Winrm Acrtive Directory Enumeration

winrm_ad_enum.py

```
#!/usr/bin/env python3
"""
WinRM Active Directory Object Enumerator (HTTPS-only)
=====================================================

Connects to a Windows Domain Controller over WinRM/HTTPS (port 5986) and
retrieves ALL user and computer objects from Active Directory.

Output formats:
    - Pretty-printed table on stdout (default)
    - CSV files (one per object type)
    - JSON files (one per object type)

Requirements:
    pip install pywinrm

Usage:
    # Pull everything, print to screen
    python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\\admin'

    # Pull everything and save to CSV in ./ad_export/
    python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\\admin' \\
        --output-dir ./ad_export --format csv

    # Save to JSON, only users
    python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\\admin' \\
        --object-type users --format json --output-dir ./ad_export

    # Strict TLS (your DC has a cert signed by a CA your client trusts)
    python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\\admin' --verify-ssl
```

```
    # Limit search to a specific OU
    python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\\admin' \\
        --search-base "OU=Employees,DC=corp,DC=local"
"""

from __future__ import annotations

import argparse
import csv
import getpass
import json
import socket
import ssl
import sys
from contextlib import closing
from datetime import datetime
from pathlib import Path

try:
    import winrm
    from winrm.exceptions import (
        InvalidCredentialsError,
        WinRMOperationTimeoutError,
        WinRMTransportError,
    )
except ImportError:
    print("[!] The 'pywinrm' package is required. Install it with:")
    print("    pip install pywinrm")
    sys.exit(1)



# ---------------------------------------------------------------------------
# Output helpers
# ---------------------------------------------------------------------------
class C:
    OK = "\033[92m"
    WARN = "\033[93m"
    FAIL = "\033[91m"
    INFO = "\033[94m"
```

```
    BOLD = "\033[1m"
    END = "\033[0m"


def banner(text: str) -> None:
    line = "=" * 72
    print(f"\n{C.BOLD}{line}\n{text}\n{line}{C.END}")


def info(msg: str) -> None:
    print(f"{C.INFO}[*]{C.END} {msg}")


def ok(msg: str) -> None:
    print(f"{C.OK}[+]{C.END} {msg}")


def warn(msg: str) -> None:
    print(f"{C.WARN}[!]{C.END} {msg}")


def fail(msg: str) -> None:
    print(f"{C.FAIL}[-]{C.END} {msg}")


# ---------------------------------------------------------------------------
# Pre-flight checks
# ---------------------------------------------------------------------------
def check_tcp_port(host: str, port: int, timeout: float = 5.0) -> bool:
    """Quick TCP probe before attempting the full TLS+WinRM handshake."""
    info(f"Testing TCP connectivity to {host}:{port} ...")
    try:
        with closing(socket.create_connection((host, port), timeout=timeout)):
            ok(f"TCP port {port} is OPEN on {host}")
            return True
    except socket.timeout:
        fail(f"TCP port {port} timed out (firewall? listener not configured?)")
    except ConnectionRefusedError:
        fail(f"TCP port {port} refused the connection (HTTPS listener not configured?)")
```

```
    except socket.gaierror:
        fail(f"Could not resolve hostname '{host}' (DNS issue)")
    except OSError as e:
        fail(f"TCP error on port {port}: {e}")
    return False


def check_tls(host: str, port: int = 5986, timeout: float = 5.0) -> None:
    """Inspect the TLS certificate on the HTTPS listener (does NOT validate it)."""
    info(f"Inspecting TLS certificate on {host}:{port} ...")
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE
    try:
        with socket.create_connection((host, port), timeout=timeout) as sock:
            with ctx.wrap_socket(sock, server_hostname=host) as ssock:
                cert = ssock.getpeercert(binary_form=False)
                if cert:
                    subject = dict(x[0] for x in cert.get("subject", []))
                    issuer = dict(x[0] for x in cert.get("issuer", []))
                    ok("TLS handshake OK.")
                    print(f"    Subject : {subject}")
                    print(f"    Issuer  : {issuer}")
                    print(f"    Valid until: {cert.get('notAfter')}")
                else:
                    der = ssock.getpeercert(binary_form=True)
                    warn(f"TLS handshake OK. Cert hidden (likely self-signed, {len(der)}B
DER).")
    except Exception as e:
        warn(f"Could not retrieve TLS info: {e}")


# ---------------------------------------------------------------------
# WinRM/HTTPS session
# ---------------------------------------------------------------------
def open_session(host: str, port: int, user: str, password: str,
                 transport: str, verify_ssl: bool) -> winrm.Session:
