Defense Evasion, Tactic TA0005 - Enterprise

Archived: 2026-04-05 13:04:39 UTC

T1548 Abuse Elevation Control Mechanism Adversaries may circumvent mechanisms designed to control elevate privileges to gain higher-level permissions. Most modern systems contain native elevation control mechanisms that are intended to limit privileges that a user can perform on a machine. Authorization has to be granted to specific users in order to perform tasks that can be considered of higher risk. An adversary can perform several methods to take advantage of built-in control mechanisms in order to escalate privileges on a system. .001 Setuid and Setgid An adversary may abuse configurations where an application has the setuid or setgid bits set in order to get code running in a different (and possibly more privileged) user's context. On Linux or macOS, when the setuid or setgid bits are set for an application binary, the application will run with the privileges of the owning user or group respectively. Normally an application is run in the current user's context, regardless of which user or group owns the application. However, there are instances where programs need to be executed in an elevated context to function properly, but the user running them may not have the specific required privileges. .002 Bypass User Account Control Adversaries may bypass UAC mechanisms to elevate process privileges on system. Windows User Account Control (UAC) allows a program to elevate its privileges (tracked as integrity levels ranging from low to high) to perform a task under administrator-level permissions, possibly by prompting the user for confirmation. The impact to the user ranges from denying the operation under high enforcement to allowing the user to perform the action if they are in the local administrators group and click through the prompt or allowing them to enter an administrator password to complete the action. .003 Sudo and Sudo Caching Adversaries may perform sudo caching and/or use the sudoers file to elevate privileges. Adversaries may do this to execute commands as other users or spawn processes with higher privileges. .004 Elevated Execution with Prompt Adversaries may leverage the `AuthorizationExecuteWithPrivileges` API to escalate privileges by prompting the user for credentials. The purpose of this API is to give application developers an easy way to perform operations with root privileges, such as for application installation or updating. This API does not validate that the program requesting root privileges comes from a reputable source or has been maliciously modified. .005 Temporary Elevated Cloud Access Adversaries may abuse permission configurations that allow them to gain temporarily elevated access to cloud resources. Many cloud environments allow administrators to grant user or service accounts permission to request just-in-time access to roles, impersonate other accounts, pass roles onto resources and services, or otherwise gain short-term access to a set of privileges that may be distinct from their own. .006 TCC Manipulation Adversaries can manipulate or abuse the Transparency, Consent, & Control (TCC) service or database to grant malicious executables elevated permissions. TCC is a Privacy & Security macOS control mechanism used to determine if the running process has permission to access the data or services protected by TCC, such as screen sharing, camera, microphone, or Full Disk Access (FDA). T1134 Access Token Manipulation Adversaries may modify access tokens to operate under a different user or system security context to perform actions and bypass access controls. 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 is the child of a different process or 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. .001 Token Impersonation/Theft Adversaries may duplicate then impersonate another user's existing token to escalate privileges and bypass access controls. For example, an

adversary can duplicate an existing token using `DuplicateToken` or `DuplicateTokenEx`. 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. .002 Create Process with Token Adversaries may create a new process with an existing token to escalate privileges and bypass access controls. Processes can be created with the token and resulting security context of another user using features such as `CreateProcessWithTokenW` and `runas`. .003 Make and Impersonate Token Adversaries may make new tokens and impersonate users to escalate privileges and bypass access controls. For example, if 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. .004 Parent PID Spoofing Adversaries may spoof the parent process identifier (PPID) of a new process to evade process-monitoring defenses or to elevate privileges. New processes are typically spawned directly from their parent, or calling, process unless explicitly specified. One way of explicitly assigning the PPID of a new process is via the `CreateProcess` API call, which supports a parameter that defines the PPID to use. This functionality is used by Windows features such as User Account Control (UAC) to correctly set the PPID after a requested elevated process is spawned by SYSTEM (typically via `svchost.exe` or `consent.exe`) rather than the current user context. .005 SID-History Injection Adversaries may use SID-History Injection to escalate privileges and bypass access controls. The Windows security identifier (SID) is a unique value that identifies a user or group account. SIDs are used by Windows security in both security descriptors and access tokens. An account can hold additional SIDs in the SID-History Active Directory attribute , allowing inter-operable account migration between domains (e.g., all values in SID-History are included in access tokens). T1197 BITS Jobs Adversaries may abuse BITS jobs to persistently execute code and perform various background tasks. Windows Background Intelligent Transfer Service (BITS) is a low-bandwidth, asynchronous file transfer mechanism exposed through Component Object Model (COM). BITS is commonly used by updaters, messengers, and other applications preferred to operate in the background (using available idle bandwidth) without interrupting other networked applications. File transfer tasks are implemented as BITS jobs, which contain a queue of one or more file operations. T1612 Build Image on Host Adversaries may build a container image directly on a host to bypass defenses that monitor for the retrieval of malicious images from a public registry. A remote `build` request may be sent to the Docker API that includes a Dockerfile that pulls a vanilla base image, such as alpine, from a public or local registry and then builds a custom image upon it. T1622 Debugger Evasion Adversaries may employ various means to detect and avoid debuggers. Debuggers are typically used by defenders to trace and/or analyze the execution of potential malware payloads. T1678 Delay Execution Adversaries may employ various time-based methods to evade detection and analysis. These techniques often exploit system clocks, delays, or timing mechanisms to obscure malicious activity, blend in with benign activity, and avoid scrutiny. Adversaries can perform this behavior within virtualization/sandbox environments or natively on host systems. T1140 Deobfuscate/Decode Files or Information Adversaries may use Obfuscated Files or Information to hide artifacts of an intrusion from analysis. They may require separate mechanisms to decode or deobfuscate that information depending on how they intend to use it. Methods for doing that include built-in functionality of malware or by using utilities present on the system. T1610 Deploy Container Adversaries may deploy a container into an environment to facilitate execution or evade defenses. In some cases, adversaries may deploy a new container to execute processes associated with a particular image or deployment, such as processes that execute or download malware. In others, an adversary may deploy a new container configured without network rules, user limitations, etc. to bypass existing defenses within the environment. In Kubernetes

environments, an adversary may attempt to deploy a privileged or vulnerable container into a specific node in order to Escape to Host and access other containers running on the node. T1006 Direct Volume Access Adversaries may directly access a volume to bypass file access controls and file system monitoring. Windows allows programs to have direct access to logical volumes. Programs with direct access may read and write files directly from the drive by analyzing file system data structures. This technique may bypass Windows file access controls as well as file system monitoring tools. T1484 Domain or Tenant Policy Modification Adversaries may modify the configuration settings of a domain or identity tenant to evade defenses and/or escalate privileges in centrally managed environments. Such services provide a centralized means of managing identity resources such as devices and accounts, and often include configuration settings that may apply between domains or tenants such as trust relationships, identity syncing, or identity federation. .001 Group Policy Modification Adversaries may modify Group Policy Objects (GPOs) to subvert the intended discretionary access controls for a domain, usually with the intention of escalating privileges on the domain. Group policy allows for centralized management of user and computer settings in Active Directory (AD). GPOs are containers for group policy settings made up of files stored within a predictable network path `\<DOMAIN>\SYSVOL\<DOMAIN>\Policies\`. .002 Trust Modification Adversaries may add new domain trusts, modify the properties of existing domain trusts, or otherwise change the configuration of trust relationships between domains and tenants to evade defenses and/or elevate privileges.Trust details, such as whether or not user identities are federated, allow authentication and authorization properties to apply between domains or tenants for the purpose of accessing shared resources. These trust objects may include accounts, credentials, and other authentication material applied to servers, tokens, and domains. T1672 Email Spoofing Adversaries may fake, or spoof, a sender’s identity by modifying the value of relevant email headers in order to establish contact with victims under false pretenses. In addition to actual email content, email headers (such as the FROM header, which contains the email address of the sender) may also be modified. Email clients display these headers when emails appear in a victim's inbox, which may cause modified emails to appear as if they were from the spoofed entity. T1480 Execution Guardrails Adversaries may use execution guardrails to constrain execution or actions based on adversary supplied and environment specific conditions that are expected to be present on the target. Guardrails ensure that a payload only executes against an intended target and reduces collateral damage from an adversary’s campaign. Values an adversary can provide about a target system or environment to use as guardrails may include specific network share names, attached physical devices, files, joined Active Directory (AD) domains, and local/external IP addresses. .001 Environmental Keying Adversaries may environmentally key payloads or other features of malware to evade defenses and constraint execution to a specific target environment. Environmental keying uses cryptography to constrain execution or actions based on adversary supplied environment specific conditions that are expected to be present on the target. Environmental keying is an implementation of Execution Guardrails that utilizes cryptographic techniques for deriving encryption/decryption keys from specific types of values in a given computing environment. .002 Mutual Exclusion Adversaries may constrain execution or actions based on the presence of a mutex associated with malware. A mutex is a locking mechanism used to synchronize access to a resource. Only one thread or process can acquire a mutex at a given time. T1211 Exploitation for Defense Evasion Adversaries may exploit a system or application vulnerability to bypass security features. Exploitation of a vulnerability occurs when an adversary takes advantage of a programming error in a program, service, or within the operating system software or kernel itself to execute adversary-controlled code. Vulnerabilities may exist in defensive security software that can be used to disable or circumvent them. T1222 File and Directory Permissions Modification Adversaries may modify file or directory permissions/attributes to evade access control lists (ACLs) and access protected files. File and

