AI FOR HACKERS

From Zero to AI-Powered Hacking

Machine Learning · Offensive Security · Prompt Injection
Automated Recon · AI-Powered Exploitation · Defensive AI

codelivly.com
2026 Editions

CHAPTER

Table of Contents

Chapter 01 — Introduction to AI for Hackers

- What is Artificial Intelligence?
- Why Hackers Need AI
- The AI Security Landscape
- How to Use This Guide

Chapter 02 — Setting Up Your AI Hacking Lab

- Required Tools & Software
- Python Environment Setup
- Key AI Libraries
- Virtual Lab Configuration

Chapter 03 — Python Crash Course for AI

- Python Basics Review
- NumPy & Pandas Essentials
- Working with Data
- Visualization with Matplotlib

Chapter 04 — Machine Learning Fundamentals

- Types of Machine Learning
- Supervised Learning
- Unsupervised Learning
- Model Evaluation & Metrics

Chapter 05 — Neural Networks & Deep Learning

- How Neural Networks Work
- Activation Functions
- Backpropagation
- Building Your First Neural Net

Chapter 06 — Large Language Models (LLMs)

- How LLMs Work
- GPT Architecture
- Tokenization & Embeddings
- Fine-Tuning Basics

Chapter 07 — Prompt Engineering for Hackers

- Crafting Effective Prompts
- Jailbreaking Techniques
- Prompt Injection Attacks
- Bypassing AI Filters

Chapter 08 — AI-Powered Reconnaissance

- OSINT with AI
- Automated Target Analysis
- Social Engineering with AI
- AI-Driven Footprinting

Chapter 09 — Vulnerability Discovery with AI

- Using AI for Code Auditing
- Automated Bug Finding
- Fuzzing with ML
- CVE Prediction Models

Chapter 10 — Offensive AI Techniques

- AI-Generated Malware
- Evasion Techniques
- Adversarial Examples
- DeepFake Attacks

Chapter 11 — Defensive AI & Detection

- Anomaly Detection
- Intrusion Detection Systems
- AI-Powered SIEM
- Threat Intelligence with AI

Chapter 12 — Real-World Projects & CTF Labs

- Project 1: AI Recon Bot
- Project 2: Password Cracker
- Project 3: AI Phishing Detector
- CTF Challenge Walkthroughs

Chapter 13 — Ethics, Legality & Responsible Disclosure

- Legal Boundaries
- Responsible Disclosure
- Bug Bounty Programs
- AI Ethics in Security

Chapter 14 — Next Steps & Resources

- Advanced Learning Paths
- Recommended Tools
- Communities & Conferences
- Final Words

CHAPTER 01

Introduction to AI for Hackers

What is Artificial Intelligence?

Artificial Intelligence (AI) refers to the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (acquiring information and rules for using it), reasoning (using rules to reach approximate or definite conclusions), and self-correction. In the context of cybersecurity, AI is rapidly becoming the most powerful force multiplier available to both attackers and defenders.

Modern AI is largely driven by a subset called **Machine Learning (ML)**, which enables systems to automatically learn and improve from experience without being explicitly programmed. Within ML, **Deep Learning** uses neural networks with many layers to process data with complex patterns — powers everything from facial recognition to natural language processing.

Key Definition

Artificial Intelligence (AI): A broad field focused on creating intelligent machines. Machine Learning (ML): A subset of AI that learns from data. Deep Learning (DL): A subset of ML using multi-layered neural networks.

The AI Hierarchy

Level	Technology	Example Use Case
Broad	Artificial Intelligence	Game playing, decision systems
Focused	Machine Learning	Spam filtering, anomaly detection
Deep	Deep Learning	Image recognition, NLP
Current	Large Language Models	ChatGPT, code generation, hacking tools

Why Hackers Need AI

The security landscape is undergoing a seismic shift. Traditional attack techniques that took hours or days can now be automated and optimized with AI in minutes. Here's why every serious security professional must understand AI:

- **Automation at Scale:** AI enables automated scanning, exploitation, and reporting across thousands of targets simultaneously.
- **Pattern Recognition:** ML models can identify vulnerabilities and attack patterns that humans would miss entirely.
- **Natural Language Power:** LLMs can generate phishing emails, write malware, craft social engineering scripts, and analyze code.

• **Evasion:** AI can generate polymorphic code and adversarial inputs that bypass traditional security tools.

• **Efficiency:** Tasks that took days — like manual code review — can be completed in minutes with AI assistance.

■ Ethical Note

All techniques in this guide are for educational purposes and authorized testing only. Unauthorized hacking is illegal. Always obtain written permission before testing any system you do not own.

The AI Security Landscape

Security and AI intersect in two major domains: AI being used **offensively** to attack systems, and AI being used **defensively** to protect them. This dual-use nature makes AI one of the most critical topics in modern cybersecurity.

On the offensive side, threat actors are already using AI to craft spear-phishing emails, generate malware variants, automate vulnerability scanning, and conduct large-scale social engineering. On the defensive side, AI powers next-generation intrusion detection, behavioral analysis, threat hunting, and automated incident response.