    endpoint = f"https://{host}:{port}/wsman"
    info(f"Opening WinRM session over HTTPS: {endpoint} (transport={transport})")
```

```
    return winrm.Session(
        endpoint,
        auth=(user, password),
        transport=transport,
        server_cert_validation="validate" if verify_ssl else "ignore",
    )


def run_ps(session: winrm.Session, script: str) -> tuple[int, str, str]:
    """Run a PowerShell script remotely. Return (exit_code, stdout, stderr)."""
    r = session.run_ps(script)
    return (
        r.status_code,
        r.std_out.decode("utf-8", errors="replace"),
        r.std_err.decode("utf-8", errors="replace"),
    )


def ping_winrm(session: winrm.Session) -> bool:
    """Confirm the session works with a trivial command before the heavy lifting."""
    try:
        code, out, err = run_ps(session, "$env:COMPUTERNAME")
        if code == 0:
            ok(f"WinRM authenticated. Remote computer: {out.strip()}")
            return True
        fail(f"WinRM command returned exit code {code}: {err.strip()}")
    except InvalidCredentialsError:
        fail("Invalid credentials.")
    except WinRMTransportError as e:
        fail(f"WinRM transport error: {e}")
    except WinRMOperationTimeoutError as e:
        fail(f"WinRM operation timed out: {e}")
    except Exception as e:
        fail(f"Unexpected WinRM error: {type(e).__name__}: {e}")
    return False


# ---------------------------------------------------------------------------
# Active Directory enumeration scripts
```

```
# ---------------------------------------------------------------------------
# We ask PowerShell to emit JSON so Python can parse it reliably.
# ConvertTo-Json -Depth 4 -Compress gives us a single-line array.
# -ResultPageSize 1000 makes the AD cmdlets page through large directories
# efficiently instead of pulling everything at once.

AD_PROBE_PS = r"""
$ErrorActionPreference = 'Stop'
try {
    Import-Module ActiveDirectory -ErrorAction Stop
    Write-Output 'MODULE_OK'
} catch {
    Write-Output "MODULE_MISSING: $($_.Exception.Message)"
    exit 2
}
"""


def build_users_query(search_base: str | None) -> str:
    base_param = f"-SearchBase '{search_base}'" if search_base else ""
    return rf"""
        $ErrorActionPreference = 'Stop'
        Import-Module ActiveDirectory

        $props = @(
            'SamAccountName','UserPrincipalName','DisplayName','GivenName','Surname',
            'EmailAddress','Title','Department','Company','Manager','Office',
            'Enabled','LockedOut','PasswordLastSet','PasswordNeverExpires',
            'LastLogonDate','WhenCreated','WhenChanged','DistinguishedName',
            'MemberOf','SID','ObjectGUID'
        )

        $users = Get-ADUser -Filter * {base_param} `
            -ResultPageSize 1000 -Properties $props |
            ForEach-Object {{
                [PSCustomObject]@{{
                    SamAccountName        = $_.SamAccountName
                    UserPrincipalName     = $_.UserPrincipalName
                    DisplayName           = $_.DisplayName
```

```
                    GivenName             = $_.GivenName
                    Surname               = $_.Surname
                    EmailAddress          = $_.EmailAddress
                    Title                 = $_.Title
                    Department            = $_.Department
                    Company               = $_.Company
                    Office                = $_.Office
                    Enabled               = $_.Enabled
                    LockedOut             = $_.LockedOut
                    PasswordNeverExpires  = $_.PasswordNeverExpires
                    PasswordLastSet       = if ($_.PasswordLastSet) {{
$_.PasswordLastSet.ToString('o') }} else {{ $null }}
                    LastLogonDate         = if ($_.LastLogonDate)   {{
$_.LastLogonDate.ToString('o') }}   else {{ $null }}
                    WhenCreated           = if ($_.WhenCreated)     {{
$_.WhenCreated.ToString('o') }}     else {{ $null }}
                    WhenChanged           = if ($_.WhenChanged)     {{
$_.WhenChanged.ToString('o') }}     else {{ $null }}
                    DistinguishedName     = $_.DistinguishedName
                    SID                   = $_.SID.Value
                    ObjectGUID            = $_.ObjectGUID.Guid
                    GroupCount            = @($_.MemberOf).Count
                }}
            }}

        # ConvertTo-Json wraps a single object in an object (not array). Force an array.
        ,@($users) | ConvertTo-Json -Depth 4 -Compress
    """


def build_computers_query(search_base: str | None) -> str:
    base_param = f"-SearchBase '{search_base}'" if search_base else ""
    return rf"""
        $ErrorActionPreference = 'Stop'
        Import-Module ActiveDirectory

        $props = @(
            'Name','DNSHostName','SamAccountName','Enabled','OperatingSystem',
            'OperatingSystemVersion','OperatingSystemServicePack','IPv4Address',