directory permissions are commonly managed by ACLs configured by the file or directory owner, or users with the appropriate permissions. File and directory ACL implementations vary by platform, but generally explicitly designate which users or groups can perform which actions (read, write, execute, etc.). .001 Windows File and Directory Permissions Modification Adversaries may modify file or directory permissions/attributes to evade access control lists (ACLs) and access protected files. File and directory permissions are commonly managed by ACLs configured by the file or directory owner, or users with the appropriate permissions. File and directory ACL implementations vary by platform, but generally explicitly designate which users or groups can perform which actions (read, write, execute, etc.). .002 Linux and Mac File and Directory Permissions Modification Adversaries may modify file or directory permissions/attributes to evade access control lists (ACLs) and access protected files. File and directory permissions are commonly managed by ACLs configured by the file or directory owner, or users with the appropriate permissions. File and directory ACL implementations vary by platform, but generally explicitly designate which users or groups can perform which actions (read, write, execute, etc.). T1564 Hide Artifacts Adversaries may attempt to hide artifacts associated with their behaviors to evade detection. Operating systems may have features to hide various artifacts, such as important system files and administrative task execution, to avoid disrupting user work environments and prevent users from changing files or features on the system. Adversaries may abuse these features to hide artifacts such as files, directories, user accounts, or other system activity to evade detection. .001 Hidden Files and Directories Adversaries may set files and directories to be hidden to evade detection mechanisms. To prevent normal users from accidentally changing special files on a system, most operating systems have the concept of a ‘hidden’ file. These files don’t show up when a user browses the file system with a GUI or when using normal commands on the command line. Users must explicitly ask to show the hidden files either via a series of Graphical User Interface (GUI) prompts or with command line switches (`dir /a` for Windows and `ls -a` for Linux and macOS). .002 Hidden Users Adversaries may use hidden users to hide the presence of user accounts they create or modify. Administrators may want to hide users when there are many user accounts on a given system or if they want to hide their administrative or other management accounts from other users. .003 Hidden Window Adversaries may use hidden windows to conceal malicious activity from the plain sight of users. In some cases, windows that would typically be displayed when an application carries out an operation can be hidden. This may be utilized by system administrators to avoid disrupting user work environments when carrying out administrative tasks. .004 NTFS File Attributes Adversaries may use NTFS file attributes to hide their malicious data in order to evade detection. Every New Technology File System (NTFS) formatted partition contains a Master File Table (MFT) that maintains a record for every file/directory on the partition. Within MFT entries are file attributes, such as Extended Attributes (EA) and Data [known as Alternate Data Streams (ADSs) when more than one Data attribute is present], that can be used to store arbitrary data (and even complete files). .005 Hidden File System Adversaries may use a hidden file system to conceal malicious activity from users and security tools. File systems provide a structure to store and access data from physical storage. Typically, a user engages with a file system through applications that allow them to access files and directories, which are an abstraction from their physical location (ex: disk sector). Standard file systems include FAT, NTFS, ext4, and APFS. File systems can also contain other structures, such as the Volume Boot Record (VBR) and Master File Table (MFT) in NTFS. .006 Run Virtual Instance Adversaries may carry out malicious operations using a virtual instance to avoid detection. A wide variety of virtualization technologies exist that allow for the emulation of a computer or computing environment. By running malicious code inside of a virtual instance, adversaries can hide artifacts associated with their behavior from security tools that are unable to monitor activity inside the virtual instance. Additionally, depending on the virtual networking implementation (ex:

bridged adapter), network traffic generated by the virtual instance can be difficult to trace back to the compromised host as the IP address and hostname might not match known values. .007 VBA Stomping Adversaries may hide malicious Visual Basic for Applications (VBA) payloads embedded within MS Office documents by replacing the VBA source code with benign data. .008 Email Hiding Rules Adversaries may use email rules to hide inbound emails in a compromised user's mailbox. Many email clients allow users to create inbox rules for various email functions, including moving emails to other folders, marking emails as read, or deleting emails. Rules may be created or modified within email clients or through external features such as the `New-InboxRule` or `Set-InboxRule` PowerShell cmdlets on Windows systems. .009 Resource Forking Adversaries may abuse resource forks to hide malicious code or executables to evade detection and bypass security applications. A resource fork provides applications a structured way to store resources such as thumbnail images, menu definitions, icons, dialog boxes, and code. Usage of a resource fork is identifiable when displaying a file’s extended attributes, using `ls -l@` or `xattr -l` commands. Resource forks have been deprecated and replaced with the application bundle structure. Non-localized resources are placed at the top level directory of an application bundle, while localized resources are placed in the `/Resources` folder. .010 Process Argument Spoofing Adversaries may attempt to hide process command-line arguments by overwriting process memory. Process command-line arguments are stored in the process environment block (PEB), a data structure used by Windows to store various information about/used by a process. The PEB includes the process command-line arguments that are referenced when executing the process. When a process is created, defensive tools/sensors that monitor process creations may retrieve the process arguments from the PEB. .011 Ignore Process Interrupts Adversaries may evade defensive mechanisms by executing commands that hide from process interrupt signals. Many operating systems use signals to deliver messages to control process behavior. Command interpreters often include specific commands/flags that ignore errors and other hangups, such as when the user of the active session logs off. These interrupt signals may also be used by defensive tools and/or analysts to pause or terminate specified running processes. .012 File/Path Exclusions Adversaries may attempt to hide their file-based artifacts by writing them to specific folders or file names excluded from antivirus (AV) scanning and other defensive capabilities. AV and other file-based scanners often include exclusions to optimize performance as well as ease installation and legitimate use of applications. These exclusions may be contextual (e.g., scans are only initiated in response to specific triggering events/alerts), but are also often hardcoded strings referencing specific folders and/or files assumed to be trusted and legitimate. .013 Bind Mounts Adversaries may abuse bind mounts on file structures to hide their activity and artifacts from native utilities. A bind mount maps a directory or file from one location on the filesystem to another, similar to a shortcut on Windows. It’s commonly used to provide access to specific files or directories across different environments, such as inside containers or chroot environments, and requires sudo access. .014 Extended Attributes Adversaries may abuse extended attributes (xattrs) on macOS and Linux to hide their malicious data in order to evade detection. Extended attributes are key-value pairs of file and directory metadata used by both macOS and Linux. They are not visible through standard tools like `Finder`, `ls`, or `cat` and require utilities such as `xattr` (macOS) or `getfattr` (Linux) for inspection. Operating systems and applications use xattrs for tagging, integrity checks, and access control. On Linux, xattrs are organized into namespaces such as `user.` (user permissions), `trusted.` (root permissions), `security.`, and `system.`, each with specific permissions. On macOS, xattrs are flat strings without namespace prefixes, commonly prefixed with `com.apple.*` (e.g., `com.apple.quarantine`, `com.apple.metadata:_kMDItemUserTags`) and used by system features like Gatekeeper and Spotlight. T1574 Hijack Execution Flow Adversaries may execute their own malicious payloads by hijacking the way operating systems run programs. Hijacking execution

