John’s Ethical Hacking Learning Lab

Compiled PDF: repository contents for upload to GitHub

Important: This repository is for ethical, educational purposes only. Do not use these materials to attack systems without explicit permission.

README.md

```
# John’s Ethical Hacking Learning Lab

**Purpose:**
This repository is a learning playground for ethical cybersecurity: lab setup instructions, safe exercises, and

> ■■ **Important:** Never use tools or techniques from this repository to attack systems without explicit autho

---

## Structure

- `README.md` — this file
- `scripts/` — safe, educational scripts (password checker, simulated log monitor)
- `lab/` — instructions to set up a local VM lab (Kali, Metasploitable or intentionally vulnerable apps)
- `resources.md` — links to learning platforms, books, and CTFs
- `LICENSE` — suggested open-source license (MIT)

---

## Quick start — Local learning lab (recommended)

1. Install virtualization: VirtualBox or VMware.
2. Create two VMs:
   - **Attacker VM**: Kali Linux (for learning tools).
   - **Target VM**: Metasploitable 2 / OWASP Juice Shop / DVWA (intentionally vulnerable).
3. Network: Use Host-only or Internal Network so the lab is isolated from the internet.
4. Follow safe practices: snapshots before exercises; disconnect host from sensitive networks.

---

## Safe scripts (in `scripts/`)

This repo includes only **educational / defensive** scripts:
- `password_strength_checker.py` — checks password strength locally.
- `simulated_log_monitor.py` — example of scanning a log file for suspicious patterns (works on sample logs onl

These illustrate programming concepts useful in security work (parsing logs, hashing, validation), not exploits

---

## Learning path & resources

- Beginner: Python for security, Linux basics, networking (TCP/IP), Bash scripting.
- Intermediate: Web app security (OWASP Top 10), SQLi/XSS basics in lab, basic reverse engineering.
- Advanced: Binary exploitation (in controlled CTFs), threat modeling, defensive engineering.

Recommended platforms:
- TryHackMe (labs and guided paths)
- Hack The Box (practical CTFs)
- OWASP Juice Shop (web security playground)
- CTFtime (CTF events and challenges)

Books:
- *The Web Application Hacker's Handbook* — for web security concepts
- *Practical Malware Analysis* — (study defensively; do not misuse)

---

## Contributing & ethics

If you add materials, ensure they are for **education** and include safe-use guidance. Use permissive license (

---
```

lab/README_lab_setup.md

```
# Local Lab Setup

This guide explains how to create an isolated local learning lab for ethical security testing.

## Requirements
- VirtualBox or VMware
- Two VMs (Attacker and Target)
- Host-only / Internal network for isolation

## Steps
1. Download and install VirtualBox (https://www.virtualbox.org/) or VMware.
2. Create two VMs:
   - Attacker VM: Kali Linux (download from https://www.kali.org/)
   - Target VM: Metasploitable 2 (intentionally vulnerable), OWASP Juice Shop, or DVWA
3. Configure the network adapters of both VMs to "Host-only Adapter" or "Internal Network" so they can talk to
4. Take snapshots before starting any exercise.
5. Practice only on the target VM. Never scan or attack systems you don't own or have explicit permission to te

## Suggested exercises
- Network scanning with `nmap` (on target VM only)
- Web app testing on OWASP Juice Shop (local browser, local network)
- Log analysis with the provided `simulated_log_monitor.py`
- Password policies and strength checks with `password_strength_checker.py`
```

resources.md

```
# Resources & Learning Links

- TryHackMe — guided security labs and paths
- Hack The Box — practical CTF-style challenges
- OWASP Juice Shop — intentionally vulnerable web app
- CTFtime — calendar and events for CTF competitions
- The Web Application Hacker's Handbook — book for web security
- Practical Malware Analysis — study defensively

Always use these resources responsibly and within the rules of each platform.
```

scripts/password_strength_checker.py

```python
#!/usr/bin/env python3
"""
password_strength_checker.py
Simple, safe, educational password strength checker.
Use locally; does NOT send data anywhere.
"""

import re
import math

def estimate_entropy(password: str) -> float:
    """Very simple entropy estimate (bits)."""
    pool = 0
    if re.search(r'[a-z]', password): pool += 26
    if re.search(r'[A-Z]', password): pool += 26
    if re.search(r'\d', password): pool += 10
    if re.search(r'\W', password): pool += 32  # approx symbols
    if pool == 0:
        return 0.0
    return math.log2(pool) * len(password)

def strength_label(entropy: float) -> str:
    if entropy < 28:
        return "Very Weak"
    if entropy < 36:
        return "Weak"
    if entropy < 60:
        return "Reasonable"
    if entropy < 128:
        return "Strong"
    return "Very Strong"

def check_password(pw: str) -> dict:
    ent = estimate_entropy(pw)
    return {
        "length": len(pw),
        "entropy_bits": round(ent, 2),
        "label": strength_label(ent),
        "has_lower": bool(re.search(r'[a-z]', pw)),
        "has_upper": bool(re.search(r'[A-Z]', pw)),
        "has_digit": bool(re.search(r'\d', pw)),
        "has_symbol": bool(re.search(r'\W', pw)),
    }

def main():
    pw = input("Enter password to evaluate (local only): ").strip()
    if not pw:
        print("No password entered.")
        return
    res = check_password(pw)
    print(f"\nLength: {res['length']}")
    print(f"Estimated entropy (bits): {res['entropy_bits']}")
    print(f"Strength: {res['label']}")
    print("Composition:", ", ".join(k for k,v in res.items() if k.startswith('has_') and v))

if __name__ == "__main__":
    main()
```

scripts/simulated_log_monitor.py

```python
#!/usr/bin/env python3
"""
simulated_log_monitor.py
Scan a provided sample log file for suspicious patterns.
This is educational — do NOT point at real /var/log without permissions.
"""

import re
import argparse

SUSPICIOUS_PATTERNS = [
    re.compile(r'Failed password'),
    re.compile(r'authentication failure'),
    re.compile(r'Invalid user'),
    re.compile(r'error: PAM'),
    re.compile(r'root:.*failed'),
]

def scan_log(path: str):
    with open(path, 'r', encoding='utf-8', errors='ignore') as f:
        for lineno, line in enumerate(f, 1):
            for pat in SUSPICIOUS_PATTERNS:
                if pat.search(line):
                    print(f"{path}:{lineno}: {line.rstrip()}")

def main():
    parser = argparse.ArgumentParser(description="Scan a sample log for suspicious lines.")
    parser.add_argument('logfile', help="Path to a sample log file (text).")
    args = parser.parse_args()
    scan_log(args.logfile)

if __name__ == "__main__":
    main()
```

LICENSE