Introduction

Red Teaming is a form of simulated cyberattack conducted by authorized security professionals to test an organization's defenses. Unlike traditional penetration testing that often focuses on finding as many vulnerabilities as possible in specific systems, Red Team operations take a holistic adversarial approach – emulating real threat actors to challenge the entire security posture. In large enterprise environments, this means not only testing on-premises networks and Windows domains, but also cloud services, identity platforms, and management tools that modern businesses rely on. The goal is to identify gaps in people, processes, and technology by staging covert, full-scope attacks that mirror advanced persistent threats. This proactive, *adversary simulation* approach helps organizations uncover critical weaknesses before malicious actors do, strengthening defenses in an era of increasingly sophisticated breaches.

Modern enterprise infrastructure has evolved to include complex hybrid-cloud systems, so Red Team assessments must adapt accordingly. Attackers today are as interested in cloud credentials and access tokens as they are in on-premises servers

A comprehensive Red Team engagement in an Azure-centric enterprise will span both Azure Active Directory (AAD) and on-prem Active Directory, leverage cloud services like Intune or ServiceNow, and target high-value secrets stored in enterprise vaults. In essence, Red Teamers need to think like an attacker who can pivot between cloud and on-premises, exploiting trust relationships and misconfigurations. This paper provides a PhD-level deep dive into Red Team strategies for large enterprises focusing on Azure infrastructure (including AAD, Microsoft Intune, and even SaaS like ServiceNow), critical secret management tools (CyberArk, Quest/Dell PPM, HashiCorp Vault, Venafi), and Windows environments that interface with Azure. We will outline methodologies, specific attack tactics, key objectives (the "attack paths" and ultimate goals), the crown-jewel targets adversaries seek, and a comparative analysis of secret-management solutions' security. Real-world inspired scenarios (leveraging the Cobalt Strike framework for command-and-control) will illustrate how these attacks can unfold in practice, followed by a brief discussion on defensive mitigations.

By understanding the tactics and techniques involved in these areas, security professionals and researchers can better prepare and fortify enterprise systems against determined attackers. A blend of industry knowledge, case studies, and trusted research is used to ensure accuracy and credibility in the guidance that follows.

Methodology

A Red Team assessment in a large enterprise setting follows a structured methodology that mimics the full kill chain of a targeted attack. This typically involves several phases: **reconnaissance**, **initial access**, **establishing persistence**, **privilege escalation**, **lateral movement**, **collection/exfiltration**, and **action on objectives** (the end goals). In practice, these phases do not always occur linearly – skilled adversaries pivot fluidly as opportunities arise – but a methodical plan ensures coverage of all potential weaknesses. Below we describe the general tactics and approach a Red Team might use when assessing an enterprise heavy on Azure and related technologies:

- **Reconnaissance**: The team quietly gathers open-source intelligence (OSINT) and internal insight about the target's footprint. This includes mapping external facing assets, enumerating domains and subdomains, identifying employee emails or credentials leaked online, and profiling technologies in use. For an Azure-focused enterprise, recon might reveal things like public Azure services (web apps, storage accounts), metadata about the AAD tenant, or exposed endpoints for services like ServiceNow. Social engineering intelligence is also gathered – for example, names of administrators or patterns in helpdesk processes – to inform phishing or pretexting later. Effective recon provides the initial attack surface and potential weak entry points without alerting the target.
- **Initial Access**: Using the recon data, Red Team operators attempt to gain a foothold. Common tactics are spear-phishing emails (or **smishing** messages to mobile phones) that trick users into revealing credentials or running malware, exploitation of vulnerable internet-facing applications, or leveraging stolen credentials found in breach data. In cloud-heavy environments, stealing a single set of valid credentials can be as impactful as malware. For instance, obtaining an employee's Office 365/Azure AD credentials (perhaps via a convincing O365 login phishing page) and bypassing multi-factor authentication can grant direct access to cloud resources

As one case demonstrated, a single infostealer-compromised password allowed an attacker to breach a corporate ServiceNow tenant and access sensitive internal data

If malware deployment is used, the Red Team might deploy a Cobalt Strike **Beacon** payload on a compromised machine, which calls back to a command-and-control (C2) server. This Beacon provides remote access for the team to issue further commands stealthily.

- **C2 and Persistence**: Once inside, maintaining access is crucial. The team will establish **persistence** mechanisms to survive host reboots or user logouts. With Cobalt Strike, this could involve installing the beacon as a service, scheduling tasks, or leveraging startup registry keys on Windows. In Azure/AAD contexts, persistence might also mean creating or backdooring credentials in the cloud – for example, adding a new **application secret** or cloud user account that the team controls. The methodology often emphasizes stealth: using living-off-the-land techniques (abusing legitimate admin tools and scripts) and carefully shaping network traffic to evade detection by the Blue Team. Encrypted C2 channels (HTTPS or named pipes, etc.) and low-and-slow operation help remain undetected. Throughout this phase, the Red Team also performs **internal reconnaissance** – enumerating directory structures, group memberships, roles, and network topology from the inside. In an Azure-integrated environment, this might include querying Azure AD for user and role information, checking Intune device lists, or identifying privileged accounts in ServiceNow or the secret vault systems.
- **Privilege Escalation and Lateral Movement**: After the initial foothold (which might be a low-level user account or a single machine), the team seeks higher privileges and broader access. In a Windows domain, this involves techniques like credential dumping (using tools like Mimikatz) to harvest hashes or tokens, exploiting misconfigurations (e.g. weak ACLs on AD objects), or leveraging enterprise solutions against themselves. For example, if the compromised user's machine is Azure AD **hybrid-joined** (joined to both on-prem AD and Azure AD), the team might extract the user's Primary Refresh Token (PRT) from memory

A PRT is an AAD token used for single sign-on; with it, an attacker can impersonate the user in Azure cloud services, potentially accessing cloud apps without needing the password again. This kind of token theft blurs the line between on-prem and cloud privilege escalation – an attacker pivoting from a Windows host to cloud resources. On the cloud side, privilege escalation might mean exploiting poorly configured Azure roles (for instance, a user who is an owner of a highly privileged application, as we'll explore) or abusing API permissions. The Red Team may target Azure AD Connect (if present) to extract credentials that synchronize on-prem AD with Azure AD, as this connector often holds a powerful account. On-premise, escalating to Domain Admin remains a classic objective; in Azure, the counterpart is Global Administrator or equivalent high-privilege roles. Lateral movement entails hopping from the initial beachhead to other systems: maybe using pass-the-hash or pass-the-ticket in a Windows domain, or using a compromised Intune admin account to push malicious software to many endpoints at once. All these tactics are executed carefully to avoid detection, often using Cobalt Strike's built-in tools or custom scripts delivered through the Beacon.

- **Actions on Objectives**: Finally, the Red Team works toward the explicit goals of the engagement – the "crown jewels" defined by the scenario. This could be obtaining certain sensitive data (e.g. exfiltrating database records or confidential files), gaining full control of core infrastructure (like domain controllers, Azure subscriptions, or vault systems), or simulating destructive actions (ransomware deployment, data corruption) depending on the scope. Upon reaching this phase, the team documents how they achieved each objective. They may demonstrate access to a **"keys to the kingdom"** asset such as extracting all passwords from a CyberArk vault, or downloading all secrets from HashiCorp Vault, or proving they can reset accounts via ServiceNow with admin access. In stealthy red team operations, it's common to avoid actually doing harm – instead, taking screenshots or hashes as proof of access. Throughout, the methodology requires maintaining operational security: covering tracks by clearing logs where possible, using credentials in ways that might blend in (for example, using a compromised admin's account during their working hours to appear normal), and constantly adapting when defensive measures are encountered.

This methodology is underpinned by frameworks like **MITRE ATT&CK**, which enumerate tactics and techniques that map to each stage of an attack. The Red Team uses such frameworks to ensure they emulate a broad range of threat behaviors and to later organize findings. By adhering to a disciplined methodology, the assessment can systematically reveal how an advanced attacker could infiltrate and navigate the enterprise – from cloud services down to the desktop, and vice versa – despite the organization's security controls.

Attack Tactics

In this section, we delve into specific attack tactics relevant to the enterprise technologies in focus. Each subsection highlights the techniques an adversary (or Red

Team) might use to exploit Azure infrastructure, endpoint management solutions like LAPS and Intune, IT service platforms like ServiceNow, secret management tools, and the interplay between Azure and Windows environments. These represent the "playbook" of attacks that a Red Team would employ during an engagement.

Azure and AAD Attacks

Microsoft Azure and Azure Active Directory (Entra ID) present a rich target set for attackers, especially as organizations migrate critical workloads to the cloud. AAD is the identity backbone for Office 365 and Azure, so compromising it can effectively grant the keys to an enterprise's cloud kingdom. Red Teamers therefore pursue Azure/AAD attacks to escalate privileges in the cloud tenant and pivot to other systems. Key tactics include:

- **Privilege Escalation via Azure AD Applications**: Azure AD allows registration of applications which have service principal identities. If an application's service principal is granted a high-privilege role (e.g. Global Administrator in AAD), and a low-privileged user account is an "Owner" of that application, this is a ripe target. The Red Team can abuse this by adding a new credential to the service principal (such as creating a client secret or uploading a certificate) via the owner user. Owners have the ability to generate app secrets

In one documented case, a low-privileged user was owner of an application whose service principal had Global Admin rights; the attacker added a secret and then logged in as that service principal, instantly gaining Global Administrator access in Azure AD

From there, the attacker effectively controlled the cloud directory. This technique exploits the misconfiguration of Azure AD application permissions and ownerships.

- **Abusing Default OAuth Consent and API Permissions**: In Azure AD, certain API permissions, if granted to a malicious app, can lead to privilege escalation or persistence. For example, an attacker who tricks a Global Admin into consenting to a rogue enterprise application or finds a way to create an application with elevated *application permissions* could bypass many traditional controls. The **Application.ReadWrite.All** permission allows creating or updating any app (and adding credentials to them), which could enable the attacker to impersonate higher-privileged apps

Similarly, **DeviceManagement** permissions can be dangerous – for instance, an app with *DeviceManagementConfiguration.ReadWrite.All* can deploy configurations to Intune-managed devices, effectively running code on devices (we'll see more on Intune shortly) Red Teams test for apps or service principals in the tenant with overly broad permissions, as these can serve as stealthy backdoors (service principals don't trigger MFA and aren't human users, so they often evade notice).

- **Token Stealing (Session Tokens and Refresh Tokens)**: Instead of directly attacking cloud infrastructure, attackers often target the *tokens* that grant access. A tactic rising in popularity is extraction of Azure AD tokens from compromised devices. For example, on a compromised Windows machine, the adversary can use tools like **Mimikatz** to retrieve a user's Primary Refresh Token (PRT) for Azure AD

With a valid PRT and associated session key material, the attacker can request access tokens for various Azure services as that user, essentially bypassing the need to authenticate. This allows lateral movement *into* cloud resources from an on-prem compromise. Likewise, if attackers capture OAuth refresh tokens (perhaps from a token cache file or memory), they can continue accessing cloud services even after a password change. Red Teams will attempt to steal session tokens from memory or illicitly leverage browser cookies and tokens stored on disk (TokenCache.dat for Azure PowerShell, etc.); these stolen credentials can be replayed to impersonate users in the Azure portal or Azure CLI.

- **Azure AD Connect and Hybrid Identity Exploitation**: In enterprises with on-prem Active Directory linked to Azure AD (hybrid identity), Azure AD Connect is a critical service. It often uses a dedicated account (sometimes called **MSOL_*** account) with directory synchronization privileges. Attackers target Azure AD Connect servers to extract the stored credentials. For example, by accessing Azure AD Connect's SQL database and decrypting the stored password (using the service's binaries or scripts), a Red Team can retrieve the plaintext password of the Azure AD sync account

- In some configurations, this account might have high privileges in Azure (in older setups it could even be a Global Admin or have directory write capabilities). By cracking Azure AD Connect, the Red Team gains another path to Azure AD compromise from on-prem. Additionally, misconfigurations like password-hash sync and pass-through authentication can be tested – e.g. capturing credentials if pass-through agents are compromised, or leveraging on-prem domain admin to inject forged credentials that sync to the cloud. The general principle is that the intersection of on-prem AD and Azure AD is a fertile ground for attacks; trust relationships (like federation or synchronization) can be manipulated.