flow can be for the purposes of persistence, since this hijacked execution may reoccur over time. Adversaries may also use these mechanisms to elevate privileges or evade defenses, such as application control or other restrictions on execution. .001 DLL Adversaries may abuse dynamic-link library files (DLLs) in order to achieve persistence, escalate privileges, and evade defenses. DLLs are libraries that contain code and data that can be simultaneously utilized by multiple programs. While DLLs are not malicious by nature, they can be abused through mechanisms such as side-loading, hijacking search order, and phantom DLL hijacking. .004 Dylib Hijacking Adversaries may execute their own payloads by placing a malicious dynamic library (dylib) with an expected name in a path a victim application searches at runtime. The dynamic loader will try to find the dylibs based on the sequential order of the search paths. Paths to dylibs may be prefixed with `@rpath`, which allows developers to use relative paths to specify an array of search paths used at runtime based on the location of the executable. Additionally, if weak linking is used, such as the `LC_LOAD_WEAK_DYLIB` function, an application will still execute even if an expected dylib is not present. Weak linking enables developers to run an application on multiple macOS versions as new APIs are added. .005 Executable Installer File Permissions Weakness Adversaries may execute their own malicious payloads by hijacking the binaries used by an installer. These processes may automatically execute specific binaries as part of their functionality or to perform other actions. If the permissions on the file system directory containing a target binary, or permissions on the binary itself, are improperly set, then the target binary may be overwritten with another binary using user-level permissions and executed by the original process. If the original process and thread are running under a higher permissions level, then the replaced binary will also execute under higher-level permissions, which could include SYSTEM. .006 Dynamic Linker Hijacking Adversaries may execute their own malicious payloads by hijacking environment variables the dynamic linker uses to load shared libraries. During the execution preparation phase of a program, the dynamic linker loads specified absolute paths of shared libraries from various environment variables and files, such as `LD_PRELOAD` on Linux or `DYLD_INSERT_LIBRARIES` on macOS. Libraries specified in environment variables are loaded first, taking precedence over system libraries with the same function name. Each platform's linker uses an extensive list of environment variables at different points in execution. These variables are often used by developers to debug binaries without needing to recompile, deconflict mapped symbols, and implement custom functions in the original library. .007 Path Interception by PATH Environment Variable Adversaries may execute their own malicious payloads by hijacking environment variables used to load libraries. The PATH environment variable contains a list of directories (User and System) that the OS searches sequentially through in search of the binary that was called from a script or the command line. .008 Path Interception by Search Order Hijacking Adversaries may execute their own malicious payloads by hijacking the search order used to load other programs. Because some programs do not call other programs using the full path, adversaries may place their own file in the directory where the calling program is located, causing the operating system to launch their malicious software at the request of the calling program. .009 Path Interception by Unquoted Path Adversaries may execute their own malicious payloads by hijacking vulnerable file path references. Adversaries can take advantage of paths that lack surrounding quotations by placing an executable in a higher level directory within the path, so that Windows will choose the adversary's executable to launch. .010 Services File Permissions Weakness Adversaries may execute their own malicious payloads by hijacking the binaries used by services. Adversaries may use flaws in the permissions of Windows services to replace the binary that is executed upon service start. These service processes may automatically execute specific binaries as part of their functionality or to perform other actions. If the permissions on the file system directory containing a target binary, or permissions on the binary itself are improperly set, then the target binary may be overwritten with another binary using user-level permissions and

executed by the original process. If the original process and thread are running under a higher permissions level, then the replaced binary will also execute under higher-level permissions, which could include SYSTEM. .011 Services Registry Permissions Weakness Adversaries may execute their own malicious payloads by hijacking the Registry entries used by services. Flaws in the permissions for Registry keys related to services can allow adversaries to redirect the originally specified executable to one they control, launching their own code when a service starts. Windows stores local service configuration information in the Registry under `HKLM\SYSTEM\CurrentControlSet\Services`. The information stored under a service's Registry keys can be manipulated to modify a service's execution parameters through tools such as the service controller, sc.exe, PowerShell, or Reg. Access to Registry keys is controlled through access control lists and user permissions. .012 COR_PROFILER Adversaries may leverage the COR_PROFILER environment variable to hijack the execution flow of programs that load the .NET CLR. The COR_PROFILER is a .NET Framework feature which allows developers to specify an unmanaged (or external of .NET) profiling DLL to be loaded into each .NET process that loads the Common Language Runtime (CLR). These profilers are designed to monitor, troubleshoot, and debug managed code executed by the .NET CLR. .013 KernelCallbackTable Adversaries may abuse the `KernelCallbackTable` of a process to hijack its execution flow in order to run their own payloads. The `KernelCallbackTable` can be found in the Process Environment Block (PEB) and is initialized to an array of graphic functions available to a GUI process once `user32.dll` is loaded. .014 AppDomainManager Adversaries may execute their own malicious payloads by hijacking how the .NET `AppDomainManager` loads assemblies. The .NET framework uses the `AppDomainManager` class to create and manage one or more isolated runtime environments (called application domains) inside a process to host the execution of .NET applications. Assemblies (`.exe` or `.dll` binaries compiled to run as .NET code) may be loaded into an application domain as executable code. T1562 Impair Defenses Adversaries may maliciously modify components of a victim environment in order to hinder or disable defensive mechanisms. This not only involves impairing preventative defenses, such as firewalls and anti-virus, but also detection capabilities that defenders can use to audit activity and identify malicious behavior. This may also span both native defenses as well as supplemental capabilities installed by users and administrators. .001 Disable or Modify Tools Adversaries may modify and/or disable security tools to avoid possible detection of their malware/tools and activities. This may take many forms, such as killing security software processes or services, modifying / deleting Registry keys or configuration files so that tools do not operate properly, or other methods to interfere with security tools scanning or reporting information. Adversaries may also disable updates to prevent the latest security patches from reaching tools on victim systems. .002 Disable Windows Event Logging Adversaries may disable Windows event logging to limit data that can be leveraged for detections and audits. Windows event logs record user and system activity such as login attempts, process creation, and much more. This data is used by security tools and analysts to generate detections. .003 Impair Command History Logging Adversaries may impair command history logging to hide commands they run on a compromised system. Various command interpreters keep track of the commands users type in their terminal so that users can retrace what they've done. .004 Disable or Modify System Firewall Adversaries may disable or modify system firewalls in order to bypass controls limiting network usage. Changes could be disabling the entire mechanism as well as adding, deleting, or modifying particular rules. This can be done numerous ways depending on the operating system, including via command-line, editing Windows Registry keys, and Windows Control Panel. .006 Indicator Blocking An adversary may attempt to block indicators or events typically captured by sensors from being gathered and analyzed. This could include maliciously redirecting or even disabling host-based sensors, such as Event Tracing for Windows (ETW), by tampering settings that control the collection and flow of

event telemetry. These settings may be stored on the system in configuration files and/or in the Registry as well as being accessible via administrative utilities such as PowerShell or Windows Management Instrumentation. .007 Disable or Modify Cloud Firewall Adversaries may disable or modify a firewall within a cloud environment to bypass controls that limit access to cloud resources. Cloud firewalls are separate from system firewalls that are described in Disable or Modify System Firewall. .008 Disable or Modify Cloud Logs An adversary may disable or modify cloud logging capabilities and integrations to limit what data is collected on their activities and avoid detection. Cloud environments allow for collection and analysis of audit and application logs that provide insight into what activities a user does within the environment. If an adversary has sufficient permissions, they can disable or modify logging to avoid detection of their activities. .009 Safe Mode Boot Adversaries may abuse Windows safe mode to disable endpoint defenses. Safe mode starts up the Windows operating system with a limited set of drivers and services. Third-party security software such as endpoint detection and response (EDR) tools may not start after booting Windows in safe mode. There are two versions of safe mode: Safe Mode and Safe Mode with Networking. It is possible to start additional services after a safe mode boot. .010 Downgrade Attack Adversaries may downgrade or use a version of system features that may be outdated, vulnerable, and/or does not support updated security controls. Downgrade attacks typically take advantage of a system's backward compatibility to force it into less secure modes of operation. .011 Spoof Security Alerting Adversaries may spoof security alerting from tools, presenting false evidence to impair defenders' awareness of malicious activity. Messages produced by defensive tools contain information about potential security events as well as the functioning status of security software and the system. Security reporting messages are important for monitoring the normal operation of a system and identifying important events that can signal a security incident. .012 Disable or Modify Linux Audit System Adversaries may disable or modify the Linux audit system to hide malicious activity and avoid detection. Linux admins use the Linux Audit system to track security-relevant information on a system. The Linux Audit system operates at the kernel-level and maintains event logs on application and system activity such as process, network, file, and login events based on pre-configured rules. .013 Disable or Modify Network Device Firewall Adversaries may disable network device-based firewall mechanisms entirely or add, delete, or modify particular rules in order to bypass controls limiting network usage. T1656 Impersonation Adversaries may impersonate a trusted person or organization in order to persuade and trick a target into performing some action on their behalf. For example, adversaries may communicate with victims (via Phishing for Information, Phishing, or Internal Spearphishing) while impersonating a known sender such as an executive, colleague, or third-party vendor. Established trust can then be leveraged to accomplish an adversary's ultimate goals, possibly against multiple victims. T1070 Indicator Removal Adversaries may delete or modify artifacts generated within systems to remove evidence of their presence or hinder defenses. Various artifacts may be created by an adversary or something that can be attributed to an adversary's actions. Typically these artifacts are used as defensive indicators related to monitored events, such as strings from downloaded files, logs that are generated from user actions, and other data analyzed by defenders. Location, format, and type of artifact (such as command or login history) are often specific to each platform. .001 Clear Windows Event Logs Adversaries may clear Windows Event Logs to hide the activity of an intrusion. Windows Event Logs are a record of a computer's alerts and notifications. There are three system-defined sources of events: System, Application, and Security, with five event types: Error, Warning, Information, Success Audit, and Failure Audit. .002 Clear Linux or Mac System Logs Adversaries may clear system logs to hide evidence of an intrusion. macOS and Linux both keep track of system or user-initiated actions via system logs. The majority of native system logging is stored under the `/var/log/` directory. Subfolders in this directory categorize logs by their related functions, such as: .003 Clear