```

```
            'IPv6Address','LastLogonDate','PasswordLastSet','WhenCreated','WhenChanged',
            'DistinguishedName','SID','ObjectGUID','Description','ManagedBy'
        )

        $computers = Get-ADComputer -Filter * {base_param} `
            -ResultPageSize 1000 -Properties $props |
            ForEach-Object {{
                [PSCustomObject]@{{
                    Name                = $_.Name
                    DNSHostName         = $_.DNSHostName
                    SamAccountName      = $_.SamAccountName
                    Enabled             = $_.Enabled
                    OperatingSystem     = $_.OperatingSystem
                    OperatingSystemVersion = $_.OperatingSystemVersion
                    OperatingSystemSP   = $_.OperatingSystemServicePack
                    IPv4Address         = $_.IPv4Address
                    IPv6Address         = $_.IPv6Address
                    Description         = $_.Description
                    ManagedBy           = $_.ManagedBy
                    LastLogonDate       = if ($_.LastLogonDate)   {{
$_.LastLogonDate.ToString('o') }}   else {{ $null }}
                    PasswordLastSet     = if ($_.PasswordLastSet) {{
$_.PasswordLastSet.ToString('o') }} else {{ $null }}
                    WhenCreated         = if ($_.WhenCreated)     {{
$_.WhenCreated.ToString('o') }}     else {{ $null }}
                    WhenChanged         = if ($_.WhenChanged)     {{
$_.WhenChanged.ToString('o') }}     else {{ $null }}
                    DistinguishedName   = $_.DistinguishedName
                    SID                 = $_.SID.Value
                    ObjectGUID          = $_.ObjectGUID.Guid
                }}
            }}

        ,@($computers) | ConvertTo-Json -Depth 4 -Compress
    """



# ---------------------------------------------------------------------
# Run queries and parse results
```

```
# ---------------------------------------------------------------------------
def parse_json_output(raw: str) -> list[dict]:
    """
    PowerShell's ConvertTo-Json sometimes emits BOM/whitespace.
    Strip and parse; always return a list of dicts (even for 0/1 items).
    """
    raw = raw.strip().lstrip("\ufeff")
    if not raw:
        return []
    data = json.loads(raw)
    if isinstance(data, dict):
        return [data]
    return list(data)


def enumerate_objects(session: winrm.Session, object_type: str,
                      search_base: str | None) -> list[dict]:
    if object_type == "users":
        info("Enumerating ALL user objects from Active Directory ...")
        script = build_users_query(search_base)
    elif object_type == "computers":
        info("Enumerating ALL computer objects from Active Directory ...")
        script = build_computers_query(search_base)
    else:
        raise ValueError(f"Unknown object_type: {object_type}")

    code, out, err = run_ps(session, script)
    if code != 0:
        fail(f"PowerShell query failed (exit {code})")
        if err.strip():
            print(f"    stderr: {err.strip()}")
        return []

    try:
        objects = parse_json_output(out)
    except json.JSONDecodeError as e:
        fail(f"Could not parse JSON returned by PowerShell: {e}")
        print("    First 500 chars of output:")
        print(f"    {out[:500]}")
```

```
        return []

    ok(f"Retrieved {len(objects)} {object_type} from Active Directory.")
    return objects


# ---------------------------------------------------------------------------
# Output writers
# ---------------------------------------------------------------------------
def print_table(objects: list[dict], object_type: str, limit: int = 20) -> None:
    """Print a brief preview table to stdout (full data goes to files)."""
    if not objects:
        warn(f"No {object_type} to display.")
        return

    if object_type == "users":
        cols = ["SamAccountName", "DisplayName", "Enabled", "LastLogonDate", "Department"]
    else:
        cols = ["Name", "DNSHostName", "OperatingSystem", "Enabled", "LastLogonDate"]

    # Compute column widths
    widths = {c: max(len(c), 8) for c in cols}
    for obj in objects[:limit]:
        for c in cols:
            v = str(obj.get(c, "") or "")
            widths[c] = min(max(widths[c], len(v)), 40)

    # Header
    print()
    header = " | ".join(c.ljust(widths[c]) for c in cols)
    print(f"{C.BOLD}{header}{C.END}")
    print("-+-".join("-" * widths[c] for c in cols))

    # Rows
    for obj in objects[:limit]:
        row = " | ".join(str(obj.get(c, "") or "")[:widths[c]].ljust(widths[c]) for c in cols)
        print(row)

    if len(objects) > limit:
```

```
        print(f"... ({len(objects) - limit} more rows not shown — see exported file)")


def write_csv(objects: list[dict], path: Path) -> None:
    if not objects:
        warn(f"Nothing to write to {path}")
        return
    # Union of keys preserves all columns even if some objects have nulls