How to Use This Guide

This guide is structured to take you from absolute beginner to confident AI-powered security practitioner. Each chapter builds on the previous, but you can also jump to specific sections that interest you. Practical code examples, labs, and projects are included throughout.

✓ Read sequentially for the best learning experience

✓ Type out all code examples — don't just copy-paste

✓ Complete the labs at the end of each major section

✓ Join the community at codelivly.com for discussion and updates

✓ Always practice in a controlled, legal environment

CHAPTER 02

Setting Up Your AI Hacking Lab

Required Tools & Software

Building your AI hacking lab requires the right combination of hardware, software, and services. Below is everything you need to get started. Most tools are free and open-source.

Category	Tool/Software	Purpose
OS	Kali Linux / Ubuntu 22.04	Primary hacking OS
Language	Python 3.10+	AI/ML programming
AI Framework	PyTorch / TensorFlow	Deep learning
ML Library	Scikit-learn	Classical ML
NLP	Hugging Face Transformers	LLM interaction
Data	Pandas, NumPy	Data manipulation
Visualization	Matplotlib, Seaborn	Data visualization
API Access	OpenAI API / Anthropic API	LLM access
Notebooks	Jupyter Lab	Interactive development
Containers	Docker	Isolated environments
Version Ctrl	Git + GitHub	Code management

Python Environment Setup

We'll use a virtual environment to keep our AI hacking tools isolated and organized. Follow these steps to set up your environment:

```
# Step 1: Install Python 3.10+
sudo apt update && sudo apt install python3.10 python3-pip -y

# Step 2: Install virtualenv
pip3 install virtualenv

# Step 3: Create your AI hacking environment
virtualenv ai_hacking_env
source ai_hacking_env/bin/activate

# Step 4: Install core AI/ML libraries
pip install numpy pandas matplotlib seaborn scikit-learn
pip install torch torchvision torchaudio
pip install transformers datasets tokenizers
pip install openai anthropic jupyter jupyterlab
pip install requests beautifulsoup4 scapy
```

Key AI Libraries Overview

NumPy: The foundational library for numerical computing in Python. Provides N-dimensional array objects and mathematical functions essential for all ML operations.

Pandas: Data manipulation and analysis library. Essential for loading, cleaning, and transforming datasets before feeding them to ML models.

Scikit-learn: The go-to library for classical machine learning. Contains implementations of hundreds of ML algorithms with consistent APIs.

PyTorch: Facebook's deep learning framework. Preferred by researchers for its dynamic computation graphs and Pythonic design.

Hugging Face Transformers: Access to thousands of pre-trained LLMs including GPT, BERT, LLaMA, and more. Essential for NLP hacking techniques.

OpenAI Python SDK: Official library to interact with GPT-4, DALL-E, and other OpenAI models via API.

Verifying Your Installation

```
# Run this verification script
import numpy as np
import pandas as pd
import sklearn
import torch
import transformers

print(f'NumPy: {np.__version__}')
print(f'Pandas: {pd.__version__}')
print(f'Scikit-learn: {sklearn.__version__}')
print(f'PyTorch: {torch.__version__}')
print(f'Transformers: {transformers.__version__}')
print(f'CUDA available: {torch.cuda.is_available()}')
print('All libraries loaded successfully!')
```

■ Pro Tip

If you have an NVIDIA GPU, install CUDA-enabled PyTorch for dramatically faster training. Visit pytorch.org for the exact install command for your CUDA version. Even without a GPU, you can use Google Colab for free GPU access.

CHAPTER 03

Python Crash Course for AI

Python Basics Review

If you're already comfortable with Python, skim this section. If not, pay close attention — Python mastery is essential for AI work. We focus on the patterns you'll use most in AI and security contexts.

Data Structures You'll Use Daily

```
# Lists - ordered, mutable sequences
targets = ['192.168.1.1', '192.168.1.2', '10.0.0.1']
targets.append('172.16.0.1')

# Dictionaries - key-value pairs (like JSON)
vuln = {'host': '192.168.1.1', 'port': 22, 'cve': 'CVE-2023-1234',
'severity': 'HIGH', 'exploitable': True}

# List comprehensions - powerful one-liners
high_severity = [v for v in vulns if v['severity'] == 'HIGH']

# Sets - unique elements, fast lookup
open_ports = {22, 80, 443, 8080}
critical_ports = {22, 23, 3389}
exposed = open_ports.intersection(critical_ports) # {22}
```

Functions, Classes & Decorators

```
# Functions with type hints (good practice in AI code)
def preprocess_payload(raw: str, max_len: int = 512) -> list:
tokens = raw.lower().split()
return tokens[:max_len]

# Classes for organizing ML pipelines
class VulnClassifier:
def __init__(self, model_path: str):
self.model = self._load_model(model_path)
self.threshold = 0.7

def predict(self, features: list) -> str:
score = self.model.predict_proba([features])[0][1]
return 'VULNERABLE' if score > self.threshold else 'SAFE'

def _load_model(self, path):
import joblib
return joblib.load(path)
```

