Persistence

From enterprise

Tactic Description

Persistence is any access, action, or configuration change to a system that gives an adversary a persistent presence on that system. Adversaries will often need to maintain access to systems through interruptions such as system restarts, loss of credentials, or other failures that would require a remote access tool to restart or alternate backdoor for them to regain access.

Techniques

Below is a list of all the Persistence techniques in enterprise:

Name	Tactics	Technical Description
.bash_profile and .bashrc	**Persistence**	`~/.bash_profile` and `~/.bashrc` are executed in a user's context when a new shell opens or when a user logs in so that their environment is set correctly. `~/.bash_profile` is executed for login shells and `~/.bashrc` is executed for interactive non-login shells. This means that when a user logs in (via username and password) to the console (either locally or remotely via something like SSH), `~/.bash_profile` is executed before the initial command prompt is returned to the user. After that, every time a new shell is opened, `~/.bashrc` is executed. This allows users more fine grained control over when they want certain commands executed. Mac's Terminal.app is a little different in that it runs a login shell by default each time a new terminal window is opened, thus calling `~/.bash_profile` each time instead of `~/.bashrc`. These files are meant to be written to by the local user to configure their own environment; however, adversaries can also insert code into these files to gain persistence each time a user logs in or opens a new shell.

Name	Tactics	Technical Description
Accessibility Features	**Persistence** Privilege Escalation	Windows contains accessibility features that may be launched with a key combination before a user has logged in (for example, when the user is on the Windows logon screen). An adversary can modify the way these programs are launched to get a command prompt or backdoor without logging in to the system. Two common accessibility programs are `C:\Windows\System32\sethc.exe`, launched when the shift key is pressed five times and `C:\Windows\System32\utilman.exe`, launched when the Windows + U key combination is pressed. The sethc.exe program is often referred to as "sticky keys", and has been used by adversaries for unauthenticated access through a remote desktop login screen.[1] Depending on the version of Windows, an adversary may take advantage of these features in different ways because of code integrity enhancements. In newer versions of Windows, the replaced binary needs to be digitally signed for x64 systems, the binary must reside in `%systemdir%\`, and it must be protected by Windows File or Resource Protection (WFP/WRP).[2] The debugger method was likely discovered as a potential workaround because it does not require the corresponding accessibility feature binary to be replaced. Examples for both methods: For simple binary replacement on Windows XP and later as well as and Windows Server 2003/R2 and later, for example, the program (e.g., `C:\Windows\System32\utilman.exe`) may be replaced with "cmd.exe" (or another program that provides backdoor access). Subsequently, pressing the appropriate key combination at the login screen while sitting at the keyboard or when connected over Remote Desktop Protocol will cause the replaced file to be executed with SYSTEM privileges.[3] For the debugger method on Windows Vista and later as well as Windows Server 2008 and later, for example, a Registry key may be modified that configures "cmd.exe," or another program that provides backdoor access, as a "debugger" for the accessibility program (e.g., "utilman.exe"). After the Registry is modified, pressing the appropriate key combination at the login screen while at the keyboard or when connected with RDP will cause the "debugger" program to be executed with SYSTEM privileges.[3] Other accessibility features exist that may also be leveraged in a similar fashion:[2] ▪ On-Screen Keyboard: `C:\Windows\System32\osk.exe` ▪ Magnifier: `C:\Windows\System32\Magnify.exe` ▪ Narrator: `C:\Windows\System32\Narrator.exe` ▪ Display Switcher: `C:\Windows\System32\DisplaySwitch.exe` ▪ App Switcher: `C:\Windows\System32\AtBroker.exe`

Name	Tactics	Technical Description
AppCert DLLs	**Persistence** Privilege Escalation	Dynamic-link libraries (DLLs) that are specified in the AppCertDLLs value in the Registry key `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager` are loaded into every process that calls the ubiquitously used application programming interface (API) functions:[4] • CreateProcess • CreateProcessAsUser • CreateProcessWithLoginW • CreateProcessWithTokenW • WinExec Similar to Process Injection, this value can be abused to obtain persistence and privilege escalation by causing a malicious DLL to be loaded and run in the context of separate processes on the computer.
AppInit DLLs	**Persistence** Privilege Escalation	Dynamic-link libraries (DLLs) that are specified in the AppInit_DLLs value in the Registry keys `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows` or `HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows` are loaded by user32.dll into every process that loads user32.dll. In practice this is nearly every program, since user32.dll is a very common library.[4] Similar to Process Injection, these values can be abused to obtain persistence and privilege escalation by causing a malicious DLL to be loaded and run in the context of separate processes on the computer.[5] The AppInit DLL functionality is disabled in Windows 8 and later versions when secure boot is enabled.[6]