    fieldnames: list[str] = []
    seen: set[str] = set()
    for obj in objects:
        for k in obj.keys():
            if k not in seen:
                seen.add(k)
                fieldnames.append(k)

    with path.open("w", newline="", encoding="utf-8") as f:
        writer = csv.DictWriter(f, fieldnames=fieldnames)
        writer.writeheader()
        writer.writerows(objects)
    ok(f"Wrote {len(objects)} rows -> {path}")


def write_json(objects: list[dict], path: Path) -> None:
    with path.open("w", encoding="utf-8") as f:
        json.dump(objects, f, indent=2, ensure_ascii=False)
    ok(f"Wrote {len(objects)} objects -> {path}")


# ---------------------------------------------------------------------------
# CLI
# ---------------------------------------------------------------------------
def parse_args() -> argparse.Namespace:
    p = argparse.ArgumentParser(
        description="Enumerate all users and computers from Active Directory over
WinRM/HTTPS.",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog=__doc__,
    )
```

```
    p.add_argument("--host", required=True, help="Target Domain Controller hostname or IP")
    p.add_argument("--user", required=True,
                   help=r"Username (DOMAIN\user, user@domain.local, etc.)")
    p.add_argument("--password", help="Password (prompted if omitted)")
    p.add_argument("--port", type=int, default=5986, help="HTTPS WinRM port (default 5986)")
    p.add_argument(
        "--transport",
        default="ntlm",
        choices=["ntlm", "kerberos", "ssl", "credssp"],
        help="WinRM auth transport (default: ntlm)",
    )
    p.add_argument("--verify-ssl", action="store_true",
                   help="Strictly validate the server's TLS certificate "
                        "(default: ignore, useful for self-signed)")
    p.add_argument(
        "--object-type",
        choices=["users", "computers", "all"],
        default="all",
        help="Which object type(s) to retrieve (default: all)",
    )
    p.add_argument("--search-base", metavar="DN",
                   help="Restrict search to an OU/DN (e.g. 'OU=Sales,DC=corp,DC=local')")
    p.add_argument("--output-dir", metavar="DIR",
                   help="Directory to write CSV/JSON files into (created if missing)")
    p.add_argument(
        "--format",
        choices=["csv", "json", "both"],
        default="csv",
        help="Output file format when --output-dir is given (default: csv)",
    )
    p.add_argument("--preview-rows", type=int, default=20,
                   help="How many rows to show in the on-screen preview (default 20)")
    return p.parse_args()


def main() -> int:
    args = parse_args()
    password = args.password or getpass.getpass(f"Password for {args.user}: ")
```

```
banner(f"WinRM/HTTPS Active Directory enumeration — {args.host}:{args.port}")

# --- pre-flight ---
if not check_tcp_port(args.host, args.port):
    fail("Aborting: HTTPS port unreachable.")
    return 1

check_tls(args.host, args.port)

# --- session ---
try:
    session = open_session(
        host=args.host,
        port=args.port,
        user=args.user,
        password=password,
        transport=args.transport,
        verify_ssl=args.verify_ssl,
    )
except Exception as e:
    fail(f"Could not build WinRM session: {e}")
    return 1

if not ping_winrm(session):
    return 1

# --- AD module probe ---
code, out, err = run_ps(session, AD_PROBE_PS)
if "MODULE_OK" not in out:
    fail("ActiveDirectory PowerShell module is not available on the remote host.")
    print(f"    stdout: {out.strip()}")
    print(f"    stderr: {err.strip()}")
    print("    Tip: target a Domain Controller, or install RSAT-AD-PowerShell.")
    return 1
ok("ActiveDirectory module is loaded on the remote host.")

# --- output dir setup ---
out_dir: Path | None = None
if args.output_dir:
```

```
        out_dir = Path(args.output_dir).expanduser().resolve()
        out_dir.mkdir(parents=True, exist_ok=True)
        info(f"Output directory: {out_dir}")

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    types_to_fetch = ["users", "computers"] if args.object_type == "all" else
[args.object_type]

    exit_code = 0
    for ot in types_to_fetch:
        banner(f"Active Directory — {ot.upper()}")
        objects = enumerate_objects(session, ot, args.search_base)
        if not objects:
            exit_code = exit_code or 2
            continue

        print_table(objects, ot, limit=args.preview_rows)

        if out_dir:
            stem = f"ad_{ot}_{timestamp}"
            if args.format in ("csv", "both"):
                write_csv(objects, out_dir / f"{stem}.csv")
            if args.format in ("json", "both"):
                write_json(objects, out_dir / f"{stem}.json")

    banner("Done")
    return exit_code


if __name__ == "__main__":
    try:
        sys.exit(main())
    except KeyboardInterrupt:
        print("\n[!] Interrupted by user.")
        sys.exit(130)
```

