

```
COUNTRY=$(echo "$RESULT" | jq -r '.YourFuckingCountryCode // empty')

if [[ "$COUNTRY" != "IL" ]]; then
    echo -e "${GREEN}[NIPE] Anonymity verified.${RESET}"
    log "Anonymity achieved — Exit IP: $EXIT_IP Country: $COUNTRY"
    return 0
else
    echo -e "${RED}[NIPE] Still IL exit. Retrying...${RESET}"
fi
```

📡 Remote Execution Architecture

Instead of scanning from the local machine, the script uses `sshpass` to authenticate non-interactively and execute commands *on the remote server*. This minimizes bandwidth usage and keeps the scanning traffic local to the target network.

Code Snippet:

```
# Execute commands remotely without user interaction
# -o StrictHostKeyChecking=no: Prevents "yes/no" prompts
sshpass -p "$SSH_PASS" ssh -o StrictHostKeyChecking=no -p "$REMOTE
_PORT" \
    "$REMOTE_USER@$REMOTE_HOST" "command_to_run"
```

🔍 Intelligent Scanning Engine (Hybrid Workflow)

The scanner uses a "Fast then Deep" logic to optimize speed.

1. **Fast Scan:** Runs a high-speed syn scan (`p-`, `-min-rate 300`) to find open ports.
2. **Extraction:** Parses the output to get a clean list of open ports.
3. **Deep Scan:** Runs vulnerability scripts (`-script vuln`) *only* on the active ports.

Code Snippet:

```
# 1. FAST SCAN: High rate, limited retries
nmap -Pn -p- -T3 --max-retries 1 --min-rate 300 ... -oG output.gnmap $TA
RGET

# 2. EXTRACT PORTS: Regex parse open ports
```

```
OPEN_PORTS=$(grep -oP '\d+(?=/open)' output.gnmap | paste -sd, -)

# 3. DEEP SCAN: Targeted vulnerability analysis
nmap -Pn -sV --script vuln -p $OPEN_PORTS -T3 -oA deep_scan $TARGET
```

💾 Fail-Safe Data Recovery

The tool features a robust `cleanup()` function hooked to system signals (`SIGINT`, `ERR`). If the script is interrupted (e.g., Ctrl+C), it doesn't just crash; it connects to the remote server, attempts to download whatever data has been gathered so far, and *then* cleans up traces.

Code Snippet:

```
# Trap signals to trigger the cleanup function
trap cleanup SIGINT
trap cleanup ERR

cleanup() {
    log "Interrupt detected — inspecting remote directory..."

    # Attempt SCP recovery of partial data before deletion
    sshpass -p "$SSH_PASS" scp -r ... "$REMOTE_USER@$REMOTE_HOS
T:$REMOTE_DIR" "$LOCAL_SAVE/"

    # Securely remove remote evidence
    sshpass ... "rm -rf $REMOTE_DIR"
}
```

📊 Automated Reporting

The script parses the raw Nmap output files to generate a readable Markdown report, highlighting discovered services and specific vulnerabilities detected by the Nmap Scripting Engine (NSE).

Code Snippet:

```
# Parse Nmap results to find Vulnerability blocks
vuln_block=$(awk '
    /VULNERABLE/ {in_v=1}
```

```
    /VULNERABLE/ {print; next}
    in_v {
        print
        if ($0 ~ /^Host:|^PORT/) in_v=0
    }
' "$NMAP_FILE")
```

3. System Requirements

- **OS:** Kali Linux (or Debian-based distros)
- **Privileges:** Root (Sudo)
- **Dependencies:** Perl, Nipe, Tor, SSHPass, Nmap, JQ, Git.