TROOPERS 2026

From Code to Coverage

A Detection Engineer's Journey Through the LDAP Wilderness

```
$ # Six months. Nine blind spots. One detection stack.
$ wc -l ~/endpoints.csv          # MDR scale
  ~5,000,000
$ ls ~/sigma/attempts/ | wc -l   # too many to count
$ grep -rc 'blind spot' ~/blog/
  9
```

[speaker] Andrew Schwartz

[handle] @4ndr3w6S

// WHOAMI

```
$ ldapsearch -x -H ldap://troopers.de \
    -b 'ou=speakers,dc=troopers,dc=de' \
    '(sAMAccountName=4ndr3w6S)'

dn: CN=Andrew Schwartz,OU=Speakers,DC=troopers,DC=de
displayName:        Andrew Schwartz
sAMAccountName:     4ndr3w6S
title:              Principal Detection Engineer
company:            Huntress
description:        AD tradecraft · LDAP · Sigma
businessCategory:   Detection · Kerberos research
memberOf:           CN=Detection Engineers
                    CN=Always Thinking Purple
comment:            Diamond Ticket · DACL · dMSA Ouroboros
info:               Tottenham Hotspur · COYS · chess
drink:              old fashioned // boulevardier

# Result: 1 entry returned   ( visited: many )
```

Six months. 5M endpoints. No baseline. The stack that finally worked.

Why I wrote dozens of broken rules

The failure story

The constraint: *5M endpoints at MDR pace — every rule has to generalize.*

Signatures: a clean Impacket rule, tested — zero hits. The DC rewrote it first.

ADWS: every Get-ADUser query logs ::1 — never the attacker.

cLDAP: ldapnomnom swept 40k usernames — zero Event 1644.

Obfuscation: ~70% of MaLDAPtive techniques produce no 1644 at all.

Stats: then one z-score caught tools I had never seen.

The fix: one source is never enough — signatures + behavior + stats + ETW.

What you leave with

- Detection patterns for Impacket, BloodHound, SOAPHound
- Whitespace-agnostic detection patterns
- SDFlags + (!FALSE) behavioral hunting
- Statistical detection that catches unknown tools
- ADWS source IP attribution (Event 5156 ↔ 1644)
- netlogon.log for LDAP Ping detection
- LDAPMon (client-side ETW) for obfuscation
- Where each log source is blind — and the fix

// THE PROBLEM

LDAP recon is every attack's first move

BloodHound | SharpHound | Impacket | SOAPHound | ldapnomnom | MaLDAPtive

50+

valid whitespace variants of one filter

::1

source IP Event 1644 shows for every ADWS query

0

Event 1644 from ldapnomnom enumerating 40k usernames

~70%

of obfuscation techniques produce NO Event 1644

Signatures break the moment you meet a real attacker. You need a different mental model.

// THE AD KILL CHAIN · WHERE THIS TALK LIVES

Out of scope: Kerberoasting, ADCS abuse, delegation paths, DCSync, lateral movement, persistence — ***all depend on recon working first.***

Reference: **github.com/infosecn1nja/AD-Attack-Defense**

Three arcs. Nine parts. One thesis.

Talk order follows the attacker's path, not the release calendar.

ARC 1 — SIGNATURE DETECTION

PUBLISHED

What tools send vs. what gets logged.

- 01 OID Transformations
- 02 Whitespace & Formatting
- 03 SDFlags Hidden Params
- 04 SOAPHound (!FALSE)

ARC 2 — BLIND SPOTS

IN PROGRESS

Where Event 1644 fundamentally fails you.

- 5A ADWS: the ::1 Blind Spot
- 5B Event 5156 — Source IP Attribution
- 06 LDAP Ping / Netlogon

ARC 3 — BEYOND SIGNATURES

IN PROGRESS

Breaks even a perfect Event 1644 setup.

- 07 Statistics / Volumetric
- 08 Low & Slow Evasion
- 09 Obfuscation / LDAPMon

One thesis: Event 1644 alone is not enough. ***You need the forest, not the tree.***

EVT:1644

Before anything else: turn on Event 1644

```
# Two registry keys. One reboot-free.
# Run as admin on the DC (EARTH-DC).

$diag = 'HKLM:\SYSTEM\CurrentControlSet\
         Services\NTDS\Diagnostics'
$param = 'HKLM:\SYSTEM\CurrentControlSet\
         Services\NTDS\Parameters'

# 1. Turn on diagnostic logging (level 5)
Set-ItemProperty $diag `
  -Name '15 Field Engineering' `
  -Value 5 -Type DWord

# 2. Set expensive/inefficient thresholds
Set-ItemProperty $param `
  -Name 'Expensive Search Results Threshold' `
  -Value 1 -Type DWord

# Lab: threshold = 1 (log everything)
# Prod: threshold = 1000 (tune to noise)
```

What you get

Event 1644 in **Directory Service** log.

Fires for every LDAP search that crosses the threshold.

Level 5 is REQUIRED — lower levels won't emit 1644.

⚠ CPU & volume caveats

- Level 5 + threshold=1 = every query logged.
- Measure EPS in lab first. Production DCs can see 10-50k EPS.
- Microsoft warns: perf impact on busy DCs. Tune threshold to your floor.
- ***Not a CIS/NIST required control — it's a diagnostic feature.***

Everything downstream assumes 1644 is on. If it isn't, the rest is theater.

// LIVE DEMO

Enable-MaxLDAPLogging.ps1

Flip Field Engineering = 5 + tune thresholds — no reboot.

