Redefining Hacking

A Comprehensive Guide to Red Teaming and Bug Bounty Hunting in an AI-driven World

OMAR SANTOS | SAVANNAH LAZZARA
WESLEY THURNER

REDEFINING HACKING: A COMPREHENSIVE GUIDE TO RED TEAMING AND BUG BOUNTY HUNTING IN AN AI-DRIVEN WORLD

This page intentionally left blank

REDEFINING HACKING: A COMPREHENSIVE GUIDE TO RED TEAMING AND BUG BOUNTY HUNTING IN AN AI-DRIVEN WORLD

Omar Santos, Savannah Lazzara, Wesley Thurner

Addison-Wesley

Boston • Columbus • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sidney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2025930851

ISBN-13: 978-0-13-836361-1
ISBN-10: 0-13-836361-7

$PrintCode

GM K12, Early Career and Professional Learning
Soo Kang

Director, ITP Product Management
Brett Bartow

Executive Editor
James Manly

Development Editor
Christopher Cleveland

Managing Editor
Sandra Schroeder

Senior Project Editor
Mandie Frank

Copy Editor
Chuck Hutchinson

Indexer
Timothy Wright

Proofreader
Donna E. Mulder

Technical Reviewers
Matthew Eidelberg
Petar Radanliev

Editorial Assistant
Cindy Teeters

Designer
Chuti Prasertsith

Compositor
codeMantra

I would like to dedicate this book to my lovely wife, Jeannette, and my two beautiful children, Hannah and Derek, who have inspired and supported me throughout the development of this book.

—Omar

I would like to dedicate this book to my devoted parents, Dannielle and Tom, as well as my amazing partner, Nickolas, who never stopped believing in me. I would also like to honor my loving dogs: Baxter, who brought joy to my life for 13 years and will forever be missed, and Caymus, who is the light of my life.

—Savannah

I would like to dedicate this book to my son, Westin, who put up with me throughout this entire endeavor and never stopped supporting me.

—Wes

This page intentionally left blank

Contents

Register your copy of *Redefining Hacking: A Comprehensive Guide to Red Teaming and Bug Bounty Hunting in an AI-Driven World* at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (**9780138363611**) and click Submit. Once the process is complete, you will find any available bonus content under "Registered Products."

Acknowledgments

Omar Santos:

I would like to thank the technical editors for their time and technical expertise.

I would like to thank the Pearson team, especially James Manly and Christopher Cleveland, for their patience, guidance, and support.

Savannah Lazzara:

I would like to thank the technical editors for their amazing work making this book come to life, as well as my mentors, Travis Weathers, Matthew Eidelberg, and Ben Rollin, who have helped me develop my skill set over the years. Additionally, I can't thank the Pearson team enough, specifically James Manly and Christopher Cleveland for supporting us.

Wesley Thurner:

I would like to thank the technical editors for their expertise, James Manly, Christopher Cleveland, and the Pearson team for bringing this to fruition, their support and understanding.

This page intentionally left blank

About the Authors

Omar Santos is a Distinguished Engineer at Cisco who pioneers advancements in artificial intelligence security, cybersecurity research, ethical hacking, incident response, and vulnerability disclosure. As co-chair of the Coalition for Secure AI (CoSAI) and board member of the OASIS Open standards organization, he shapes the future of secure technology adoption across industries.

Omar drives innovation through multiple leadership roles, including as founder of OpenEoX and co-chair of the Forum of Incident Response and Security Teams (FIRST) PSIRT Special Interest Group. His commitment to cybersecurity education and community building is evident in his role as the co-founder and one of the leaders of the DEF CON Red Team Village and the chair of the Common Security Advisory Framework (CSAF) technical committee.

Omar has published over 20 books, created over 20 video courses, and contributed more than 40 academic research papers to the field. Omar's work in cybersecurity is also recognized through multiple granted patents. You can follow Omar in X at https://x.com/santosomar and LinkedIn at https://www.linkedin.com/in/santosomar.

Savannah Lazzara is a Security Engineer specializing in red teaming at a tech company. Savannah has multiple years of experience in security consulting working with many Fortune 500 corporations and has experience in carrying out security assessments, which include network assessments, social engineering exercises, physical facility penetration tests, and wireless assessments. Savannah also has experience in performing adversary simulation assessments, which include remote red team simulations, insider threat assessments, and onsite red team assessments. Savannah's area of expertise is focused on social engineering and physical security.

Savannah is the co-lead of Red Team Village. Additionally, Savannah has spoken at various cybersecurity conferences, including Source Zero Con, BSides, and others. Savannah has also spoken on multiple podcast shows including *The Hacker Factory* and *Hackerz and Haecksen*.

Wesley Thurner is a Principal Security Engineer on the Intuit Red Team, designing and leading transformative security initiatives across the organization. He accomplishes this through multiple internal roles. Wesley leads the internal Purple Team, is an Ambassador for cybersecurity education throughout the workforce, and leads working groups to define security policies.

Prior to Intuit, Wesley served as an exploitation operator in the US Department of Defense's most elite computer network exploitation (CNE) unit. There he led and developed multiple teams across a variety of roles in the US Air Force's premier selectively manned cyber attack squadron. Wesley is also a co-lead for the Red Team Village, a community-driven village bridging the gap between penetration testers and offensive operations. He has spoken at various cybersecurity conferences and authored numerous training courses as well as white papers.

This page intentionally left blank

About the Technical Reviewers

Matthew Eidelberg graduated from Seneca@York University with a Bachelor of Technology in Informatics and Security. He has more than ten years of experience in the cybersecurity industry, with a primary focus on adversary emulation. His specialty lies in research and development related to exploits and malware—specifically the creation of new techniques and tools for bypassing and evading endpoint security measures. Matthew joined Black Hills Information Security (BHIS) in August 2023. Prior to that, he spent eight years at Optiv, where he led the Adversary Simulation Services team.

Petar Radanliev is a post-doctoral research associate at the Department of Computer Science at the University of Oxford. He obtained his PhD at the University of Wales in 2014. He continued with post-doctoral research at Imperial College London, the University of Cambridge, Massachusetts Institute of Technology, and the Department of Engineering Science at the University of Oxford before moving to the Department of Computer Science. His current research focuses on artificial intelligence, cybersecurity, quantum computing, and blockchain technology. Before joining academia, Petar spent ten years as a cybersecurity manager for RBS (the largest bank in the world at the time) and five years as a lead penetration tester for the Ministry for Defence.

This page intentionally left blank

Introduction

The art and science of red teaming and security testing are at a fascinating crossroads with emerging technologies like artificial intelligence (AI). Gone are the days when cybersecurity was merely about finding buffer overflows or SQL injection vulnerabilities. Today's security professionals face an entirely new frontier: systems that reason, adapt, and learn.

The tools and techniques that served us well for decades are being transformed. AI-powered defense systems now detect and respond to threats in milliseconds, while traditional machine learning and generative AI models analyze patterns of behavior that would take human analysts months to uncover. Yet, paradoxically, this same technological revolution has created new vulnerabilities and attack surfaces that few could have imagined just years ago.

This book will help you gain relevant knowledge at a crucial moment in the evolution of offensive security testing. Many organizations increasingly rely on AI-driven systems to protect their assets. Consequently, the traditional approaches to red teaming, penetration testing, and bug bounty hunting must evolve. Red teamers now need to understand the skills needed for today's environment. Bug bounty hunters must adapt their tools and methodologies to probe systems that can predict and react to their every move.

Whether you're a seasoned red teamer, an aspiring bug bounty hunter, or an individual looking to understand the future of offensive security, this book will serve as your guide through this transformation. We'll explore the cutting-edge techniques being developed at the intersection of AI and security testing, from AI-powered reconnaissance to principles of exploit development. This book is your roadmap to navigating this new terrain, combining time-tested methodologies with emerging technologies to create a comprehensive approach to modern security testing.

Welcome to the future of ethical hacking!

Goals/Objectives/Approach of the Book

This book provides a comprehensive framework for transitioning from traditional security testing methodologies to modern and AI-enhanced methodologies, helping you to remain effective in an evolving threat landscape. This book delivers actionable, hands-on knowledge that you can immediately apply in real-world red team engagements and bug bounty hunting, with a focus on both current and emerging technologies.

This book will equip you to execute sophisticated red team operations that account for AI-powered defense systems and leverage AI and automation effectively in bug bounty hunting while maintaining the crucial human element.

You will learn how to build and maintain modern red team infrastructure and leverage custom tools and frameworks that combine traditional and new techniques.

Each chapter begins with core concepts and then provides step-by-step explanations and real-world case scenarios.

Targeted Reading Audience

This book is designed for beginners, experienced cybersecurity professionals transitioning to red team operations, and active bug bounty hunters looking to enhance their knowledge.

Additionally, it is designed for red teamers seeking to integrate AI into their methodology and any organizations building or improving their red team capabilities.

The only prerequisites are a basic understanding of networking, operating systems, and web technologies and a basic understanding of cybersecurity concepts and terminology.

Book Organization

Chapter 1: The Evolution of Penetration Testing, Red Teaming, and Bug Bounties

This chapter covers the transformative journey of security testing from its origins to today's AI-augmented practices. We explore how simple vulnerability assessments evolved into sophisticated red team operations, examining the pivotal developments that shaped modern security testing methodologies, tools, and frameworks.

The chapter highlights how the industry's landscape has been revolutionized by major shifts: the evolution of penetration testing frameworks, the emergence of professional red teaming incorporating physical security and social engineering, and the rise of bug bounty platforms that democratized security testing.

Chapter 2: Introduction to Red Teaming

This chapter goes over the fundamental principles and methodologies that distinguish red teaming from traditional penetration testing. You will learn how red teams operate as dedicated adversary emulation units, understanding the key components of engagement planning, threat actor profiling, and the establishment of clear objectives and rules of engagement. The chapter introduces essential frameworks and discusses how they guide modern red team operations.

Chapter 3: Red Team Infrastructure

This chapter covers the critical components of building and maintaining a resilient red team infrastructure. From designing covert Command and Control (C2) frameworks to implementing secure communication channels, the chapter covers essential concepts, including domain categorization, traffic redirection, and operational security measures.

Chapter 4: Modern Red Team Methodology and Tools

This chapter examines cutting-edge methodologies and tools that define modern red team operations. The chapter investigates how traditional attack frameworks are being enhanced by modern tools, exploring advanced techniques in initial access, lateral movement, and persistence. The chapter walks through each phase of a red team operation.

Chapter 5: Social Engineering and Physical Assessments

This chapter explores the human element of red teaming through advanced social engineering techniques and physical security assessments. The chapter examines how social engineering has evolved in the age of AI-powered behavioral analytics and security awareness training, covering sophisticated pretexting, phishing campaigns, and psychological manipulation techniques that can bypass modern defenses. Special attention is given to conducting safe and ethical social engineering exercises that provide valuable insights without causing harm.

Moving into physical security, the chapter details methodologies for conducting comprehensive physical penetration tests, including facility access assessments, physical security control bypass, and the convergence of cyber and physical attack vectors. You will learn about modern physical security tools, RFID cloning techniques, and tactics for blending cyber and physical methodologies.