- **Cloud Admin Misconfigurations**: Red Teams also look for configuration mistakes in Azure tenant setup. This includes things like unused Global Administrator accounts (that could be quietly compromised via password spraying), lack of multi-factor authentication on admin roles, or the presence of legacy authentication protocols that allow bypassing MFA. They will attempt password spray attacks against Azure AD accounts, exploitation of legacy endpoints (like using SMTP/IMAP with old credentials), or try well-known attacks such as **Golden SAML** (if ADFS is used for Azure – forging authentication tokens if they compromise ADFS). In pure Azure AD, Golden SAML is not applicable, but similar concepts exist for forging tokens if one gains the signing certificates for AAD applications or the tenant. Another target is **Azure AD Conditional Access** policies – if misconfigured, a Red Team might find that certain networks or apps don't enforce MFA, providing an easier route in.

In summary, Azure/AAD attacks revolve around abusing identity and access management weaknesses. Because Azure AD's roles and app model can be complex, attackers often find subtle misconfigurations that yield privilege escalation (like the service principal example) or persistence. With Azure in play, a Red Team's purview extends to any cloud resource accessible via compromised credentials – from Azure Key Vaults to storage accounts – but identity is typically the first focus. Once Global Admin is obtained, the attacker can leverage that to grant themselves roles in Azure subscriptions (via the User Access Administrator role toggle, or API calls) , thus pivoting from Azure AD control to full Azure resource control as well. Essentially, owning Azure AD can equate to owning the cloud environment, making it a primary target.

LAPS Exploitation (Local Administrator Password Solution)

Microsoft's Local Administrator Password Solution (LAPS) is a security feature designed to periodically randomize and securely store the local Administrator passwords of domain-joined Windows machines in Active Directory. It mitigates the risk of lateral movement via shared local admin credentials by ensuring each machine has a unique password that domain admins or authorized users can retrieve when needed. However, if not implemented with least privilege in mind, LAPS can itself become an avenue for attack. A Red Team will examine how LAPS is deployed in the enterprise to identify any weaknesses to exploit. Key tactics include:

- **Abusing Privileged Access to LAPS Passwords**: In a proper LAPS deployment, only specific security principals (like a dedicated AD group for helpdesk or system administrators) have permission to read the confidential attribute (typically ms-MCS-AdmPwd) on computer objects that stores the password. Red Teams will attempt to compromise any user that has these rights. For example, if a helpdesk user account with rights to view LAPS passwords is phished or its NTLM hash is captured, the team can retrieve the LAPS-managed password for targeted hosts

There have been real cases where members of built-in groups like "Account Operators" inadvertently had read access to LAPS passwords for many computers An attacker who gains such an account can immediately escalate to local administrator on those machines by grabbing the ms-MCS-AdmPwd value from AD. This turns what would have been a single-account compromise into potential control over numerous endpoints.

- **LDAP Relay to Dump LAPS Passwords**: A more advanced technique is leveraging network attacks to obtain LAPS secrets. If the Red Team can coerce a privileged user (say, a helpdesk admin) to authenticate to a machine under their control (using tricks like LLMNR/NBNS poisoning or malicious UNC paths), they might capture an NTLM challenge-response. Tools like **Impacket's ntlmrelayx** can relay an NTLM authentication to LDAP on a domain controller and perform actions as that user. Praetorian researchers added a module to dump LAPS passwords via such relayed credentials

In essence, even without knowing the cleartext password of a privileged user, the team could use the one-time authenticated session to query AD for any LAPS password that the user has rights to read

This allows a stealthy extraction of local admin creds across many machines. After dumping, Red Team operators will have admin access on those machines, which facilitates lateral movement (for example, use those creds to RDP or run remote commands on the systems).

- **Targeting LAPS Client-Side**: Another angle is on the endpoint. LAPS periodically runs a process on each domain-joined system to update the password in AD. By default, that process runs as SYSTEM. If an attacker already has code execution on a Windows host (but not admin), they might try to intercept the LAPS process or its stored information. However, LAPS is fairly secure on the endpoint (password not stored plaintext, it's immediately sent to AD and then cleared). Still, a misconfiguration on a client (like storing the password in event logs or improper permissions on the LAPS binary) could be checked, though such issues are not common in modern LAPS deployments.

- **Bypassing LAPS**: If LAPS is properly implemented, it actually hardens the environment significantly by preventing the reuse of local admin passwords

Attackers then might try to *bypass* the need for those creds entirely via other means (for instance, using pass-the-hash with captured NTLM hashes if any machine still shares a local account, or using token impersonation on already compromised admin sessions). But a Red Team will note if LAPS is not universally applied – any machine not using LAPS or left with a default local admin password becomes low-hanging fruit.

In summary, the security of LAPS hinges on the access control of the stored passwords in AD. A Red Team focuses on those control gaps. If a subset of users or computers has excessive rights, that becomes an attack path. The consequences of a LAPS compromise are significant: the attacker gains administrator access on endpoints without needing to crack any passwords, effectively turning the organization's own defensive tool into an offensive windfall. Testing LAPS in an engagement often reveals whether the "passwords" it protects are truly secure or if an insider threat (or malware) could harvest them due to permission missteps.

Intune and ServiceNow Admin Compromise

Enterprise IT environments frequently rely on cloud-based management platforms like **Microsoft Intune** (part of Endpoint Manager) for device management and **ServiceNow** for IT service management. Individuals with administrative access to these platforms hold a lot of power: Intune admins can control thousands of computers and deploy software, while ServiceNow admins can access sensitive data and even integrate with other systems. A Red Team will seek to compromise such admin roles, as they offer potent opportunities for privilege escalation and pivoting.

- **Abusing Microsoft Intune (Endpoint Manager)**: Intune is Microsoft's cloud MDM/MAM service used to manage endpoints (Windows, mobile devices) via Azure AD. An Intune administrator (or Global Admin, since Global Admins also have Intune rights by default) can push configurations, software, and scripts to managed devices. This makes Intune a powerful attack vector once an attacker has admin access to it. A Red Team that gains **Intune Admin** privileges (say by compromising an account with that role) can effectively “remote control” endpoints. One tactic is to deploy a malicious PowerShell script to devices. Intune allows uploading PowerShell scripts that execute as SYSTEM on enrolled Windows machines

In practice, the Red Team could upload a Cobalt Strike Beacon loader or other malware through Intune and assign it to a group of devices (for example, all devices or a subset like servers). As documented by researchers, when such a script is added, it will run automatically on target machines (typically the script executes during the next device check-in, or on reboot)

This means an attacker can move from a cloud compromise to an on-premises compromise: using Intune to distribute payloads that give control of devices inside the corporate network. In one real scenario, an attacker with Azure Global Admin rights leveraged Intune by pushing a Cobalt Strike beacon to a hybrid-joined workstation; when the machine checked in and ran the script, the attacker got a shell on that on-prem system, bridging the gap from cloud to the internal network

Even without Global Admin, a dedicated Intune Administrator role could do the same. Another angle is using Intune to harvest information: Intune stores device inventories and even user info for enrolled devices, which could be recon for further attacks (e.g., identifying a device used by a Domain Admin). Intune admin portals might also contain API tokens or integration keys that, if stolen, allow persistent access via Microsoft Graph API.

- **Persistence via Intune**: Beyond immediate code execution, a compromised Intune can facilitate persistence. For example, an attacker could **create a new Intune policy** or modify a script that continually deploys a payload (so even if one beacon is cleaned, it will respawn on the next check-in). They could also add a new admin account or device enrollment credentials if possible. Essentially, Intune becomes the attacker’s own deployment system. This is stealthy because it uses the organization’s trusted management channel to do the attacker’s bidding – endpoints will see the actions as coming from a legitimate Intune service. From a Red Team perspective, exploiting Intune is a powerful way to scale control and reach systems that were not directly accessible. It’s a modern twist on “live off the land,” using the organization’s cloud admin tools maliciously.

- **ServiceNow Admin Account Abuse**: ServiceNow is a widely used IT Service Management (ITSM) platform, often used to handle incidents, requests, and it can integrate with identity management or run scripts in workflows. An administrator on ServiceNow has access to potentially very sensitive data: user information, support tickets (which might include passwords or system details in their text), configuration management databases, and sometimes integration credentials for other systems. A Red Team that compromises a ServiceNow admin (via phishing, credential reuse, etc.) can perform several actions. One, they can **exfiltrate data**: read incident tickets or HR cases for valuable intel (as happened in a noted incident where an attacker accessed Microsoft's ServiceNow and obtained employee information and internal communications).
- This data can be used for further social engineering or to understand the network. Two, they might find **stored credentials or keys** – ServiceNow often has integrations (for example, it might have a privileged account stored to create accounts in AD or to query monitoring systems). If these credentials are accessible or can be extracted via a script injection, the attacker can pivot to those systems. Three, a ServiceNow admin could potentially create a malicious **payload through the platform**. For instance, if ServiceNow is integrated with email, an attacker could send phishing emails from a trusted ServiceNow domain to employees. Or if ServiceNow has automation (like running PowerShell on a MID server or using Orchestration), an attacker could leverage that to execute code in the environment. However, these possibilities depend on what modules the victim organization has licensed and configured. The simplest impactful action is often data theft. Real-world threat actors have shown interest in ServiceNow: by logging in with a stolen admin credential, they quietly accessed emails, support logs, and more, all through that single cloud interface

For a Red Team, proving access to such data underscores a serious gap in identity security.

- **Privilege Escalation via ITSM**: Sometimes ServiceNow or similar platforms can even lead to higher access. Consider that ServiceNow might be used by IT to perform password resets or changes for users (via integration with AD). If an attacker has admin control, they could attempt to reset a privileged account's password through the normal IT workflow (impersonating a helpdesk function). This requires understanding the workflows and possibly abusing them, but it's a vector: using the ITSM's legitimate capabilities to elevate privileges (for example, resetting the password of a Domain Admin account if the system allows it). If multi-factor is not required or if the ServiceNow has an automation account in AD with rights to do that, the attacker effectively turns ServiceNow into an attack tool. At a minimum, the Red Team will evaluate how far an intruder could go with ServiceNow admin control.

In summary, administrative access to management platforms like Intune and ServiceNow can be just as dangerous as access to a domain controller – albeit in different ways. Intune gives direct computer control (and by extension, potential domain control if domain-joined computers are compromised and used to get credentials), while ServiceNow gives expansive visibility and indirect influence over IT operations. These are attractive targets during Red Team ops. A successful attack on these can also often be achieved **without malware**: simply abusing legitimate functionality with stolen credentials, which makes detection even harder.

Secret Management System Attacks

Enterprise-grade secret management tools – such as **CyberArk Privileged Access Manager (PAM)**, **Quest/Dell Privileged Password Manager (PPM)**, **HashiCorp Vault**, and **Venafi** (for machine identities) – are meant to be bastions of security, safeguarding the organization's most sensitive keys, passwords, and certificates. Naturally, this makes them high-priority targets for an attacker. If a Red Team can compromise the confidentiality or integrity of these systems, the payoff is enormous: the attackers may obtain a trove of privileged credentials (allowing them to impersonate admins or services at will) or manipulate trust (issuing fraudulent certificates, etc.). We discuss the attack tactics for each in brief:

- **CyberArk Vault Attacks**: CyberArk's PAM solution typically includes a **Digital Vault** server and web interfaces (PVWA – Password Vault Web Access) along with components like credential providers and PVWA plugins on clients. An attack on CyberArk can be direct or indirect. A direct attack might target vulnerabilities in the CyberArk infrastructure itself. For instance, older versions of CyberArk's Password Vault Web Access had a known critical vulnerability allowing remote code execution via crafted requests (an issue where a serialized .NET object in an HTTP header could execute commands)

A Red Team (or real adversary) exploiting such a flaw could gain control over the vault server without any credentials, which would be game over (they could then extract all stored secrets). Aside from exploiting known CVEs, attackers may attempt to **brute-force** or use default credentials if the vault or its components were improperly configured (e.g., default admin password left unchanged – though CyberArk installations typically enforce strong creds). Indirect attacks focus on endpoints that interface with CyberArk. For example, many organizations deploy CyberArk's **Credential Provider** or **Endpoint Privilege Manager (EPM)** agents on servers to retrieve passwords on behalf of applications. If an attacker compromises such an endpoint with local admin access, they might extract sensitive info from the agent's memory or configuration. Indeed, a past vulnerability in CyberArk's Credential Provider could allow an attacker to decrypt stored credentials because of inadequate encryption

Essentially, if the attacker can steal the local encryption key or abuse the API that the agent uses, they may get plaintext passwords for that system's accounts. Another angle is abusing CyberArk's design: if an attacker obtains *any* high-level credential that is stored in CyberArk (for example, a domain admin whose password is managed by CyberArk), they could log into the vault with that account's permissions via the API and extract other credentials. Red Teams will attempt to phish or capture a CyberArk administrator's login as well – once you can log into the vault's web interface as an admin, you can typically retrieve or reset passwords for many accounts.