Command History In addition to clearing system logs, an adversary may clear the command history of a compromised account to conceal the actions undertaken during an intrusion. Various command interpreters keep track of the commands users type in their terminal so that users can retrace what they've done. .004 File Deletion Adversaries may delete files left behind by the actions of their intrusion activity. Malware, tools, or other non-native files dropped or created on a system by an adversary (ex: Ingress Tool Transfer) may leave traces to indicate to what was done within a network and how. Removal of these files can occur during an intrusion, or as part of a post-intrusion process to minimize the adversary's footprint. .005 Network Share Connection Removal Adversaries may remove share connections that are no longer useful in order to clean up traces of their operation. Windows shared drive and SMB/Windows Admin Shares connections can be removed when no longer needed. Net is an example utility that can be used to remove network share connections with the `net use \system\share /delete` command. .006 Timestomp Adversaries may modify file time attributes to hide new files or changes to existing files. Timestomping is a technique that modifies the timestamps of a file (the modify, access, create, and change times), often to mimic files that are in the same folder and blend malicious files with legitimate files. .007 Clear Network Connection History and Configurations Adversaries may clear or remove evidence of malicious network connections in order to clean up traces of their operations. Configuration settings as well as various artifacts that highlight connection history may be created on a system and/or in application logs from behaviors that require network connections, such as Remote Services or External Remote Services. Defenders may use these artifacts to monitor or otherwise analyze network connections created by adversaries. .008 Clear Mailbox Data Adversaries may modify mail and mail application data to remove evidence of their activity. Email applications allow users and other programs to export and delete mailbox data via command line tools or use of APIs. Mail application data can be emails, email metadata, or logs generated by the application or operating system, such as export requests. .009 Clear Persistence Adversaries may clear artifacts associated with previously established persistence on a host system to remove evidence of their activity. This may involve various actions, such as removing services, deleting executables, Modify Registry, Plist File Modification, or other methods of cleanup to prevent defenders from collecting evidence of their persistent presence. Adversaries may also delete accounts previously created to maintain persistence (i.e. Create Account). .010 Relocate Malware Once a payload is delivered, adversaries may reproduce copies of the same malware on the victim system to remove evidence of their presence and/or avoid defenses. Copying malware payloads to new locations may also be combined with File Deletion to cleanup older artifacts. T1202 Indirect Command Execution Adversaries may abuse utilities that allow for command execution to bypass security restrictions that limit the use of command-line interpreters. Various Windows utilities may be used to execute commands, possibly without invoking cmd. For example, Forfiles, the Program Compatibility Assistant (`pcalua.exe`), components of the Windows Subsystem for Linux (WSL), `Scriptrunner.exe` , as well as other utilities may invoke the execution of programs and commands from a Command and Scripting Interpreter, Run window, or via scripts. Adversaries may also abuse the `ssh.exe` binary to execute malicious commands via the `ProxyCommand` and `LocalCommand` options, which can be invoked via the `-o` flag or by modifying the SSH config file. T1036 Masquerading Adversaries may attempt to manipulate features of their artifacts to make them appear legitimate or benign to users and/or security tools. Masquerading occurs when the name or location of an object, legitimate or malicious, is manipulated or abused for the sake of evading defenses and observation. This may include manipulating file metadata, tricking users into misidentifying the file type, and giving legitimate task or service names. .001 Invalid Code Signature Adversaries may attempt to mimic features of valid code signatures to increase the chance of deceiving a user, analyst, or tool. Code signing provides a level of authenticity on a binary from the developer and a guarantee that the binary has not been

tampered with. Adversaries can copy the metadata and signature information from a signed program, then use it as a template for an unsigned program. Files with invalid code signatures will fail digital signature validation checks, but they may appear more legitimate to users and security tools may improperly handle these files. .002 Right-to-Left Override Adversaries may abuse the right-to-left override (RTLO or RLO) character (U+202E) to disguise a string and/or file name to make it appear benign. RTLO is a non-printing Unicode character that causes the text that follows it to be displayed in reverse. For example, a Windows screensaver executable named `March 25 \u202Excod.scr` will display as `March 25 rcs.docx`. A JavaScript file named `photo_high_re\u202Egnp.js` will be displayed as `photo_high_resj.png`. .003 Rename Legitimate Utilities Adversaries may rename legitimate / system utilities to try to evade security mechanisms concerning the usage of those utilities. Security monitoring and control mechanisms may be in place for legitimate utilities adversaries are capable of abusing, including both built-in binaries and tools such as PSExec, AutoHotKey, and IronPython. It may be possible to bypass those security mechanisms by renaming the utility prior to utilization (ex: rename `rundll32.exe`). An alternative case occurs when a legitimate utility is copied or moved to a different directory and renamed to avoid detections based on these utilities executing from non-standard paths. .004 Masquerade Task or Service Adversaries may attempt to manipulate the name of a task or service to make it appear legitimate or benign. Tasks/services executed by the Task Scheduler or systemd will typically be given a name and/or description. Windows services will have a service name as well as a display name. Many benign tasks and services exist that have commonly associated names. Adversaries may give tasks or services names that are similar or identical to those of legitimate ones. .005 Match Legitimate Resource Name or Location Adversaries may match or approximate the name or location of legitimate files, Registry keys, or other resources when naming/placing them. This is done for the sake of evading defenses and observation. .006 Space after Filename Adversaries can hide a program's true filetype by changing the extension of a file. With certain file types (specifically this does not work with .app extensions), appending a space to the end of a filename will change how the file is processed by the operating system. .007 Double File Extension Adversaries may abuse a double extension in the filename as a means of masquerading the true file type. A file name may include a secondary file type extension that may cause only the first extension to be displayed (ex: `File.txt.exe` may render in some views as just `File.txt`). However, the second extension is the true file type that determines how the file is opened and executed. The real file extension may be hidden by the operating system in the file browser (ex: explorer.exe), as well as in any software configured using or similar to the system’s policies. .008 Masquerade File Type Adversaries may masquerade malicious payloads as legitimate files through changes to the payload's formatting, including the file’s signature, extension, icon, and contents. Various file types have a typical standard format, including how they are encoded and organized. For example, a file’s signature (also known as header or magic bytes) is the beginning bytes of a file and is often used to identify the file’s type. For example, the header of a JPEG file, is `0xFF 0xD8` and the file extension is either `.JPE`, `.JPEG` or `.JPG`. .009 Break Process Trees An adversary may attempt to evade process tree-based analysis by modifying executed malware's parent process ID (PPID). If endpoint protection software leverages the "parent-child" relationship for detection, breaking this relationship could result in the adversary’s behavior not being associated with previous process tree activity. On Unix-based systems breaking this process tree is common practice for administrators to execute software using scripts and programs. .010 Masquerade Account Name Adversaries may match or approximate the names of legitimate accounts to make newly created ones appear benign. This will typically occur during Create Account, although accounts may also be renamed at a later date. This may also coincide with Account Access Removal if the actor first deletes an account before re-creating one with the same name. .011 Overwrite Process Arguments Adversaries may modify a process's in-memory

arguments to change its name in order to appear as a legitimate or benign process. On Linux, the operating system stores command-line arguments in the process's stack and passes them to the `main()` function as the `argv` array. The first element, `argv[0]`, typically contains the process name or path - by default, the command used to actually start the process (e.g., `cat /etc/passwd`). By default, the Linux `/proc` filesystem uses this value to represent the process name. The `/proc/<PID>/cmdline` file reflects the contents of this memory, and tools like `ps` use it to display process information. Since arguments are stored in user-space memory at launch, this modification can be performed without elevated privileges. .012 Browser Fingerprint Adversaries may attempt to blend in with legitimate traffic by spoofing browser and system attributes like operating system, system language, platform, user-agent string, resolution, time zone, etc. The HTTP User-Agent request header is a string that lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent. T1556 Modify Authentication Process Adversaries may modify authentication mechanisms and processes to access user credentials or enable otherwise unwarranted access to accounts. The authentication process is handled by mechanisms, such as the Local Security Authentication Server (LSASS) process and the Security Accounts Manager (SAM) on Windows, pluggable authentication modules (PAM) on Unix-based systems, and authorization plugins on MacOS systems, responsible for gathering, storing, and validating credentials. By modifying an authentication process, an adversary may be able to authenticate to a service or system without using Valid Accounts. .001 Domain Controller Authentication Adversaries may patch the authentication process on a domain controller to bypass the typical authentication mechanisms and enable access to accounts. .002 Password Filter DLL Adversaries may register malicious password filter dynamic link libraries (DLLs) into the authentication process to acquire user credentials as they are validated. .003 Pluggable Authentication Modules Adversaries may modify pluggable authentication modules (PAM) to access user credentials or enable otherwise unwarranted access to accounts. PAM is a modular system of configuration files, libraries, and executable files which guide authentication for many services. The most common authentication module is `pam_unix.so`, which retrieves, sets, and verifies account authentication information in `/etc/passwd` and `/etc/shadow`. .004 Network Device Authentication Adversaries may use Patch System Image to hard code a password in the operating system, thus bypassing of native authentication mechanisms for local accounts on network devices. .005 Reversible Encryption An adversary may abuse Active Directory authentication encryption properties to gain access to credentials on Windows systems. The `AllowReversiblePasswordEncryption` property specifies whether reversible password encryption for an account is enabled or disabled. By default this property is disabled (instead storing user credentials as the output of one-way hashing functions) and should not be enabled unless legacy or other software require it. .006 Multi-Factor Authentication Adversaries may disable or modify multi-factor authentication (MFA) mechanisms to enable persistent access to compromised accounts. .007 Hybrid Identity Adversaries may patch, modify, or otherwise backdoor cloud authentication processes that are tied to on-premises user identities in order to bypass typical authentication mechanisms, access credentials, and enable persistent access to accounts. .008 Network Provider DLL Adversaries may register malicious network provider dynamic link libraries (DLLs) to capture cleartext user credentials during the authentication process. Network provider DLLs allow Windows to interface with specific network protocols and can also support add-on credential management functions. During the logon process, Winlogon (the interactive logon module) sends credentials to the local `mpnotify.exe` process via RPC. The `mpnotify.exe` process then shares the credentials in cleartext with registered credential managers when notifying that a logon event is happening. .009 Conditional Access Policies Adversaries may disable or modify conditional access policies to enable persistent access to compromised accounts. Conditional access policies are additional verifications used by identity providers and identity and