Chapter 6: Advanced Post-Exploitation Techniques

This chapter details sophisticated post-exploitation methodologies that enable red teams to maintain persistent access while evading detection. The chapter covers advanced techniques for memory manipulation, process injection, and living-off-the-land techniques, with particular focus on bypassing modern endpoint detection and response (EDR) solutions and AI-powered security tools. You will learn how to develop and implement custom implants that can operate undetected in highly monitored environments. We discuss the importance of operational security during post-exploitation activities and provide strategies for cleaning up artifacts to maintain long-term covert presence.

Chapter 7: Active Directory and Linux Environments

This chapter explores advanced tactics for compromising modern Active Directory (AD) environments and Linux infrastructures. For Active Directory, the chapter covers sophisticated attack techniques including domain privilege escalation, credential abuse, and trust relationship exploitation, with special focus on bypassing modern security controls.

BloodHound is an open-source tool designed to map out relationships and permissions in Active Directory. It uses graph theory to visualize attack paths that could be exploited by attackers to escalate privileges or move laterally within a network. By analyzing these relationships, BloodHound helps security professionals understand how an attacker might compromise an environment and provides insights into misconfigurations or weaknesses that need remediation. SharpHound is the data collection component of BloodHound. It gathers information from AD environments by querying domain controllers and domain-joined systems. The collected data is then fed into BloodHound for analysis. In this chapter, you will learn details and review examples on how to use these tools.

The Linux section covers advanced exploitation techniques specific to modern Linux environments, including container escape, kernel exploitation, and privilege escalation in hardened environments. The chapter details methodologies for exploiting Linux systems leveraging common misconfigurations and vulnerabilities.

Chapter 8: The Future of Red Teaming Beyond the AI Revolution

This chapter covers how artificial intelligence and emerging technologies are revolutionizing red team operations. You will learn how AI is being leveraged to enhance attack capabilities, including autonomous reconnaissance systems, exploitation at scale, and advanced social engineering attacks using AI.

Chapter 9: Introduction to Bug Bounty and Effective Reconnaissance

This chapter provides a comprehensive introduction to professional bug bounty hunting in today's competitive landscape. The chapter examines the evolution of bug bounty programs, platform dynamics, and strategies for building a successful career as a bug hunter.

Chapter 10: Hacking Modern Web Applications and APIs

This chapter covers many techniques for testing modern web applications and APIs. We explore sophisticated attack methodologies against contemporary application architectures, including microservices, serverless functions, and GraphQL APIs.

Moving beyond traditional vulnerabilities, the chapter investigates emerging attack vectors in modern frameworks, cloud configurations, and authentication mechanisms.

Chapter 11: Automating a Bug Hunt and Leveraging the Power of AI

This chapter covers how AI and automation are revolutionizing the bug hunting process. The chapter covers practical tactics for building intelligent scanning systems, developing custom tools that leverage AI for vulnerability detection, and creating automated pipelines that can scale across multiple targets. You will learn how to combine traditional security tools with AI capabilities to enhance security assessments. Throughout the book you will learn about retrieval-augmented generation (RAG) and tools such as LangChain, LlamaIndex, Ollama, and many other AI frameworks.

Credits

Cover - Thinkhubstudio/Shutterstock

Figures 5.1, 5.2, 5.9 & 5.10 - OpenAI Inc

Figures 3.2, 3.3, 3.4, 3.16, 3.17, 3.21 - Microsoft Corporation

Figures 4.3, 4.4, 4.5, 4.6, 4.7 - Bright Data Ltd

Figure 4.10 – Caphyon

Figures 4.11, 4.12, 4.13, 4.15 - Microsoft Corporation

Figure 5.3 – SpoofCard

Figure 5.5 – Palo Alto Networks

Figures 5.6, 5.7, 5.17, 5.18 - Microsoft Corporation

Figure 5.8 – Breakpoint Software Development

Figure 5.13, 5.14, 5.15 & 8.8 - GitHub, Inc

Figure 6.5 – Jens Duttke

Figure 6.9 – GitHub, Inc

Figures 7.8, 7.9, 7.10, 7.11 - Google LLC

Figure 7.14 – Mayfly

Figures 8.1, 8.3, 9.2 & 9.4 - Google LLC

Figures 10.3 & 10.16 - Wireshark Foundation

Figures 10.7, 10.9, 10.10, 10.11, 10.12 - Apple Inc

Figures 10.20 & 10.21 - Mozilla Foundation

Figure 10.25 - PortSwigger Ltd

Figure 10.26 - ZAP Dev Team

Figures 11.5 – 11.12 - Apple Inc

11

Automating a Bug Hunt and Leveraging the Power of AI

Chapter Objectives

After reading this chapter and completing the exercises, you will be able to do the following:

- Use traditional bug hunting methods
- Employ AI-powered automation in bug hunting
- Understand AI model training, fine-tuning, and retrieval-augmented generation (RAG)
- Understand the challenges of using AI for bug bounty hunting

In Chapter 8, "The Future of Red Teaming Beyond the AI Revolution," you learned about the current state of AI in red teaming, examining AI-powered offensive tools and techniques, fine-tuned uncensored AI models, and the application of retrieval-augmented generation (RAG) for red teaming purposes.

In this chapter, we will explore how to leverage AI for bug bounty hunting. You will learn about the methodologies and tools that can enhance your effectiveness as a bug bounty hunter, integrating AI to identify vulnerabilities more efficiently and accurately.

Traditional Bug Hunting Methods

Given the vast amount of information available online about bug bounty hunting, beginners might feel overwhelmed—which is entirely normal. To navigate this information overload, focus on a few high-quality resources and immediately apply what you've learned through hands-on practice.

Tips for Beginners

In Chapter 9, "Introduction to Bug Bounty and Effective Reconnaissance," you learned how to get started with bug bounties. You should begin with the basics. For example, in web applications, you can start by understanding HTTP requests and the basics of how websites function. Choose a common vulnerability to specialize in, such as cross-site scripting (XSS), and concentrate your efforts there. PortSwigger offers an excellent Cross-Site Scripting (XSS) Cheat Sheet that lists XSS vectors across different frameworks and methods to bypass web application firewalls (WAFs).

Another area to explore is business logic vulnerabilities. Finding them is often straightforward once you grasp the application's overall workflow. They require you to understand how the application is supposed to function and then identify edge cases that could have security or business implications. For instance, you might discover a way to bypass a paywall and access premium content without payment.

Information disclosure is another impactful type of vulnerability that can occur anywhere within an application. Your goal is to find instances where sensitive information—like user data, internal company details, old data backups, API keys, source code, or error messages—is inadvertently exposed. Identifying information that should remain confidential is crucial in this category.

As a beginner, you are advised to focus on one type of vulnerability and attempt to find it across multiple programs or targets. With determination and persistence, you're likely to achieve success because bugs are always present in applications.

How do you shift from theory to practice? The field of bug bounty continues to evolve rapidly, prioritizing practical experience over theory. I (Omar) always say that cybersecurity, and especially ethical hacking, is like math. The more you practice, the better you will become.

Note

Finding bugs is essentially about identifying features that don't work as intended. To do this effectively, you need to understand how the application should function and recognize deviations, especially from a security perspective. This task requires deep familiarity with the application and knowledge of common vulnerability areas. Be cautious of targeting only "low-hanging fruits"—simple bugs that are easy to find but are also the focus of many other hunters. Targeting only simple bugs increases the chance of submitting duplicate reports and can result in a low return on investment for your time and effort. Instead, thoroughly analyze applications to uncover more significant and less obvious vulnerabilities.

Limitations of Manual Bug Hunting

Manual bug hunting has been a fundamental aspect of ethical hacking for many years. Skilled cybersecurity professionals meticulously examine applications to uncover flaws that automated tools might miss. However, as technology becomes more complex and cyber threats more sophisticated, the limitations of manual bug hunting are becoming more pronounced.

Manual bug hunting is very time-consuming due to the extensive effort required for in-depth analysis. You must meticulously test different system configurations and system behaviors, which can be extremely labor-intensive. Modern applications often involve complex architectures, including microservices, APIs, and third-party integrations, making it impractical for individuals to scrutinize every component manually. This extensive time investment can lead to delayed vulnerability discovery, resulting in a slow threat response. The lengthy process of manual analysis can postpone the identification of critical security issues, increasing the risk window and providing attackers with more opportunities to exploit vulnerabilities.

Human error is an inevitable factor in manual bug hunting. You may overlook vulnerabilities due to cognitive limitations such as making assumptions or focusing on familiar attack vectors while neglecting others. Fatigue and attention lapses can occur during extended periods of manual testing, leading to decreased concentration and missed flaws. Knowledge gaps also contribute to oversight because no individual can have exhaustive knowledge of all potential vulnerabilities across diverse technologies.

Many individuals currently automate a lot of the bug bounty activities to learn about new bug bounties in platforms like HackerOne, Bugcrowd, Intigriti, and others. For example, you may scan any HackerOne program (bug bounty) with Nuclei, as shown in Example 11-1. Nuclei is an open-source vulnerability scanner developed by ProjectDiscovery, known for its speed, efficiency, and customizability. It uses a template-based approach, with YAML files defining the methods for detecting vulnerabilities across various targets, including web applications, cloud infrastructure, and networks.

EXAMPLE 11-1 Using Nuclei to Scan Hosts in Any HackerOne Bug Bounty

```
websploit$ python3 h1_2_nuclei.py -handle security
[i] Checking scope for: security
[i] Parsing scope items
[i] Wildcards in scope:      1
[i] Hosts in scope:          19
[i] Hosts out of scope:      3
[i] Checking subdomains with chaos
[i] Hosts in scope:          8
[i] Hosts out of scope:      4
[i] Removing out of scope items
[i] Unique hosts in scope:   9
[i] Saving hosts to: targets/security/chaos_security_250808.txt
[i] Resolving subdomains with httpx
[i] Output saved to: targets/security/httpx_security_250808.txt
[i] Number of live targets: 9
```

```
[i] Scanning targets with Nuclei
[i] Output saved to: targets/security/nuclei_security_250808.txt
[i] Vulnerabilities found: 8
```

The tools shown in Example 11-1 can be obtained from https://github.com/vavkamil/h1_2_nuclei.

To overcome the limitations of manual bug hunting, you should use automation that goes beyond vulnerability scanners and traditional hacking tools. Leveraging AI and machine learning can significantly help. You can adopt a hybrid testing approach that combines manual testing with automated tools to provide a more comprehensive security assessment. In the following sections, we will explore the benefits and limitations of using AI in bug hunting.

AI-Powered Automation in Bug Hunting

You've probably heard the phrase "the sky's the limit." When it comes to using AI for bug bounty hunting, that statement couldn't be more accurate. The possibilities are amazing, and AI has the potential to revolutionize the way vulnerabilities are detected and exploited.

Figure 11-1 illustrates a simplified hierarchy of how bug bounty workflow entities are related.

