## Access Token Manipulation

Windows uses access tokens to determine the ownership of a running process. A user can manipulate access tokens to make a running process appear as though it belongs to someone other than the user that started the process. When this occurs, the process also takes on the security context associated with the new token. For example, Microsoft promotes the use of access tokens as a security best practice. Administrators should log in as a standard user but run their tools with administrator privileges using the built-in access token manipulation command runas.

Adversaries may use access tokens to operate under a different user or system security context to perform actions and evade detection. An adversary can use built-in Windows API functions to copy access tokens from existing processes; this is known as token stealing. An adversary must already be in a privileged user context (i.e. administrator) to steal a token. However,adversaries commonly use token stealing to elevate their security context from the administrator level to the SYSTEM level. An adversary can use a token to authenticate to a remote system as the account for that token if the account has appropriate permissions on the remote system.

Access tokens can be leveraged by adversaries through three methods:

**Token Impersonation/Theft** - An adversary creates a new access token that duplicates an existing token using DuplicateToken(Ex). The token can then be used with ImpersonateLoggedOnUser to allow the calling thread to impersonate a logged on user's security context, or with SetThreadToken to assign the impersonated token to a thread. This is useful for when the target user has a non-network logon session on the system.

**Create Process with a Token** - An adversary creates a new access token with DuplicateToken(Ex) and uses it with CreateProcessWithTokenW to create a new process running under the security context of the impersonated user. This is useful for creating a new process under the security context of a different user.

**Make and Impersonate Token** - An adversary has a username and password but the user is not logged onto the system. The adversary can then create a logon session for the user using the LogonUser function. The function will return a copy of the new session's access token and the adversary can use SetThreadToken to assign the token to a thread.

Any standard user can use the runas command, and the Windows API functions, to create impersonation tokens; it does not require access to an administrator account.

Metasploit's Meterpreter payload allows arbitrary token manipulation and uses token impersonation to escalate privileges. The Cobalt Strike beacon payload allows arbitrary token impersonation and can also create tokens.

## Related Events

| date | host | pid | activity |
|---|---|---|---|
| 02/08 20:14 | MALWARE-TEST-PC | 516 | revert token |

## Mitigation

Access tokens are an integral part of the security system within Windows and cannot be turned off. However, an attacker must already have administrator level access on the local system to make full use of this technique; be sure to restrict users and accounts to the least privileges they require to do their job.

Any user can also spoof access tokens if they have legitimate credentials. Follow mitigation guidelines for preventing adversary use of Valid Accounts. Limit permissions so that users and user groups cannot create tokens. This setting should be defined for the local system account only. GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignment: Create a token object. Also define who can create a process level token to only the local and network service through GPO: Computer Configuration > [Policies] > Windows Settings > Security Settings > Local Policies > User Rights Assignment: Replace a process level token.

Also limit opportunities for adversaries to increase privileges by limiting Privilege Escalation opportunities.

## Detection Methods

If an adversary is using a standard command-line shell, analysts can detect token manipulation by auditing command-line activity. Specifically, analysts should look for use of the runas command. Detailed command-line logging is not enabled by default in Windows.

If an adversary is using a payload that calls the Windows token APIs directly, analysts can detect token manipulation only through careful analysis of user network activity, examination of running processes, and correlation with other endpoint and network behavior.

There are many Windows API calls a payload can take advantage of to manipulate access tokens (e.g., LogonUser, DuplicateTokenEx, and ImpersonateLoggedOnUser). Please see the referenced Windows API pages for more information.

Query systems for process and thread token information and look for inconsistencies such as user owns processes impersonating the local SYSTEM account.

## Reference

- Tactic: T1134

## Credential Dumping

Credential dumping is the process of obtaining account login and password information, normally in the form of a hash or a clear text password, from the operating system and software. Credentials can then be used to perform Lateral Movement and access restricted information.

Several of the tools mentioned in this technique may be used by both adversaries and professional security testers. Additional custom tools likely exist as well.

**SAM (Security Accounts Manager)**
The SAM is a database file that contains local accounts for the host, typically those found with the 'net user' command. To enumerate the SAM database, system level access is required.