EVT:1644

Enable-MaxLDAPLogging — run + verify

! LIVE

DEMO 01 · Enable-MaxLDAPLogging

EVT:1644

Raw Event 1644 — from my DC, right now

```
Log Name:      Directory Service
Source:        Microsoft-Windows-ActiveDirectory_DomainService
Event ID:      1644
Message:
  Internal event: A client issued a search operation with the
  following options.

  Client:       [::1]:60780
  Starting node:
    DC=marvel.local,cn=MicrosoftDNS,DC=DomainDnsZones,DC=marvel,...
  Filter:
    ( & (objectCategory=CN=Dns-Node,CN=Schema,CN=Configuration,
        DC=marvel,DC=local) (uSNChanged>=572542) )
  Search scope:  onelevel
  Attribute selection:  dc, dnsRecord, nTSecurityDescriptor, ...
  Visited entries: 53     Returned entries: 53
  Used indexes:    idx_objectCategory:53:N
  Search time (ms): 31
  User:  MARVEL\loki
```

Read every column

CLIENT
[::1] — query came via ADWS,
not the real attacker IP.

FILTER
Bitwise op already normalized.
Chapter 01 problem.

ATTRIBUTES
nTSecurityDescriptor present.
ACL recon footprint.

CONTROLS
SDFlags = the BloodHound tell.
Chapter 03 detection.

STATS
Visited / Returned / Pages.
Chapter 07 math fodder.

Note: Client shows **[::1]** — *this query came via ADWS. That's the Chapter 5A problem.*

CHAPTER 01

OID Transformations

Your rule never had a chance. The DC rewrote the query before logging.

EVT:1644

What the attacker sends vs. what you log

IMPACKET SENDS

```
(&(objectCategory=person)
 (objectClass=user)
 (userAccountControl:
  1.2.840.113556.1.4.803:
  =524288))
```

EVENT 1644 LOGS

```
(&(objectCategory=person)
 (objectClass=user)
 (
  userAccountControl&524288
 ))
```

LDAP RFC 4515 extensible match
1.2.840.113556.1.4.803 = bitwise AND

OID has been normalized to the & operator
Your regex for "1.2.840" matches nothing.

LESSON: Write rules against the logged form, not the wire form. Know both.

Four transformation OIDs

OID	Operator	Appears in 1644 as
1.2.840.113556.1.4.803	Bitwise AND	attr&value
1.2.840.113556.1.4.804	Bitwise OR	attr|value
1.2.840.113556.1.4.1941	RULE_IN_CHAIN	attr:chain:=value
1.2.840.113556.1.4.2253	DN_WITH_DATA	attr:dnlink:=value

WORKING SIGMA RULE

```
title: Impacket LDAP Recon
logsource:
  product: windows
  service: ldap
detection:
  sel_bitwise:
    EventID: 1644
    SearchFilter|re:
     '&\d{3,}'
  condition: sel_bitwise
level: medium

# matches '&524288' etc.
# not '1.2.840.113556...'
```

CHAPTER 02

Whitespace & Formatting

Fifty ways to write the same query. One rule to match them all.

EVT:1644

Same query. Seven of fifty valid forms.

```
#01  (&(objectClass=user)(cn=admin))
#02  ( & (objectClass=user) (cn=admin) )
#03  (&  (objectClass=user)  (cn=admin))
#04  (&(ObjectClass=user)(CN=admin))
#05  (&(objectClass=USER)(cn=ADMIN))
#06  (&\n(objectClass=user)\n(cn=admin))
#07  (&(objectClass=\75ser)(cn=admin))
```

All seven: semantically identical to AD. All seven: break a naive `|contains` rule.

EVT:1644

Normalize, then match

NAIVE — BREAKS ON FORMATTING

```
detection:
  sel:
    SearchFilter|contains:
     '(&(objectClass=user)(cn='
  condition: sel
```

CORRECT — NORMALIZE FIRST

```
detection:
  sel:
    SearchFilter|re|i:
     '\(\s*&\s*\(\s*objectclass'
  condition: sel
```

The performance tradeoff

- Regex runs 3–4× slower than |contains on high-volume 1644 data
- Pre-filter with EventID and an anchor token (e.g. 'objectclass'), then regex
- Measure events/sec on YOUR DC before shipping — regex on 50k EPS hurts

CHAPTER 03

SDFlags

The server-side control BloodHound asks for. Detection gold.

EVT:1644

SDFlags: the BloodHound tell

What it is

Server-side LDAP control (OID 1.2.840.113556.1.4.801).

Tells AD which parts of nTSecurityDescriptor to return:

```
0x01  owner
0x02  group
0x04  DACL  ← access control
0x08  SACL  ← audit
```

SharpHound asks for 0x4 or 0x5. SoaPy and SOAPHound both ask for 0x7.

Over ADWS, nTSecurityDescriptor is impossible WITHOUT this control — its presence proves SDFlags was set.

What the event looks like

```
Event 1644
Client: 10.1.1.13
Attributes:
  nTSecurityDescriptor
Controls:
  SDFlags=0x4  (DACL only)
  SDFlags=0x5  (DACL+Owner)
  SDFlags=0x7  (DACL+Owner+Group)
Entries returned: 3,812
Entries visited:  4,201

# SDFlags 0x4/0x5/0x7 catches the family.
SharpHound (0x4/0x5); SoaPy & SOAPHound
(0x7).
```

EVT:1644

Why attackers WANT nTSecurityDescriptor

What's inside it

The security descriptor for every AD object:

Owner	— who owns the object
Group	— primary group
DACL	**— WHO can do WHAT**
SACL	— audit (rarely set)

The DACL is the attack graph.
Every ACE = a potential edge
in BloodHound's path-finder.

Why they want it

Every ACE = a possible privesc.

- GenericAll / WriteDACL
- WriteOwner / ForceChangePwd
- AddSelf / AddMember
- RBCD source SID writes
- AdminSDHolder bypass

Reading the DACL is how they pick their next move.

Why SDFlags is forced

Query nTSecurityDescriptor without SDFlags:

1. AD tries all 4 parts (default)
2. SACL needs SeSecurityPriv
3. Standard user → DENIED
4. AD returns NOTHING

SDFlags = 0x4 or 0x5 skips SACL → success. So they always set it.