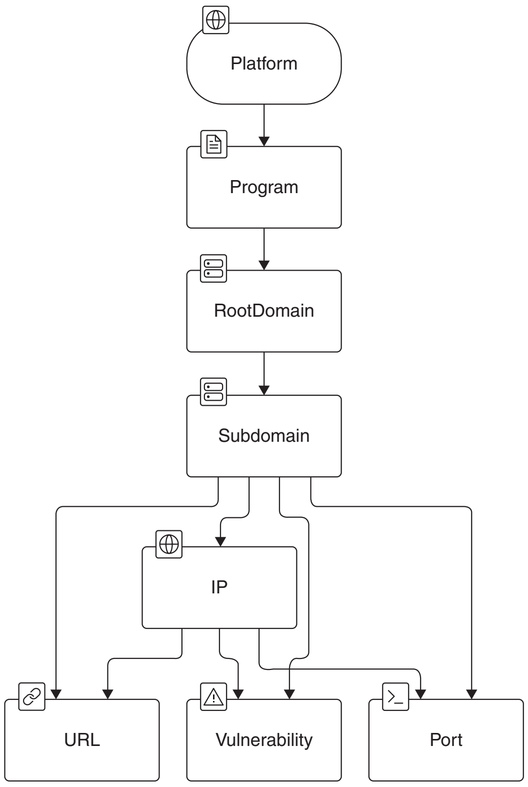


Figure 11-1
The Bug Bounty Hunter Workflow

At the top of the hierarchy are the bug bounty platforms like Bugcrowd, HackerOne, Intigriti, and others. Next, the program is the specific bug bounty program offered by a company or organization on the platform. For example, a company may set up a program with Bugcrowd to allow researchers to test their web applications for vulnerabilities.

The root domain represents the main domain or scope of the program. For example, if the company's main website is websploit.org, this would be the root domain in scope for bug testing. Programs often list which root domains are in scope for testing.

Beneath the root domain are subdomains, which are more specific areas of the website (such as api.websploit.org or admin.websploit.org). Bug bounty programs may specify which subdomains are part of the scope because certain subdomains may expose different services or applications.

The IP address represents the network layer, where subdomains are resolved to one or more IP addresses. This allows ethical hackers to probe network-level configurations, services, and open ports that could be vulnerable.

The URL represents specific web pages or API endpoints within the subdomain. These URLs may have parameters, authentication mechanisms, or other inputs that hackers target for testing vulnerabilities like XSS or SQL injection.

Ports represent the specific network ports on the IP address that could be open and potentially expose services. For example, port 80 (HTTP) and port 443 (HTTPS) are often tested.

Let's assume that now you want to automate the discovery, scanning, enumeration, and processing of these tasks using AI agents. You can use platforms like LangGraph and LangGraph Studio to create agents that will help accelerate these tasks with low-code requirements. Let's take a look at the diagram in Figure 11-2.

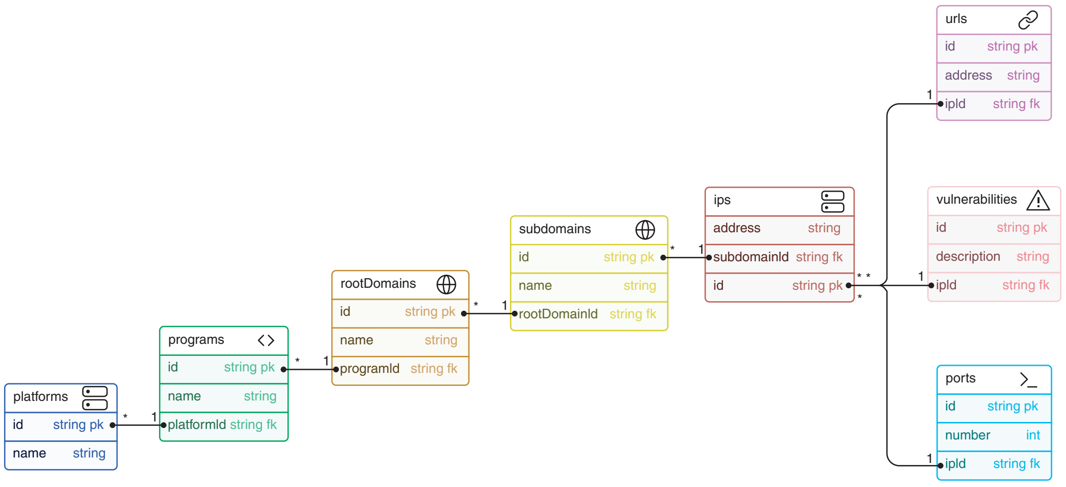


Figure 11-2

High-Level Building Blocks of an Automated Recon System

Figure 11-2 represents the architecture of the building blocks for an automated system to automate reconnaissance and vulnerability in bug bounty programs.

The platforms table represents the various bug bounty platforms such as Bugcrowd, HackerOne, and similar platforms. The following are the *platforms* fields:

- **id**: A unique identifier for each platform (primary key)
- **name**: The name of the platform (such as Bugcrowd, HackerOne)

The *programs* table represents individual bug bounty programs run by different organizations on the platforms. Each program specifies the scope and rules for identifying and reporting vulnerabilities. These are specific bug bounties listed under the *platforms*. The following are the *programs* fields:

- **id**: A unique identifier for each program (primary key)
- **name**: The name of the bug bounty program
- **platformId**: A foreign key that links the program to a specific platform

The *root domains* are the primary domains that are in scope for a bug bounty program. Each program may cover one or multiple root domains that you are allowed to test. The following are the *root domains* fields:

- **id**: A unique identifier for each root domain (primary key)
- **name**: The name of the root domain (such as websploit.org)
- **programId**: A foreign key linking the root domain to a specific bug bounty program

These are the *subdomains* under the root domains that are also in scope for the bug bounty. A root domain like websploit.org may have subdomains such as api.websploit.org, which are part of the same scope for vulnerability testing. The following are the *subdomains* fields:

- **id**: A unique identifier for each subdomain (primary key)
- **name**: The name of the subdomain
- **rootDomainId**: A foreign key linking the subdomain to a specific root domain

The *IPs* table stores IP addresses associated with subdomains. These IP addresses are the targets that may be probed for vulnerabilities such as open ports, services, and misconfigurations. The following are the fields in the *IPs* table:

- **id**: A unique identifier for each IP address of the corresponding subdomain
- **address**: The IP address of the subdomain
- **subdomainId**: A foreign key linking the IP address to a specific subdomain

The *ports* table contains information about open ports on specific IP addresses. Identifying open ports is critical for understanding the services running on the target, which can reveal vulnerabilities such as misconfigurations or exposure of sensitive services. The following are the fields for *ports:*

- **id**: A unique identifier for each port
- **number**: The port number (such as 80, 443, 22)
- **ipId**: A foreign key linking the port to a specific IP

The *URLs* table tracks specific web addresses within the subdomains that may need further inspection for vulnerabilities such as XSS, CSRF, or SQL injection. The following are the URLs fields:

- **id**: A unique identifier for each URL
- **address**: The specific URL address (such as https://api.websploit.org/login)
- **ipId**: A foreign key linking the URL to a specific IP address

The *vulnerabilities* table stores information about specific vulnerabilities found during the bug hunting process. This is one of the core elements of this automated system because it represents the outcomes of ethical hacking and bug hunting efforts. The following are the URL fields*:*

- **id**: A unique identifier for each vulnerability (primary key)
- **description**: A textual description of the vulnerability (such as "SQL Injection in the login page").
- **ipId**: A foreign key linking the vulnerability to a specific IP

How can AI help with bug hunting in this architecture? AI can correlate information between different components, such as IPs, URLs, vulnerabilities flagged, and open ports, to paint a complete picture of potential attack surfaces. It can even analyze historical data from other scans to similar applications. AI could predict which services or ports are more likely to have security flaws based on the type of system, configuration, or past exploitation data.

AI models can be trained to identify common web vulnerabilities, such as XSS or SQL injection, by scanning and analyzing the URLs, parameters, and request-response patterns. AI can automate the process of identifying malicious input or output behavior, improving detection accuracy and reducing false positives.

AI can accelerate subdomain enumeration by using predictive models that recognize patterns in domain naming conventions, automatically suggesting potential subdomains or previously undiscovered targets. AI-driven DNS reconnaissance tools can quickly uncover new subdomains linked to root domains by cross-referencing large datasets from DNS history and certificates.

What if we give the explanation of Figure 11-2 to tools such as Cursor, GitHub Copilot, or OpenAI's models such as o1? I used OpenAI's o1, and it created Python code that uses SQLAlchemy, an object-relational mapping (ORM) library. Example 11-2 shows the Python code. Additionally, the code includes functions to process and interact with the data, such as adding records and querying vulnerabilities.