A number of tools can be used to retrieve the SAM file through in-memory techniques:

- pwdumpx.exe
- gsecdump
- Mimikatz
- secretsdump.py

Alternatively, the SAM can be extracted from the Registry with Reg:

- reg saveHKLM\sam sam
- reg save HKLM\system system

Creddump7 can then be used to process the SAM database locally to retrieve hashes.

Notes:
Rid 500 account is the local, in-built administrator.
Rid 501 is the guest account.
User accounts start with a RID of 1,000+.

**Cached Credentials**
The DCC2 (Domain Cached Credentials version 2) hash, used by Windows Vista and newer caches credentials when the domain controller is unavailable. The number of default cached credentials varies, and this number can be altered per system. This hash does not allow pass-the-hash style attacks.

A number of tools can be used to retrieve the SAM file through in-memory techniques.

- pwdumpx.exe
- gsecdump
- Mimikatz

Alternatively, reg.exe can be used to extract from the Registry and Creddump7 used to gather the credentials.

Notes:
Cached credentials for Windows Vista are derived using PBKDF2.

**Local Security Authority (LSA) Secrets**
With SYSTEM access to a host, the LSA secrets often allows trivial access from a local account to domain-based account credentials. The Registry is used to store the LSA secrets.

When services are run under the context of local or domain users, their passwords are stored in the Registry. If auto-logon is enabled, this information will be stored in the Registry as well.

A number of tools can be used to retrieve the SAM file through in-memory techniques.

- pwdumpx.exe
- gsecdump
- Mimikatz
- secretsdump.py

Alternatively, reg.exe can be used to extract from the Registry and Creddump7 used to gather the credentials.

Notes:
The passwords extracted by his mechanism are UTF-16 encoded, which means that they are returned in plaintext.
Windows 10 adds protections for LSA Secrets described in Mitigation.

**NTDS from Domain Controller**
Active Directory stores information aboutmembers of the domain including devices and users to verify credentials and define access rights. The Active Directory domain database is stored in the NTDS.dit file. By default the NTDS file will be located in %SystemRoot%\NTDS\Ntds.dit of a domain controller.

The following tools and techniques can be used to enumerate the NTDS file and the contents of the entire Active Directory hashes.

- Volume Shadow Copy
- secretsdump.py
- Using the in-built Windows tool, ntdsutil.exe
- Invoke-NinjaCopy

**Group Policy Preference (GPP) Files**
Group Policy Preferences (GPP) are tools that allowed administrators to create domain policies with embedded credentials. These policies, amongst other things, allow administrators to set local accounts.

These group policies are stored in SYSVOL on a domain controller, this means that any domain user can view the SYSVOL share and decrypt the password (the AES private key was leaked on-line.

The following tools and scripts can be used to gather and decrypt the password file from Group Policy Preference XML files:

- Metasploit's post exploitation module: "post/windows/gather/credentials/gpp"
- Get-GPPPassword
- gpprefdecrypt.py

Notes:
On the SYSVOL share, the following can be used to enumerate potential XML files.
dir /s *.xml

**Service Principal Names (SPNs)**
See Kerberoasting.

**Plaintext Credentials**
After a user logs on to a system, a variety of credentials are generated and stored in the Local Security Authority Subsystem Service (LSASS) process in memory. These credentials can be harvested by a administrative user or SYSTEM.

SSPI (Security Support Provider Interface) functions as a common interface to several Security Support Providers (SSPs): A Security Support Provider is a dynamic-link library (DLL) that makes one or more security packages available to applications.

The following SSPs can be used to access credentials:

Msv: Interactive logons, batch logons, and service logons are done through the MSV authentication package.
Wdigest: The Digest Authentication protocol is designed for use with Hypertext Transfer Protocol (HTTP) and Simple Authentication Security Layer (SASL) exchanges.
Kerberos: Preferred for mutual client-server domain authentication in Windows 2000 and later.
CredSSP: Provides SSO and Network Level Authentication for Remote Desktop Services.

The following tools can be used to enumerate credentials:

- Windows Credential Editor
- Mimikatz

As well as in-memory techniques, the LSASS process memory can be dumped from the target host and analyzed on a local system.