What this script does

HTTPS-only by design — no port 5985 fallback, no HTTP code paths. It connects to port 5986, verifies the TLS handshake, then enumerates **every** user and computer object from Active

Directory.

Pipeline

1. **TCP probe** on port 5986 — fails fast if the HTTPS listener isn't configured.
2. **TLS inspection** — pulls the cert subject/issuer/expiry so you can see what's on the wire (works even with self-signed certs).
3. **WinRM session** over HTTPS with NTLM (default) or Kerberos/CredSSP/SSL auth.
4. **AD module probe** — gracefully bails if the host isn't a DC and doesn't have RSAT installed.
5. **Bulk enumeration** — uses PowerShell's `Get-ADUser -Filter * -ResultPageSize 1000` (and the same for `Get-ADComputer`), which **pages through the entire directory** rather than hitting the LDAP page-size limit (default 1000).
6. **JSON-over-WinRM** — PowerShell emits `ConvertTo-Json -Compress`, Python parses it directly into dicts. No fragile text-table scraping.
7. **Output** — preview table on screen, optional CSV and/or JSON files in `--output-dir`.

Fields retrieved

For each user (22 fields): SamAccountName, UserPrincipalName, DisplayName, GivenName, Surname, EmailAddress, Title, Department, Company, Office, Enabled, LockedOut, PasswordLastSet, PasswordNeverExpires, LastLogonDate, WhenCreated, WhenChanged, DistinguishedName, SID, ObjectGUID, GroupCount.