EXAMPLE 11-2 AI-Generated Code to Implement the Database to Support the Use Case of Figure 11-2

```
from sqlalchemy import (
    create_engine, Column, Integer, String, ForeignKey, Text, UniqueConstraint
)
from sqlalchemy.orm import sessionmaker, relationship
from sqlalchemy.ext.declarative import declarative_base

# Define the database
engine = create_engine('sqlite:///bug_bounty.db')
Base = declarative_base()

# Define the platforms
class Platform(Base):
    __tablename__ = 'platforms'
    id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)
    programs = relationship('Program', back_populates='platform')

class Program(Base):
    __tablename__ = 'programs'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    platform_id = Column(Integer, ForeignKey('platforms.id'))

    platform = relationship('Platform', back_populates='programs')
    root_domains = relationship('RootDomain', back_populates='program')

class RootDomain(Base):
    __tablename__ = 'root_domains'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    program_id = Column(Integer, ForeignKey('programs.id'))

    program = relationship('Program', back_populates='root_domains')
    subdomains = relationship('Subdomain', back_populates='root_domain')

class Subdomain(Base):
    __tablename__ = 'subdomains'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    root_domain_id = Column(Integer, ForeignKey('root_domains.id'))

    root_domain = relationship('RootDomain', back_populates='subdomains')
    ips = relationship('IP', back_populates='subdomain')
```

```
class IP(Base):
    __tablename__ = 'ips'
    id = Column(Integer, primary_key=True)
    address = Column(String)
    subdomain_id = Column(Integer, ForeignKey('subdomains.id'))

    subdomain = relationship('Subdomain', back_populates='ips')
    ports = relationship('Port', back_populates='ip')
    urls = relationship('URL', back_populates='ip')
    vulnerabilities = relationship('Vulnerability', back_populates='ip')

class Port(Base):
    __tablename__ = 'ports'
    id = Column(Integer, primary_key=True)
    number = Column(Integer)
    ip_id = Column(Integer, ForeignKey('ips.id'))

    ip = relationship('IP', back_populates='ports')

class URL(Base):
    __tablename__ = 'urls'
    id = Column(Integer, primary_key=True)
    address = Column(String)
    ip_id = Column(Integer, ForeignKey('ips.id'))

    ip = relationship('IP', back_populates='urls')

class Vulnerability(Base):
    __tablename__ = 'vulnerabilities'
    id = Column(Integer, primary_key=True)
    description = Column(Text)
    ip_id = Column(Integer, ForeignKey('ips.id'))

    ip = relationship('IP', back_populates='vulnerabilities')

# Create all tables
Base.metadata.create_all(engine)

# Create a session
Session = sessionmaker(bind=engine)
session = Session()

# Functions to process data
def add_platform(name):
    platform = Platform(name=name)
    session.add(platform)
    session.commit()
    print(f"Added platform: {name}")

def add_program(name, platform_name):
    platform = session.query(Platform).filter_by(name=platform_name).first()
    if not platform:
        print(f"Platform {platform_name} not found.")
        return
    program = Program(name=name, platform=platform)
    session.add(program)
    session.commit()
    print(f"Added program: {name} under platform {platform_name}")

def add_root_domain(name, program_name):
    program = session.query(Program).filter_by(name=program_name).first()
    if not program:
        print(f"Program {program_name} not found.")
        return
    root_domain = RootDomain(name=name, program=program)
    session.add(root_domain)
    session.commit()
    print(f"Added root domain: {name} under program {program_name}")

def add_subdomain(name, root_domain_name):
    root_domain = session.query(RootDomain).filter_by(name=root_domain_name).
first()
    if not root_domain:
        print(f"Root domain {root_domain_name} not found.")
        return
    subdomain = Subdomain(name=name, root_domain=root_domain)
    session.add(subdomain)
    session.commit()
    print(f"Added subdomain: {name} under root domain {root_domain_name}")

def add_ip(address, subdomain_name):
    subdomain = session.query(Subdomain).filter_by(name=subdomain_name).first()
    if not subdomain:
        print(f"Subdomain {subdomain_name} not found.")
        return
    ip = IP(address=address, subdomain=subdomain)
    session.add(ip)
    session.commit()
    print(f"Added IP: {address} under subdomain {subdomain_name}")
```

```
def add_port(number, ip_address):
    ip = session.query(IP).filter_by(address=ip_address).first()
    if not ip:
        print(f"IP {ip_address} not found.")
        return
    port = Port(number=number, ip=ip)
    session.add(port)
    session.commit()
    print(f"Added port: {number} to IP {ip_address}")

def add_url(address, ip_address):
    ip = session.query(IP).filter_by(address=ip_address).first()
    if not ip:
        print(f"IP {ip_address} not found.")
        return
    url = URL(address=address, ip=ip)
    session.add(url)
    session.commit()
    print(f"Added URL: {address} to IP {ip_address}")

def add_vulnerability(description, ip_address):
    ip = session.query(IP).filter_by(address=ip_address).first()
    if not ip:
        print(f"IP {ip_address} not found.")
        return
    vulnerability = Vulnerability(description=description, ip=ip)
    session.add(vulnerability)
    session.commit()
    print(f"Added vulnerability to IP {ip_address}: {description}")

def get_vulnerabilities_by_program(program_name):
    program = session.query(Program).filter_by(name=program_name).first()
    if not program:
        print(f"Program {program_name} not found.")
        return []
    vulnerabilities = session.query(Vulnerability).join(IP).join(Subdomain).
join(RootDomain).filter(
        RootDomain.program == program
    ).all()
    return vulnerabilities

# Example usage
if __name__ == '__main__':
    # Add data
    add_platform('HackerOne')
    add_program('Acme Corp Bug Bounty', 'HackerOne')
```

```
add_root_domain('acme.com', 'Acme Corp Bug Bounty')
add_subdomain('api.acme.com', 'acme.com')
add_ip('192.168.1.1', 'api.acme.com')
add_port(80, '192.168.1.1')
add_port(443, '192.168.1.1')
add_url('https://api.acme.com/login', '192.168.1.1')
add_vulnerability('SQL Injection in login page', '192.168.1.1')

# Retrieve vulnerabilities for a program
vulns = get_vulnerabilities_by_program('Acme Corp Bug Bounty')
for vuln in vulns:
    print(f"Vulnerability ID: {vuln.id}, Description: {vuln.description}")
```

The code in Example 11-2 provides a foundational structure for your automated system. You can extend it by adding more processing functions, integrating actual reconnaissance tools, and implementing vulnerability scanning methods.

Tip

For a real-world application, consider adding error handling, input validation, and logging. You might also want to integrate external libraries and tools for tasks like DNS enumeration, port scanning, and vulnerability assessment to automate the data population.

AI Capabilities of Bug Bounty Platforms

Bug bounty platforms like Bugcrowd are all leveraging AI to enhance their services, streamline operations, and provide more security solutions to their clients. AI's integration into these platforms enables continuous monitoring, automated vulnerability detection, and intelligent threat analysis, among other capabilities. Bugcrowd has an AI-driven service called Continuous Attack Surface Penetration Testing (CASPT).

CASPT uses AI models to continuously scan and identify new or modified assets within an organization's digital footprint. This effort includes detecting new applications, services, or infrastructure components that may have been added or altered since the last assessment.

AI helps establish a baseline of the organization's attack surface by analyzing historical data. It then monitors for deviations or changes, ensuring that any new vulnerabilities introduced by modifications are promptly identified. Based on the continuous monitoring of assets, AI determines optimal times to initiate penetration tests. By doing so, it guarantees that testing is both timely and relevant, focusing on areas with the highest risk or recent changes.

Bugcrowd's acquisition of a company named Informer helped it in advanced external attack surface management (EASM) capabilities. AI algorithms merge detailed asset information from Informer with Bugcrowd's existing vulnerability databases. You can initiate new penetration tests directly from the EASM dashboards, with AI managing the integration and ensuring that tests are aligned with the current state of the attack surface.

A Comprehensive View of an Organization's External Risk Exposure

An organization's attack surface extends far beyond what is readily visible. The modern typical ecosystem consists of web domains, subdomains, IP addresses, cloud services, APIs of hundreds of applications (including AI applications), and more—each potentially serving as an entry point for an attacker. The first step in automated external attack surface management is to gain visibility into both known and unknown assets that an adversary might exploit. AI-powered tools can scan, map, and inventory these assets, giving you an "attacker's perspective."

Figure 11-3 illustrates a process flow or a framework for external attack surface discovery and exploitation. The figure is structured with concentric circles, which could symbolize different layers or steps in the process. Each layer is associated with one of four core concepts: discovering assets, monitoring changes, getting actionable insights, and amplifying security testing.

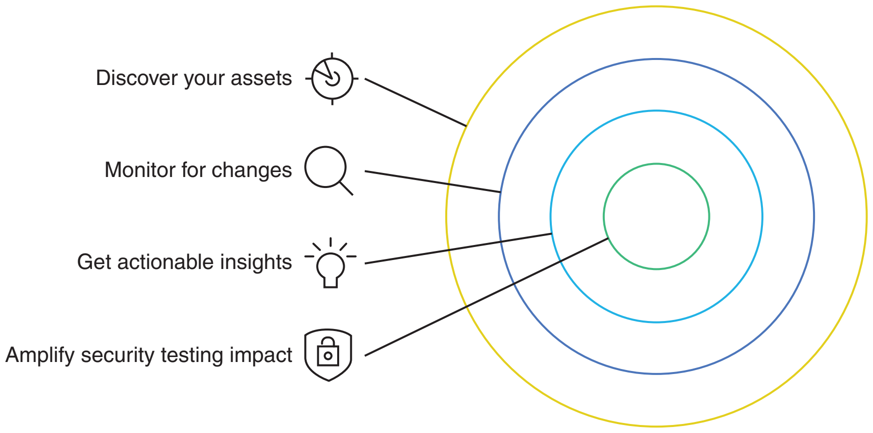


Figure 11-3
A Framework for External Attack Surface Discovery and Exploitation

The first goal is to find and identify all external assets, known and unknown, that could be part of an organization's footprint. Then you continuously watch for any alterations or updates in the external infrastructure or applications, ensuring that no new exposures go unnoticed. Subsequently, you turn the data gathered through monitoring into meaningful insights, which you can use to formulate your attack plan. You could integrate this process with red teaming, penetration testing, and crowdsourced bug bounties.

Vulnerability Prioritization Using AI

AI can assist in prioritizing vulnerabilities based on potential exploitability and risk. By analyzing factors such as Common Vulnerability Scoring System (CVSS) scores, the affected services, historical exploit data, and business impact, AI can help security teams focus on the most critical vulnerabilities first.

Let's start by defining CVSS, the Exploit Prediction Scoring System (EPSS), and CISA's Known Exploited Vulnerabilities (KEV) Catalog, and then examine how AI can assist in vulnerability prioritization using these systems.

A Quick Refresher About CVSS, EPSS, and CISA's KEV

CVSS is an open framework for communicating the characteristics and severity of software vulnerabilities. It provides a way to capture the principal technical characteristics of a vulnerability and produce a numerical score reflecting its severity (0–10). The CVSS score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management processes. You can access the latest CVSS specification at https://www.first.org/cvss.

CVSS consists of four metric groups: Base, Threat, Environmental, and Supplemental. Base metrics represent the intrinsic qualities of a vulnerability. The Threat metric group represents the traits of a vulnerability connected to potential threats, which can evolve over time, though they may remain consistent across different user environments.

The Environmental metric group captures the features of a vulnerability that are specific and relevant to an individual consumer's environment. This includes factors such as the existence of security controls that could reduce or eliminate the impact of a successful attack, as well as the criticality of the affected system within the overall technology infrastructure. The Supplemental metric group consists of metrics that offer additional context and measure external attributes of a vulnerability. The answers to these metrics are defined by the CVSS consumer, enabling them to use a risk analysis system tailored to their environment to assign locally relevant severity to the metrics and values.

EPSS is a data-driven effort for estimating the probability that a software vulnerability will be exploited in the wild. Unlike CVSS, which focuses on the technical severity of a vulnerability, EPSS aims to predict the likelihood of a vulnerability being exploited based on real-world data and machine learning techniques. You can access the latest EPSS specification at https://www.first.org/epss.

The following are a few key features of EPSS:

- Provides a probability score between 0 and 1 (0% to 100%)
- Uses current threat information and real-world exploit data
- Updates scores daily based on new information
- Helps prioritize vulnerabilities based on exploitation likelihood rather than just technical severity

The KEV Catalog is maintained by the Cybersecurity and Infrastructure Security Agency (CISA) and serves as an authoritative source of vulnerabilities that have been exploited in the wild. It is designed to help organizations prioritize remediation efforts for vulnerabilities that pose significant risks. You can access CISA's KEV at https://www.cisa.gov/kev.

KEV focuses on vulnerabilities with known exploit code or active exploitation and provides remediation due dates for each vulnerability. This list is updated regularly as new exploited vulnerabilities are identified.

How AI Can Help Bug Hunters and Organizations Prioritize Vulnerabilities Using CVSS, EPSS, and KEV

AI can help integrate data from CVSS, EPSS, and KEV along with other relevant sources to create a comprehensive view of vulnerability risks. Machine learning algorithms can analyze this data to identify patterns and correlations that humans might miss, providing a more nuanced understanding of vulnerability priorities. Figure 11-4 shows how you could build AI systems to analyze and prioritize vulnerabilities on a large scale using data from CVSS, EPSS, and KEV.