access management systems to determine whether a user should be granted access to a resource. T1578 Modify Cloud Compute Infrastructure An adversary may attempt to modify a cloud account's compute service infrastructure to evade defenses. A modification to the compute service infrastructure can include the creation, deletion, or modification of one or more components such as compute instances, virtual machines, and snapshots. .001 Create Snapshot An adversary may create a snapshot or data backup within a cloud account to evade defenses. A snapshot is a point-in-time copy of an existing cloud compute component such as a virtual machine (VM), virtual hard drive, or volume. An adversary may leverage permissions to create a snapshot in order to bypass restrictions that prevent access to existing compute service infrastructure, unlike in Revert Cloud Instance where an adversary may revert to a snapshot to evade detection and remove evidence of their presence. .002 Create Cloud Instance An adversary may create a new instance or virtual machine (VM) within the compute service of a cloud account to evade defenses. Creating a new instance may allow an adversary to bypass firewall rules and permissions that exist on instances currently residing within an account. An adversary may Create Snapshot of one or more volumes in an account, create a new instance, mount the snapshots, and then apply a less restrictive security policy to collect Data from Local System or for Remote Data Staging. .003 Delete Cloud Instance An adversary may delete a cloud instance after they have performed malicious activities in an attempt to evade detection and remove evidence of their presence. Deleting an instance or virtual machine can remove valuable forensic artifacts and other evidence of suspicious behavior if the instance is not recoverable. .004 Revert Cloud Instance An adversary may revert changes made to a cloud instance after they have performed malicious activities in attempt to evade detection and remove evidence of their presence. In highly virtualized environments, such as cloud-based infrastructure, this may be accomplished by restoring virtual machine (VM) or data storage snapshots through the cloud management dashboard or cloud APIs. .005 Modify Cloud Compute Configurations Adversaries may modify settings that directly affect the size, locations, and resources available to cloud compute infrastructure in order to evade defenses. These settings may include service quotas, subscription associations, tenant-wide policies, or other configurations that impact available compute. Such modifications may allow adversaries to abuse the victim’s compute resources to achieve their goals, potentially without affecting the execution of running instances and/or revealing their activities to the victim. T1666 Modify Cloud Resource Hierarchy Adversaries may attempt to modify hierarchical structures in infrastructure-as-a-service (IaaS) environments in order to evade defenses. T1112 Modify Registry Adversaries may interact with the Windows Registry as part of a variety of other techniques to aid in defense evasion, persistence, and execution. T1601 Modify System Image Adversaries may make changes to the operating system of embedded network devices to weaken defenses and provide new capabilities for themselves. On such devices, the operating systems are typically monolithic and most of the device functionality and capabilities are contained within a single file. .001 Patch System Image Adversaries may modify the operating system of a network device to introduce new capabilities or weaken existing defenses. Some network devices are built with a monolithic architecture, where the entire operating system and most of the functionality of the device is contained within a single file. Adversaries may change this file in storage, to be loaded in a future boot, or in memory during runtime. .002 Downgrade System Image Adversaries may install an older version of the operating system of a network device to weaken security. Older operating system versions on network devices often have weaker encryption ciphers and, in general, fewer/less updated defensive features. T1599 Network Boundary Bridging Adversaries may bridge network boundaries by compromising perimeter network devices or internal devices responsible for network segmentation. Breaching these devices may enable an adversary to bypass restrictions on traffic routing that otherwise separate trusted and untrusted networks. .001 Network Address Translation Traversal Adversaries may

bridge network boundaries by modifying a network device’s Network Address Translation (NAT) configuration. Malicious modifications to NAT may enable an adversary to bypass restrictions on traffic routing that otherwise separate trusted and untrusted networks. T1027 Obfuscated Files or Information Adversaries may attempt to make an executable or file difficult to discover or analyze by encrypting, encoding, or otherwise obfuscating its contents on the system or in transit. This is common behavior that can be used across different platforms and the network to evade defenses. .001 Binary Padding Adversaries may use binary padding to add junk data and change the on-disk representation of malware. This can be done without affecting the functionality or behavior of a binary, but can increase the size of the binary beyond what some security tools are capable of handling due to file size limitations. .002 Software Packing Adversaries may perform software packing or virtual machine software protection to conceal their code. Software packing is a method of compressing or encrypting an executable. Packing an executable changes the file signature in an attempt to avoid signature-based detection. Most decompression techniques decompress the executable code in memory. Virtual machine software protection translates an executable's original code into a special format that only a special virtual machine can run. A virtual machine is then called to run this code. .003 Steganography Adversaries may use steganography techniques in order to prevent the detection of hidden information. Steganographic techniques can be used to hide data in digital media such as images, audio tracks, video clips, or text files. .004 Compile After Delivery Adversaries may attempt to make payloads difficult to discover and analyze by delivering files to victims as uncompiled code. Text-based source code files may subvert analysis and scrutiny from protections targeting executables/binaries. These payloads will need to be compiled before execution; typically via native utilities such as ilasm.exe, csc.exe, or GCC/MinGW. .005 Indicator Removal from Tools Adversaries may remove indicators from tools if they believe their malicious tool was detected, quarantined, or otherwise curtailed. They can modify the tool by removing the indicator and using the updated version that is no longer detected by the target's defensive systems or subsequent targets that may use similar systems. .006 HTML Smuggling Adversaries may smuggle data and files past content filters by hiding malicious payloads inside of seemingly benign HTML files. HTML documents can store large binary objects known as JavaScript Blobs (immutable data that represents raw bytes) that can later be constructed into file-like objects. Data may also be stored in Data URLs, which enable embedding media type or MIME files inline of HTML documents. HTML5 also introduced a download attribute that may be used to initiate file downloads. .007 Dynamic API Resolution Adversaries may obfuscate then dynamically resolve API functions called by their malware in order to conceal malicious functionalities and impair defensive analysis. Malware commonly uses various Native API functions provided by the OS to perform various tasks such as those involving processes, files, and other system artifacts. .008 Stripped Payloads Adversaries may attempt to make a payload difficult to analyze by removing symbols, strings, and other human readable information. Scripts and executables may contain variables names and other strings that help developers document code functionality. Symbols are often created by an operating system’s `linker` when executable payloads are compiled. Reverse engineers use these symbols and strings to analyze code and to identify functionality in payloads. .009 Embedded Payloads Adversaries may embed payloads within other files to conceal malicious content from defenses. Otherwise seemingly benign files (such as scripts and executables) may be abused to carry and obfuscate malicious payloads and content. In some cases, embedded payloads may also enable adversaries to Subvert Trust Controls by not impacting execution controls such as digital signatures and notarization tickets. .010 Command Obfuscation Adversaries may obfuscate content during command execution to impede detection. Command-line obfuscation is a method of making strings and patterns within commands and scripts more difficult to signature and analyze. This type of obfuscation can be included within commands executed by delivered payloads (e.g., Phishing and