Name	Tactics	Technical Description
Application Shimming	**Persistence** Privilege Escalation	The Microsoft Windows Application Compatibility Infrastructure/Framework (Application Shim) was created to allow backward compatibility of programs as Windows updates and changes its code. For example, the application shimming feature allows developers to apply fixes to applications (without rewriting code) that were created for Windows XP so that it will work with Windows 10.[4] Within the framework, shims are created to act as a buffer between the program (or more specifically, the Import Address Table) and the Windows OS. When a program is executed, the shim cache is referenced to determine if the program requires the use of the shim database (.sdb). If so, the shim database uses Hooking to redirect the code as necessary in order to communicate with the OS. A list of all shims currently installed by the default Windows installer (sdbinst.exe) is kept in: ▪ `%WINDIR%\AppPatch\sysmain.sdb` ▪ `hklm\software\microsoft\windows nt\currentversion\appcompatflags\installedsdb` Custom databases are stored in: ▪ `%WINDIR%\AppPatch\custom & %WINDIR%\AppPatch\AppPatch64\Custom` ▪ `hklm\software\microsoft\windows nt\currentversion\appcompatflags\custom` To keep shims secure, Windows designed them to run in user mode so they cannot modify the kernel and you must have administrator privileges to install a shim. However, certain shims can be used to Bypass User Account Control (UAC) (RedirectEXE), inject DLLs into processes (InjectDLL), disable Data Execution Prevention (DisableNX) and Structure Exception Handling (DisableSEH), and intercept memory addresses (GetProcAddress). Similar to Hooking, utilizing these shims may allow an adversary to perform several malicious acts such as elevate privileges, install backdoors, disable defenses like Windows Defender, etc.
Authentication Package	**Persistence**	Windows Authentication Package DLLs are loaded by the Local Security Authority (LSA) process at system start. They provide support for multiple logon processes and multiple security protocols to the operating system.[7] Adversaries can use the autostart mechanism provided by LSA Authentication Packages for persistence by placing a reference to a binary in the Windows Registry location `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\` with the key value of `"Authentication Packages"=<target binary>`. The binary will then be executed by the system when the authentication packages are loaded.

Name	Tactics	Technical Description
Bootkit	**Persistence**	A bootkit is a malware variant that modifies the boot sectors of a hard drive, including the Master Boot Record (MBR) and Volume Boot Record (VBR).[8] Adversaries may use bootkits to persist on systems at a layer below the operating system, which may make it difficult to perform full remediation unless an organization suspects one was used and can act accordingly. **Master Boot Record** The MBR is the section of disk that is first loaded after completing hardware initialization by the BIOS. It is the location of the boot loader. An adversary who has raw access to the boot drive may overwrite this area, diverting execution during startup from the normal boot loader to adversary code.[9] **Volume Boot Record** The MBR passes control of the boot process to the VBR. Similar to the case of MBR, an adversary who has raw access to the boot drive may overwrite the VBR to divert execution during startup to adversary code.
Browser Extensions	Collection **Persistence**	Browser extensions or plugins are small programs that can add functionality and customize aspects of internet browsers. They can be installed directly or through a browser's app store. Extensions generally have access and permissions to everything that the browser can access.[10][11] Malicious extensions can be installed into a browser through malicious app store downloads masquerading as legitimate extensions, through social engineering, or by an adversary that has already compromised a system. Security can be limited on browser app stores so may not be difficult for malicious extensions to defeat automated scanners and be uploaded.[12] Once the extension is installed, it can browse to websites in the background,[13][14] steal all information that a user enters into a browser, to include credentials,[15][16] and be used as an installer for a RAT for **persistence**. There have been instances of botnets using a persistent backdoor through malicious Chrome extensions.[17] There have also been similar examples of extensions being used for command & control [18].