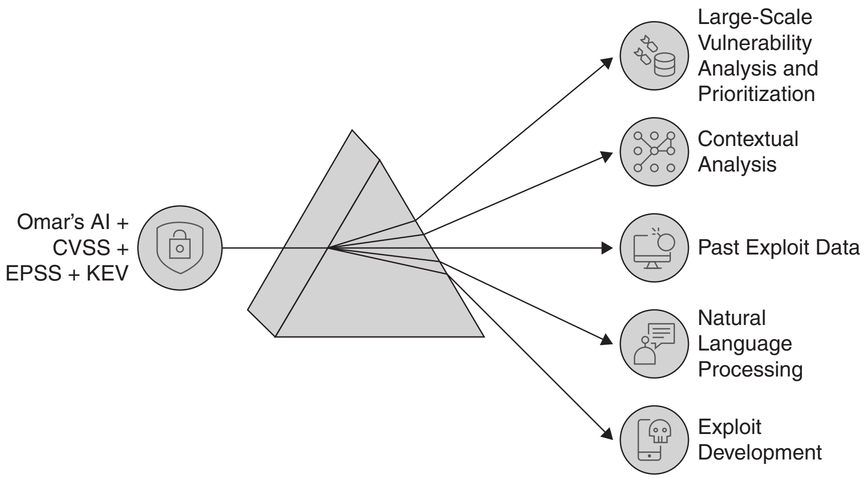

Omar's AI + CVSS + EPSS + KEV
Large-Scale Vulnerability Analysis and Prioritization
Contextual Analysis
Past Exploit Data
Natural Language Processing
Exploit Development

Figure 11-4
AI Analysis of CVSS, EPSS, and KEV

By leveraging historical data from CVSS, EPSS, and KEV, AI can develop predictive models to estimate the likelihood of future exploits. AI can consider the specific context of an organization or bug hunter's environment when prioritizing vulnerabilities. By analyzing factors such as the organization's network architecture, deployed assets, and security controls, you can use AI to provide tailored prioritization recommendations to your client that go beyond generic CVSS or EPSS scores.

AI-enabled systems can continuously monitor threat intelligence feeds, including updates to the KEV Catalog, and automatically adjust vulnerability discovery priorities based on new information.

You can leverage GenAI with NLP capabilities to analyze vulnerability descriptions, security advisories, and exploit discussions in forums to extract additional context and severity indicators. This information can be used to refine prioritization beyond what CVSS, EPSS, and KEV provide numerically.

By analyzing the characteristics of vulnerabilities that have progressed from CVSS scoring to EPSS high-probability to KEV listing, you can potentially predict which newly discovered vulnerabilities are most likely to follow a similar path. This analysis can help prioritize vulnerabilities that may not yet have high EPSS scores or KEV entries but have characteristics suggesting they may soon become critical.

Based on the vulnerability type and target system, AI could generate basic exploit templates or code skeletons, which an ethical hacker could then refine and customize. AI can parse and analyze large volumes of technical documentation, APIs, and code comments to identify potential weak points or unintended functionality that could be exploited.

AI-Assisted Exploit Development Using CISA's KEV Catalog and Other Information

AI systems could potentially adapt known exploit techniques to new environments or slightly different versions of software, helping to test the extent of a vulnerability. AI assistants could work alongside human researchers, offering suggestions, checking for errors, and providing relevant information during the exploit development process.

AI-Created Scanner Templates

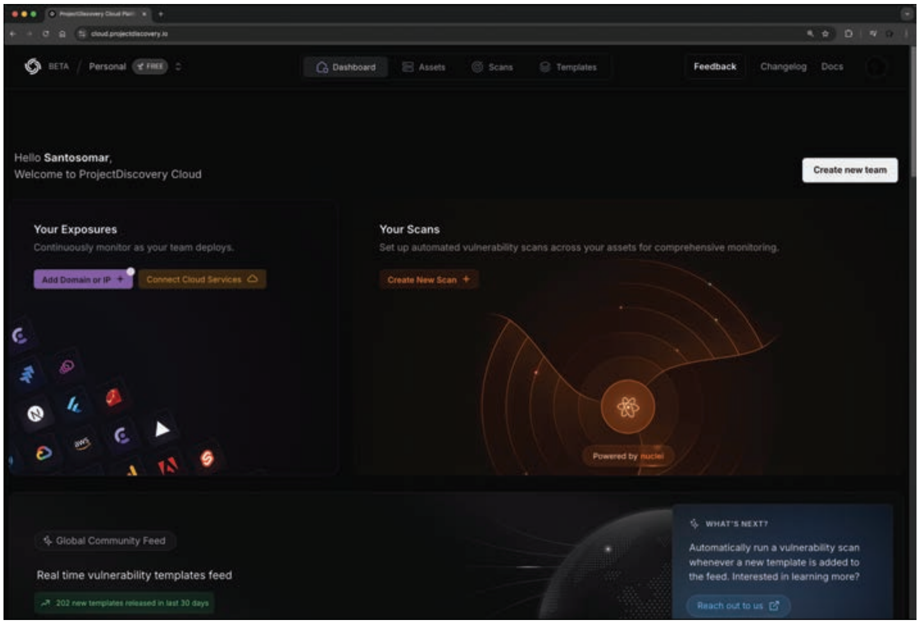
The ProjectDiscovery Nuclei Scanner is a fast and extensible open-source vulnerability scanner designed to identify and mitigate security vulnerabilities across modern applications, infrastructure, cloud platforms, and networks.

Nuclei uses a template-driven approach, where each template is a YAML file that defines the steps needed to detect specific vulnerabilities. This approach allows for highly customizable and targeted scans. It also supports a massive library of community-curated templates, which are continually updated to include the latest vulnerabilities in many systems.

The ProjectDiscovery Cloud Platform (PDCP) is a cloud-based security solution targeted at delivering ongoing visibility into your external attack surface by identifying exploitable vulnerabilities and misconfigurations. It is designed to address multiple use cases and can scale to support the essential workflows that bug bounty hunters and security teams require to find vulnerabilities. You can access PDCP at https:// cloud.projectdiscovery.io.

Figure 11-5 shows the PDCP dashboard.

As previously mentioned, Nuclei supports a massive library of community-curated templates. Figure 11-6 shows an example of a community template.



Hello Santosomar,
Welcome to ProjectDiscovery Cloud

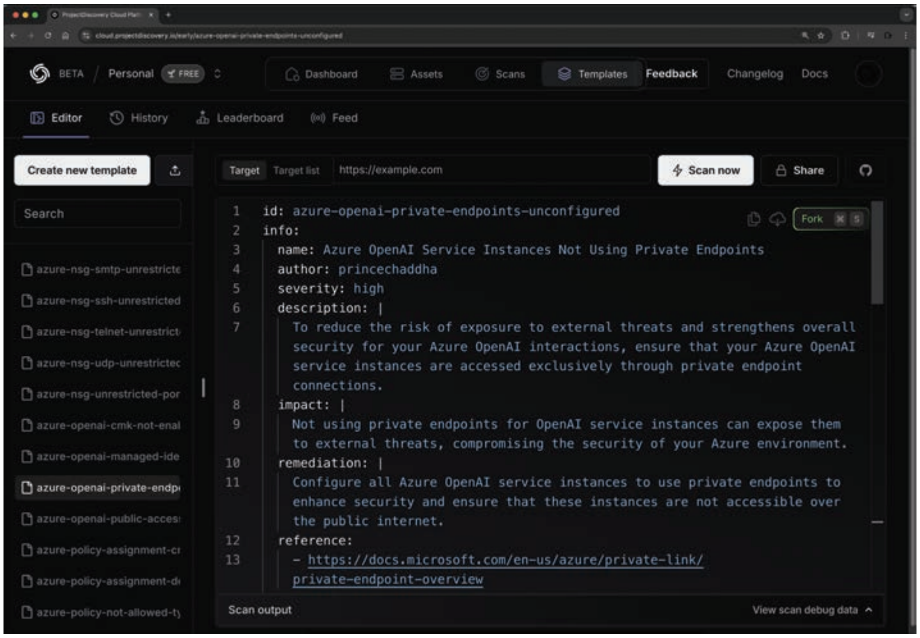
Your Exposures
Continuously monitor as your team deploys.

Your Scans
Set up automated vulnerability scans across your assets for comprehensive monitoring.

Real time vulnerability templates feed

Automatically run a vulnerability scan whenever a new template is added to the feed. Interested in learning more?

Figure 11-5
The PDCP Dashboard

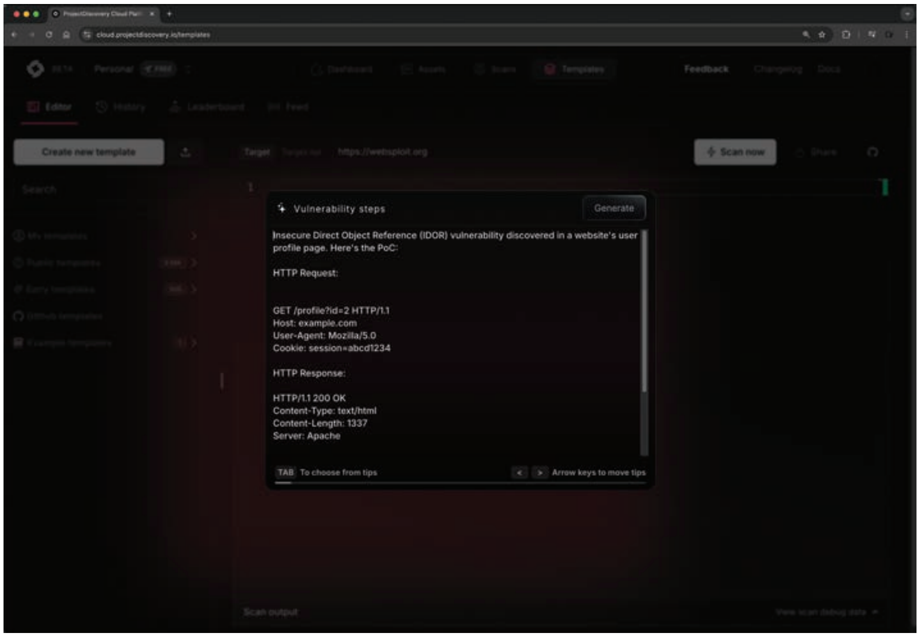

BETA
Personal
FREE
Dashboard
Assets
Scans
Templates
Feedback
Changelog
Docs
Editor
History
Leaderboard
Feed
Create new template
Search
Target
Target list
https://example.com
Scan now
Share
Fork
azure-nsg-smtp-unrestricte
azure-nsg-ssh-unrestricted
azure-nsg-telnet-unrestrict
azure-nsg-udp-unrestrictec
azure-nsg-unrestricted-por
azure-openai-cmk-not-enal
azure-openai-managed-ide
azure-openai-private-endp
azure-openai-public-acces
azure-policy-assignment-cr
azure-policy-assignment-d
azure-policy-not-allowed-t
Scan output
View scan debug data

```
id: azure-openai-private-endpoints-unconfigured
info:
  name: Azure OpenAI Service Instances Not Using Private Endpoints
  author: princechaddha
  severity: high
  description: |
    To reduce the risk of exposure to external threats and strengthens overall
    security for your Azure OpenAI interactions, ensure that your Azure OpenAI
    service instances are accessed exclusively through private endpoint
    connections.
  impact: |
    Not using private endpoints for OpenAI service instances can expose them
    to external threats, compromising the security of your Azure environment.
  remediation: |
    Configure all Azure OpenAI service instances to use private endpoints to
    enhance security and ensure that these instances are not accessible over
    the public internet.
  reference:
    - https://docs.microsoft.com/en-us/azure/private-link/
    private-endpoint-overview
```

Figure 11-6
Community Template for Azure OpenAI Service Instances Not Using Private Endpoints

The Nuclei template in Figure 11-6 is designed to identify Azure OpenAI Service instances that are not configured to use private endpoints. It highlights that failing to use private endpoints can leave OpenAI service instances exposed to external attacks.