Drive-by Compromise) or interactively via Command and Scripting Interpreter. .011 Fileless Storage Adversaries may store data in "fileless" formats to conceal malicious activity from defenses. Fileless storage can be broadly defined as any format other than a file. Common examples of non-volatile fileless storage in Windows systems include the Windows Registry, event logs, or WMI repository. Shared memory directories on Linux systems (`/dev/shm` , `/run/shm` , `/var/run` , and `/var/lock`) and volatile directories on Network Devices (`/tmp` and `/volatile`) may also be considered fileless storage, as files written to these directories are mapped directly to RAM and not stored on the disk.. .012 LNK Icon Smuggling Adversaries may smuggle commands to download malicious payloads past content filters by hiding them within otherwise seemingly benign windows shortcut files. Windows shortcut files (.LNK) include many metadata fields, including an icon location field (also known as the `IconEnvironmentDataBlock`) designed to specify the path to an icon file that is to be displayed for the LNK file within a host directory. .013 Encrypted/Encoded File Adversaries may encrypt or encode files to obfuscate strings, bytes, and other specific patterns to impede detection. Encrypting and/or encoding file content aims to conceal malicious artifacts within a file used in an intrusion. Many other techniques, such as Software Packing, Steganography, and Embedded Payloads, share this same broad objective. Encrypting and/or encoding files could lead to a lapse in detection of static signatures, only for this malicious content to be revealed (i.e., Deobfuscate/Decode Files or Information) at the time of execution/use. .014 Polymorphic Code Adversaries may utilize polymorphic code (also known as metamorphic or mutating code) to evade detection. Polymorphic code is a type of software capable of changing its runtime footprint during code execution. With each execution of the software, the code is mutated into a different version of itself that achieves the same purpose or objective as the original. This functionality enables the malware to evade traditional signature-based defenses, such as antivirus and antimalware tools.

Other obfuscation techniques can be used in conjunction with polymorphic code to accomplish the intended effects, including using mutation engines to conduct actions such as Software Packing, Command Obfuscation, or Encrypted/Encoded File. .015 Compression Adversaries may use compression to obfuscate their payloads or files. Compressed file formats such as ZIP, gzip, 7z, and RAR can compress and archive multiple files together to make it easier and faster to transfer files. In addition to compressing files, adversaries may also compress shellcode directly - for example, in order to store it in a Windows Registry key (i.e., Fileless Storage). .016 Junk Code Insertion Adversaries may use junk code / dead code to obfuscate a malware’s functionality. Junk code is code that either does not execute, or if it does execute, does not change the functionality of the code. Junk code makes analysis more difficult and time-consuming, as the analyst steps through non-functional code instead of analyzing the main code. It also may hinder detections that rely on static code analysis due to the use of benign functionality, especially when combined with Compression or Software Packing. .017 SVG Smuggling Adversaries may smuggle data and files past content filters by hiding malicious payloads inside of seemingly benign SVG files. SVGs, or Scalable Vector Graphics, are vector-based image files constructed using XML. As such, they can legitimately include `<script>` tags that enable adversaries to include malicious JavaScript payloads. However, SVGs may appear less suspicious to users than other types of executable files, as they are often treated as image files. T1647 Plist File Modification Adversaries may modify property list files (plist files) to enable other malicious activity, while also potentially evading and bypassing system defenses. macOS applications use plist files, such as the `info.plist` file, to store properties and configuration settings that inform the operating system how to handle the application at runtime. Plist files are structured metadata in key-value pairs formatted in XML based on Apple's Core Foundation DTD. Plist files can be saved in text or binary format. T1542 Pre-OS Boot Adversaries may abuse Pre-OS Boot mechanisms as a way to establish persistence on a system. During the

booting process of a computer, firmware and various startup services are loaded before the operating system. These programs control flow of execution before the operating system takes control. .001 System Firmware Adversaries may modify system firmware to persist on systems.The BIOS (Basic Input/Output System) and The Unified Extensible Firmware Interface (UEFI) or Extensible Firmware Interface (EFI) are examples of system firmware that operate as the software interface between the operating system and hardware of a computer. .002 Component Firmware Adversaries may modify component firmware to persist on systems. Some adversaries may employ sophisticated means to compromise computer components and install malicious firmware that will execute adversary code outside of the operating system and main system firmware or BIOS. This technique may be similar to System Firmware but conducted upon other system components/devices that may not have the same capability or level of integrity checking. .003 Bootkit Adversaries may use bootkits to persist on systems. A bootkit is a malware variant that modifies the boot sectors of a hard drive, allowing malicious code to execute before a computer's operating system has loaded. Bootkits reside at a layer below the operating system and may make it difficult to perform full remediation unless an organization suspects one was used and can act accordingly. .004 ROMMONkit Adversaries may abuse the ROM Monitor (ROMMON) by loading an unauthorized firmware with adversary code to provide persistent access and manipulate device behavior that is difficult to detect. .005 TFTP Boot Adversaries may abuse netbooting to load an unauthorized network device operating system from a Trivial File Transfer Protocol (TFTP) server. TFTP boot (netbooting) is commonly used by network administrators to load configuration-controlled network device images from a centralized management server. Netbooting is one option in the boot sequence and can be used to centralize, manage, and control device images. T1055 Process Injection Adversaries may inject code into processes in order to evade process-based defenses as well as possibly elevate privileges. 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. .001 Dynamic-link Library Injection Adversaries may inject dynamic-link libraries (DLLs) into processes in order to evade process-based defenses as well as possibly elevate privileges. DLL injection is a method of executing arbitrary code in the address space of a separate live process. .002 Portable Executable Injection Adversaries may inject portable executables (PE) into processes in order to evade process-based defenses as well as possibly elevate privileges. PE injection is a method of executing arbitrary code in the address space of a separate live process. .003 Thread Execution Hijacking Adversaries may inject malicious code into hijacked processes in order to evade process-based defenses as well as possibly elevate privileges. Thread Execution Hijacking is a method of executing arbitrary code in the address space of a separate live process. .004 Asynchronous Procedure Call Adversaries may inject malicious code into processes via the asynchronous procedure call (APC) queue in order to evade process-based defenses as well as possibly elevate privileges. APC injection is a method of executing arbitrary code in the address space of a separate live process. .005 Thread Local Storage Adversaries may inject malicious code into processes via thread local storage (TLS) callbacks in order to evade process-based defenses as well as possibly elevate privileges. TLS callback injection is a method of executing arbitrary code in the address space of a separate live process. .008 Ptrace System Calls Adversaries may inject malicious code into processes via ptrace (process trace) system calls in order to evade process-based defenses as well as possibly elevate privileges. Ptrace system call injection is a method of executing arbitrary code in the address space of a separate live process. .009 Proc Memory Adversaries may inject malicious code into processes via the /proc filesystem in order to evade process-based defenses as well as possibly elevate privileges. Proc memory injection is a method of executing

arbitrary code in the address space of a separate live process. .011 Extra Window Memory Injection Adversaries may inject malicious code into process via Extra Window Memory (EWM) in order to evade process-based defenses as well as possibly elevate privileges. EWM injection is a method of executing arbitrary code in the address space of a separate live process. .012 Process Hollowing Adversaries may inject malicious code into suspended and hollowed processes in order to evade process-based defenses. Process hollowing is a method of executing arbitrary code in the address space of a separate live process. .013 Process Doppelgänging Adversaries may inject malicious code into process via process doppelgänging in order to evade process-based defenses as well as possibly elevate privileges. Process doppelgänging is a method of executing arbitrary code in the address space of a separate live process. .014 VDSO Hijacking Adversaries may inject malicious code into processes via VDSO hijacking in order to evade process-based defenses as well as possibly elevate privileges. Virtual dynamic shared object (vdso) hijacking is a method of executing arbitrary code in the address space of a separate live process. .015 ListPlanting Adversaries may abuse list-view controls to inject malicious code into hijacked processes in order to evade process-based defenses as well as possibly elevate privileges. ListPlanting is a method of executing arbitrary code in the address space of a separate live process. Code executed via ListPlanting may also evade detection from security products since the execution is masked under a legitimate process. T1620 Reflective Code Loading Adversaries may reflectively load code into a process in order to conceal the execution of malicious payloads. Reflective loading involves allocating then executing payloads directly within the memory of the process, vice creating a thread or process backed by a file path on disk (e.g., Shared Modules). T1207 Rogue Domain Controller Adversaries may register a rogue Domain Controller to enable manipulation of Active Directory data. DCShadow may be used to create a rogue Domain Controller (DC). DCShadow is a method of manipulating Active Directory (AD) data, including objects and schemas, by registering (or reusing an inactive registration) and simulating the behavior of a DC. Once registered, a rogue DC may be able to inject and replicate changes into AD infrastructure for any domain object, including credentials and keys. T1014 Rootkit Adversaries may use rootkits to hide the presence of programs, files, network connections, services, drivers, and other system components. Rootkits are programs that hide the existence of malware by intercepting/hooking and modifying operating system API calls that supply system information. T1679 Selective Exclusion Adversaries may intentionally exclude certain files, folders, directories, file types, or system components from encryption or tampering during a ransomware or malicious payload execution. Some file extensions that adversaries may avoid encrypting include `.dll`, `.exe`, and `.lnk`. T1553 Subvert Trust Controls Adversaries may undermine security controls that will either warn users of untrusted activity or prevent execution of untrusted programs. Operating systems and security products may contain mechanisms to identify programs or websites as possessing some level of trust. Examples of such features would include a program being allowed to run because it is signed by a valid code signing certificate, a program prompting the user with a warning because it has an attribute set from being downloaded from the Internet, or getting an indication that you are about to connect to an untrusted site. .001 Gatekeeper Bypass Adversaries may modify file attributes and subvert Gatekeeper functionality to evade user prompts and execute untrusted programs. Gatekeeper is a set of technologies that act as layer of Apple's security model to ensure only trusted applications are executed on a host. Gatekeeper was built on top of File Quarantine in Snow Leopard (10.6, 2009) and has grown to include Code Signing, security policy compliance, Notarization, and more. Gatekeeper also treats applications running for the first time differently than reopened applications. .002 Code Signing Adversaries may create, acquire, or steal code signing materials to sign their malware or tools. Code signing provides a level of authenticity on a binary from the developer and a guarantee that the binary has not been tampered with. The certificates used during an operation may be created, acquired, or stolen by the