Legitimate admins query specific objects. Attackers query everything. SDFlags presence is the difference.

EVT:1644

What the rule actually looks like

```yaml
title: BloodHound Family ACL Enumeration
id: 8fa99195-83b7-4c0e-b8b9-4026b444ce19
status: stable
author: Andrew Schwartz
tags:
  - attack.discovery
  - attack.t1087.002
  - tool.sharphound
  - tool.soapy
logsource:
  product: windows
  service: directory-service
detection:
  selection_base:
    EventID: 1644
    Controls|contains: '1.2.840.113556.1.4.801'
    # AttributeList check removed — SDFlags presence sufficient
  selection_sdflags:
    ControlValue: ['0x4', '0x5', '0x7']  # all 3
  condition: selection_base and selection_sdflags
level: high
```

What it catches

- SharpHound (default + DCOnly)
- BloodHound.py
- SOAPHound (ACL runs)
- Manual ACL recon

➔ *0x4, 0x5, 0x7 catches all.*

Where it lives

Huntress production rule.

Plus 6 more in the repo:

- Generic SDFlags
- ADCS template recon
- Whitespace-agnostic OID

Not pseudocode. Not pseudo-Sigma. This is the rule.

CHAPTER 04

SOAPHound

A clever (!FALSE) optimization that becomes a fingerprint.

EVT:1644

The (!FALSE) fingerprint

The clever-to-a-fault optimization

SOAPHound enumerates with **(!soaphound=*)** — a presence check for an attribute that doesn't exist. Since nothing ever has soaphound=, AD's query optimizer collapses the inner expression to **FALSE** and the outer **(!FALSE)** becomes TRUE — returning every object.

```
detection:
  sel:
    EventID: 1644
    SearchFilter|re: '\(\s*!\s*\(\s*FALSE'
  condition: sel
---
correlation:
  type: event_count
  rules: [soaphound_false]
  group-by: User
  timespan: 5m
  condition: { gte: 10 }
```

Stop hunting names. Hunt behavior.

SIGNATURE — dies on rename	PATTERN — survives evasion
String 'SOAPHound' in filter	(!(nonexistent=*)) with high entry count
User-agent = 'SharpHound'	SDFlags ∈ {0x4,0x5,0x7} OR nTSecurityDescriptor in attrs
Specific OID 1.2.840.113556.1.4.803	&\d{3,} — any bitwise operator + value
Query matches known tool fingerprint	EntriesVisited/EntriesReturned ratio

EVT:1644

Same moment — attacker view + DC view

// ATTACKER · SanctuaryII-RTO

```
C:\Users\loki.MARVEL\Desktop> SOAPHound.exe ^
   --user loki@marvel.local --password ******** ^
   --dc 10.1.1.11 --bhdump -o bh_output --nolaps

Loading cache from disk
Loaded cache with stats: 264 ID to type mappings.
 264 name to SID mappings.
-------------
Gathering AD data
ADWS request with ldapbase (DC=marvel,DC=local),
ldapquery: (!soaphound=*) and ldapproperties:
[name, sAMAccountName, cn, dNSHostName, objectSid,
 nTSecurityDescriptor, member, ... gPOptions]
Gathering AD data complete
-------------
Gathering DomainTrusts data
ldapquery: (trustType=*)
Gathering DomainTrusts data complete
Output files with prefix "full" -> bh_output\
```

// DEFENDER · EARTH-DC · Event 1644

```
# Directory Service log
# Credential: MARVEL\loki

Filter:  ( ! (FALSE) )

Attributes:  name, sAMAccountName,
  cn, dNSHostName, objectSid,
  objectGUID, primaryGroupID,
  distinguishedName, pwdLastSet,
  servicePrincipalName, sIDHistory,
  nTSecurityDescriptor,
  userAccountControl, member,
  adminCount, msDS-AllowedToDelegateTo,
  gPLink, gPOptions, objectClass
  + 13 more  (32 attributes total)

Server controls: SDflags:0x7  ← full SD (ACL dump)
Entries Visited:  3,815
Entries Returned: 334
```

SOAPHound's (!soaphound=) becomes (!(FALSE)). The fingerprint AD writes for you.* ***Two sides of the same moment — from my lab.***

// LIVE DEMO

soaphound.exe --bhdump

Build the cache → dump BloodHound — (!(FALSE)) lands in 1644.

EVT:1644

soaphound --bhdump — the (!FALSE) tell, live

! LIVE

DEMO 02 · SOAPHound (!FALSE) capture

CHAPTER 5A

PowerShell → ADWS

Get-ADUser is the attacker's love letter. Event 1644 logs ::1 — not them.

EVT:1644

PowerShell → ADWS → ::1 → 1644

“Source IP attribution for ADWS queries is not really possible.”

— BloodHound community Slack

...until you look at a different log.

The BloodHound Slack exchange

anon

what's your preferred way of opsec-safe AD enum — ADSI or ADWS?

anon

ADWS. but client always appears as the DC itself — service talks to LDAP on loopback.

anon

I've not found a way to correlate the originating host. alerting is very difficult.

Charlie Clark

surely you could with a network event? 5156?

me

Challenge accepted.

The data was always there. Nobody was correlating it.

CHAPTER 5B

Event 5156 — Port Correlation

Join the external port to the proxied 1644. Source IP restored.

MULTI-EVT

Event 5156 restores the source IP

```
# (1) External 5156 — the real attacker hits ADWS
EventID: 5156  SrcIP: 10.1.1.13  SrcPort: 49871  DstIP: 10.1.1.11  DstPort: 9389
```

```
# (2) Internal 5156 — ADWS proxies to local LDAP
EventID: 5156  SrcIP: ::1  SrcPort: 52318  DstIP: ::1  DstPort: 389
```

```
# (3) 1644 — logs localhost and the proxied source port 52318
EventID: 1644  Client: ::1:52318  Filter: (&(objectClass=user))
```

JOIN (2).SrcPort == (3).ClientPort · JOIN (1) ≈ (2) within 60–80 ms

// LIVE DEMO

Get-ADWSAttribution-dual.ps1

Restore the source IP for ADWS queries — port match + timestamp fallback.