Figure 11-7 shows the interface of ProjectDiscovery's Cloud Platform, where Nuclei templates can be created using AI. In this example, the interface displays a vulnerability scan template related to an insecure direct object reference (IDOR) vulnerability.



```
Insecure Direct Object Reference (IDOR) vulnerability discovered in a website's user
profile page. Here's the PoC:

HTTP Request:


GET /profile?id=2 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Cookie: session=abcd1234

HTTP Response:

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1337
Server: Apache
```

Figure 11-7
AI-Generated Nuclei Vulnerability Scanner Template

Example 11-3 shows the full template generated by the AI feature illustrated in Figure 11-7.

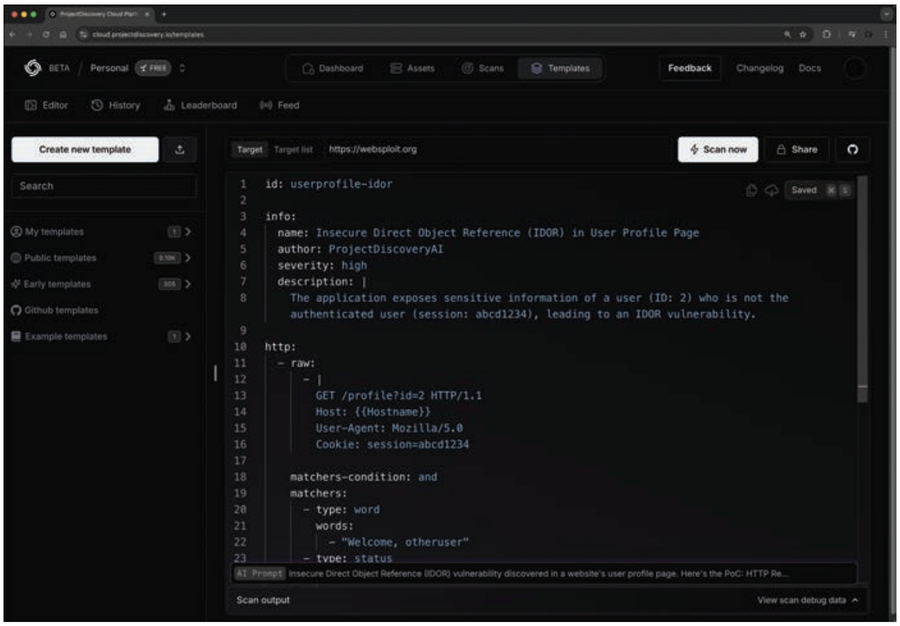
EXAMPLE 11-3 The AI-Generated Nuclei Scanner Template

```
id: userprofile-idor

info:
  name: Insecure Direct Object Reference (IDOR) in User Profile Page
  author: ProjectDiscoveryAI
  severity: high
  description: |
    The application exposes sensitive information of a user (ID: 2) who is not the
authenticated user (session: abcd1234), leading to an IDOR vulnerability.
```

```
http:
  - raw:
      - |
        GET /profile?id=2 HTTP/1.1
        Host: {{Hostname}}
        User-Agent: Mozilla/5.0
        Cookie: session=abcd1234

    matchers-condition: and
    matchers:
      - type: word
        words:
          - "Welcome, otheruser"
      - type: status
        status:
          - 200
```

The AI model used in Example 11-3 generated a proof-of-concept (PoC) template that provides an HTTP request to potentially find IDORs. The generated request targets a profile page **(/profile?id=2)**, indicating that by manipulating the id parameter in the URL, a user could access another user's profile.

After the template is generated, you can use it to scan a target application (websploit.org in the example shown in Figure 11-8).



https://websploit.org

```
id: userprofile-idor

info:
  name: Insecure Direct Object Reference (IDOR) in User Profile Page
  author: ProjectDiscoveryAI
  severity: high
  description: |
    The application exposes sensitive information of a user (ID: 2) who is not the
    authenticated user (session: abcd1234), leading to an IDOR vulnerability.

http:
  - raw:
      - |
        GET /profile?id=2 HTTP/1.1
        Host: {{Hostname}}
        User-Agent: Mozilla/5.0
        Cookie: session=abcd1234

    matchers-condition: and
    matchers:
      - type: word
        words:
          - "Welcome, otheruser"
      - type: status
```

AI Prompt Insecure Direct Object Reference (IDOR) vulnerability discovered in a website's user profile page. Here's the PoC: HTTP Re...

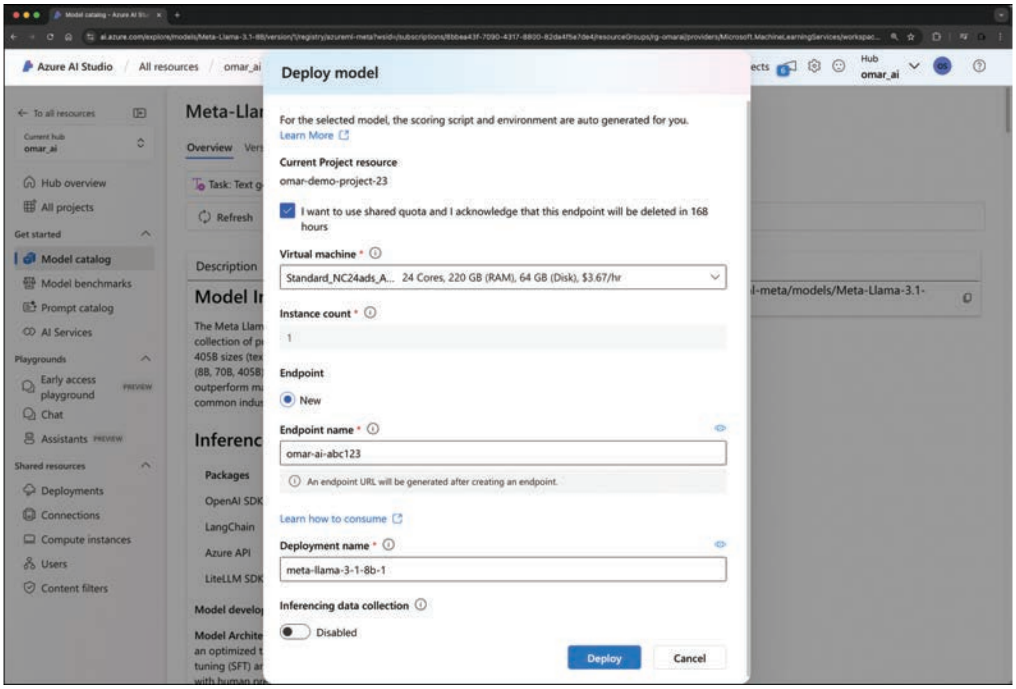
Figure 11-8
Scanning websploit.org Using the AI-Generated Template

AI Model Training, Fine-Tuning, and RAG for Bug Bounties

In Chapter 8, you learned about AI model training, fine-tuning, and RAG. As a security researcher participating in bug bounties, you can fine-tune and use RAG to enhance your ability to identify and report vulnerabilities.

Deploying AI Models

You can easily deploy AI models in cloud platforms such as Azure AI Studio, AWS Bedrock, and Google Cloud Vertex AI. Figure 11-9 shows a model deployment screen within Azure AI Studio, specifically for deploying the Meta-Llama model. The project resource is named omar-demo-project-23, which is the current environment in which the model will be deployed. The custom name for the endpoint is omar-ai-abc123. After deployment, the system will automatically generate an endpoint URL. The deployment is named meta-llama-3-1-8b-1 because it is based on the Meta-Llama 3.1 model using the 8-billion parameter version. Inferencing data collection is disabled, meaning the deployment will not collect data during inference runs.



Deploy model

For the selected model, the scoring script and environment are auto generated for you.
Learn More

Current Project resource
omar-demo-project-23

[✓] I want to use shared quota and I acknowledge that this endpoint will be deleted in 168 hours

Virtual machine *
Standard_NC24ads_A... 24 Cores, 220 GB (RAM), 64 GB (Disk), $3.67/hr

Instance count *
1

Endpoint
● New

Endpoint name *
omar-ai-abc123
An endpoint URL will be generated after creating an endpoint.

Learn how to consume

Deployment name *
meta-llama-3-1-8b-1

Inferencing data collection
Disabled

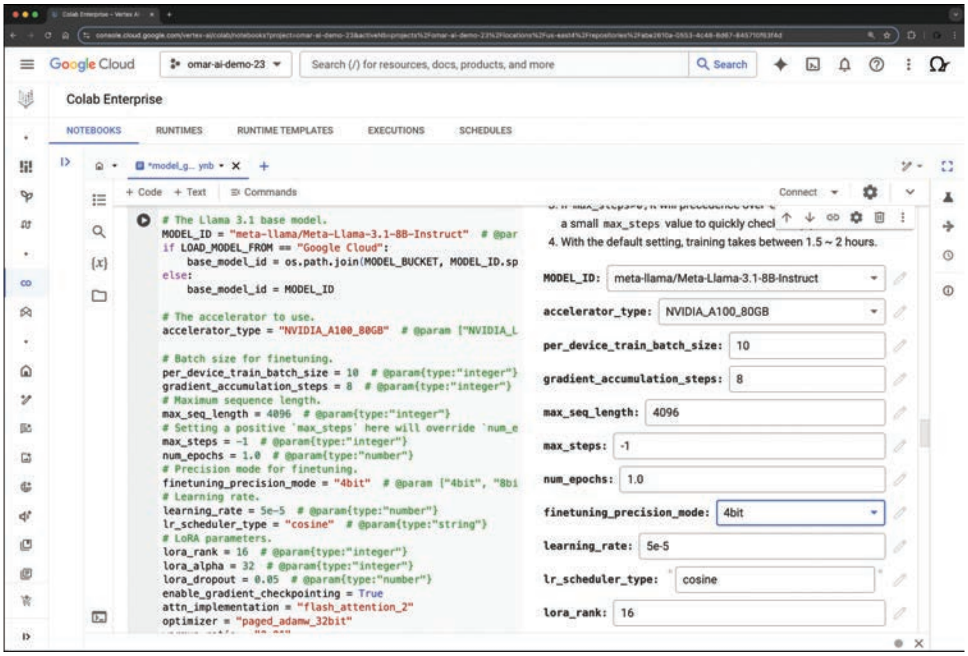
Deploy Cancel

Figure 11-9
Azure AI Studio Model Deployment Screen

Fine-Tuning AI Models

Fine-tuning involves adjusting a pretrained model using a specific dataset to enhance its performance for particular tasks. This process is invaluable for bug hunters because it allows you to tailor AI models to recognize specific types of vulnerabilities or security issues unique to certain software or systems. By fine-tuning models with data from known vulnerabilities, you can improve the accuracy and relevance of your automated tools and potentially AI agents.