adversary. Unlike Invalid Code Signature, this activity will result in a valid signature. .003 SIP and Trust Provider Hijacking Adversaries may tamper with SIP and trust provider components to mislead the operating system and application control tools when conducting signature validation checks. In user mode, Windows Authenticode digital signatures are used to verify a file's origin and integrity, variables that may be used to establish trust in signed code (ex: a driver with a valid Microsoft signature may be handled as safe). The signature validation process is handled via the WinVerifyTrust application programming interface (API) function, which accepts an inquiry and coordinates with the appropriate trust provider, which is responsible for validating parameters of a signature. .004 Install Root Certificate Adversaries may install a root certificate on a compromised system to avoid warnings when connecting to adversary controlled web servers. Root certificates are used in public key cryptography to identify a root certificate authority (CA). When a root certificate is installed, the system or application will trust certificates in the root's chain of trust that have been signed by the root certificate. Certificates are commonly used for establishing secure TLS/SSL communications within a web browser. When a user attempts to browse a website that presents a certificate that is not trusted an error message will be displayed to warn the user of the security risk. Depending on the security settings, the browser may not allow the user to establish a connection to the website. .005 Mark-of-the-Web Bypass Adversaries may abuse specific file formats to subvert Mark-of-the-Web (MOTW) controls. In Windows, when files are downloaded from the Internet, they are tagged with a hidden NTFS Alternate Data Stream (ADS) named `Zone.Identifier` with a specific value known as the MOTW. Files that are tagged with MOTW are protected and cannot perform certain actions. For example, starting in MS Office 10, if a MS Office file has the MOTW, it will open in Protected View. Executables tagged with the MOTW will be processed by Windows Defender SmartScreen that compares files with an allowlist of well-known executables. If the file is not known/trusted, SmartScreen will prevent the execution and warn the user not to run it. .006 Code Signing Policy Modification Adversaries may modify code signing policies to enable execution of unsigned or self-signed code. Code signing provides a level of authenticity on a program from a developer and a guarantee that the program has not been tampered with. Security controls can include enforcement mechanisms to ensure that only valid, signed code can be run on an operating system. T1218 System Binary Proxy Execution Adversaries may bypass process and/or signature-based defenses by proxying execution of malicious content with signed, or otherwise trusted, binaries. Binaries used in this technique are often Microsoft-signed files, indicating that they have been either downloaded from Microsoft or are already native in the operating system. Binaries signed with trusted digital certificates can typically execute on Windows systems protected by digital signature validation. Several Microsoft signed binaries that are default on Windows installations can be used to proxy execution of other files or commands. .001 Compiled HTML File Adversaries may abuse Compiled HTML files (.chm) to conceal malicious code. CHM files are commonly distributed as part of the Microsoft HTML Help system. CHM files are compressed compilations of various content such as HTML documents, images, and scripting/web related programming languages such VBA, JScript, Java, and ActiveX. CHM content is displayed using underlying components of the Internet Explorer browser loaded by the HTML Help executable program (hh.exe). .002 Control Panel Adversaries may abuse control.exe to proxy execution of malicious payloads. The Windows Control Panel process binary (control.exe) handles execution of Control Panel items, which are utilities that allow users to view and adjust computer settings. .003 CMSTP Adversaries may abuse CMSTP to proxy execution of malicious code. The Microsoft Connection Manager Profile Installer (CMSTP.exe) is a command-line program used to install Connection Manager service profiles. CMSTP.exe accepts an installation information file (INF) as a parameter and installs a service profile leveraged for remote access connections. .004 InstallUtil Adversaries may use InstallUtil to proxy execution of code through a trusted

Windows utility. InstallUtil is a command-line utility that allows for installation and uninstallation of resources by executing specific installer components specified in .NET binaries. The InstallUtil binary may also be digitally signed by Microsoft and located in the .NET directories on a Windows system: `C:\Windows\Microsoft.NET\Framework\v\InstallUtil.exe` and `C:\Windows\Microsoft.NET\Framework64\v\InstallUtil.exe` . .005 Mshta Adversaries may abuse mshta.exe to proxy execution of malicious .hta files and Javascript or VBScript through a trusted Windows utility. There are several examples of different types of threats leveraging mshta.exe during initial compromise and for execution of code .007 Msiexec Adversaries may abuse msiexec.exe to proxy execution of malicious payloads. Msiexec.exe is the command-line utility for the Windows Installer and is thus commonly associated with executing installation packages (.msi). The Msiexec.exe binary may also be digitally signed by Microsoft. .008 Odbcconf Adversaries may abuse odbcconf.exe to proxy execution of malicious payloads. Odbcconf.exe is a Windows utility that allows you to configure Open Database Connectivity (ODBC) drivers and data source names. The Odbcconf.exe binary may be digitally signed by Microsoft. .009 Regsvcs/Regasm Adversaries may abuse Regsvcs and Regasm to proxy execution of code through a trusted Windows utility. Regsvcs and Regasm are Windows command-line utilities that are used to register .NET Component Object Model (COM) assemblies. Both are binaries that may be digitally signed by Microsoft. .010 Regsvr32 Adversaries may abuse Regsvr32.exe to proxy execution of malicious code. Regsvr32.exe is a command-line program used to register and unregister object linking and embedding controls, including dynamic link libraries (DLLs), on Windows systems. The Regsvr32.exe binary may also be signed by Microsoft. .011 Rundll32 Adversaries may abuse rundll32.exe to proxy execution of malicious code. Using rundll32.exe, vice executing directly (i.e. Shared Modules), may avoid triggering security tools that may not monitor execution of the rundll32.exe process because of allowlists or false positives from normal operations. Rundll32.exe is commonly associated with executing DLL payloads (ex: `rundll32.exe {DLLname, DLLfunction}`). .012 Verclsid Adversaries may abuse verclsid.exe to proxy execution of malicious code. Verclsid.exe is known as the Extension CLSID Verification Host and is responsible for verifying each shell extension before they are used by Windows Explorer or the Windows Shell. .013 Mavinject Adversaries may abuse mavinject.exe to proxy execution of malicious code. Mavinject.exe is the Microsoft Application Virtualization Injector, a Windows utility that can inject code into external processes as part of Microsoft Application Virtualization (App-V). .014 MMC Adversaries may abuse mmc.exe to proxy execution of malicious .msc files. Microsoft Management Console (MMC) is a binary that may be signed by Microsoft and is used in several ways in either its GUI or in a command prompt. MMC can be used to create, open, and save custom consoles that contain administrative tools created by Microsoft, called snap-ins. These snap-ins may be used to manage Windows systems locally or remotely. MMC can also be used to open Microsoft created .msc files to manage system configuration. .015 Electron Applications Adversaries may abuse components of the Electron framework to execute malicious code. The Electron framework hosts many common applications such as Signal, Slack, and Microsoft Teams. Originally developed by GitHub, Electron is a cross-platform desktop application development framework that employs web technologies like JavaScript, HTML, and CSS. The Chromium engine is used to display web content and Node.js runs the backend code. T1216 System Script Proxy Execution Adversaries may use trusted scripts, often signed with certificates, to proxy the execution of malicious files. Several Microsoft signed scripts that have been downloaded from Microsoft or are default on Windows installations can be used to proxy execution of other files. This behavior may be abused by adversaries to execute malicious files that could bypass application control and signature validation on systems. .001 PubPrn Adversaries may use PubPrn to proxy execution of malicious remote files. PubPrn.vbs is a Visual Basic script that publishes a