Name	Tactics	Technical Description
Change Default File Association	**Persistence**	When a file is opened, the default program used to open the file (also called the file association or handler) is checked. File association selections are stored in the Windows Registry and can be edited by users, administrators, or programs that have Registry access.[19][20] Applications can modify the file association for a given file extension to call an arbitrary program when a file with the given extension is opened. System file associations are listed under `HKEY_CLASSES_ROOT\.[extension]`, for example `HKEY_CLASSES_ROOT\.txt`. The entries point to a handler for that extension located at `HKEY_CLASSES_ROOT\[handler]`. The various commands are then listed as subkeys underneath the shell key at `HKEY_CLASSES_ROOT\[handler]\shell\[action]\command`. For example: ▪ `HKEY_CLASSES_ROOT\txtfile\shell\open\command` ▪ `HKEY_CLASSES_ROOT\txtfile\shell\print\command` ▪ `HKEY_CLASSES_ROOT\txtfile\shell\printto\command` The values of the keys listed are commands that are executed when the handler opens the file extension. Adversaries can modify these values to execute arbitrary commands.
Component Firmware	Defense Evasion **Persistence**	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 that may not have the same capability or level of integrity checking. Malicious device firmware could provide both a persistent level of access to systems despite potential typical failures to maintain access and hard disk re-images, as well as a way to evade host software-based defenses and integrity checks.
Component Object Model Hijacking	Defense Evasion **Persistence**	The Microsoft Component Object Model (COM) is a system within Windows to enable interaction between software components through the operating system.[21] Adversaries can use this system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence. Hijacking a COM object requires a change in the Windows Registry to replace a reference to a legitimate system component which may cause that component to not work when executed. When that system component is executed through normal system operation the adversary's code will be executed instead.[22] An adversary is likely to hijack objects that are used frequently enough to maintain a consistent level of persistence, but are unlikely to break noticeable functionality within the system as to avoid system instability that could lead to detection.
Create Account	**Persistence**	Adversaries with a sufficient level of access may create a local system or domain account. Such accounts may be used for persistence that do not require persistent remote access tools to be deployed on the system. The `net user` commands can be used to create a local or domain account.

Name	Tactics	Technical Description
DLL Search Order Hijacking	Defense Evasion **Persistence** Privilege Escalation	Windows systems use a common method to look for required DLLs to load into a program.[23] Adversaries may take advantage of the Windows DLL search order and programs that ambiguously specify DLLs to gain privilege escalation and persistence. Adversaries may perform DLL preloading, also called binary planting attacks, [24] by placing a malicious DLL with the same name as an ambiguously specified DLL in a location that Windows searches before the legitimate DLL. Often this location is the current working directory of the program. Remote DLL preloading attacks occur when a program sets its current directory to a remote location such as a Web share before loading a DLL.[25] Adversaries may use this behavior to cause the program to load a malicious DLL. Adversaries may also directly modify the way a program loads DLLs by replacing an existing DLL or modifying a .manifest or .local redirection file, directory, or junction to cause the program to load a different DLL to maintain persistence or privilege escalation.[26][27][28] If a search order-vulnerable program is configured to run at a higher privilege level, then the adversary-controlled DLL that is loaded will also be executed at the higher level. In this case, the technique could be used for privilege escalation from user to administrator or SYSTEM or from administrator to SYSTEM, depending on the program. Programs that fall victim to path hijacking may appear to behave normally because malicious DLLs may be configured to also load the legitimate DLLs they were meant to replace.
Dylib Hijacking	**Persistence** Privilege Escalation	macOS and OS X use a common method to look for required dynamic libraries (dylib) to load into a program based on search paths. Adversaries can take advantage of ambiguous paths to plant dylibs to gain privilege escalation or persistence. A common method is to see what dylibs an application uses, then plant a malicious version with the same name higher up in the search path. This typically results in the dylib being in the same folder as the application itself. [29][30] If the program is configured to run at a higher privilege level than the current user, then when the dylib is loaded into the application, the dylib will also run at that elevated level. This can be used by adversaries as a privilege escalation technique.
External Remote Services	**Persistence**	Remote services such as VPNs, Citrix, and other access mechanisms allow users to connect to internal enterprise network resources from external locations. There are often remote service gateways that manage connections and credential authentication for these services. Services such as Windows Remote Management can also be used externally. Adversaries may use remote services to access and persist within a network.[31] Access to Valid Accounts to use the service is often a requirement, which could be obtained through credential pharming or by obtaining the credentials from users after compromising the enterprise network. Access to remote services may be used as part of Redundant Access during an operation.