- **Quest/Dell Privileged Password Manager (PPM) Attacks**: PPM (sometimes part of the older Dell **TPAM** suite – Privileged Access Management Appliance) serves a similar purpose to CyberArk: it stores and rotates privileged passwords. Being an older solution, one challenge for attackers is that PPM/TPAM might not be as well-monitored or up-to-date. If the enterprise still uses PPM, a Red Team will first assess if it's fully patched. Legacy systems often suffer from unpatched CVEs. Suppose PPM's interface has a known flaw (SQL injection, command injection, etc.), the team could exploit that to dump the password database. If direct exploitation isn't available, the next approach is to target credentials that unlock it. PPM usually had an admin web UI; cracking a weak admin password or using credentials obtained elsewhere might let the attacker in. Additionally, because TPAM/PPM was sometimes delivered as a hardened appliance, teams might try *out-of-band* attacks – for example, if they can get console access to the appliance or exploit the underlying OS (though it's hardened, but perhaps not impervious). Indirectly, similar to CyberArk, any system that fetches secrets from PPM could be abused. If an automation script pulls a password from PPM by authenticating with an API key or service account, capturing those credentials gives the attacker a path to query the vault. The main tactic is recognizing that PPM holds the "skeleton key ring" for the enterprise – domain admins, root accounts, service accounts – so compromising it yields administrative access everywhere. Red Teams will treat the PPM system itself as a crown jewel target and possibly use *patient* techniques: for instance, establishing keyloggers or packet captures on an admin's machine hoping to catch them logging into PPM, or exploiting trust (some PPM deployments might integrate with AD – if domain compromise happens, the attacker may then impersonate PPM service accounts).

- **HashiCorp Vault Attacks**: HashiCorp Vault is a newer-age secrets manager used in DevOps and cloud environments. It's designed with strong security in mind (requiring unseal keys, using robust encryption)

Still, there are attack vectors. One is exploiting vulnerabilities in Vault's code or misconfiguration. For example, a severe Vault bug (CVE-2024-7594) was found in the SSH secrets engine, which could allow an attacker to bypass authentication to retrieve secrets A Red Team would check the Vault version for known CVEs and attempt to exploit if unpatched. Another common weakness is poor security around the **unseal keys** or root tokens. Vault operates on a model where it's locked (sealed) until several key holders provide their unseal keys. If an attacker can find those keys (perhaps from an IT admin's notes or a poorly secured key share, or maybe from a compromised Terraform script if the keys were kept as plain variables), they can **unlock the Vault** and get everything. Similarly, if a "root token" (an all-powerful API token) is left lying around (for example, in someone's home directory or an ops wiki), that's a gold mine. Red Teams will scour file shares, configuration files, and even code repositories for any hints of Vault tokens or keys. Another tactic is targeting **authentication methods**: Vault can be set to trust LDAP or cloud auth or Kubernetes for issuing tokens. If the team already has Domain Admin, they might create a fake LDAP user that has Vault access or manipulate the identity backend of Vault. Or, if Vault's policies are overly permissive, a lesser token might be able to read more than it should. Vault's strength is granular access policies, but misconfiguration by admins (e.g., a default policy that is too open) could be leveraged to extract secrets that were not intended to be accessible. Finally, attacking the **server environment**: Vault typically runs on a Linux server or as a container. If an attacker cannot get to Vault via the API, they might try to compromise the underlying server (through OS exploits or stolen SSH access) and then directly access Vault's data files or memory. Vault's data is encrypted at rest, but the encryption key resides in memory when

Vault is unsealed. So, a root compromise of the Vault server while it's running could allow an attacker to dump memory and retrieve the master key. In summary, the tactics for Vault are: exploit known bugs if unpatched, find the keys/tokens through side channels, or get control of the host it runs on. With any of those, the attacker can decrypt and access the secrets, which might include API keys, database passwords, certificates – everything needed to impersonate or access critical services.

- **Venafi (Machine Identity Management) Attacks**: Venafi is somewhat different – it's a platform for managing digital certificates and cryptographic keys (machine identities). An admin in Venafi can issue certificates for any configured domain, which could allow an attacker to create valid certificates for internal services or even intercept traffic by issuing themselves a trusted TLS cert. So, a Red Team targeting Venafi will likely aim to **abuse its certificate issuance capabilities**. If the Venafi platform is integrated with an organization's Certificate Authority (CA) or external CAs, compromising Venafi effectively lets the attacker request certificates as any entity. For example, they could get a code-signing certificate in the name of the company, which would allow them to sign malware to bypass signature checking. Or they could issue a client auth cert for a privileged user account if the organization uses certificate-based authentication. The attack vectors include: stealing a Venafi admin's credentials (through phishing or keylogging) to log into the Venafi web portal, exploiting the Venafi platform (though publicly documented exploits are rare – it's a specialized system), or manipulating its configuration via another integrated account. Venafi often ties into Active Directory (for permissions or placing issued certificates). If the Red Team has Active Directory control, they might not need Venafi access – they could instead abuse AD Certificate Services directly. But assuming Venafi is the gatekeeper, one interesting tactic could be to look at **orphaned or expired certificates**. Venafi manages renewal; if something is misconfigured, an attacker might generate a certificate signing request (CSR) for an important domain and see if Venafi will sign it due to an automated policy. Another subtle attack: if an organization hasn't secured the Venafi server, an attacker on the network could try to impersonate it or exploit weak encryption between agents and the server (though Venafi is typically well-secured). The consequences of Venafi compromise were exemplified in the analysis of the Snowden breach – essentially, misuse of digital certificates enabled stealthy access

In a simulation, a Red Team with Venafi admin could create fake identities for critical servers, enabling man-in-the-middle attacks or decrypting sensitive traffic. They could also revoke certificates to cause outages or hide their tracks by making legitimate admins lose trust in logs (if log signing keys are managed). The core tactic is to turn the organization's trust infrastructure to the attacker's advantage.

Overall, secret management tools are a double-edged sword: they greatly enhance security when properly used, but if compromised, they hand over the *organization's secrets on a silver platter*. Red Team operations treat these systems as high-value targets, often mapping them in the "Crown Jewels" phase. The tactics involve a combination of exploiting technical flaws, stealing credentials/keys, and abusing legitimate functionality (issuing credentials, extracting stored secrets) to achieve a complete breach of confidentiality. The next section will compare these tools in terms of their security features and weaknesses, giving a clearer picture of their resilience.

Windows Attacks in Azure-Integrated Environments

Traditional Windows and Active Directory attacks remain crucial even in a cloud-focused enterprise. In fact, the interplay between on-prem Windows environments and Azure services expands the attack surface. Red Teams will leverage classic tactics (pass-the-hash, Kerberos ticket attacks, etc.) in tandem with cloud techniques. Some notable Windows-specific attack vectors related to Azure setups include:

- **Pass-the-Hash / Pass-the-Ticket in Hybrid Environments**: If an attacker compromises a Windows machine that is *Azure AD Hybrid Joined* (meaning it's joined to the on-prem AD domain and registered in Azure AD), they might obtain credentials that work in both realms. For example, dumping credentials on such a machine could yield an NTLM hash of a domain account (usable for lateral movement in the on-prem network) and also potentially a session to Azure AD (the PRT as discussed). With Mimikatz and similar tools, the Red Team can perform **pass-the-hash** to move laterally to other Windows systems (if local admin accounts or certain service accounts share the hash) while simultaneously performing **pass-the-PRT** to access cloud resources as that user

In assessments, it's common to compromise a user's workstation, then reuse their domain credentials to pivot to a server, then maybe compromise an account with higher privileges on AD, culminating in Domain Admin – all typical Windows attack steps – and then use that Domain Admin status to affect Azure (via AD Connect or altering federation trusts). Conversely, using an obtained cloud token to call Intune (as described) or Azure management functions can lead to new Windows malware deployment or on-prem account creation. The well-known **“Golden Ticket”** (forging a Kerberos TGT using the krbtgt hash from a Domain Controller) remains extremely powerful. If the Red Team gets a Golden Ticket for the on-prem AD, they effectively have indefinite domain control. They can then create an account or modify an existing account to have an Azure AD role (if directory sync promotes new users or if a federated domain trust exists). Another vector is **“Silver Tickets”** (forging service tickets) – if ADFS (Active Directory Federation Services) is used for Azure authentication and the team steals the ADFS service account hash, they could forge SAML tokens to log into Azure services as any user (the so-called Golden SAML attack). So Windows attacks like these can indirectly give cloud access.

- **Active Directory Certificate Services (ADCS) and Azure**: Many organizations with AD have Certificate Services deployed for enterprise PKI. Compromising ADCS can allow forging certificate-based authentication. If Azure AD Connect or ADFS use certificate auth, a Red Team that has compromised ADCS could issue a certificate that Azure trusts for a given user. Recent research shows ADCS is often misconfigured, allowing low-privilege users to enroll for certificates that map to domain admins – an attacker can leverage that to escalate on-prem (known as ESC8/ESC9 attacks on ADCS). Once domain admin, the attacker might register a new device in Azure AD (as a Azure AD Join) to get a foothold in the cloud tenant as well. While this is a multi-step process, Red Teams often include ADCS in their target list when Azure AD is in scope, because certificate trust exploitation can seamlessly bridge on-prem and cloud identity.
- **Kerberos Replication to Cloud (Azure AD Kerberos)**: Azure AD has a feature to allow cloud Kerberos ticket issuance for Azure AD joined devices (for accessing on-prem file shares without direct line-of-sight DC). The feature involves a trust where a key (the Azure AD Kerberos ticket granting ticket) is shared. If an attacker can compromise that trust or the keys involved, they might forge tickets that Azure AD will accept for cloud auth or vice versa. This is an emerging area and likely beyond the scope of most Red Team engagements currently, but it’s worth noting as Azure AD continues to integrate with AD.
- **Lateral Movement via Azure-connected Machines**: On a simpler note, a Windows tactic is leveraging any *Azure-related agents or credentials on a machine* to move laterally. For example, Azure Virtual Machines may have an extension that manages them (like an Azure VM agent). If the enterprise uses Azure Arc or other cloud management for on-prem servers, those agents might have service principals or certificates for authentication. An attacker on that server could extract those and

possibly use them to authenticate to Azure (to the management plane, or to impersonate that machine in the cloud). Additionally, credentials for Azure Automation runbooks or Logic Apps might be found on Windows systems if scripts are stored or if run-as accounts exist – compromising those can shift the attack from on-prem to cloud quickly.

- **Denial of Service / Destructive Actions**: As part of attack simulation, Red Teams might demonstrate the potential **destructive** impact once they have control of Windows and Azure. For instance, using Domain Admin on-prem, an attacker could **deploy ransomware** or wipe systems. Using Azure admin, they could shut down VMs or delete cloud resources. While these are "actions on objectives" rather than tactics to gain further access, they are important in showing the risk. A coordinated attack on Windows and Azure could cripple a hybrid environment: encrypting on-prem files while simultaneously removing cloud backups or snapshots in Azure to prevent recovery.