MULTI-EVT

Get-ADWSAttribution-dual.ps1 - recorded

▶ VIDEO

DEMO 03 · Get-ADWSAttribution.ps1

MULTI-EVT

The event chain — captured, not theorized

```
SUSPICIOUS ADWS QUERY DETECTED  —  MARVEL.local  (12/15 lab capture)

  Source IP:   10.1.1.13    User:  MARVEL\loki
  Correlation: TIMESTAMP | High (61.6 ms)

  [5156] 17:30:08.214 - Security log (WFP Connection)
         Src: 10.1.1.13:48594  ->  DC:9389 (ADWS)
         App: microsoft.activedirectory.webservices.exe
                                                    | 46.4 ms
  [1138] 17:30:08.260 - Directory Service (LDAP Op Start)
         Client: [::1]:53149    User SID: S-1-5-21-...-1105
         >> ADWS proxy to LDAP (localhost)
                                                    | 15.2 ms
  [1644] 17:30:08.275 - Directory Service (Expensive Search)
         Client: [::1]:53149    User: MARVEL\loki
         Filter:     (&(objectClass=user)(objectCategory=...))
         Attributes: [all_with_list]nTSecurityDescriptor
         Controls:   SDflags:0x7  ← full SD (BloodHound)
         Stats:      Visited 53 / Returned 53 / Pages 1967

  [!]  Indicators: ACL enumeration, Full SD (BloodHound pattern)
```

Three events. ~62ms apart. Joined back to 10.1.1.13 / MARVEL\loki. ***Not theoretical — this is from my lab.***

BONUS · CH 5B

Multi-Event Correlation

When 1644 isn't enough: stitch 1138, 1139, 5156, and 1644 into one story.

The full event chain

} bonus events on client (optional)

Sysmon 7*	[client†]	Image load of Microsoft.ActiveDirectory.Management.dll (if Sysmon config logs it)
PS 4104	[client]	PowerShell script block logging: Get-ADUser -Filter *
5156	[client]	Outbound TCP to DC:9389 — DLL connects to ADWS
1138	[DC]	ADWS received connection — authenticated user attribution
1139	[DC]	ADWS request — cmdlet name, application identity
5156	[DC]	ADWS proxies to ::1:389 — ephemeral port recorded
1644	[DC]	LDAP query logged with client=::1:<port>

Correlate by timestamp + user + port. Full tool + source + user + query attribution.

** Sysmon Event 7 fires only if Sysmon config tracks DLL loads (most do not by default). PowerShell 4104 requires script block logging via GPO.*

CHAPTER 06

ldapnomnom

Event 1644 is blind by design. netlogon.log catches every query.

NETLOGON

LDAP Ping is not LDAP search

Regular LDAP Search

- Requires authentication
- Queries the AD directory tree
- Processed by ntdsa.dll in lsass
- **Event 1644 fires**

Used by: BloodHound, SharpHound, Impacket, PowerView

LDAP Ping (cLDAP)

- ANONYMOUS — no auth needed
- Does NOT touch the directory
- Intercepted by Netlogon service
- **Event 1644 NEVER fires**

Used by: ldapnomnom (TCP), cldap_ping.py (UDP), Windows DC Locator (UDP)

NETLOGON

netlogon.log sees every username

Enable with: nltest /dbflag:0x2080ffff

```
# %windir%\debug\netlogon.log  —  EARTH-DC, MARVEL.local

# HIT — valid enabled account:
[MAILSLOT] Received ping from (null) thor on UDP LDAP
[MAILSLOT] EARTH-DC: Ping response 'Sam Logon Response Ex' thor

# MISS — account does not exist:
[MAILSLOT] Received ping from (null) magneto on UDP LDAP
[MAILSLOT] EARTH-DC: Ping response 'Sam User Unknown Ex' magneto

# DISABLED — defender sees, attacker does NOT:
[MAILSLOT] Received ping from (null) hawkeye on UDP LDAP
[MISC]     EARTH-DC: NlSamVerifyUserAccountEnabled: hawkeye DISABLED
[MAILSLOT] EARTH-DC: Ping response 'Sam User Unknown Ex' hawkeye

# HONEYTOKEN — critical detection opportunity:
[MAILSLOT] Received ping from (null) svc_backup on UDP LDAP
[MAILSLOT] EARTH-DC: Ping response 'Sam Logon Response Ex' svc_backup
```

Always follow the code

THE README

"No Windows audit logs generated."

Wrong.

THE SOURCE CODE

ldapnomnom uses TCP, not UDP.

Verified.

THE EVENT LOGS

Event 5156 fires on every connection.

Confirmed.

Don't trust tool READMEs. Read the source. Verify with the logs.

The defender advantage: honeytoken gold

Disabled accounts + honeytokens

The attacker gets `'Sam User Unknown Ex'` for any account that is disabled OR doesn't exist — the same response. ldapnomnom treats it as a miss. BUT netlogon.log shows **`NlSamVerifyUserAccountEnabled`** before the response — naming the disabled account explicitly.

*Plant a disabled account named **`svc_backup`**. Watch netlogon.log. Catch the attacker.*

Caveats you need to know

- Source IP = `(null)`. netlogon.log doesn't record it. Pair with Event 5156 for TCP attribution.
- Transport label says "UDP LDAP" even for TCP connections. *Format quirk — don't trust it.*
- Enable with `nltest /dbflag:0x2080ffff`. Logs to `%windir%\debug\netlogon.log`

// LIVE DEMO

ldapnomnom -d sorpanos.local

Spray usernames over cLDAP — netlogon fills, 1644 stays empty.

NETLOGON

ldapnomnom — netlogon fires, 1644 silent

! LIVE

DEMO 04 · ldapnomnom (zero 1644)

CHAPTER 07

The Math Never Lies

Statistics catch tools you've never seen. Here's why it works.

