Winrm Active Directory Enumeration using HTTP (5985)

winrm_ad_enum-py

```
#!/usr/bin/env python3
"""
WinRM Active Directory Object Enumerator (HTTP-only, port 5985)
================================================================

Connects to a Windows Domain Controller over WinRM/HTTP (port 5985) and
retrieves ALL user and computer objects from Active Directory.

About transport security:
    Port 5985 is plain HTTP at the transport level, but Windows WinRM still
    wraps the WS-Management SOAP payload in NTLM or Kerberos message-level
    encryption by default. That means credentials and AD data are NOT in
    cleartext on the wire, even though the URL is http://. The exception is
    the 'basic' transport: that one DOES send credentials base64-encoded and
    should never be used over HTTP unless you've explicitly enabled
    'AllowUnencrypted=true' on the server (don't).

    Default transport here: NTLM with message-level encryption — safe for
    lab use and typical for AD-joined environments.

Requirements:
    pip install pywinrm

Usage:
    # Pull everything, preview on screen
    python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\\admin'

    # Save to CSV in ./ad_export/
    python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\\admin' \\
        --output-dir ./ad_export --format csv
```

```
    # Only computers, JSON output
    python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\\admin' \\
        --object-type computers --format json --output-dir ./ad_export

    # Restrict to a specific OU
    python winrm_ad_enum_http.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 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 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 (HTTP 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


# ---------------------------------------------------------------------
# WinRM/HTTP session
# ---------------------------------------------------------------------
def open_session(host: str, port: int, user: str, password: str,
                 transport: str) -> winrm.Session:
    endpoint = f"http://{host}:{port}/wsman"
    info(f"Opening WinRM session over HTTP: {endpoint} (transport={transport})")
    if transport == "basic":
        warn("Using 'basic' over HTTP — credentials will be sent base64-encoded "
             "in cleartext. Only do this in a trusted lab.")
    return winrm.Session(
        endpoint,
        auth=(user, password),
        transport=transport,
        # server_cert_validation is irrelevant for HTTP, but pywinrm accepts it
        server_cert_validation="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:
        msg = str(e)
        fail(f"WinRM transport error: {msg}")
        # Common HTTP-specific hint
        if "401" in msg or "Unauthorized" in msg:
            print("    Hint: on HTTP, the server often requires NTLM/Kerberos "
                  "(not Basic). Try --transport ntlm.")
        elif "AllowUnencrypted" in msg:
            print("    Hint: server requires encryption. NTLM/Kerberos provide "
                  "message-level encryption — use --transport ntlm.")
    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
# ---------------------------------------------------------------------------
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 on a single object emits an object, not an array.
        # The leading comma forces an array context.
        ,@($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]:
    """Strip BOM/whitespace and always return a list of dicts."""
    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"]

    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)

    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))

    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
    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 AD over WinRM/HTTP (port 5985).",
        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=5985, help="HTTP WinRM port (default 5985)")
    p.add_argument(
        "--transport",
        default="ntlm",
        choices=["ntlm", "kerberos", "basic", "credssp"],
        help="WinRM auth transport (default: ntlm). 'basic' is INSECURE over HTTP.",
    )
    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/HTTP Active Directory enumeration — {args.host}:{args.port}")
    warn("Using HTTP transport (port 5985). NTLM/Kerberos provide message-level "
         "encryption; Basic does not.")

    if not check_tcp_port(args.host, args.port):
        fail("Aborting: HTTP port unreachable.")
        return 1

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

    if not ping_winrm(session):
        return 1

    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.")

    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 changed vs the HTTPS version

The script is identical in **what** it does — enumerate every user and computer with the same fields, same CSV/JSON output, same pagination — but the transport is different:

	HTTPS version	HTTP version (this one)
URL scheme	`https://...`	`http://...`
Default port	5986	**5985**
TLS check	`check_tls()` inspects cert	removed (no TLS to inspect)
`--verify-ssl` flag	yes	removed (no TLS)
`--transport ssl` option	yes	removed (no TLS)
`--transport basic`	available, encrypted by TLS	available, **explicit warning** — credentials base64-only

About "HTTP" being a misleading label

The most common misconception about WinRM on port 5985 is that it sends data in cleartext. It usually doesn't. Here's the actual picture:

- **NTLM transport** (this script's default) wraps the SOAP payload in **NTLM message-level encryption** (SPNEGO/GSSAPI sealing). Even though the URL says `http://`, the body on the wire is encrypted with a session key derived from your credentials. Someone sniffing the network sees ciphertext.
- **Kerberos transport** does the same with Kerberos session keys.
- **CredSSP transport** also encrypts, plus enables credential delegation.
- **Basic transport** is the *only* option that sends credentials base64-encoded and the payload in cleartext. Windows refuses Basic over HTTP by default (`AllowUnencrypted=false`). That's why I added the explicit warning when `--transport basic` is selected — and you really shouldn't use it outside a closed lab.

So in practice, this HTTP script is approximately as confidential as the HTTPS one *as long as you stick with NTLM or Kerberos*. The HTTPS version still wins for two reasons: it doesn't rely on the auth-protocol's sealing being correctly negotiated, and TLS gives you server-identity verification (the cert), which NTLM alone does not.

Usage examples

bash

```
# Default: NTLM auth over HTTP/5985
python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\admin'

# Save everything to CSV
python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\admin' \
    --output-dir ./ad_export --format csv

# Only computers, JSON output, narrowed to one OU
python winrm_ad_enum_http.py --host dc01.corp.local --user 'CORP\admin' \
    --object-type computers --format json \
    --search-base "OU=Servers,DC=corp,DC=local" \
    --output-dir ./ad_export

# Kerberos auth (Linux client needs valid ticket from `kinit`)
python winrm_ad_enum_http.py --host dc01.corp.local --user admin@CORP.LOCAL \
    --transport kerberos

# Lab-only: Basic auth (server must have AllowUnencrypted=true and Basic enabled)
python winrm_ad_enum_http.py --host dc01.local --user admin --transport basic
```

If you get a 401 or "AllowUnencrypted" error

The script catches `WinRMTransportError` and prints a hint, but here's the field reference:

- **401 Unauthorized with NTLM** → The server may require Kerberos, or the user has no logon right. Try `--transport kerberos` (after `kinit`) or check the account's "Allow log on through Windows Remote Management" privilege.
- **"The WS-Management service does not support the request"** with Basic → `AllowUnencrypted` is false on the server (the secure default). Either switch to `--transport ntlm`, or on the Windows server: `Set-Item WSMan:\LocalHost\Service\AllowUnencrypted -Value $true` and enable Basic in `WSMan:\LocalHost\Service\Auth\Basic`. (Lab only — don't do this in production.)
- **403 / "TrustedHosts"** errors generally come from the **client** side on Windows, not Linux/pywinrm — not applicable here.

You now have three scripts

```
winrm_ad_test.py         # Both ports + connectivity tests + many AD queries
winrm_ad_enum.py         # HTTPS-only, bulk user/computer export
winrm_ad_enum_http.py    # HTTP-only,  bulk user/computer export  ← this one
```

Want me to merge the two enumeration scripts into a single dual-mode tool with a `--scheme http|https` flag so you only have to maintain one file?

Revision #1
Created 19 May 2026 16:02:58 by Admin
Updated 19 May 2026 16:04:10 by Admin