In essence, Windows-specific attacks remain foundational. The difference in an Azure-integrated enterprise is that the attacker's path may start or end in the cloud. The Red Team might start by compromising an Office 365 account, then move on-prem (via Intune or stolen VPN creds) to hit AD, then escalate in AD to get more cloud access. Or start on a Windows PC, get Domain Admin, then target Azure AD Connect to get Global Admin in cloud. The tactics employed on Windows (phishing, credential dumping, exploiting trust like pass-the-ticket, abusing admin tools like PowerShell remoting, etc.) don't radically change because of Azure – they just gain new significance in how they can be chained with cloud vectors. A thorough Red Team engagement will cover both domains (cloud and on-prem) and use the full arsenal of Windows penetration techniques alongside cloud attacks to achieve objectives.

Goals of Attacks and Key Objectives

When conducting an in-depth security assessment, a Red Team defines specific **goals (objectives)** that mimic what real adversaries would aim for. In large enterprises, these goals typically align with obtaining maximum access, sensitive data, or disruptive capabilities. Below, we break down key attack paths and their final objectives, categorized by the primary focus of the attack. These illustrate what the "endgame" looks like for various avenues of compromise:

- **Azure AD / Cloud Attack Paths**: The ultimate objective is often to gain **Global Administrator** privileges in the tenant or equivalent high-level cloud roles (e.g. Azure Subscription Owner). Attack paths include compromising a privileged cloud account via phishing or token theft, abusing a vulnerable application or service principal to elevate privileges (as with the service principal secret addition exploit), or bypassing MFA to use stolen credentials. The final goal once Global Admin is achieved is full control over the cloud environment – for example, the ability to create or delete users, assign roles, exfiltrate data from SharePoint/OneDrive, or integrate a malicious application at the tenant level. A variant objective could be compromising a critical SaaS application (like the ServiceNow tenant or an HR system) to steal intellectual property or PII. In our context, an Azure AD focus might specifically aim to **control Entra ID** and thereby indirectly control connected services (Exchange Online, Teams, etc.). From there, objectives can branch: one path might pivot to on-prem (cloud-to-ground attack, using Intune or credentials synced downward), while another could purely be cloud-resident (accessing sensitive cloud-only resources such as an Azure SQL Database containing customer data). Success is measured by the extent of cloud dominance – e.g. has the team achieved the ability to act as any user or access any confidential data in the cloud?
- **LAPS/On-Prem Windows Attack Paths**: Here the goal is classic **Domain Dominance** – typically obtaining *Domain Admin* or *Enterprise Admin* rights in Active Directory. The attack paths start from an initial foothold on a Windows endpoint and escalate by exploiting LAPS or other weaknesses: for example, phish a helpdesk user, use their privileges to dump a LAPS password, use that local admin access to move to a server, from there exploit a misconfiguration to get a service account, eventually capture a domain admin’s credentials. The final objective might be control of a Domain Controller (dumping the NTDS.dit database of all domain credentials), which proves the Red Team can impersonate any user in the enterprise. With domain admin, the team could also create persistence by creating hidden accounts or backdooring AD objects (like adding a rogue Kerberos trust or credentials into AD). A parallel objective along this path is **data theft from servers**: if the engagement scope is more data-centric, once they have local admin on key servers (perhaps by using LAPS credentials), they might copy entire file shares or database contents. For instance, using a LAPS password to get into a finance server and then exfiltrating financial records. But typically, the crown jewel from on-prem

escalation is the domain controller itself, because from there any other data or system can be accessed or impacted.

- **Intune/ServiceNow Attack Paths**: The objectives for these paths can be twofold. One is **pivoting and escalation** – e.g. using Intune to reach Domain Admin as described. In that case, Intune is not the final goal but a means to an end (the end being Domain Admin or system control). The other is **sensitive data access or business process compromise**. With ServiceNow admin rights, the final goal could be to retrieve a set of highly sensitive incident reports or employee data that only resides on ServiceNow (this might be the case if we treat ServiceNow as the "crown jewel data source"). Or it could be to demonstrate the ability to **manipulate IT operations** – for example, proving you can create a fake emergency change ticket and trick IT into installing malware, effectively showing that the ServiceNow compromise can lead to physical changes in the environment. In summary, the goal for ServiceNow focus might be *data breach (confidential records)* or *indirect access (through resetting creds or issuing IT tasks)*. For Intune, a likely final objective is *complete control of a set of endpoints* or *extraction of credentials from those endpoints*, which then leads to domain or cloud admin. For example, a specific Intune-focused end goal might be: "Obtain a persistent beacon on CEO's laptop via Intune deployment and use it to collect the CEO's data and credentials." That demonstrates a high-impact outcome (the CEO's device is fully compromised via the MDM).
- **Secret Management Vault Attack Paths**: The overarching goal here is **access to the vault contents**. For CyberArk or PPM, the ideal end state is that the Red Team can retrieve all stored privileged credentials. That means, for example, they can pull out the domain admin password, all local admin passwords, cloud admin API keys, etc., that are safeguarded in the vault. With those in hand, the attackers effectively hold *every key to every system*. A concrete final objective might be: using a stolen CyberArk admin credential, export the entire password list (or specifically grab, say, the root account password of a core database and then use it to log into that database). Achieving a vault compromise often *is* considered the crown jewel, since from there you can breach anything else almost trivially. For HashiCorp Vault, the goal would be similar – dump all secrets (which might include cloud service keys, encryption keys, certificates). Sometimes the objective is framed as *persistence*: for instance, if an attacker can steal the master encryption key of HashiCorp Vault, they can come back at any time to decrypt future secrets, representing an ongoing threat. For Venafi, the objective might be to *issue a wildcard certificate for the*

company's primary domain, or *mint a code signing certificate as Microsoft* (if the enterprise's Venafi has authority over such things), thereby achieving a capability that can bypass trust elsewhere. Essentially, with Venafi, the goal is to subvert trust: either eavesdrop on encrypted communications by issuing fraudulent certs or impersonate services. A Red Team might demonstrate this by actually issuing a certificate for, say, "vpn.company.com" and then using that in a man-in-the-middle proxy to show they can intercept VPN traffic. Or they might obtain the certificate that signs internal software updates, showing they could push a fake update.

- **Hybrid/Bridge Attack Paths**: Many ultimate goals involve bridging between on-prem and cloud. For example, an attack path might start on-prem (phish user -> get domain admin) and end with cloud compromise (use domain admin to seize Azure Global Admin via AD Connect or ADFS). The final objective in such a case is explicitly framed: *Global Admin rights obtained via on-prem compromise*. Conversely, a path might start in cloud (phish O365 user -> get Global Admin) and end with on-prem DA (using Intune to deliver a payload to a DC). In that case, the goal is *Domain Admin obtained via cloud compromise*. These hybrid goals are increasingly relevant in enterprise scenarios; the Red Team report might have a section for "From O365 tenant to Domain Controller: achieved?" and detail that as a goal.

It's useful to categorize objectives by focus, but in a full engagement these often intertwine. For instance, obtaining the secrets from a vault (secret management focus) might itself be the method to achieve Domain Admin (on-prem focus) and then that leads to cloud admin. A PhD-level assessment recognizes these multi-step chained goals. Therefore, Red Teams often set *progressive objectives*: initial (access one user's data), intermediate (control of critical management tool), and ultimate (enterprise "crown jewel" like domain or global admin, or extraction of the most sensitive data). Each attack path we've discussed is essentially how you get to one of those ultimate goals.

To summarize in a categorized way, typical final goals include:

- **Complete Cloud Takeover** – verified by Global Admin in Azure AD and Owner role on major Azure subscriptions.
- **Complete Domain Takeover** – verified by Domain/Enterprise Admin in AD and control of domain controllers.
- **Compromise of Privileged Management Platforms** – e.g. full access to CyberArk vault or ServiceNow, demonstrating ability to retrieve or alter any secrets/records.

- **Extraction of Crown Jewel Data** – e.g. download of all customer records from a database, or access to proprietary source code, or emails from the CEO's mailbox.
- **Demonstrated Persistence** – e.g. the placement of stealth backdoors in both cloud and on-prem (such as hidden Azure AD app credentials and on-prem scheduled tasks) showing the attacker can maintain long-term access undetected.
- **Destructive Actions (simulated)** – e.g. illustrate ability to deploy ransomware enterprise-wide or turn off cloud resources, thereby highlighting the potential impact (usually this is done as a proof-of-concept rather than actually pulling the trigger).

Each of these goals maps to what a real-world adversary might consider their "mission success," whether that's espionage-driven (steal data), sabotage-driven (destroy or disrupt infrastructure), or takeover-driven (monetary gain via ransomware or persistent control for future use). By achieving these in a controlled manner, the Red Team demonstrates where the organization's defenses failed and provides valuable lessons for remediation.

Crown Jewels: Highest-Value Targets

"Crown Jewels" refer to the assets an attacker covets the most – the systems, accounts, or data whose compromise would be most devastating to the organization. In a large enterprise with Azure and hybrid infrastructure, crown jewels span both cloud and on-prem realms. Identifying and securing these is critical, and a Red Team exercise often uses them as the finish line. Based on the attack scenarios we're focusing on, the crown jewels typically include:

- **Active Directory Domain Controllers and Kerberos Keys**: The on-prem AD domain controller (and the AD DS database) is a perennial crown jewel. It contains authentication data for the entire enterprise. With it, an attacker can impersonate any user, decrypt enterprise Kerberos traffic, and essentially own the network. Domain Admin groups, the *KRBTGT account* (which signs Kerberos tickets), and backup files of the domain are all part of this critical category. If a Red Team fully compromises a domain controller, they've hit a crown jewel because recovery from that scenario often means rebuilding the AD from scratch.
- **Azure AD Tenant and Global Administrator Accounts**: In the cloud arena, the Azure AD tenant itself is the crown jewel. A Global Administrator (or the highly privileged Azure roles like User Access Administrator or Subscription Owner) is the

cloud equivalent of Domain Admin. Compromise of the tenant means the attacker can access any O365 mailbox, any SharePoint site, create or delete any user, and modify any cloud resource. This could facilitate wide-reaching impacts (like impersonating the CEO via email or spinning up costly resources for cryptomining as a distraction). Within Azure, other jewels include **Azure Key Vaults** that store master keys or secrets for applications – if not protected by separate HSMs, Key Vault compromise yields encryption keys that might secure databases or disks. For organizations using Azure's Managed Identities or service principals, those with contributor/owner rights on key subscriptions are jewels too (one might call them "cloud crown jewels").

- **Privileged Access Management Vaults (CyberArk, PPM)**: These systems concentrate the most sensitive credentials. The **CyberArk Master Vault Server** and its data store (or the TPAM appliance) are crown jewels because they hold the literal keys to other systems. Also, the *accounts managed by CyberArk themselves* are jewels – e.g. the domain admin credentials stored inside. One could say the vault is like a treasure chest: the chest (vault server) and the treasure (passwords inside) are both of highest value. In a Red Team context, if you compromise CyberArk, you likely automatically compromise many other jewels (AD, Unix root accounts, etc.), hence it's an uber target.
- **ServiceNow (or similar ITSM) containing Sensitive Records**: While ServiceNow itself might not always be treated as a crown jewel compared to domain or cloud admin, in certain contexts it is – especially if it harbors critical incident data, customer PII from support tickets, or credentials in scripts. If, for example, ServiceNow has integration that allows it to reset passwords or has an update set that contains secrets, that specific facet becomes a jewel. More broadly, **sensitive data stores** such as databases with customer information, trade secrets in file servers or SharePoint, and email archives of top executives are all crown jewels. These are the targets a nation-state APT might go after for espionage, or a cybercriminal for extortion.
- **Intellectual Property Repositories**: Many enterprises have code repositories (Git, etc.), design documents, or formulas that are the core of their business. Those might not be directly mentioned in our Azure/Windows focus, but if they are hosted on-prem (maybe in a file share or on an Azure DevOps server), then a chain of attacks leading to them would mark them as crown jewels. For instance, if the company's crown jewel is its source code, then obtaining domain admin and thus

access to the build servers would be an intermediary – the final jewel being the source code itself exfiltrated.