EVT:1644

Reminder: Step 0 from pre-flight

Two registry paths

```
# (1) Diagnostic logging level
HKLM\SYSTEM\CurrentControlSet\
  Services\NTDS\Diagnostics

  '15 Field Engineering' = 5
  # level 5 REQUIRED for 1644

# (2) Query thresholds
HKLM\SYSTEM\CurrentControlSet\
  Services\NTDS\Parameters

  'Expensive Search
   Results Threshold'    = 10000
  'Inefficient Search
   Results Threshold'    = 1000
  'Search Time
   Threshold (msecs)'    = 30000
```

The reality

Default: DISABLED (level 0)

*Only level 5 emits 1644 — **levels 1-4 don't.***

*Default thresholds are **noisy-safe** (10k entries, 30s). Tune lower once baseline is stable.*

Event lives in: **Directory Service log (not Security)**

You cannot detect what you do not log. Step 0: enable 1644.

EVT:1644

Performance impact — and how to tune it

Microsoft: *"Logging Event ID 1644 events might impact server performance."*

Tune thresholds for your env

BASELINE — *Safe. Quiet. Misses fast recon.*

Default (10k / 1k / 30s)

INVESTIGATE — *Catches most recon tools*

Medium (500 / 100 / 1s)

HUNT — *SOAPHound/BloodHound caught. CPU cost.*

Aggressive (100 / 50 / 50ms)

DEBUG — *LAB ONLY. Logs everything.*

Max (1 / 1 / 1ms)

What drives the cost

- Query volume on the DC
- Lower thresholds = more events
- Event log write overhead
- Directory Service log size

No benchmark from MS. Measure in your env.

Start at INVESTIGATE. Watch DC CPU. Tighten once stable. You are the environment expert.

Four chapters of whack-a-mole

Ch 01	OID transformations	→ *AD rewrites it — the log never shows the string*
Ch 02	Whitespace-agnostic regex	→ *Add one space — the pattern misses*
Ch 03	SDFlags parameter match	→ *Change the flag — 0x4 / 0x5 / 0x7*
Ch 04	SOAPHound (!FALSE)	→ *Infinite nonsense attributes to choose from*

"We've been playing whack-a-mole with syntax for four chapters. Time for a paradigm shift."

The paradigm shift

> *“Attackers need the data. Getting that data creates mathematical patterns that cannot be hidden, regardless of the tool or technique used.”*

What every recon tool must do:

~5,000	~3,000	~500	10-30 each
Enumerate all users	Enumerate all computers	Enumerate all groups	Read 10-30 attrs per object

The environmental ceiling

Your AD size × your attribute density = the cost of reconnaissance.

No tool can pay that cost quietly.

Dimension 1: Volume

Example environment (illustrative):

```
Users:     5,234
Computers: 3,847
Groups:    892
SPNs:      127
```

BloodHound needs ALL of these.

Dimension 2: Density

Attrs collected per object (illustrative):

```
avg attrs/user:     23
avg attrs/computer: 31
```

Leaks as **pages/object** in 1644:

```
admin query:    5–20
BloodHound:     98+
```

Reduce volume → miss the graph. Reduce attributes → miss the attack paths. Can't dodge both.

EVT:1644

Every 1644 tells a story: the 5 W's

Six questions turn raw counters into intent.

Question	1644 field(s)	Why it matters
WHO made the query?	Client IP, User	Attribution & targeting
WHAT are they after?	Search Filter, Returned	Attack intent
WHEN did it happen?	TimeCreated	Pattern detection
WHERE are they looking?	Base DN, Search Scope	Scope of recon
WHY does it matter?	Visited / Returned / Pages	Tool sophistication
HOW are they doing it?	Used Indexes	Query optimization

WHO + WHAT + WHERE = the target. WHY + HOW = how good they are. A whole investigation from one event.

EVT:1644

Event 1644's secret weapon: the counters

```
Event 1644 — Search Performance
---------------------------------
Client: 10.1.1.13
User:   MARVEL\loki
Filter: (!(FALSE))
Scope:  subtree

Entries Returned:    334
Entries Visited:   3,815
Used Indexes:      0
Search Time:       1274ms

Pages Referenced: 8421
Pages Read:       512
```

Every 1644 event carries performance telemetry. Most defenders ignore it.

Why this matters

EntriesReturned = what the attacker got

EntriesVisited = what AD had to scan ← the signal

Used Indexes = 0 means full tree walk

High visited + 0 indexes = recon. No real app queries this way.

EVT:1644

Index reference: what 'Used Indexes' reveals

The HOW dimension, decoded — reading the 'Used Indexes' field.

Index	What it signals	Recon weight
Ancestors_index	Subtree enumeration	High
dnt_index	Targeted or bulk lookups	Medium
obj_index	Class-based enumeration	High
pdnt_index	OU structure mapping	Medium
tuple_index	Specific attribute recon	High
memberOf_index	Privilege mapping	CRITICAL
servicePrincipalName_index	Kerberoasting prep	CRITICAL
Container_index	OU reconnaissance	Medium

memberOf and servicePrincipalName are the loudest indexes — privilege mapping and Kerberoast prep. See them, look closer.

EVT:1644

Your environment is your baseline

Same tool, three envs (illustrative)

SMB 200 users / 150 computers

Normal **μ=8 entries** → BloodHound **x=140** *(×17)*

MID 5k users / 3k computers

Normal **μ=57 entries** → BloodHound **x=3,815** *(×67)*

ENT 50k users / 40k computers

Normal **μ=430 entries** → BloodHound **x=41,200** *(×96)*

The factor (×17, ×67, ×96) is what matters — not the raw count.