For each computer (18 fields): Name, DNSHostName, SamAccountName, Enabled, OperatingSystem, OperatingSystemVersion, OperatingSystemServicePack, IPv4Address, IPv6Address, Description, ManagedBy, LastLogonDate, PasswordLastSet, WhenCreated, WhenChanged, DistinguishedName, SID, ObjectGUID.

All datetimes are converted to ISO-8601 (`.ToString('o')`) so they round-trip cleanly into CSV/JSON and can be parsed in any language.

Usage examples

bash

```
# Pull everything, just preview to screen
python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\admin'

# Pull everything, save to ./ad_export as CSV
python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\admin' \
    --output-dir ./ad_export --format csv

# Both CSV and JSON, only users
python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\admin' \
```

```
    --object-type users --format both --output-dir ./ad_export

# Limit to a single OU
python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\admin' \
    --search-base "OU=Employees,DC=corp,DC=local" \
    --output-dir ./ad_export

# Production: strict TLS verification (cert signed by a CA your client trusts)
python winrm_ad_enum.py --host dc01.corp.local --user 'CORP\admin' --verify-ssl

# Kerberos auth from a Linux box with valid ticket (kinit beforehand)
python winrm_ad_enum.py --host dc01.corp.local --user admin@CORP.LOCAL \
    --transport kerberos
```

Sample output (preview)

When you run it against a real DC, the on-screen preview looks like:

```
================================================================================
Active Directory — USERS
================================================================================
[*] Enumerating ALL user objects from Active Directory ...
[+] Retrieved 247 users from Active Directory.

SamAccountName | DisplayName         | Enabled | LastLogonDate              | Department
-+-------------+---------------------+---------+----------------------------+----------
Administrator  | Administrator       | True    | 2026-05-19T08:12:03+00:00  |
asmith         | Alice Smith         | True    | 2026-05-19T07:30:11+00:00  | Sales
bwilliams      | Bob Williams        | True    | 2026-05-15T14:10:55+00:00  | Engineering
...
... (227 more rows not shown — see exported file)

[+] Wrote 247 rows -> /home/you/ad_export/ad_users_20260519_153022.csv
```

Output files

If you pass `--output-dir`, you'll get timestamped files like:

```
ad_export/
├── ad_users_20260519_153022.csv
├── ad_users_20260519_153022.json
├── ad_computers_20260519_153022.csv
└── ad_computers_20260519_153022.json
```

The CSV uses the **union of all keys** across objects as the header row, so even sparsely-populated attributes (e.g., a user with no `Department`) keep their columns aligned. The JSON is `indent=2` and UTF-8, ready to load into pandas, jq, or any analytics tool.

A few things to know

- **Pagination is automatic.** `-ResultPageSize 1000` tells the AD cmdlets to fetch the directory in 1000-object chunks, which avoids hitting the LDAP server's `MaxPageSize` policy. For directories with tens of thousands of objects, this just works — no manual paging needed.
- **Memory.** A directory with 50,000 users will produce a JSON blob of roughly 30-50 MB transferred over WinRM. WinRM's default message size limit (`MaxEnvelopeSizeKb` = 500) should be fine since `-Compress` is used, but if you have huge directories and hit `WSManFault`, raise it on the server: `Set-Item WSMan:\localhost\MaxEnvelopeSizekb 8192`.
- **Permissions.** Authenticated users can read most of the directory by default, but `LastLogonDate`, `PasswordLastSet`, and `LockedOut` may show as null if the account doesn't have read access to those attributes. Run as a Domain Admin (or a delegated read account) to see everything.
- **`--verify-ssl`.** With self-signed certs, leave this off — the connection still uses TLS encryption, you just don't validate the cert chain. For production, install a CA-signed cert on the DC and use `--verify-ssl`.

Want me to add anything else — for example, also pulling **groups** with their member lists, or filtering by `LastLogonDate` to find stale accounts before exporting?

Revision #1
Created 19 May 2026 16:00:42 by Admin
Updated 19 May 2026 16:02:53 by Admin