- **Certificate Authorities and Trust Stores**: The **Venafi platform** and any integrated Certificate Authorities (whether an internal Microsoft CA or external CA credentials) count as crown jewels of trust. By compromising these, an attacker can forge identities at will. The ability to mint certificates that are trusted by the organization's systems (for VPN, code signing, etc.) means the attacker can undermine fundamental security assumptions. Thus, Venafi and the root certificate keys it manages are extremely high value. The same goes for cryptographic keys (e.g., the key used to sign SAML tokens in ADFS, or JWTs in Azure AD if one could get it – though Azure AD's own keys aren't accessible to tenant admins, in hybrid scenarios keys on ADFS or OIDC providers are jewels).
- **Global Infrastructure Controls**: This includes things like the admin credentials for hypervisors or cloud infrastructure beyond Azure (if multi-cloud, maybe AWS root keys, etc.). Since our focus is Azure, the analog is the **Azure Subscription Owner** for critical subscriptions (like production environment). If an attacker gets that, they can delete resources, or open environments to the public. For example, an Azure subscription hosting all customer databases – Owner access there is a crown jewel because they could copy or destroy all those databases.

In summary, the crown jewels can be summarized as: *the accounts with the broadest privileges (Domain Admin, Global Admin), the systems that store those privileges (vaults, AD, AAD), and the data stores of highest sensitivity or trust (sensitive data repositories, certificate authorities).* A successful Red Team exercise should identify these in advance (with the organization) so that the team knows what "winning" looks like, and so the organization is aware of what assets require the strongest protection. In the course of an engagement, as we saw, multiple jewels may be targeted sequentially – e.g. compromise a lesser target to get to a bigger jewel. Ultimately, demonstrating compromise of any single crown jewel is usually enough to prove a serious security gap. When multiple are compromised in one campaign, it underlines how interconnected the security of these assets is (for instance, one vault breach leads to many other jewels falling).

Protecting crown jewels requires a defense-in-depth strategy, which we'll touch on in a later section. The Red Team's findings around these highest-value targets drive the most critical remediation recommendations.

Comparative Analysis of Secret Management Tools

Enterprise secret management solutions each have distinct architectures and security features. Here we compare CyberArk, Quest/Dell PPM, HashiCorp Vault, and Venafi in terms of their security strengths, known weaknesses, and how an attacker might approach each. This analysis highlights how their differing designs affect their resilience under Red Team scrutiny:

CyberArk Privileged Access Manager (Vault)

Overview & Strengths: CyberArk is a market leader in Privileged Access Management (PAM). Its Digital Vault is a hardened server (often running on Windows with hardened OS settings) that stores credentials in an encrypted database. CyberArk emphasizes isolation – the Vault server is meant to be isolated from the domain (not domain-joined) , use its own authentication, and be firewalled to only allow the PVWA (web interface) and authorized clients. It provides robust access controls, workflow for check-out and check-in of credentials, and auditing of every access. One of CyberArk's strengths is its **security layers**: even if an attacker gets domain admin, they *shouldn't* directly get into the Vault because it's standalone and heavily secured (e.g. requires separate credentials or physical token to login). CyberArk also has features like **Session Manager** that isolate privileged sessions, and **rotational policies** so passwords change after use, reducing the window of abuse. The cryptography in CyberArk is proprietary but has withstood scrutiny for many years. All secrets in the vault are encrypted with strong algorithms and a master key that is split (with recovery requiring quorum of key holders, if configured). In short, CyberArk's design assumes it will be targeted, so it builds a fortress around the keys.

Weaknesses & Attack Surface: Despite its strong design, CyberArk is not invulnerable. A notable weakness is the complexity – it requires agents, web servers, and integration points, each of which can be a target. For example, the **Password Vault Web Access (PVWA)** interface is accessible to admins and sometimes users; any web application vulnerabilities here (like the 2018 deserialization RCE) are critical. CyberArk has had a few CVEs over the years, though they are relatively few for its age. Another weakness is *credential footprint*: the CyberArk **Credential Provider** installed on servers caches credentials to serve applications. If an attacker compromises a server with a Provider, they might exploit something like CVE-2021-31796 , where the encryption of cached creds was found to be insufficient in some versions, leading to potential decryption of passwords. Also, CyberArk often integrates with AD for user authentication to the vault (though the vault has its own internal user DB, many deploy LDAP integration for user convenience). If AD is compromised, an attacker might leverage a synced CyberArk admin account to get into the vault (this depends on deployment specifics; best practice is to keep vault users separate or require 2FA). Additionally, because CyberArk is so critical, many organizations give it some of the highest privileges – for instance, CyberArk might have an automation account that is a domain admin to reset passwords. That account itself becomes a target. If an attacker finds those credentials (say in a config file or by intercepting traffic if not properly encrypted), they could misuse it outside of CyberArk. Lastly, while the Vault server is hardened, it's still a Windows server – a local exploit or an unpatched OS vulnerability could give an attacker a foothold on it. CyberArk attempts to minimize this by allowing only console login with special procedures, but a determined attacker with network access might try anyway. In summary, CyberArk's weaknesses are usually through **adjacent systems or default passwords** rather than easily exploitable flaws in the core vault. But if any part cracks, the impact is huge.

Attacker Methodology: A Red Team targeting CyberArk will first try to enumerate its presence: identify the PVWA URL, version, etc. They might run a vulnerability scan for known CVEs. They'll also attempt to gather credentials for any CyberArk users. Often, companies have "break-glass" accounts in CyberArk whose credentials are sealed in envelopes – if any such credentials were ever temporarily exposed (perhaps in documentation), an attacker would love to have them. Another method is social engineering: phish a CyberArk administrator by sending a fake "CyberArk login" page to capture their vault password or intercept a 2FA token. If the team has some level of domain compromise, they will sniff network traffic to the vault (though communications are usually encrypted and signed). They might also examine any endpoints for traces of CyberArk usage – e.g., if someone checked out a password, maybe it's temporarily stored in memory or in a log (unlikely, but part of thorough checking). If an insider-level scenario is allowed, the Red Team might even attempt to plug into the vault network segment to see if any services (like NTP or DNS) that the vault uses can be poisoned – but again, CyberArk hardening usually limits this. Overall, the attacker's approach is either to *find a direct exploit* (a vulnerability or misconfiguration that allows entry) or *harvest credentials that unlock it*. One more thing: attackers can target the **workstations of privileged users** who use CyberArk. If a user is logged into PVWA via a web browser, maybe an XSS or token theft could be done by infecting that user's machine. All angles are considered because a straightforward breach is tough if CyberArk is well-maintained.

Quest/Dell Privileged Password Manager (PPM)

Overview & Strengths: Quest's Privileged Password Manager (PPM), often part of the TPAM (Total Privileged Access Management) appliance, was one of the earlier PAM solutions. It's typically delivered as a hardened Linux appliance with two main components: PPM (for password management) and PSM (Privileged Session Manager) for session proxying. The strengths of PPM include its **appliance model** – being a black-box appliance means it has a minimal attack surface (in theory) as users don't get shell access or the ability to install software on it. It also enforces password rotation and can randomly generate strong passwords for accounts, reducing the chance of guessable passwords. PPM has role-based access control to restrict who can retrieve which credentials, and it keeps an audit log of activity. Since it's often isolated on a management network, an attacker might have a hard time even reaching it without prior network access. When Dell/Quest built it, they intended it to run on a stripped-down platform to avoid typical OS vulnerabilities. It supports 2FA and other enhancements as well. In terms of cryptography, it securely stores passwords (using encryption in its database). One might say its strength was simplicity and the fact that it's not as feature-rich (which sometimes can reduce complexity-based bugs).

Weaknesses & Known Issues: PPM is older and not as actively developed now (Quest's TPAM has been succeeded by other One Identity products). Legacy often means unpatched issues: if the appliance firmware isn't updated, it might have outdated software (like old OpenSSL, old Apache/Tomcat if it uses those). Weakness may lie in the web interface that admins use to manage PPM – if it's vulnerable to an injection or buffer overflow, that's a direct vector. The appliance's OS, while minimal, could have default accounts or known backdoor passwords (some older appliances had default root passwords which if not changed could be exploited via console). Another weakness: PPM's integration with AD – if configured to sync users from AD for login, a domain compromise could translate to PPM admin login (similar to the CyberArk LDAP integration issue). Also, PPM may not have had as granular role separation as CyberArk – so one admin role might have broad access, increasing the impact if that one admin account is compromised. Since PPM's popularity has waned, the pool of people looking for 0-day in it is smaller, which is double-edged: fewer public CVEs, but also less scrutiny might mean undiscovered issues. One publicly documented issue to note: historically, some PAM appliances had trouble with the infamous *Java deserialization* exploits or *outdated libraries* that could be hit by tools. If PPM's UI was Java-based, it might be prone to such attacks if not updated. Additionally, PPM depends on internal secrets to function (encryption keys for its DB) – if an attacker gets a backup of the appliance or the config, they might extract those keys offline and decrypt the vault. Weak backup protection is a risk: if backups of TPAM are stored on a network share without strong protection, an attacker with access to that share could potentially restore it and extract passwords.

Attacker Approach: A Red Team assessing PPM will typically start by identifying the version of the TPAM appliance and researching any known exploits or default credentials. They may attempt a **credential stuffing or default password** on the web UI (maybe “Administrator:Administrator” or similar, as some appliances have a well-known initial password). Next, they might port-scan the appliance to see if any service beyond the management UI is open (SSH might be present for support – if so, perhaps there’s a known SSH exploit or default creds). If the web UI is accessible, they’ll try common web attacks: SQL injection on login, command injection on any diagnostic functionalities, or an authentication bypass. Given it’s closed-source, they might use fuzzing tools to find anomalies. If they already have some internal access, another tactic is to intercept traffic between the appliance and directory or database – but in TPAM, the DB is internal and directory communication (LDAP) would be interesting if configured: intercepting that could yield AD creds if not using LDAPS. One more approach is to utilize any API that PPM provides (some PAMs have an API/CLI); if that is less secure or not well known, maybe it can be abused with an admin token that the team somehow acquires. But often, the simplest is social engineering: find someone who has a bunch of admin passwords but is supposed to check them out via PPM. If that person can be tricked into revealing one or writing it down outside of PPM, the whole purpose is defeated – but that’s more a policy fail than a tech fail. In terms of *incidents*, we haven’t heard as much about PPM breaches in the wild, but a Red Team will treat it as “if I own this, I own everything.” So they might devote significant effort to getting in, potentially even a physical attack – e.g., if the TPAM appliance is on-prem, a malicious insider could attach a USB or connect a monitor to see if console access is possible. While extreme, such out-of-band tactics are not unheard of for crown jewels.

HashiCorp Vault

Overview & Strengths: HashiCorp Vault is a modern, open-source (and enterprise) secrets manager that is heavily used in DevOps and cloud environments. Its primary strength is its **strong security model**: everything in Vault is encrypted at rest with a master key that can be split among multiple key holders (Shamir’s secret sharing for unseal). It uses in-memory encryption for data in use, and access is mediated by strict policies. Vault is also highly **extensible** – it has numerous secret engines (for key/value, databases, cloud credentials, etc.) and authentication methods (tokens, LDAP, cloud auth, etc.), which allow fine-tuned control. From a design perspective, Vault's cryptography is meant to stand up to nation-state level attackers , and it's regularly assessed by third parties due to its popularity. When properly configured, no one person's credentials should trivially unlock everything; for example, an operator might have a root token but that can be disabled or tightly scoped in enterprise use. Another strength is Vault's **dynamic secret generation** – it can generate ephemeral credentials (like short-lived DB logins), meaning even if those get stolen, they expire quickly. This reduces the window of opportunity for an attacker. Vault's audit logging is comprehensive as well – every access can be logged (though if an attacker gets root on the Vault server they could tamper with logs, as with any system). The community and HashiCorp respond quickly to vulnerabilities, and the fact that it's open source means it gets a lot of eyes (though also means vulnerabilities, when found, are public).