Your baseline depends on:

- **Users** — *Base query volume scales linearly*
- **Computers** — *BH computer enumeration floor*
- **Groups** — *Group membership query depth*
- **SPNs** — *Kerberoast target universe*
- **Admin-count=1** — *Tier-0 enumeration size*
- **Cert templates** — *ADCS ESC* enumeration size*
- **Legit tools** — *SCCM / backup / scanner floors*

"You still have to understand your env." — me.

EVT:1644

Building the baseline — 5 knobs

1	**COLLECTION PERIOD** 14 days (min 24-48 hrs)	*Capture daily + weekly cycles*
2	**DATA POINT** EntriesReturned per 1644	*Plus Visited / Pages / Indexes*
3	**EXCLUSIONS** SCCM · backup · scanners	*Their volume skews μ high*
4	**SCOPE** Per source IP · per user	*Global σ hides the signal*
5	**WINDOW** Rolling 14-day · Welford's	*Baseline drifts as env grows*

The one-liner

```
# Pull 14 days of 1644
$d = Get-WinEvent ...

# Compute baseline
$stats = $d |
  Measure-Object `
  -Average `
  -StandardDeviation

# μ, σ in hand.
```

Built-in. No ML stack required.

Per-IP μ and σ. Exclude known tools. Rolling 14-day window. Alert when z > 3.

EVT:1644

The only formula you need: z-score

$$z = (x - \mu) / \sigma$$

x

WHAT JUST HAPPENED

what just happened: 3,815 entries visited

μ

WHAT'S NORMAL

what's normal for this user: 57 entries

σ

HOW MUCH IT WOBBLES

how much it normally varies: ±31 entries

x, μ, σ are object counts. z has no unit — just 'wobbles from normal.' Alert at z > 3.

Worked example: counter → z-score

1

Build μ from 14 days of normal

loki's sessions normally visit ~57 objects (μ = 57).
That number varies day-to-day by ±31 entries (σ = 31).

2

Watch what just happened

This session: BloodHound visited 3,815 objects (x = 3,815).

3

Compute how weird 'weird' is

z = (3,815 − 57) / 31 = 121.2 σ
99.7% of normal lives within 3σ. We are 40× past that.

4

Why this can't be hidden

To stay under z > 3, BloodHound has to touch < 150 objects.
That's not 'sneaky BloodHound.' That's 'broken BloodHound.'

EVT:1644

Why z > 3? The 99.7% rule.

The empirical rule

In a normal distribution:

68% of values within ±1σ

95% of values within ±2σ

99.7% of values within ±3σ

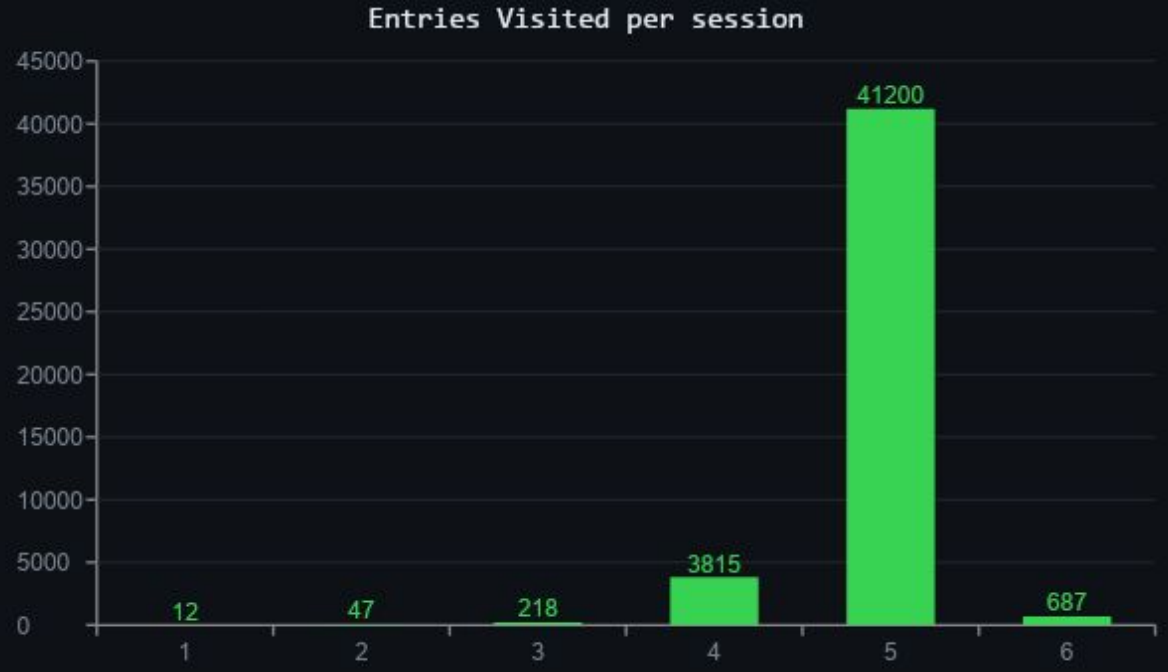
Alerting at z > 3 means you alert on the 0.3% of behavior that is statistically abnormal.

Worked example

```
# loki's 14-day baseline:
μ  = 57  objects visited
σ  = 31  objects

# BloodHound session:
x  = 3,815

z  = (3815 − 57) / 31
z  = 3758 / 31
z  = 121.2  σ

# alert threshold z > 3
# we're 40× over. ALERT.
```

EVT:1644

The picture: baseline vs. outliers



THE READ

Normal user ⟶ Admin script: **one order of magnitude**

BloodHound ⟶ SOAPHound: **four orders of magnitude**

You don't need ML to see this. ***You need a baseline and a calculator.***

// LIVE DEMO

Start-BloodHoundDetection.ps1

Live z-score analysis + composite probability against MARVEL.local.

EVT:1644

BloodHoundDetector.ps1 — live

! LIVE

DEMO 05 · Start-BloodHoundDetector.ps1

Four signals an attacker can't hide

One signal false-positives. All four at once — reconnaissance.