For example, on thetarget host use procdump:

- procdump -ma lsass.exe lsass_dump

Locally, mimikatz can be run:

- sekurlsa::Minidump lsassdump.dmp
- sekurlsa::logonPasswords

**DCSync**
DCSync is a variation on credential dumping which can be used to acquire sensitive information from a domain controller. Rather than executing recognizable malicious code, the action works by abusing the domain controller's application programming interface (API) to simulate the replication process from a remote domain controller. Any members of the Administrators, Domain Admins, Enterprise Admin groups or computer accounts on the domain controller are able to run DCSync to pull password data from Active Directory, which may include current and historical hashes of potentially useful accounts such as KRBTGT and Administrators. The hashes can then in turn be used to create a Golden Ticket for use in Pass the Ticket or change an account's password as noted in Account Manipulation. DCSync functionality has been included in the "lsadump" module in Mimikatz. Lsadump also includes NetSync, which performs DCSync over a legacy replication protocol.

## Related Events

| date | host | pid | activity |
|---|---|---|---|
| 02/08 19:59 | MALWARE-TEST-PC | 3104 | dump hashes |

## Mitigation

Monitor/harden access to LSASS and SAM table with tools that allow process whitelisting. Limit credential overlap across systems to prevent lateral movement opportunities using Valid Accounts if passwords and hashes are obtained. Ensure that local administrator accounts have complex, unique passwords across all systems on the network. Do not put user or admin domain accounts in the local administrator groups across systems unless they are tightly controlled, as this is often equivalent to having a local administrator account with the same password on all systems. Follow best practices for design and administration of an enterprise network to limit privileged account use across administrative tiers.

On Windows 8.1 and Windows Server 2012 R2, enable Protected Process Light for LSA.

Identify and block potentially malicious software that may be used to dump credentials by using whitelisting tools, like AppLocker, or Software Restriction Policies where appropriate.

With Windows 10, Microsoft implemented new protections called Credential Guard to protect the LSA secrets that can be used to obtain credentials through forms of credential dumping. It is not configured by default and has hardware and firmware system requirements. It also does not protect against all forms of credential dumping.

Manage the access control list for "Replicating Directory Changes" and other permissions associated with domain controller replication.

Consider disabling or restricting NTLM traffic.

## Detection Methods

Common credential dumpers such as Mimikatz access the LSA Subsystem Service (LSASS) process by opening the process, locating the LSA secrets key, and decrypting the sections in memory where credential details are stored. Credential dumpers may also use methods for reflective Process Injection to reduce potential indicators of malicious activity.

Hash dumpers open the Security Accounts Manager (SAM) on the local file system (%SystemRoot%/system32/config/SAM) or create a dump of the Registry SAM key to access stored account password hashes. Some hash dumpers will open the local file system as a device and parse to the SAM table to avoid file access defenses. Others will make an in-memory copy of the SAM table before reading hashes. Detection of compromised Valid Accounts in-use by adversaries may help as well.

On Windows 8.1 and Windows Server 2012 R2, monitor Windows Logs for LSASS.exe creation to verify that LSASS started as a protected process.

Monitor processes and command-line arguments for program execution that may be indicative of credential dumping. Remote access tools may contain built-in features or incorporate existing tools like Mimikatz. PowerShell scripts also exist that contain credential dumping functionality, such as PowerSploit's Invoke-Mimikatz module, which may require additional logging features to be configured in the operating system to collect necessary information for analysis.

Monitor domain controller logs for replication requests and other unscheduled activity possibly associated with DCSync. Note: Domain controllers may not log replication requests originating from the default domain controller account.. Also monitor for network protocols and other replication requests from IPs not associated with known domain controllers.

## Reference

- Tactic: T1003

## Pass the Hash

Pass the hash (PtH) is a method of authenticating as a user without having access to the user's cleartext password. This method bypasses standard authentication steps that require a cleartext password, moving directly into the portion of the authentication that uses the password hash. In this technique, valid password hashes for the account being used are captured using a Credential Access technique. Captured hashes are used with PtH to authenticate as that user. Once authenticated, PtH may be used to perform actions on local or remote systems.

Windows 7 and higher with KB2871997 require valid domain user credentials or RID 500 administrator hashes.