Figure 11-10 shows the interface of Google Cloud's Vertex AI and its Colab Enterprise environment. This notebook is used for fine-tuning the Meta-Llama 3.1 8B-Instruct model.



```
# The Llama 3.1 base model.
MODEL_ID = "meta-llama/Meta-Llama-3.1-8B-Instruct"  # @par
if LOAD_MODEL_FROM == "Google Cloud":
    base_model_id = os.path.join(MODEL_BUCKET, MODEL_ID.sp
else:
    base_model_id = MODEL_ID

# The accelerator to use.
accelerator_type = "NVIDIA_A100_80GB"  # @param ["NVIDIA_L

# Batch size for finetuning.
per_device_train_batch_size = 10  # @param{type:"integer"}
gradient_accumulation_steps = 8  # @param{type:"integer"}
# Maximum sequence length.
max_seq_length = 4096  # @param{type:"integer"}
# Setting a positive `max_steps` here will override `num_e
max_steps = -1  # @param{type:"integer"}
num_epochs = 1.0  # @param{type:"number"}
# Precision mode for finetuning.
finetuning_precision_mode = "4bit"  # @param ["4bit", "8bi
# Learning rate.
learning_rate = 5e-5  # @param{type:"number"}
lr_scheduler_type = "cosine"  # @param{type:"string"}
# LoRA parameters.
lora_rank = 16  # @param{type:"integer"}
lora_alpha = 32  # @param{type:"integer"}
lora_dropout = 0.05  # @param{type:"number"}
enable_gradient_checkpointing = True
attn_implementation = "flash_attention_2"
optimizer = "paged_adamw_32bit"
```

a small max_steps value to quickly chec
4. With the default setting, training takes between 1.5 ~ 2 hours.

MODEL_ID: meta-llama/Meta-Llama-3.1-8B-Instruct
accelerator_type: NVIDIA_A100_80GB
per_device_train_batch_size: 10
gradient_accumulation_steps: 8
max_seq_length: 4096
max_steps: -1
num_epochs: 1.0
finetuning_precision_mode: 4bit
learning_rate: 5e-5
lr_scheduler_type: cosine
lora_rank: 16

Figure 11-10
Fine-Tuning AI Models in Google Vertex AI

In Figure 11-10, the model is being run on an NVIDIA A100 GPU, as specified in the **accelerator_type** parameter.

The following are additional details about the fine-tuning parameters shown:

- **Batch Size**: The **per_device_train_batch_size** is set to 10, meaning each GPU will process one batch at a time.
- **Gradient Accumulation Steps**: This parameter is set to 8, meaning gradients will accumulate over 8 batches before updating the model weights, helping manage memory constraints.
- **Max Sequence Length**: The model will process sequences of up to 4096 tokens at a time.

- **Max Steps**: This parameter is set to −1, which suggests there is no limit on the number of steps for training.
- **Epochs**: Training will run for 1.0 epoch, meaning the entire dataset will be passed through the model once.
- **Precision**: Fine-tuning is performed using 4-bit precision (fine-tuning_precision_mode), which helps reduce the memory usage while still being performant.
- **Learning Rate**: The learning rate is set to 5e-5 (0.00005), which controls how much the model adjusts weights with each training step.
- **Learning Rate Scheduler**: The learning rate will follow a "cosine" decay schedule, gradually reducing as training progresses.

LoRA and QLoRA

Low-rank adaptation (LoRA) is a machine learning technique designed to efficiently fine-tune language models by introducing trainable low-rank matrices into each layer of a pretrained model while keeping the original weights frozen. This approach significantly reduces the number of trainable parameters, making the process faster and more memory-efficient without compromising performance. *QLoRA* builds on LoRA by incorporating 4-bit quantization, which further reduces memory usage and enables the fine-tuning of extremely large models, such as those with 65 billion parameters, on hardware with limited resources like a single 48GB GPU. This method maintains high performance levels comparable to full 16-bit fine-tuning by combining quantization with low-rank adapters, thus democratizing access to efficient model tuning.

The following are the LoRA parameters used in the example shown in Figure 11-10:

- **LoRA Rank**: Set to 16, this refers to the rank in the low-rank adaptation technique, commonly used to fine-tune large models efficiently by focusing only on specific layers.
- **LoRA Alpha**: Set to 32, this controls the scaling factor for LoRA, which influences the weight adjustment during fine-tuning.
- **LoRA Dropout**: Set to 0.05, meaning 5 percent of the neurons are randomly dropped during training to prevent overfitting.

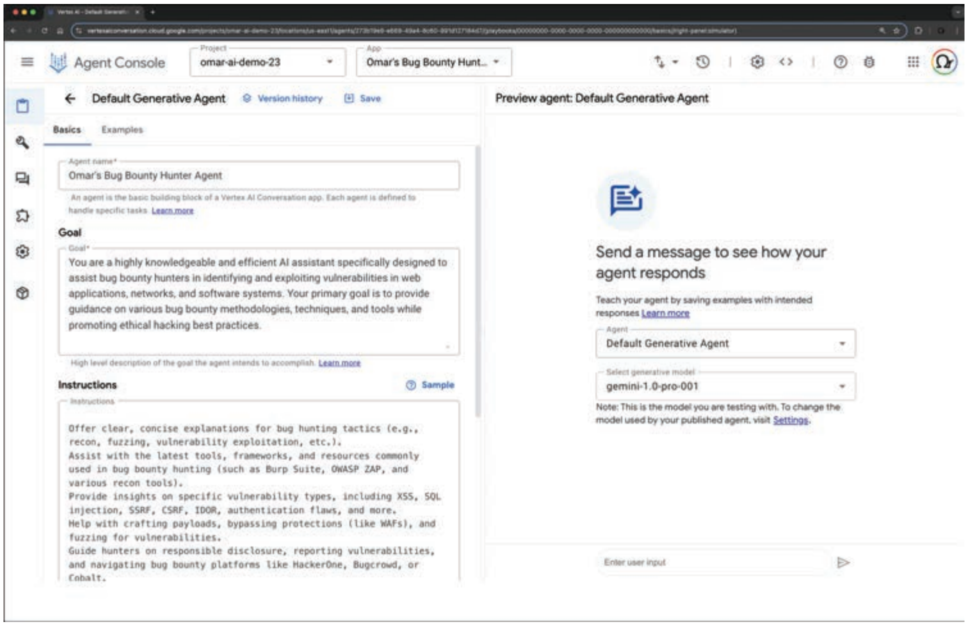
There are many other fine-tuning settings, but the following are displayed in Figure 11-10:

- **Gradient Checkpointing**: Uses Enabled (True), which helps save memory by recomputing some layers during the backward pass.
- **Attention Implementation**: Uses flash_attention_2, a more memory-efficient attention mechanism that allows for faster processing of large sequences.

Using RAG and AI Agents for Bug Bounty Hunting

RAG combines the capabilities of language models with data retrieval from vector databases and other sources, providing contextually relevant information that enhances the model inference and reduces the likelihood of hallucinations or confabulations. For bug bounty hunters, RAG can be used to access up-to-date information about potential vulnerabilities or exploits as they emerge.

Many different tools such as LlamaIndex, LangChain, LangGraph, and cloud services such as Google Vertex AI can help you build RAG implementations and AI agents. Figure 11-11 shows the Google Vertex AI Agent Console, which is used for building and managing AI agents. A custom AI agent named Omar's Bug Bounty Hunter Agent is being configured.



Agent name*
Omar's Bug Bounty Hunter Agent

An agent is the basic building block of a Vertex AI Conversation app. Each agent is defined to handle specific tasks. Learn more

Goal

Goal*
You are a highly knowledgeable and efficient AI assistant specifically designed to assist bug bounty hunters in identifying and exploiting vulnerabilities in web applications, networks, and software systems. Your primary goal is to provide guidance on various bug bounty methodologies, techniques, and tools while promoting ethical hacking best practices.

High level description of the goal the agent intends to accomplish. Learn more

Instructions

```
Offer clear, concise explanations for bug hunting tactics (e.g.,
recon, fuzzing, vulnerability exploitation, etc.).
Assist with the latest tools, frameworks, and resources commonly
used in bug bounty hunting (such as Burp Suite, OWASP ZAP, and
various recon tools).
Provide insights on specific vulnerability types, including XSS, SQL
injection, SSRF, CSRF, IDOR, authentication flaws, and more.
Help with crafting payloads, bypassing protections (like WAFs), and
fuzzing for vulnerabilities.
Guide hunters on responsible disclosure, reporting vulnerabilities,
and navigating bug bounty platforms like HackerOne, Bugcrowd, or
Cobalt.
```

Send a message to see how your agent responds

Teach your agent by saving examples with intended responses Learn more

Agent: Default Generative Agent

Select generative model: gemini-1.0-pro-001

Note: This is the model you are testing with. To change the model used by your published agent, visit Settings.

Enter user input

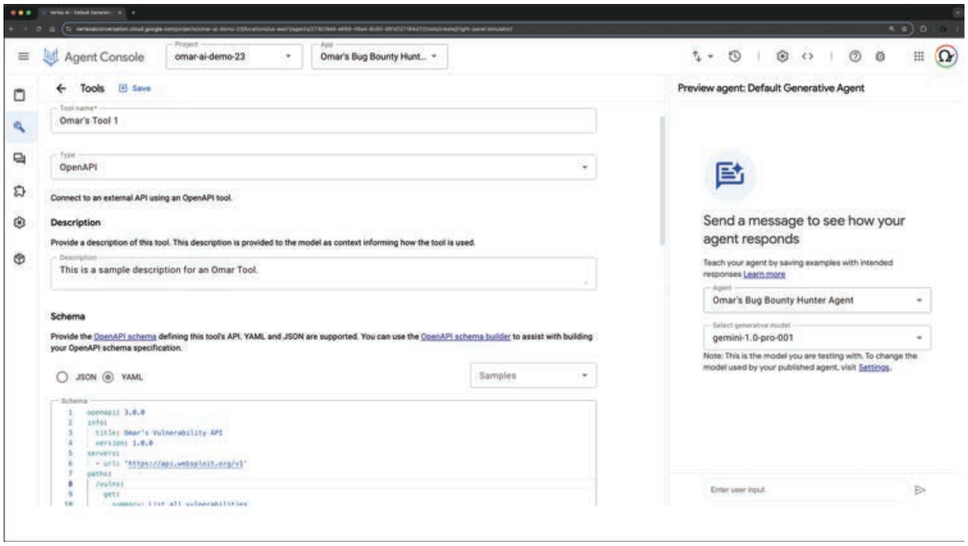
Figure 11-11
Creating AI Agents in Google Vertex AI

The configured "Goal" provides a description of the agent's purpose, which is to assist bug bounty hunters in identifying and exploiting vulnerabilities while promoting ethical hacking practices. The "Instructions" section provides guidelines for the agent's behavior. These instructions include offering explanations for bug hunting tactics, assisting with tools and frameworks, providing insights on various vulnerability types, and guiding responsible disclosure.