Weaknesses & Notable Vulnerabilities: Vault's flexibility can be its weakness. Misconfiguration by users is a common risk: for example, an admin might leave a broad policy that allows reading more secrets than intended, or enable an auth method that isn't properly constrained (like trusting any LDAP user to get a token). If an attacker finds a path to log in (say, the Vault is configured to trust GitHub logins and the attacker gets a token for a GitHub org), they might gain some access. Another inherent risk is the **single point of failure/trust** – the master key. Vault has to decrypt data in memory to serve it; if an attacker can run code on the Vault server (via OS exploit or a malicious plugin loaded into Vault), they could potentially extract secrets or keys from memory. So Vault is only as secure as the server it runs on and the unseal key holders. Speaking of unseal keys, human processes around those can be weaknesses (people writing them down or one person holding all shards). We saw a mention of CVE-2024-7594

cycognito.com

– an *authentication bypass in the SSH secrets engine*. This is a specific bug: it implies that an attacker could authenticate to Vault's SSH secrets backend without proper auth, which could give out credentials. Vault has had a few such issues; earlier, in 2022, there was a bug in the Auto-unseal using certain KMS providers that could potentially leak unseal keys under specific conditions. However, no catastrophic flaws in core crypto have been public. Another potential weakness is **user error in handling Vault tokens**: Vault often returns secrets or tokens to the client; if applications or scripts fail to secure those (e.g., leaving them in bash history or in config files), an attacker may grab them. So while not Vault's fault, the ecosystem around it can leak. Compared to CyberArk, Vault is more **API-centric**, which means if an attacker can get any privileged API token, they can automate extraction of all secrets quickly. So the speed of compromise, if it occurs, could be very fast (with CyberArk, an interactive GUI might slow an attacker down, whereas Vault one API call can list hundreds of secrets). Also, some organizations might neglect to enable Vault's audit logging or not monitor it, so an attacker could try many things against Vault's API without detection if not monitored.

Attacker Methodology: A Red Team targeting Vault will often begin by trying to see if Vault is accessible from their current position. Many times, Vault is on an internal network only. If the team has a foothold on a dev's machine, they might find that dev has a Vault CLI configured (with a token cached). They'll try to use that token or capture it. If they have to attack Vault head-on, they'll check the version and compare against known exploits. If CVE-2024-7594 is applicable (Vault version in use with SSH secrets engine and unpatched), they could attempt to exploit it to bypass auth and extract at least the secrets from that engine. If unsuccessful, they may try a **brute force on API** – Vault has rate limiting on login by default, but perhaps some auth backends (like LDAP) might let many attempts through or reveal timing differences. Another vector is looking at **Vault's storage backend**: Vault can use Consul, etcd, or others to store encrypted data. If the team can't get into Vault directly, maybe they can access the storage backend. For example, if Vault stores data in Consul and the Consul UI or API is not protected, an attacker might enumerate the encrypted secrets. They can't decrypt them without the master key, but if they later compromise the master key, they have the data at hand. In extreme cases, an attacker might try to extract memory from the running Vault process (if they have OS-level access via something like a local privilege escalation on the Vault host). That could reveal master keys or loaded secrets. So a multi-step: compromise host -> dump memory -> find key -> use key on stored data to get secrets. Also, targeting how Vault starts: if auto-unseal is configured with a cloud KMS (like AWS KMS), the team might try to get access to that KMS key in AWS (via cloud credentials obtained elsewhere). Then they could decrypt Vault's data without needing unseal keys. Essentially, because Vault often ties into other systems (cloud KMS, databases for dynamic secrets, etc.), those integrations broaden the attack surface. An example: if Vault can generate AWS API keys and an attacker can trick Vault into doing so with high privileges (via a policy mistake), they could get AWS admin keys through Vault if a policy wasn't limited. Red Teams look for these logic flaws. On the flip side, if nothing obvious is open, a Red Team might simply phish a developer or Ops engineer for their Vault credentials (which might be their LDAP password if that's how they login to Vault). Then they login as that user and see what they can access. Many secrets might be segmented by team, so maybe that gets only some secrets, but possibly among them could be something like a database admin password. That password might allow lateral movement to that database server, where further escalation can occur. In summary, attacking Vault is half about finding technical exploits and half about leveraging any **operational security lapses** (like poor key custody or leftover tokens).

Venafi (Machine Identity Management)

Overview & Strengths: Venafi Trust Protection Platform focuses on managing and protecting machine identities – things like SSL/TLS certificates, code signing certificates, and SSH keys. Its strength lies in centralizing certificate issuance and automating renewal, which means organizations can enforce policies (e.g., key lengths, approved CAs) and have an inventory of all certificates. From a security standpoint, Venafi helps by ensuring private keys are properly stored (often in hardware security modules, HSMs, integrated with the platform) and not sprawled across servers. It often acts as a broker between certificate authorities (internal or external) and the enterprise systems that need certs. Venafi's platform includes role-based access so that only authorized teams can request or approve certain certificates. One big strength is that it can detect anomalous certificate creation or usage, which could indicate a compromise. It also secures **code signing keys** – treating them carefully because a stolen code signing cert is extremely dangerous. If integrated with HSMs or cloud KMS, Venafi ensures even Venafi admins might not directly touch the raw key material – they just orchestrate. The product is mature and has deployments in very security-conscious orgs, so it's built with strong authentication and auditing in mind.

Weaknesses & Potential Risks: A core weakness is that Venafi, by necessity, must interface with many systems (CAs, DevOps pipelines, etc.), so it has many integration points that if misconfigured, could be abused. For example, Venafi might have an automation that when a new certificate is requested for a certain common name, it auto-approves and issues via an internal Microsoft CA. If an attacker can fake a request that meets those criteria (maybe by having a certain IP or using an API with minimal auth), they could get a certificate without proper approval. Also, while Venafi secures keys, the keys ultimately reside somewhere – if not in an HSM, then on disk or in a database. A breach of the underlying storage (like SQL database) could expose encrypted keys or certificates. If the encryption guarding those is weak or if the attacker also gets the application server, they might decrypt them. Another potential issue is **certificate trust**: if Venafi manages a root or intermediate CA for the org, compromise of Venafi could mean the attacker can generate certs that all systems trust (because they chain to the root). This is similar to how Stuxnet allegedly used stolen certificates – an attacker could do that at scale. There have been incidents and research highlighting how abused certificates and keys allow stealthy breaches – Venafi itself publicized how Snowden likely used forged certificates to remain undetected

If an attacker got Venafi, they could repeat such tactics in an enterprise. On the software side, Venafi is not as commonly discussed in CVE feeds, but that might mean issues could be underreported. The platform complexity (web dashboards, agents on hosts to auto-enroll for certs, etc.) means vulnerabilities could exist in those agents or interfaces. One risk: if Venafi agents on servers accept instructions from the Venafi server to install a new cert or run a key generation, perhaps an attacker who controls the server could impersonate the Venafi server or manipulate agent config to get a cert for a different identity. It's somewhat theoretical, but possible if mutual authentication between agent and server is not strong. Also, Venafi often ties into AD for user authentication; thus, compromise of admin AD accounts could give access to Venafi admin portal unless 2FA is enforced.

Attacker Methodology: A Red Team aiming at Venafi would likely treat it as an *indirect target* initially. They might not go straight at Venafi software (unless a known exploit is available). Instead, they will seek to misuse its capabilities via a legitimate interface. For example, if the team has some foothold in IT, they might see if they can request a certificate from Venafi for a domain they shouldn't (like a wildcard cert for *.company.com). If the approval process is lax, that request might get fulfilled. That gives them a certificate they can use in attacks (like setting up a spoofed VPN or web server that looks legitimate). Another approach: compromise an account that is a Venafi administrator or certificate manager. Then through the Venafi UI or API, issue a cert or retrieve an existing key. Venafi's interface might allow exporting a key (for backup or for use on another server) – if an admin can do that, an attacker with that admin's privileges might export, say, the code signing key used for software releases. That's a massive jewel – they could sign malware as if it were the company. So the team will definitely look at what roles exist and what they can do. On the network side, they might attempt to sniff or intercept traffic between Venafi and an internal CA. If Venafi requests a cert from a Microsoft CA, it likely uses the CA's RPC or web interface. Is there an opportunity for a man-in-the-middle? Possibly not if Kerberos or certificate-based auth is used. But if an attacker already has domain admin, they could attempt to subvert the CA directly (e.g., issue a certificate for a desired name via certutil). But with Venafi in play, the org might have locked down manual issuance – still, domain admin can override. So for a Red Team, if they reach domain admin, they can indirectly get certificates anyway (Venafi or not). Perhaps more interesting is if the Red Team is *not* domain admin but manages to get Venafi. That could allow them to escalate to many scenarios: e.g., issue a valid certificate for the domain controller's LDAP service, then use that in an LDAPS man-in-the-middle to capture credentials. Or, issue certificates for user authentication if any system use certificate-based user auth. If targeting Venafi software directly, they'd identify the Venafi Trust Server version and check for known vulnerabilities or default credentials. If Venafi uses a database, maybe default SA password? Unlikely, but they'd check. They might also examine any web endpoints for injection flaws. But due to the nature of the product, an easier method is often just to *use it against the org*, rather than break it.

Summary of Comparative Security: Summarizing the above:

- **CyberArk**: Very strong enterprise-grade security with isolation and maturity. Hard to compromise directly without a serious flaw or credential theft. Needs diligent patching (past RCEs) and good operational discipline (no bypass of its workflows). Usually a prime target due to concentration of credentials.
- **Quest PPM**: Older, somewhat out-of-date technology. Security depends on appliance integrity. Fewer known exploits publicly, but likely less sophisticated security measures than others. If still in use, must ensure it's updated and isolated. Red Teams might find legacy weaknesses (like default creds or old software vulnerabilities) to exploit.
- **HashiCorp Vault**: Modern and secure by design, but heavily reliant on correct setup. When properly configured (especially Enterprise version with HSM auto-unseal), it's very resilient. However, any lapse (exposed root token, unsealed server compromise, known CVE unpatched) can lead to immediate loss of all secrets. Attackers often exploit user misconfiguration or chain other system compromises to get Vault.
- **Venafi**: Focused on a different domain (trust of machine identities). Strong integration with security infrastructure (HSMs for keys, etc.) means it can be robust, but if taken over, yields potent attack tools (forgery of identities). Likely attack vectors involve abusing its functionality more than breaking its crypto. Should be

treated as critical infrastructure (with strong access controls and monitoring of certificate issuance).

Each tool has a unique role, and often enterprises use more than one (e.g., CyberArk for passwords, Vault for app secrets, Venafi for certificates). From a Red Team perspective, any one of them can be the stepping stone to complete enterprise compromise, which is why understanding their security is paramount. In practice, combining their use (like storing the Vault unseal keys in CyberArk, or using Venafi with an HSM) can mitigate some risks but also creates new dependencies (one being compromised could affect the other).

Example Scenarios (Real-World Case Studies)

To illustrate how these attacks and tactics come together, we present two example scenarios that simulate determined adversaries targeting a large enterprise. Each scenario highlights the use of the Cobalt Strike framework for command-and-control (C2) during initial access, persistence, and post-exploitation. These cases draw on real techniques observed in Red Team engagements and threat research, combining multiple attack vectors discussed earlier.

Scenario 1: Cloud-to-Ground Attack – From Azure AD to Domain Admin

Initial Access: The Red Team begins by quietly compromising a low-privileged Azure AD account via a phishing campaign. The phish was a well-crafted email with a link to a fake Office 365 login page, which an employee in the finance department fell for, providing credentials. Unfortunately for the team, the account had MFA (a mobile authenticator), but the Red Team executed an MFA fatigue attack – repeatedly sending push notifications late at night until the user inadvertently approved one. With the authenticated cloud session, the team had user-level access to Office 365. They discovered this user had access to a Microsoft Teams channel where an IT admin had posted a snippet of a PowerShell command containing what looked like a password. The Red Team dumped the Teams chat and obtained credentials of an IT support user. Using these, they were able to login to the Azure portal. The support user was not a Global Admin, but had the **Intune Administrator** role (as support often handles device issues).

Establishing C2 in Azure: The team now leveraged Cobalt Strike's Beacon in a novel way. They couldn't directly run code on a cloud service, but with Intune admin rights, they had Intune push a small PowerShell snippet to a pilot group of devices (which included a couple of IT laptops). This snippet, when executed on those machines, fetched and executed a Cobalt Strike Beacon payload in memory. Essentially, the team used Intune as their “launcher” to get a foothold on on-premises devices

Within an hour, one of the targeted laptops checked in, downloaded the malicious script, and executed the Beacon. The Beacon established a C2 connection over HTTPS to the Red Team’s server, masquerading as normal Microsoft telemetry. Now the Red Team had a Beacon running as SYSTEM on an internal Windows 10 machine used by an IT staff member.