### Related Events

| date | host | pid | activity |
|---|---|---|---|
| 02/08 20:14 | MALWARE-TEST-PC | 516 | run mimikatz's sekurlsa::pth /user:malware-test /domain:MALWARE-TEST-PC / ntlm:329153f560eb329c0e1deea55e88a1e9 / run:"%COMSPEC% /c echo 2a21a367221 > \\.\pipe\ef02b8" command |

### Mitigation

Monitor systems and domain logs for unusual credential logon activity. Prevent access to Valid Accounts. Apply patch KB2871997 to Windows 7 and higher systems to limit the default access of accounts in the local administrator group.

Enable pass the hash mitigations to apply UAC restrictions to local accounts on network logon. The associated Registry key is located HKLM\SOFTWARE\Microsoft\Windows \CurrentVersion\Policies\System\LocalAccountTokenFilterPolicy Through GPO: Computer Configuration > [Policies] > Administrative Templates > SCM: Pass the Hash Mitigations: Apply UAC restrictions to local accounts on network logons.

Limit credential overlap across systems to prevent the damage of credential compromise and reduce the adversary's ability to perform Lateral Movement between systems. Ensure that built-in and created local administrator accounts have complex, uniquepasswords. Do not allow a domain user to be in the local administrator group on multiple systems.

### Detection Methods

Audit all logon and credential use events and review for discrepancies. Unusual remote logins that correlate with other suspicious activity (such as writing and executing binaries) may indicate malicious activity. NTLM LogonType 3 authentications that are not associated to a domain login and are not anonymous logins are suspicious.

## Reference

- Tactic: T1075

## Process Hollowing

Process hollowing occurs when a process is created in a suspended state then its memory is unmapped and replaced with malicious code. Similar to Process Injection, execution of the malicious code is masked under a legitimate process and may evade defenses and detection analysis.

### Related Events

| date | host | pid | activity |
|---|---|---|---|
| 02/08 19:59 | MALWARE-TEST-PC | 3104 | dump hashes |
| 02/08 20:05 | MALWARE-TEST-PC | 3104 | spawn (x86) windows/beacon_http/reverse_http (10.0.2.15:8080) |
| 02/08 20:14 | MALWARE-TEST-PC | 516 | run mimikatz's sekurlsa::pth /user:malware-test /domain:MALWARE-TEST-PC /ntlm:329153f560eb329c0e1deea55e88a1e9 /run:"%COMSPEC% /c echo 2a21a367221 > \\.\pipe\ef02b8" command |

### Mitigation

This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of operating system design features. For example, mitigating specific API calls will likely have unintended side effects, such as preventing legitimate software (i.e., security products) from operating properly. Efforts should be focused on preventing adversary tools from running earlier in the chain of activity and on identifying subsequent malicious behavior.

Although process hollowing may be used to evade certain types of defenses, it is still good practice to identify potentially malicious software that may be used to perform adversarial actions and audit and/or block it by using whitelisting tools, like AppLocker, or Software Restriction Policies where appropriate.

### Detection Methods

Monitoring API calls may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. API calls that unmap process memory, such as ZwUnmapViewOfSection or NtUnmapViewOfSection, and those that can be used to modify memory within another process, such as WriteProcessMemory, may be used for this technique.

Analyze process behavior to determine if a process is performing actions it usually does not, such as opening network connections, reading files, or other suspicious actions that could relate to post-compromise behavior.

## Reference

- Tactic: T1093

## Process Injection

Process injection is a method of executing arbitrary code in the address space of a separate live process. Running code in the context of another process may allow access to the process's memory, system/network resources, and possibly elevated privileges. Execution via process injection may also evade detection from security products since the execution is masked under a legitimate process.