40% VOLUME — entries returned

How many objects came back per query (z-score vs baseline)

Normal: z < 2 (μ ± σ) **Recon:** z > 3 (rare)
→ *Attackers need the data. Tiny queries miss attack paths.*

30% EFFICIENCY — visited/returned

How precisely they aimed (ratio of visited to returned)

Normal: 5–20× (broad) **Recon:** <2× (laser-aimed)
→ *Reconnaissance is targeted by design. Sloppy is slow.*

20% INDEX — used indexes

Which AD indexes the query hit (Ancestors / SPN / objectClass)

Normal: varied indexes **Recon:** Ancestors_index or 0
→ *Subtree enumeration leaves an index fingerprint AD always logs.*

10% PAGES — pages referenced

How much raw data per object (proxy for attribute volume)

Normal: 5–20 pages/object **Recon:** 50+ pages/object
→ *Pulling DACL + all attributes is heavy. The pages tell on you.*

CHAPTER 08

Low & Slow Evasion

Patient attackers stay under every threshold. Event 1644 detects tools, not tradecraft.

EVT:1644

Event 1644 only logs expensive queries

Three independent thresholds. Trip ANY one and it logs. Trip none, you're invisible:

Volume / Efficiency / Time — results returned, visited:returned ratio, or search-time ms

Defaults are huge — labs drop them to ~100 / ~50 / ~10ms to surface activity

secretsdump scopes which accounts to pull with -ldapfilter. Bulk scope is loud; targeted scope is silent.

```
# bulk DCSync scoping — caught, but ONLY by the time threshold:
secretsdump -ldapfilter (objectClass=user)   → 57 ret, 46ms → LOGGED
# not expensive, not inefficient — just slow enough. one signal.

# targeted DCSync scoping — crosses ZERO of three:

secretsdump -ldapfilter (memberOf=Domain Admins) → 4 ret, 0ms → NO 1644
# 4 results, indexed, 0ms. Then it DCSyncs just those. 4662 catches that.
```

MULTI-EVT

The chicken-and-egg — and the fix

The attacker's problem

To query individually you need the username list first. And to get the list, you query bulk — which trips 1644.

Unless they have the list — or skip Domain entirely →

The defender's fix

Frequency detection. 1,000 single-user lookups from one IP in 60 seconds is its own signal — even without 1644.

Count queries per source per minute. Alert on anomaly.

```
# Event 5156 logs every TCP connection — direct LDAP and ADWS proxy
detection:
  sel:
    EventID: 5156
    DestinationPort:
      - 389       # direct LDAP (Impacket, BH.py)
      - 9389      # ADWS (Get-ADUser, SOAPHound)
  timeframe: 1m
  condition: sel | count() by SourceAddress > 100
```

PARTITION

AD is three trees. You watch one.

Almost all Event 1644 monitoring is scoped to the Domain partition. Attackers know it.

Partition	Base DN	What attackers hunt here
Domain	DC=marvel,DC=local	Users, groups, ACLs, BloodHound paths
Configuration	CN=Configuration,DC=marvel,DC=local	Cert templates → ESC1-8, trusts, sites
Schema	CN=Schema,CN=Configuration,...	Class definitions — mostly noise

The attack: base-scope walk the cert templates under Configuration, one per query, sleep between. Full ADCS recon, zero Event 1644 — and no bulk phase to catch, because that path is identical in every forest.

3 AXES

Three axes. Thresholds fight one.

An attacker can move along three independent axes. The literature only covers the first.

Axis	What it defeats	Cost to attacker
Volume / Time	Event 1644 thresholds	Speed — smaller, slower queries
Partition	Domain-scoped monitoring + SACLs	None — if they know where to look
Scope: base vs subtree	Anomaly detectors hunting subtree	Speed — more queries

Real low-and-slow stacks all three: base-scope reads against the Configuration partition, one object at a time, with delays between.

MULTI-EVT

When recon is invisible, catch the theft

4662 = the DCSync safety net

Event 4662 fires on DS-Replication-Get-Changes-All. It catches the credential theft — not the recon.

NOT a recon detector. The enumeration itself stays silent.

Stop building single-event rules

4624 logon + 1644 targeted query + 4662 replication, same source, inside 60 seconds.

Near-impossible to fake. No single event has to cross a threshold.

```
# the sequence that is almost certainly an attack
T+0s   4624  — network logon from 10.1.1.13
T+5s   1644  — targeted query: Domain Admins membership
T+10s  4662  — DS-Replication-Get-Changes-All
T+15s  4662  — replication: krbtgt, Administrator, DA

# correlate: EventCode IN (4624, 1644, 4662)
#   by src_ip, span=60s, where event_types >= 3
# no single event is actionable. the pattern is.
```

CHAPTER 09

Obfuscation Blind Spot

Event 1644 catches the loud. LDAPMon catches the quiet. You need both.

EVT:1644

MaLDAPtive vs Event 1644: ~70% blind

Test suite: 21 obfuscation techniques. Log source comparison:

Technique	Event 1644	LDAPMon ETW
Hex-encoded values (\75ser = user)	NORMALIZED	RAW — seen
Null byte injection (\00)	STRIPPED	RAW — seen
OID attribute names (1.2.840...1.4.1)	NORMALIZED	RAW — seen
Whitespace padding / newlines	NORMALIZED	RAW — seen
Random case (ObjectClass / CN)	NORMALIZED	RAW — seen
Double negation (!(!(x=y)))	LOGGED	RAW — seen
Redundant parens ((((x=y))))	NORMALIZED	RAW — seen
Tautology injection ((&(1=1)...))	NORMALIZED	RAW — seen

Double negation survives (forces a full scan). The other ~70% vanish three ways: DC normalizes · indexed ≠ expensive · optimizer drops it.

// LIVE DEMO

Invoke-MaLDAPtiveTests.ps1

I built this expecting Event 1644 to catch most of them. Place your bet before the reveal.