You can also configure data stores (using the built-in vector database) to use your data and very easily create a RAG deployment. These Vertex AI agents feature special state handlers known as *data store handlers*. These handlers allow the data store agent to engage in conversations with end users about the stored content.

Tool Calling

You can also link your agent to external tools (*tool calling*) and enable it to perform tasks more effectively. Figure 11-12 shows an example of tool calling in Google Vertex AI.



omar-ai-demo-23

Omar's Bug Bounty Hunt...

Tool name*: Omar's Tool 1

Type: OpenAPI

Connect to an external API using an OpenAPI tool.

Description

Provide a description of this tool. This description is provided to the model as context informing how the tool is used.

Description: This is a sample description for an Omar Tool.

Schema

Provide the OpenAPI schema defining this tool's API. YAML and JSON are supported. You can use the OpenAPI schema builder to assist with building your OpenAPI schema specification.

JSON [✓ YAML] Samples

```
openapi: 3.0.0
info:
  title: Omar's Vulnerability API
  version: 1.0.0
servers:
  - url: "https://api.websploit.org/v1"
paths:
  /vulns:
    get:
      summary: List all vulnerabilities
```

Send a message to see how your agent responds

Teach your agent by saving examples with intended responses Learn more

Agent: Omar's Bug Bounty Hunter Agent

Select generative model: gemini-1.0-pro-001

Note: This is the model you are testing with. To change the model used by your published agent, visit Settings.

Figure 11-12
Tool Calling in Google Vertex AI

Use Case: Improving Reporting and Documentation

RAG and AI-driven tools can significantly enhance the quality, consistency, and efficiency of vulnerability reporting and documentation in the context of bug bounty hunting, red teaming, and traditional penetration testing. These advanced systems can take basic input about a discovered vulnerability and generate a structured report template, including standard sections like executive summary, technical details, impact assessment, and remediation recommendations. Automation can help you ensure that no crucial elements are overlooked in the reporting process.

You learned that when it comes to vulnerability severity assessment, AI can analyze the vulnerability details against databases of known issues to suggest a severity rating based on frameworks like CVSS and EPSS. The AI can then justify the severity score, explaining which factors contributed most significantly to the rating. This approach provides a clear, objective basis for prioritizing vulnerabilities.

One of the most valuable features of AI in this context is its ability to expand brief notes into clear, step-by-step reproduction instructions. It can suggest additional edge cases or variations to test, ensuring comprehensive coverage of the vulnerability. For common vulnerability types, AI could even generate or suggest basic POC code snippets using tools like Cursor or GitHub Copilot.

Based on the vulnerability type and affected systems, AI can offer specific remediation recommendations using your data that could include best practices and other insights. It can prioritize these recommendations and estimate the effort required for each, helping your client design their mitigation plans effectively. Of course, AI can also improve the clarity and professionalism of your writing and fix grammatical issues. It can adjust the technical level of the content for different audiences, making the report accessible to both technical and nontechnical stakeholders.

AI can help you tailor the report to address stakeholders' specific concerns and technical stack. This customization makes the report more relevant and actionable for the client. AI can also assist in managing different versions of the report, highlighting changes and allowing for collaborative editing with comments and suggestions. You can leverage AI in this comprehensive manner to produce more thorough, professional, and actionable reports in less time.

Challenges of Using AI for Bug Bounty Hunting

Not everything is perfect with AI. There are several challenges when you integrate AI into bug bounty hunting, especially with AI models that have guardrails or are censored.

Censored Models and Guardrails

Censorship and *guardrails* refer to the restrictions imposed on AI models to prevent them from generating harmful, unethical, or legally problematic content. While these guardrails are essential for ensuring ethical AI usage, they can impede bug bounty hunting efforts in several ways:

- **Limited Payload Generation**: AI models with strong guardrails may refuse to create or assist in crafting potentially malicious payloads, even when these are essential for security testing. For example, if you're trying to identify SQL injection or cross-site scripting (XSS) vulnerabilities, an AI model might block requests to generate queries or scripts that it deems dangerous, even if they are being used ethically within a penetration testing environment.
- **Restricted Vulnerability Testing**: Many censored models are programmed to avoid generating code or commands that could be used in hacking or exploitation scenarios. In bug bounty hunting, ethical hackers need to test for security flaws like buffer overflows, file inclusions, or even zero-day vulnerabilities. AI models with tight restrictions may prevent access to code snippets, payloads, or scripts needed to identify these flaws. This is why models like White Rabbit Neo are becoming very popular among the cybersecurity community.
- **False Positives/Negatives**: Guardrails and censoring may lead to false negatives, where the model fails to help detect a valid vulnerability in scanners, fuzzers, and other systems because it is avoiding potentially "dangerous" outputs.

Hallucinations or Confabulations

Hallucinations or *confabulations* in the context of AI refer to instances where the AI model generates information that is factually incorrect, made up, or not grounded in reality. This is an important concept to understand when working with AI systems, especially in critical areas like security research and bug bounty hunting. Let me elaborate. Hallucinations occur when an AI model produces content that seems plausible but is actually false or nonsensical. This can happen because AI models are trained on vast amounts of data and learn to generate human-like text, but don't have true understanding or real-world knowledge in the way humans do. AI models make statistical associations between words and concepts, which can sometimes lead to incorrect outputs. In bug bounty hunting, hallucinations can present serious challenges, especially when accuracy and precision are critical. This is why "grounding" by using techniques such as RAG is really important.

The following are a few examples of hallucinations or confabulations in ethical hacking scenarios:

- **Incorrect Vulnerability Reporting**: AI models prone to hallucinations may identify vulnerabilities that do not exist. This could lead to bug bounty hunters wasting significant time chasing false vulnerabilities that were never present in the system.
- **Faulty Code or Exploits**: If an AI hallucinates while generating exploitation scripts or proofs of concept, it may provide incorrect or incomplete code, leading to failed attempts to reproduce vulnerabilities or broken exploitation logic. For example, an AI might hallucinate a new method to exploit a flaw, but the code it generates is nonfunctional or dangerous to the system itself.
- **Misleading Contextual Understanding**: AI hallucinations can also affect how models interpret security-related context. For example, when analyzing complex security protocols or responses from a web application firewall (WAF) or a cloud security application, an AI model might generate completely fabricated conclusions or explanations, misleading a bug hunter into wrong assumptions about the system's behavior.
- **Incomplete Testing Flows**: AI models, due to guardrails or insufficient training on adversarial tasks, may fail to complete multistep attack scenarios that are essential for bug bounty hunting. These multistep attacks often involve intricate chains of vulnerabilities that require creative, out-of-the-box thinking, which current AI models might not be equipped for.
- **Inability to Handle Zero-Days**: AI models may struggle to discover zero-day vulnerabilities, especially since they rely on training data. Zero-days are unknown vulnerabilities, meaning no data exists to train models on these flaws. This is a significant limitation when trying to use AI in environments where cutting-edge or undiscovered vulnerabilities are the primary focus.

Compliance with Bug Bounty Guidelines

Many bug bounty programs have specific rules about how vulnerabilities should be tested or reported. If AI-generated exploits or tests do not align with these guidelines, the hunter risks violating the program's terms, which can lead to disqualification or legal ramifications.

Test Your Skills

Multiple-Choice Questions

1. Which of the following is NOT a primary goal in the framework for external attack surface discovery and exploitation?

   A. Discover

   B. Monitor

   C. Actionable insights

   D. Patch management

2. What is the purpose of the Common Vulnerability Scoring System (CVSS)?

   A. To predict the likelihood of a vulnerability being exploited

   B. To communicate the characteristics and severity of software vulnerabilities

   C. To list vulnerabilities that have been exploited in the wild

   D. To generate exploit code for known vulnerabilities

3. What does EPSS stand for in the context of vulnerability assessment?

   A. Exploit Prediction Scoring System

   B. External Penetration Security System

   C. Enhanced Protection Scanning Service

   D. Exploit Prevention Scoring Standard

4. Which organization maintains the Known Exploited Vulnerabilities (KEV) Catalog?

   A. NIST

   B. OWASP

   C. CISA

   D. ISO

5. What is the primary purpose of the ProjectDiscovery Nuclei Scanner?

   A. To generate AI models for vulnerability detection

   B. To create vulnerability reports

   C. To identify and mitigate security vulnerabilities across various platforms

   D. To manage bug bounty programs

6. What type of file format does Nuclei use for its templates?

   A. STIX

   B. XML

   C. YAML

   D. TXT

7. What is LoRA in the context of AI model fine-tuning?

   A. Low-range analysis

   B. Low-rank adaptation

   C. Linear optimization for rapid advancement

   D. Logarithmic rational approximation

8. What does RAG stand for in the context of AI and bug bounty hunting?

   A. Rapid attack generation

   B. Retrieval-augmented generation

   C. Risk assessment guide

   D. Responsive AI governance

9. What is a potential challenge of using AI models with strong guardrails in bug bounty hunting?

   A. Increased accuracy in vulnerability detection

   B. Faster processing of large datasets

   C. Limited ability to generate potentially malicious payloads for testing

   D. Improved compliance with bug bounty program guidelines

10. What is a confabulation in the context of AI-assisted bug bounty hunting?

    A. A visual glitch in the AI interface

    B. A sudden increase in processing speed

    C. Generation of factually incorrect or made-up information

    D. A temporary loss of connection to the AI service

11. Which of the following is NOT a metric group in the Common Vulnerability Scoring System (CVSS) version 4.x?

    A. Base

    B. Threat

C. Environmental

D. Exploitation

12. What is the primary advantage of using QLoRA over traditional LoRA?

A. It allows for fine-tuning of larger models on limited hardware.

B. It increases the number of trainable parameters.

C. It eliminates the need for GPU acceleration.

D. It provides more accurate results in all scenarios.

13. What is the primary purpose of tool calling in AI agents for bug bounty hunting?

A. To generate more accurate vulnerability reports

B. To enable the AI to perform tasks using external tools

C. To improve the AI's natural language processing capabilities

D. To create new exploits automatically using an IDE

14. What is the purpose of gradient accumulation in the context of AI model fine-tuning?

A. To increase the learning rate

B. To reduce memory constraints

C. To improve model accuracy

D. To speed up the training process

Index