**Windows**
There are multiple approaches to injecting code into a live process. Windows implementations include:

- **Dynamic-link library (DLL) injection** involves writing the path to a malicious DLL inside a process then invoking execution by creating a remote thread.
- **Portable executable injection** involves writing malicious code directly into the process (without a file on disk) then invoking execution with either additional code or by creating a remote thread. The displacement of the injected code introduces the additional requirement for functionality to remap memory references. Variations of this method such as reflective DLL injection (writing a self-mapping DLL into a process) and memory module (map DLL when writing into process) overcome the address relocation issue.
- **Thread execution hijacking** involves injecting malicious code or the path to a DLL into a thread of a process. Similar to Process Hollowing, the thread must first be suspended.
- **Asynchronous Procedure Call** (APC) injection involves attaching malicious code to the APC Queue of a process's thread. Queued APC functions are executed when the thread enters an alterable state. AtomBombing is a variation that utilizes APCs to invoke malicious code previously written to the global atom table.
- **Thread Local Storage** (TLS) callback injection involves manipulating pointersinside a portable executable (PE) to redirect a process to malicious code before reaching the code's legitimate entry point.

**Mac and Linux**
Implementations for Linux and OS X/macOS systems include:

- **LD_PRELOAD, LD_LIBRARY_PATH** (Linux), "'DYLD_INSERT_LIBRARIES"' (Mac OS X) environment variables, or the dlfcn application programming interface (API) can be used to dynamically load a library (shared object) in a process which can be used to intercept API calls from the running process.
- **Ptrace system calls** can be used to attach to a running process and modify it in runtime.
- **/proc/[pid]/mem** provides access to the memory of the process and can be used to read/write arbitrary data to it. This technique is very rare due to its complexity.
- **VDSO hijacking** performs runtime injection on ELF binaries by manipulating code stubs mapped in from the linux-vdso.so shared object.

Malware commonly utilizes process injection to access system resources through which Persistence and other environment modifications can be made. More sophisticated samples may perform multiple process injections to segment modules and further evade detection, utilizing named pipes or other inter-process communication (IPC) mechanisms as a communication channel.

## Related Events

| date | host | pid | activity |
|---|---|---|---|
| 02/08 19:59 | MALWARE-TEST-PC | 3104 | dump hashes |

## Mitigation

This type of attack technique cannot be easily mitigated with preventive controls since it is based on the abuse of operating system design features. For example, mitigating specific Windows API calls will likely have unintended side effects, such as preventing legitimate software (i.e., security products) from operating properly. Efforts should be focused on preventing adversary tools from running earlier in the chain of activity and on identification of subsequent malicious behavior.

Identify or block potentially malicious software that may contain process injection functionality by using whitelisting tools, like AppLocker, or Software Restriction Policies where appropriate.

Utilize Yama to mitigate ptrace based process injection by restricting the use of ptrace to privileged users only. Other mitigation controls involve the deployment of security kernel modules that provide advanced access control and process restrictions such as SELinux , grsecurity , and AppAmour .

## Detection Methods

Monitoring Windows API calls indicative of the various types of code injection may generate a significant amount of data and may not be directly useful for defense unless collected under specific circumstances for known bad sequences of calls, since benign use of API functions may be common and difficult to distinguish from malicious behavior. API calls such as CreateRemoteThread, SuspendThread/SetThreadContext/ ResumeThread, QueueUserAPC, and those that can be used to modify memory within another process, such as WriteProcessMemory, may be used for this technique.

Monitoring for Linux specific calls such as the ptrace system call, the use of LD_PRELOAD environment variable, or dlfcn dynamic linking API calls, should not generate large amounts of data due to their specialized nature, and can be a very effective method to detect some of the common process injection methods.

Monitor for named pipe creation and connection events (Event IDs 17 and 18) for possible indicators of infected processes with external modules.

Monitor processes and command-line arguments for actions that could be done before or after code injection has occurred and correlate the information with related event information. Code injection may also be performed using PowerShell with tools such as PowerSploit, so additional PowerShell monitoring may be required to cover known implementations of this behavior.

## Reference

- Tactic: T1055

# LICENSE



## DISCLAIMERS

MITRE does not claim ATT&CK enumerates all possibilities for the types of actions and behaviors documented as part of its adversary model and framework of techniques. Using the information contained within ATT&CK to address or cover full categories of techniques will not guarantee full defensive coverage as there may be undisclosed techniques or variations on existing techniques not documented by ATT&CK.

ALL DOCUMENTS AND THE INFORMATION CONTAINED THEREIN ARE PROVIDED ON AN "AS IS" BASIS AND THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE MITRE CORPORATION, ITS BOARD OF TRUSTEES, OFFICERS, AGENTS, AND EMPLOYEES, DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.