Persistence: The first thing the operators did was establish persistence on that machine, in case the Intune policy was noticed and revoked. Through the Beacon, they created a new Windows service that would launch the Beacon payload on startup. They also took note that this machine was hybrid Azure AD joined (so it had a device identity in Azure AD). They added an additional scheduled task that attempted to leverage the device’s Azure AD identity to request a token periodically (which could be a backup method to regain a foothold via cloud if needed). These actions were quiet and did not trigger antivirus.

Privilege Escalation (On-Prem): The compromised IT laptop had a user logged in who was a member of some privileged groups (the user had local admin and was a member of a group that allowed some AD actions like resetting passwords). Using the Beacon, the Red Team dumped credentials from LSASS memory (via an in-memory execution of Mimikatz). They extracted an NTLM hash for the logged-in support user and also found a Kerberos ticket for that user still valid. With the NTLM hash, they performed a pass-the-hash to connect to a domain controller (leveraging the fact this support user was often allowed to log into DCs for troubleshooting). On the DC, they ran the **DCSync** command (Mimikatz lsadump::dcsync) to simulate being a domain controller and pulled the credentials of a Domain Admin account. This succeeded because the support user’s group had been misconfigured with replication rights. Now the Red Team had the **hash of a Domain Admin** account. They used the classic technique to forge a Kerberos TGT (Golden Ticket) for that Domain Admin, injecting it into the Beacon’s process. Instantly, their Beacon had Domain Admin privileges.

Lateral Movement and Further Escalation: With Domain Admin, persistence was virtually assured – but they went further. They wanted to get to Azure Global Admin as well, achieving a full cloud-to-ground circle. They remembered the Intune admin account they phished initially did not have Global Admin. However, as Domain Admins on-prem, they had control of Azure AD Connect. They RDP’d to the Azure AD Connect server (with

Domain Admin rights) and ran a PowerShell extractor (similar to XPN's method) to retrieve the Azure AD synchronization account's credentials. This account (typically ending in @tenant.onmicrosoft.com) was found to have an older, known-weak password that the Red Team cracked (if it hadn't been possible, they could also attempt to pass the hash or use the decrypted password directly). Using those credentials, they logged into Azure AD PowerShell as that sync account. This account had directory write permissions. The Red Team added their phished support user to the **Company Administrator** role (Global Admin) – effectively a backdoor promotion using the sync account's privileges. Now the support user (and thus the Red Team) was a Global Administrator in Azure AD. They confirmed this by logging into the Azure portal with the support user's credentials, which passed MFA since they still had the session token. They toggled "User Access Administrator" as allowed (exploiting the fact that as Global Admin they could elevate Azure RBAC roles) , giving the support user full Azure subscription access too.

At this point, the Red Team has *domain admin on-prem* and *global admin on Azure*. They have multiple beacons: one on the IT laptop, one deployed on a server for persistence (they had also deployed a Beacon on the AD Connect server, in case anything went awry, by using the Domain Admin to push it via psexec).

Actions on Objectives: With this level of access, the operators proceeded to enumerate and exfiltrate the agreed-upon targets. For instance, they used the Domain Admin to access a file server containing proprietary research documents, encrypt and copy a subset of those files to exfiltrate over their C2 channel. Simultaneously, using Global Admin, they accessed several executives' Office 365 mailboxes (by assigning eDiscovery rights to the support account) and pulled down emails related to upcoming financial results. All these operations were done through the Beacon, staging data and then exfiltrating in chunks that looked like normal HTTPS traffic. Finally, they demonstrated potential destructive capability: from Azure, they had the ability to shut down VMs or delete cloud resources; from on-prem, they could have deployed ransomware via Group Policy. They didn't execute those, but documented the pathways.

Notable Cobalt Strike Techniques: Throughout this scenario, Cobalt Strike was the workhorse enabling the moves. The Beacon on the IT laptop was configured in *sleep* mode to only beacon infrequently and use an HTTP profile mimicking Windows Update traffic. This helped it avoid detection. When they performed the hash dump, they used Cobalt Strike's execute-assembly to run safetyKatz (an in-memory Mimikatz variant) to avoid writing any binaries to disk. For lateral movement, they utilized Beacon's psexec with a reflective loading to push beacons to remote systems using the stolen credentials – a technique that often bypasses older detection rules. For persistence, the new services and tasks were created via Beacon's spawnas and shell commands, carefully using Windows commands that are native (sc.exe, schtasks) to remain low profile. In one instance, they attempted to run another Beacon on a different host and the organization's EDR flagged it, but the team reacted by using Cobalt Strike's *evasion* script to migrate the Beacon into dllhost.exe (a less suspicious process) which stopped further alerts. The coordination of multiple Beacons was done through Cobalt Strike's team server, allowing the operators to manage all compromised assets concurrently.

Outcome: In this scenario, the Red Team achieved all major objectives: global admin in cloud, domain admin on-prem, data access, etc., demonstrating the risk of a breach that spans the hybrid environment. They did so by chaining together multiple tactics: phishing, Intune abuse, token theft, misconfig exploit (Azure AD Connect), and classic domain attacks – with Cobalt Strike providing a unified platform to carry it out covertly. The engagement revealed gaps in monitoring (no one noticed Intune being used strangely, nor the lateral movement) and weaknesses in config (excess privileges for support staff, old passwords, lack of MFA for the sync account, etc.).

Scenario 2: Vault Heist and ServiceNow Subversion – On-Prem to Cloud

Initial Access: The Red Team's initial foothold came through a targeted **spear-phishing email** to a system engineer. The email pretended to be an internal alert about "VPN certificate renewal" requiring the user to go to a (fake) internal portal. The engineer downloaded and ran a provided "VPN Certificate Installer", which was actually a Cobalt Strike dropper. A Beacon was established on this engineer's workstation. This user was a local admin on their machine (common for engineers) but not a domain admin. However, the user had recently used **CyberArk PVWA** (Privileged Vault Web Access) to retrieve a password. In doing so, they had an active login cookie for CyberArk in their browser. The Red Team's beacon stole that session cookie from memory. They used the cookie to impersonate the user's web session to the CyberArk vault (which was internally reachable). As luck would have it, the user had access to some passwords of mid-tier servers. Among them, the team found credentials for a ServiceNow integration account (an account used by ServiceNow to query AD for user data). This account had read rights in AD and belonged to a group "ServiceNow_Integration". No obvious admin privileges yet, but it was a foothold.

Pivot: ServiceNow Compromise: Using the stolen ServiceNow integration credentials, the Red Team logged into the corporate ServiceNow instance (the account was a service account but had an interactive login to ServiceNow with admin rights, because it was used to configure the AD integration). Now inside ServiceNow as an admin, the team explored the data. They found in the "Incident" table multiple entries referring to "Vault password reset" and even an incident where a CyberArk admin pasted a one-time login token. They also accessed the "Credentials" store table of ServiceNow and found an entry storing a privileged account for *HashiCorp Vault* integration – encrypted, but ServiceNow allowed them to trigger a decryption by testing the connection. Through that, they retrieved a Vault API token that ServiceNow was using to pull secrets for automation. Armed with this Vault token, the team moved to target HashiCorp Vault.

Vault Breach: The Vault was accessible only from certain IPs, but the ServiceNow server was one of them. The Red Team leveraged their control of ServiceNow: they crafted a small script within a ServiceNow business rule that, when executed, would use the Vault token to read secrets from Vault's API. They executed this by impersonating the ServiceNow integration's functionality. Almost immediately, they got a trove of secrets from Vault – because the token they acquired was a highly privileged one (ServiceNow was pulling various secrets). Among these, crucially, was an **Azure service principal secret** that had Contributor rights in the Azure subscription, and a couple of domain account passwords (for service accounts) that were stored in Vault for safe keeping. Essentially, by compromising Vault via the token, they now had cloud and on-prem credentials.

Establishing C2 & Persistence Internally: Meanwhile, on the engineer's machine, the Beacon persisted by injecting into a legitimate process and setting up a run key. The team pivoted using the domain accounts from Vault – one was an account that had local admin rights on a bunch of servers (a service account). They used Cobalt Strike's lateral movement to push beacons to two servers where that account had admin (confirmed via SMB session tests). Now they had beacons on an application server and on a database server. The database server Beacon was used to dump data (customer info database) as a target proof. The application server Beacon was used to create a **reverse port forward** to the internal network (enabling the team's external C2 server to reach internal services via the beacon's connection). They then leveraged the port forward to access the HashiCorp Vault web UI (since they had the token, they could log in via API and see if any UI or further secrets were needed). They also port-forwarded RDP through Beacon to access the

ServiceNow Windows server with the stolen credentials, to ensure long-term control there.

Attack on Secret Vault (CyberArk): Now aiming higher, the team attempted to fully compromise CyberArk. Using the domain service account from Vault, they noticed that account was part of a group that had some privileges on the CyberArk server (maybe for backup tasks). They port-scanned the CyberArk vault server (which was on a segregated network but reachable from the app server Beacon due to firewall rules for backups) and found the API port open. They tried a known exploit for CyberArk (the 2018 RCE via serialized object) but the server was patched. Plan B: they used the stolen ServiceNow data to see if any CyberArk admin credentials were stored or mentioned. One ticket mentioned a default Master password that hadn't been changed during an update – jackpot. They tried those credentials on the CyberArk web login and surprisingly got in as a Vault administrator (this was a scenario of misconfiguration – perhaps a test vault user was left enabled). Once in the CyberArk UI, they utilized the vault's own tools to **export** all passwords in certain safes. This gave them domain admin (which they already had indirectly by now), local admin creds, and even some Unix root passwords.

Cloud Maneuvers with Stolen Secrets: Among the secrets from Vault was an Azure service principal as mentioned, which had Contributor to the main subscription. The Red Team used that credential (client ID and secret) to obtain an Azure access token and then escalated privileges in Azure. They created a new Azure AD application, added it as an Owner to the subscription (since Contributor could create a role assignment for Owner on itself due to a mis-scoped role), effectively giving them a backdoor global admin (Owner on subscription and with application credentials, they could try to further elevate in AAD if needed). They deployed a small VM in Azure and connected a Beacon to it as well (demonstrating they can use cloud resources for C2 if internal gets shut down).

Cobalt Strike Highlights: This scenario saw Cobalt Strike used in a blended attack – both on endpoints and through the cloud. The Beacon that pivoted through ServiceNow script wasn't a typical one; instead, the team used the credentials to retrieve data rather than running Beacon on ServiceNow (to avoid detection on that high-profile system). They did however run Beacon on standard servers to maintain footholds. The use of Cobalt Strike's pivoting (SSH pivot or sock proxy) allowed them to route traffic to Vault and other restricted systems without needing new malware there. They also leveraged Beacon's ability to impersonate tokens: for instance, when moving laterally on Windows servers, they used token impersonation to escalate privileges of processes (from service account to SYSTEM). At one point, an EDR on the database server flagged suspicious memory injection (Beacon's reflectively loaded DLL). The Red Team responded by using Cobalt Strike's spawn to create a sacrificial process that mimicked a Windows Update, then migrated the Beacon into it and killed the original, clearing the EDR alert. They also used the **Malleable C2** feature to adjust their beacon traffic pattern when they realized the EDR was doing behavioral detection on network traffic – switching to a different profile that blended with ServiceNow API calls. These adjustments kept them stealthy enough to complete objectives.

Outcome: In this second scenario, the Red Team demonstrated how an on-prem compromise can lead to a cloud and vault compromise by abusing centralized systems (ServiceNow, Vault). They managed to empty the secret stores (both CyberArk and Vault), gaining essentially every credential in the company. They showed the ability to forge machine identities by obtaining certificate keys (they exported a couple of private keys from Venafi through CyberArk – as one of the safes contained an HSM operator card PIN which they used on the Venafi console to extract a test certificate). Persistence was established at multiple levels: application backdoors, new cloud credentials, and data exfiltration channels. The attack flew under the radar because it leveraged legitimate tools – the ServiceNow and Vault accesses were via existing allowed channels (the integration accounts), and Cobalt Strike kept a low profile on standard servers.