NumPy & Pandas Essentials

NumPy and Pandas are the backbone of data processing in AI. You will use them constantly to prepare datasets for machine learning models.

```
import numpy as np
import pandas as pd

# NumPy arrays - the foundation of ML
features = np.array([[1, 0, 443, 1], [0, 1, 22, 0], [1, 1, 80, 1]])
labels = np.array([1, 0, 1]) # 1=attack, 0=normal

# Array operations (vectorized = fast)
normalized = (features - features.mean(axis=0)) / features.std(axis=0)

# Pandas DataFrames for structured data
df = pd.read_csv('network_logs.csv')
print(df.head()) # First 5 rows
print(df.describe()) # Statistical summary
print(df.isnull().sum()) # Check for missing values

# Feature engineering
df['is_suspicious'] = (df['bytes_sent'] > 10000) & (df['port'] == 22)
df['hour'] = pd.to_datetime(df['timestamp']).dt.hour

# Select features for model
X = df[['bytes_sent', 'bytes_recv', 'port', 'duration', 'hour']].values
y = df['label'].values
```

Key Concept: Feature Engineering

Feature engineering is the process of transforming raw data into meaningful inputs for ML models. In security, good features might include: packet size, connection duration, time of day, protocol type, and byte ratios. Bad features produce bad models, regardless of algorithm choice.

CHAPTER 04

Machine Learning Fundamentals

Types of Machine Learning

Machine learning algorithms fall into three major paradigms. Understanding which to use is the first critical skill of any AI practitioner:

Type	How It Works	Security Use Cases
Supervised	Learns from labeled examples	Malware classification, phishing detection
Unsupervised	Finds patterns without labels	Anomaly detection, network clustering
Reinforcement	Learns via rewards/penalties	Automated penetration testing, evasion
Semi-supervised	Mix of labeled & unlabeled	Threat intel with limited labels
Self-supervised	Labels from data itself	Log analysis, LLM pre-training

Supervised Learning

Supervised learning is the most commonly used paradigm in security applications. You provide labeled training data, and the algorithm learns a mapping from inputs to outputs. Here's a complete example of a network intrusion classifier:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import pandas as pd, numpy as np

# Load the NSL-KDD intrusion detection dataset
df = pd.read_csv('nsl_kdd_train.csv')

# Feature selection
features = ['duration','src_bytes','dst_bytes','land','wrong_fragment',
'urgent','hot','num_failed_logins','logged_in','num_compromised']
X = df[features].values
y = (df['label'] != 'normal').astype(int).values # 1=attack, 0=normal

# Split data
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42, stratify=y)

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train Random Forest
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred,
target_names=['Normal', 'Attack']))
```

Model Evaluation & Metrics

Choosing the right evaluation metric is critical in security. A 99% accurate model sounds great — but if 1% of attacks slip through, your system is compromised.

Metric	Formula	When to Use
Accuracy	Correct / Total	Balanced classes only
Precision	TP / (TP + FP)	Minimize false alarms
Recall	TP / (TP + FN)	Minimize missed attacks
F1-Score	2 * P*R / (P+R)	Balance precision/recall
AUC-ROC	Area under ROC curve	Threshold-independent eval

Security Focus: Recall vs Precision

In intrusion detection, HIGH RECALL is usually more important than high precision. Missing a real attack (false negative) is more dangerous than a false alarm (false positive). Tune your decision threshold accordingly — don't just use the default 0.5.

CHAPTER 05

Neural Networks & Deep Learning

How Neural Networks Work

Neural networks are inspired by the human brain. They consist of layers of interconnected nodes (neurons) that process information by passing weighted signals through activation functions. The power of deep learning comes from stacking many such layers, allowing the network to learn increasingly abstract representations of data.

Every neural network has three core component types:

- **Input Layer:** Receives raw features (e.g., network packet features)
- **Hidden Layers:** Transform inputs through weighted connections and activation functions
- **Output Layer:** Produces final predictions (e.g., attack probability)

Building Your First Neural Network with PyTorch

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Define a simple intrusion detection neural network
class IntrusionDetector(nn.Module):
def __init__(self, input_dim: int):
super(IntrusionDetector, self).__init__()
self.network = nn.Sequential(
nn.Linear(input_dim, 128),
nn.ReLU(),
nn.Dropout(0.3),
nn.Linear(128, 64),
nn.ReLU(),
nn.Dropout(0.2),
nn.Linear(64, 32),
nn.ReLU(),
nn.Linear(32, 1),
nn.Sigmoid() # Output: attack probability
)

def forward(self, x):
return self.network(x)

# Training loop
model = IntrusionDetector(input_dim=41)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.BCELoss()

for epoch in range(50):
model.train()
for batch_X, batch_y in train_loader:
optimizer.zero_grad()
predictions = model(batch_X).squeeze()
loss = criterion(predictions, batch_y.float())
loss.backward()
optimizer.step()
```

Activation Functions

Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns. Choosing the right activation function matters:

Function	Range	Best Used For
ReLU	[0, inf)	Hidden layers (default choice)
Sigmoid	(0, 1)	Binary classification output
Softmax	(0, 1) summing to 1	Multi-class classification output
Tanh	(-1, 1)	Hidden layers, NLP tasks