ETW:LDAP

Invoke-MaLDAPtiveTests — actual output

! LIVE

DEMO 06 · Invoke-MaLDAPtiveTests

MULTI-EVT

I tested all 21 — here's what 1644 missed

Full MaLDAPtive suite (Bohannon & Elezaj) plus Charlie Clark's, via his PowerView. How Event 1644 handles each:

Key	Technique	Event 1644 status
Z	Prepend zeros to numbers	DC normalizes
N	Names to ANR	Logged but undetectable
x	Extensible match empty rule	DC mangles
T	Tautology replacement	OID bitwise unrecognized
A	Approximate match ~=	Normalized to =
B	Boolean condition padding	Partial — unrecognized
s	Substring split	Survives as cn=jo*hn
W	Whitespace injection	DC strips all whitespace
C	Case randomization	LDAPMon only
O	OID attribute names	LDAPMon only
X	Hex value encoding	LDAPMon only — rule exists
I	Equality to range (>=/<=)	Covered — Ch 01

Event 1644 normalizes, strips, or optimizes most of these away. LDAPMon sees all 21 raw on the wire — the fix, next slide.

ETW:LDAP

LDAPMon — client ETW sees the raw wire

The architecture

Provider: Microsoft-Windows-LDAP-Client

GUID: 099614a5-5dd7-4788-8bc9-e29f43db28fc

Captures: raw LDAP filter BEFORE the DC sees it. Obfuscation intact.

Tool: LDAPMon by Jonny Johnson · github.com/jonny-jhnson/LDAPMon

```
# Enable per process with registry key:
HKLM\System\CurrentControlSet\Services\ldap\Tracing\<ProcessName.exe>

# Start ETW trace:
logman start LDAPMon -p Microsoft-Windows-LDAP-Client -o ldap.etl -ets

# Convert to events with Event ID 30 = LDAP SearchRequest
tracerpt ldap.etl -o ldap.xml
```

ETW:LDAP

Sigma rules LDAPMon can carry

Deploy via LDAPMon by Jonny Johnson · github.com/jonny-jhnson/LDAPMon

```
title: Hex-Encoded LDAP Filter
logsource:
  service: ldap-client
detection:
  sel:
    EventID: 30
    Filter|re: '\\[0-9a-fA-F]{2}'
  filter_lsass:
    ProcessName: lsass.exe
  condition: sel and not filter_lsass
```

```
title: Null Byte Injection
logsource:
  service: ldap-client
detection:
  sel:
    EventID: 30
    Filter|contains: '\00'
  condition: sel
level: high
```

```
title: OID Attribute Notation
logsource:
  service: ldap-client
detection:
  sel:
    EventID: 30
    Filter|re: '\(1\.2\.840\.113556'
  condition: sel
```

```
title: Double Negation
detection:
  sel_etw:          # LDAP-Client ETW
    EventID: 30
    Filter|re: '\(!\s*\(!'
  sel_1644:         # Directory Service log
    EventID: 1644
    SearchFilter|re: '\(!\s*\(!'
  condition: 1 of sel_*
```

1644 cheatsheet: read every column

Field	What it reveals	Attacker footprint	Where in talk
Client IP	WHO ran the query	::1 means ADWS proxy	Ch 5A · 5B
Search Filter	WHAT they're after	(!FALSE), OID syntax, bitwise	Ch 01 · 04
Search Scope	Scope of search	Subtree = full enum	Ch 07
AttributeList	Fields they pulled	nTSecurityDescriptor = ACL recon	Ch 03
Controls	Server-side flags	SDFlags 0x4/0x5/0x7 OR nTSecurityDescriptor	Ch 03 · 04
Visited / Returned	How much data extracted	Low ratio + high volume = recon	Ch 07

Six fields. Nine chapters. One log source. Read every column.

The forest: every blind spot, every fix

Part	Blind spot	Fix
01	OID transformations break signatures	Regex the normalized form
02	Whitespace/case variants evade |contains	Whitespace-agnostic regex
03	SDFlags is invisible to string rules	Parse Controls field
04	SOAPHound !FALSE dumps without signature	Pattern-based rule
5A/B	ADWS hides source IP behind ::1	Event 5156 port join + 1138/1139 chain
06	LDAP Ping never reaches the LDAP engine	netlogon.log + 5156
07	Unknown tools evade all signatures	Statistical z-score
08	Low-and-slow beats every 1644 threshold	Event 5156 frequency
09	~70% of obfuscation is normalized away	LDAPMon ETW

No single log source is complete. Layer three: Event 1644 + Event 5156 + LDAPMon ETW.

// CONCLUSION

Three tools. Know when to reach for each.

SIGNATURES

fast, specific, brittle

USE WHEN

Known OIDs, tool strings, fixed patterns

AVOID WHEN

Anything rename or whitespace breaks

BEHAVIOR

patterns over strings

USE WHEN

SDFlags ∈ {0x4,0x5,0x7}, (!FALSE), nTSD in attrs

AVOID WHEN

When you don't understand the protocol yet

STATISTICS

catches the unknown

USE WHEN

EntriesVisited/Returned, volume z-scores

AVOID WHEN

When you can't build a clean baseline

Six things to take home

1. Event 1644 logs the NORMALIZED filter. Rules match what AD writes, not what's sent.
2. Whitespace, case, hex, OID aliases all survive the parser. Your rule must too.
3. Server-side controls (SDFlags) and query patterns ((!FALSE)) reveal intent.
4. EntriesVisited / EntriesReturned is the most underused field in Windows auditing.
5. ADWS attribution isn't impossible. Event 5156 port-matching solves it.
6. Event 1644 + Event 5156 + LDAPMon ETW. That's the stack. Deploy all three.

Don't stare at one tree. Walk the forest.

Questions

(and arguments about whether regex counts as statistics)

@4ndr3w6S · huntress.com/authors/andrew-schwartz · github.com/4ndr3w6S