These scenarios underscore how a Red Team weaves together different threads – cloud, on-prem, user accounts, and powerful management systems – to achieve a full compromise. Cobalt Strike's role as an enabler is evident: it provides the silent footholds and the post-exploitation toolkit to escalate and pivot at will

In real incidents, threat actors have similarly used Cobalt Strike beacons to maintain persistence while moving laterally and even proxying other tool traffic through the beacons to avoid direct connections By studying these examples, defense teams can better anticipate such multi-faceted attacks.

Defensive Strategies and Mitigations

After examining the many ways an enterprise can be compromised, it's crucial to consider how to defend against these scenarios. A layered defense strategy is required – there is no single silver bullet. Below is a brief outline of defensive measures corresponding to the tactics and weaknesses discussed, which an organization can implement to improve resilience:

- **Strengthen Identity Security**: Enforce phishing-resistant **multi-factor authentication** for all users, especially privileged roles

This includes methods like FIDO2 tokens or certificate-based auth that are harder to phish than simple push notifications. Implement **conditional access** policies in Azure AD to block login attempts from unusual locations or legacy protocols, thereby cutting off many credential abuse pathways. For on-prem AD, disable NTLM where possible and require signing/sealing for LDAP to prevent relay attacks (mitigating things like LAPS relays). Regularly review membership of privileged groups (Domain Admins, Global Admins, Account Operators) to minimize who has high access; remove any unnecessary accounts that can read sensitive data like LAPS passwords, and use tiered admin models to limit the blast radius if one admin is compromised.

- **Hardening Cloud Configuration**: Apply the principle of least privilege in Azure AD and Azure resources. Ensure no low-level user is owner of highly privileged service principals ; implement Azure AD Privileged Identity Management (PIM) so that even Global Admins must elevate just-in-time and their usage is audited. Monitor for creation of new application credentials or consent to new enterprise apps – these can signal abuse of the kind SpecterOps described. Limit and audit Intune Administrators; if possible, require separate approval for pushing high-risk scripts. In Intune, enable notifications or logging for new scripts added or devices rebooted unexpectedly, which might catch the Intune attack chain mid-step. For SaaS like ServiceNow, integrate it with the corporate SSO and enforce MFA, and monitor admin activities (like creation of new integration credentials or export of data).
- **Network Segmentation and Host Monitoring**: Segregate critical servers (domain controllers, CyberArk vault, CA servers, etc.) on network segments with strict access control. For example, the Vault and PAM systems should only be reachable from a jump-host by admins, not directly from user subnets. Implement LAPS properly but also monitor LDAP queries for the ms-MCS-AdmPwd attribute – any account unexpectedly reading those might indicate an attack. Use host-based intrusion detection or EDR on servers to detect Mimikatz and other memory dumping techniques – modern EDRs can often catch signatures of tools that grab LSASS contents. Additionally, ensure Windows Defender Credential Guard is

enabled on administrative workstations to protect creds from being dumped. Enable and collect Windows Event Logs like 4724 (password reset) and others that might indicate if an attacker is creating backdoor accounts or touching domain policy. In the cloud, enable Azure AD log auditing and sign-in logs; feed these to a SIEM and create alerts for anomalous events (like a disabled account suddenly logging in, or an Intune admin performing mass actions).

- **Protect Secret Management Systems**: For CyberArk/PPM, keep them patched religiously so known exploits won't work. Rotate and secure the Master passwords – consider using an HSM or hardware-backed module for the vault master key. Conduct regular permission audits on who can access what in the vault; remove any default or emergency accounts from daily use (store those credentials offline so they're not lingering electronically). Turn on CyberArk's own threat detection if available – it can sometimes detect suspicious vault access patterns. For HashiCorp Vault, use **MFA on Vault** for any privileged users, implement namespace and policy separation to contain breaches, and preferably use the Enterprise version's MFA and encryption features. The unseal keys should be split among multiple owners or secured in a KMS/HSM (auto-unseal) – and even there, control access to that KMS key tightly (with alerts on any usage). Monitor Vault audit logs for irregular token usage (like a token reading lots of secrets it normally doesn't). Vault's servers should run on dedicated machines with minimal software to reduce OS attack surface. For Venafi, ensure any integrated CAs have their own safeguards (HSM for the CA private key, admin approval for any high-value cert issuance). Activate Venafi's anomaly detection: it can flag unusual certificate requests. Use role-based access so that, for instance, obtaining a code-signing certificate requires a higher level of approval. And certainly, use 2FA for Venafi administrators.
- **Detect and Respond to C2 Activity**: Since tools like Cobalt Strike are so prevalent, organizations must invest in detecting their tell-tale behaviors. This includes monitoring for network patterns (e.g., Beacon's periodic check-ins) – though these can be masked, anomalies like a host making regular small HTTPS POST requests to an unknown domain can be spotted by an attentive NDR (Network Detection & Response) system or even carefully tuned firewall analytics. Endpoint monitoring can catch things like suspicious parent-child processes (e.g., Word spawning a powershell, a common initial access sign) or unexpected use of LOLBins (Living-off-the-land binaries) like rundll32 launching from a user temp directory. Cobalt Strike beacons often attempt to inject into known processes (explorer.exe, svchost.exe); EDR can alarm on injection events or on strange process memory protections being

changed. Employ decoy accounts and honeytokens as tripwires – for example, an AD account that shouldn't ever be accessed, if its hash gets dumped and used, that's an immediate red flag. Similarly, a fake secret in Vault or a dummy admin credential in CyberArk can serve as a canary – if someone tries to use it, you know the vault was breached.

- **Incident Response Preparedness**: Lastly, having an incident response plan and performing regular **red team vs blue team exercises** (purple teaming) helps. An organization should periodically test how quickly and effectively they can detect and stop a simulated attack. This might include drills where the security team has to spot a Beacon in the network or track an Intune misuse scenario. The idea is to practice and refine detection engineering continuously. Also, keep an eye on threat intelligence: many real attacks (like those by nation-states) are shared by agencies (e.g., CISA's advisories) and carry recommendations that can be directly applied.

In summary, defenses must address people, process, and technology. Regular training can help users resist phishing and recognize social engineering (so that initial foothold is harder to get)

Strong processes like thorough change management and least privilege administration reduce the chance of misconfigurations (like the ones exploited in our scenarios). And a multi-layered set of technical controls – from hardened configurations to continuous monitoring – can catch the subtle signs when an attacker slips through. The key is to assume breach: design the network such that even if an attacker gets in, it's hard for them to escalate and easy for you to isolate and eradicate them quickly.

Conclusion

Red Team operations in large enterprise environments reveal the complex interplay between cloud services, on-premises networks, and specialized security tools. In this paper, we dissected how a determined adversary can navigate through Azure Active Directory, exploit endpoint management solutions like Intune or LAPS, compromise critical IT platforms such as ServiceNow, and subvert secret management systems (CyberArk, PPM, Vault, Venafi) to achieve full enterprise dominance. The **introduction** established that Red Teaming is a realistic simulation of threats , and modern enterprises must be prepared for attacks that seamlessly blend cloud and on-prem techniques

We outlined a comprehensive **methodology** covering reconnaissance through actions on objectives, emphasizing how tactics like social engineering, credential theft, token replay, and lateral movement via legitimate channels are employed in concert.

The core **attack tactics** section provided a detailed look at specific vectors: from Azure AD application abuse where a low-privilege user can escalate to Global Admin , to exploiting LAPS by extracting stored local admin passwords ,to weaponizing Intune's device management against the enterprise's own endpoints , and hijacking ServiceNow to access sensitive internal data

We also examined how vaults and key management systems can be breached, despite their strong security, through vulnerabilities or misconfigurations, and how traditional Windows attacks (like pass-the-hash and AD CS abuse) remain relevant in an Azure-connected world.

In the **goals** section, we categorized what attackers aim for: cloud takeover, domain dominance, vault control, and data exfiltration. Each attack path we explored leads to one or more of these "crown jewel" outcomes. Speaking of **crown jewels**, we identified the highest-value targets such as domain controllers, Global Admin accounts, and the contents of secret vaults – essentially the assets that, if owned, give an adversary maximal control or leverage.

We then performed a **comparative analysis of secret management tools**. CyberArk's fortress-like vault and extensive PAM features make it powerful but also a prime target ; older solutions like PPM require careful isolation to remain secure; HashiCorp Vault offers strong crypto and flexibility but demands correct configuration and patching diligence ; and Venafi, while protecting machine identities, can be turned into an attack platform if an attacker abuses its certificate issuance capabilities Understanding these nuances helps defenders prioritize hardening measures for each system.

The **example scenarios** grounded the discussion in real-world style attack narratives. In Scenario 1, we saw a Global Admin and Domain Admin compromise achieved by chaining an Intune attack with on-prem privilege escalation, with Cobalt Strike beacons orchestrating the multi-stage operation. Scenario 2 showed an attack starting on-prem with a vault and ITSM breach that led to widespread credential theft and cloud abuse. These case studies highlighted how Cobalt Strike is leveraged by attackers for stealthy persistence and control , and how multiple attack vectors converge to amplify the impact of a breach. They illustrate that no single defense could have prevented those multifaceted assaults – a combination of preventive and detective controls was needed.

Finally, we outlined **defensive strategies** that correspond to each aspect of the attack chain. In summary, organizations must implement strong identity controls (e.g., MFA everywhere, least privilege, monitoring of admin activities), harden their cloud and on-prem configurations (patch systems, secure vaults with 2FA and network isolation, monitor risky actions), and maintain an agile detection and response capability (EDR, SIEM alerts on suspicious behavior, user training to resist social engineering). Regular red teaming and purple teaming exercises are invaluable for validating that these defenses work against realistic threats. As the CISA red team case showed, even well-secured organizations have room for improvement in detecting lateral movement and abuse of legitimate credentials

In conclusion, the assessments and techniques discussed underscore a key principle: **security is only as strong as the weakest link in a deeply interconnected chain**. Azure AD and cloud services offer incredible power and agility, but missteps in their configuration or integration with on-prem systems can open serious holes. Secret management tools greatly reduce risk in day-to-day operations, yet they become irresistible targets – concentration of secrets is like honey for bears. A PhD-level analysis of red teaming in such contexts affirms that a holistic approach is needed. One must consider the attacker's perspective across the entire enterprise terrain – from cloud tokens to domain protocols, from human factors to hardware. By doing so, and by rigorously testing and reinforcing each layer of defense, enterprises can raise the bar so high that even advanced adversaries will be detected and foiled before they reach the crown jewels. Ultimately, the value of Red Team exercises lies in revealing these insights proactively, so that real incidents can be prevented. Each weakness uncovered is an opportunity to fortify the enterprise and protect the integrity, confidentiality, and availability of its critical systems.

References:

1. IBM – Anderson, E. *What is red teaming?* (IBM, 2024).
2. Jumpsec Labs – *Red Teaming the Cloud: A Shift in Perspective* (2022).
3. SpecterOps – Hausknecht, R. *Attacking Azure & Azure AD, Part II* (2020).
4. SpecterOps – Robbins, A. *Death from Above: Lateral Movement from Azure to On-Prem AD* (2020).

5. Praetorian – Crosser, A. *Obtaining LAPS Passwords Through LDAP Relaying Attacks* (2020).
6. Savvy Security – Caraballo, J. *Microsoft ServiceNow Incident: Lesson in Credential Hygiene* (2024).
7. Tenable – Van de Wiele, T. *Dangerous Permissions (Intune) in Entra ID* (2023).
8. Vulmon – *CyberArk Password Vault Web Access RCE CVE-2018-9843* (2018).
9. Vulmon – *CyberArk Credential Provider Encryption Weakness CVE-2021-31796* (2021).
10. Cycognito – *Emerging Threat: HashiCorp Vault SSH Bypass CVE-2024-7594* (2024).
11. Venafi – *Press Release: How Snowden Breached the NSA (Venafi research)* (2013).
12. OmniCyber – *Case Study: Inside a Red Team Engagement* (2022).
13. CISA – *CSA: Red Team Shares Key Findings* AA23-059A (2023).