printer to Active Directory Domain Services. The script may be signed by Microsoft and is commonly executed through the Windows Command Shell via `Cscript.exe`. For example, the following code publishes a printer within the specified domain: `cscript pubprn Printer1 LDAP://CN=Container1,DC=Domain1,DC=Com`. .002 SyncAppvPublishingServer Adversaries may abuse SyncAppvPublishingServer.vbs to proxy execution of malicious PowerShell commands. SyncAppvPublishingServer.vbs is a Visual Basic script associated with how Windows virtualizes applications (Microsoft Application Virtualization, or App-V). For example, Windows may render Win32 applications to users as virtual applications, allowing users to launch and interact with them as if they were installed locally. T1221 Template Injection Adversaries may create or modify references in user document templates to conceal malicious code or force authentication attempts. For example, Microsoft's Office Open XML (OOXML) specification defines an XML-based format for Office documents (.docx, xlsx, .pptx) to replace older binary formats (.doc, .xls, .ppt). OOXML files are packed together ZIP archives compromised of various XML files, referred to as parts, containing properties that collectively define how a document is rendered. T1205 Traffic Signaling Adversaries may use traffic signaling to hide open ports or other malicious functionality used for persistence or command and control. Traffic signaling involves the use of a magic value or sequence that must be sent to a system to trigger a special response, such as opening a closed port or executing a malicious task. This may take the form of sending a series of packets with certain characteristics before a port will be opened that the adversary can use for command and control. Usually this series of packets consists of attempted connections to a predefined sequence of closed ports (i.e. Port Knocking), but can involve unusual flags, specific strings, or other unique characteristics. After the sequence is completed, opening a port may be accomplished by the host-based firewall, but could also be implemented by custom software. .001 Port Knocking Adversaries may use port knocking to hide open ports used for persistence or command and control. To enable a port, an adversary sends a series of attempted connections to a predefined sequence of closed ports. After the sequence is completed, opening a port is often accomplished by the host based firewall, but could also be implemented by custom software. .002 Socket Filters Adversaries may attach filters to a network socket to monitor then activate backdoors used for persistence or command and control. With elevated permissions, adversaries can use features such as the `libpcap` library to open sockets and install filters to allow or disallow certain types of data to come through the socket. The filter may apply to all traffic passing through the specified network interface (or every interface if not specified). When the network interface receives a packet matching the filter criteria, additional actions can be triggered on the host, such as activation of a reverse shell. T1127 Trusted Developer Utilities Proxy Execution Adversaries may take advantage of trusted developer utilities to proxy execution of malicious payloads. There are many utilities used for software development related tasks that can be used to execute code in various forms to assist in development, debugging, and reverse engineering. These utilities may often be signed with legitimate certificates that allow them to execute on a system and proxy execution of malicious code through a trusted process that effectively bypasses application control solutions. .001 MSBuild Adversaries may use MSBuild to proxy execution of code through a trusted Windows utility. MSBuild.exe (Microsoft Build Engine) is a software build platform used by Visual Studio. It handles XML formatted project files that define requirements for loading and building various platforms and configurations. .002 ClickOnce Adversaries may use ClickOnce applications (.appref-ms and .application files) to proxy execution of code through a trusted Windows utility. ClickOnce is a deployment that enables a user to create self-updating Windows-based .NET applications (i.e, .XBAP, .EXE, or .DLL) that install and run from a file share or web page with minimal user interaction. The application launches as a child process of DFSVC.EXE, which is responsible for installing, launching, and updating the application. .003 JamPlus Adversaries may use `JamPlus` to proxy the execution of a malicious script. `JamPlus` is a build utility

tool for code and data build systems. It works with several popular compilers and can be used for generating workspaces in code editors such as Visual Studio. T1535 Unused/Unsupported Cloud Regions Adversaries may create cloud instances in unused geographic service regions in order to evade detection. Access is usually obtained through compromising accounts used to manage cloud infrastructure. T1550 Use Alternate Authentication Material Adversaries may use alternate authentication material, such as password hashes, Kerberos tickets, and application access tokens, in order to move laterally within an environment and bypass normal system access controls. .001 Application Access Token Adversaries may use stolen application access tokens to bypass the typical authentication process and access restricted accounts, information, or services on remote systems. These tokens are typically stolen from users or services and used in lieu of login credentials. .002 Pass the Hash Adversaries may "pass the hash" using stolen password hashes to move laterally within an environment, bypassing normal system access controls. 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. .003 Pass the Ticket Adversaries may "pass the ticket" using stolen Kerberos tickets to move laterally within an environment, bypassing normal system access controls. Pass the ticket (PtT) is a method of authenticating to a system using Kerberos tickets without having access to an account's password. Kerberos authentication can be used as the first step to lateral movement to a remote system. .004 Web Session Cookie Adversaries can use stolen session cookies to authenticate to web applications and services. This technique bypasses some multi-factor authentication protocols since the session is already authenticated. T1078 Valid Accounts Adversaries may obtain and abuse credentials of existing accounts as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion. Compromised credentials may be used to bypass access controls placed on various resources on systems within the network and may even be used for persistent access to remote systems and externally available services, such as VPNs, Outlook Web Access, network devices, and remote desktop. Compromised credentials may also grant an adversary increased privilege to specific systems or access to restricted areas of the network. Adversaries may choose not to use malware or tools in conjunction with the legitimate access those credentials provide to make it harder to detect their presence. .001 Default Accounts Adversaries may obtain and abuse credentials of a default account as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion. Default accounts are those that are built-into an OS, such as the Guest or Administrator accounts on Windows systems. Default accounts also include default factory/provider set accounts on other types of systems, software, or devices, including the root user account in AWS, the root user account in ESXi, and the default service account in Kubernetes. .002 Domain Accounts Adversaries may obtain and abuse credentials of a domain account as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion. Domain accounts are those managed by Active Directory Domain Services where access and permissions are configured across systems and services that are part of that domain. Domain accounts can cover users, administrators, and services. .003 Local Accounts Adversaries may obtain and abuse credentials of a local account as a means of gaining Initial Access, Persistence, Privilege Escalation, or Defense Evasion. Local accounts are those configured by an organization for use by users, remote support, services, or for administration on a single system or service. .004 Cloud Accounts Valid accounts in cloud environments may allow adversaries to perform actions to achieve Initial Access, Persistence, Privilege Escalation, or Defense Evasion. Cloud accounts are those created and configured by an organization for use by users, remote support, services, or for administration of resources within a cloud service provider or SaaS application. Cloud Accounts can exist solely in the cloud; alternatively, they may be hybrid-joined between on-premises systems and the cloud through syncing or federation with other identity sources such

as Windows Active Directory. T1497 Virtualization/Sandbox Evasion Adversaries may employ various means to detect and avoid virtualization and analysis environments. This may include changing behaviors based on the results of checks for the presence of artifacts indicative of a virtual machine environment (VME) or sandbox. If the adversary detects a VME, they may alter their malware to disengage from the victim or conceal the core functions of the implant. They may also search for VME artifacts before dropping secondary or additional payloads. Adversaries may use the information learned from Virtualization/Sandbox Evasion during automated discovery to shape follow-on behaviors. .001 System Checks Adversaries may employ various system checks to detect and avoid virtualization and analysis environments. This may include changing behaviors based on the results of checks for the presence of artifacts indicative of a virtual machine environment (VME) or sandbox. If the adversary detects a VME, they may alter their malware to disengage from the victim or conceal the core functions of the implant. They may also search for VME artifacts before dropping secondary or additional payloads. Adversaries may use the information learned from Virtualization/Sandbox Evasion during automated discovery to shape follow-on behaviors. .002 User Activity Based Checks Adversaries may employ various user activity checks to detect and avoid virtualization and analysis environments. This may include changing behaviors based on the results of checks for the presence of artifacts indicative of a virtual machine environment (VME) or sandbox. If the adversary detects a VME, they may alter their malware to disengage from the victim or conceal the core functions of the implant. They may also search for VME artifacts before dropping secondary or additional payloads. Adversaries may use the information learned from Virtualization/Sandbox Evasion during automated discovery to shape follow-on behaviors. .003 Time Based Checks Adversaries may employ various time-based methods to detect virtualization and analysis environments, particularly those that attempt to manipulate time mechanisms to simulate longer elapses of time. This may include enumerating time-based properties, such as uptime or the system clock. T1600 Weaken Encryption Adversaries may compromise a network device’s encryption capability in order to bypass encryption that would otherwise protect data communications. .001 Reduce Key Space Adversaries may reduce the level of effort required to decrypt data transmitted over the network by reducing the cipher strength of encrypted communications. .002 Disable Crypto Hardware Adversaries disable a network device’s dedicated hardware encryption, which may enable them to leverage weaknesses in software encryption in order to reduce the effort involved in collecting, manipulating, and exfiltrating transmitted data. T1220 XSL Script Processing Adversaries may bypass application control and obscure execution of code by embedding scripts inside XSL files. Extensible Stylesheet Language (XSL) files are commonly used to describe the processing and rendering of data within XML files. To support complex operations, the XSL standard includes support for embedded scripting in various languages.