Name	Tactics	Technical Description
File System Permissions Weakness	**Persistence** Privilege Escalation	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. Adversaries may use this technique to replace legitimate binaries with malicious ones as a means of executing code at a higher permissions level. If the executing process is set to run at a specific time or during a certain event (e.g., system bootup) then this technique can also be used for persistence. **Services** Manipulation of Windows service binaries is one variation of this technique. Adversaries may replace a legitimate service executable with their own executable to gain persistence and/or privilege escalation to the account context the service is set to execute under (local/domain account, SYSTEM, LocalService, or NetworkService). Once the service is started, either directly by the user (if appropriate access is available) or through some other means, such as a system restart if the service starts on bootup, the replaced executable will run instead of the original service executable. **Executable Installers** Another variation of this technique can be performed by taking advantage of a weakness that is common in executable, self-extracting installers. During the installation process, it is common for installers to use a subdirectory within the `%TEMP%` directory to unpack binaries such as DLLs, EXEs, or other payloads. When installers create subdirectories and files they often do not set appropriate permissions to restrict write access, which allows for execution of untrusted code placed in the subdirectories or overwriting of binaries used in the installation process. This behavior is related to and may take advantage of DLL Search Order Hijacking. Some installers may also require elevated privileges that will result in privilege escalation when executing adversary controlled code. This behavior is related to Bypass User Account Control. Several examples of this weakness in existing common installers have been reported to software vendors.[32][33]

Name	Tactics	Technical Description
Hidden Files and Directories	Defense Evasion **Persistence**	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). **Windows** Users can mark specific files as hidden by using the attrib.exe binary. Simply do `attrib +h filename` to mark a file or folder as hidden. Similarly, the "+s" marks a file as a system file and the "+r" flag marks the file as read only. Like most windows binaries, the attrib.exe binary provides the ability to apply these changes recursively "/S". **Linux/Mac** Users can mark specific files as hidden simply by putting a "." as the first character in the file or folder name [34][35]. Files and folder that start with a period, '.', are by default hidden from being viewed in the Finder application and standard command-line utilities like "ls". Users must specifically change settings to have these files viewable. For command line usages, there is typically a flag to see all files (including hidden ones). To view these files in the Finder Application, the following command must be executed: `defaults write com.apple.finder AppleShowAllFiles YES`, and then relaunch the Finder Application. **Mac** Files on macOS can be marked with the UF_HIDDEN flag which prevents them from being seen in Finder.app, but still allows them to be seen in Terminal.app[36]. Many applications create these hidden files and folders to store information so that it doesn't clutter up the user's workspace. For example, SSH utilities create a .ssh folder that's hidden and contains the user's known hosts and keys. Adversaries can use this to their advantage to hide files and folders anywhere on the system for persistence and evading a typical user or system analysis that does not incorporate investigation of hidden files.

Name	Tactics	Technical Description
Hooking	Credential Access **Persistence** Privilege Escalation	Windows processes often leverage application programming interface (API) functions to perform tasks that require reusable system resources. Windows API functions are typically stored in dynamic-link libraries (DLLs) as exported functions. Hooking involves redirecting calls to these functions and can be implemented via: ▪ **Hooks procedures**, which intercept and execute designated code in response to events such as messages, keystrokes, and mouse inputs.[37][4] ▪ **Import address table (IAT) hooking**, which use modifications to a process's IAT, where pointers to imported API functions are stored.[4][38][39] ▪ **Inline hooking**, which overwrites the first bytes in an API function to redirect code flow.[4][40][39] Similar to Process Injection, adversaries may use hooking to load and execute malicious code within the context of another process, masking the execution while also allowing access to the process's memory and possibly elevated privileges. Installing hooking mechanisms may also provide **Persistence** via continuous invocation when the functions are called through normal use. Malicious hooking mechanisms may also capture API calls that include parameters that reveal user authentication credentials for Credential Access.[41] Hooking is commonly utilized by Rootkits to conceal files, processes, Registry keys, and other objects in order to hide malware and associated behaviors.[42]
Hypervisor	**Persistence**	A type-1 hypervisor is a software layer that sits between the guest operating systems and system's hardware.[43] It presents a virtual running environment to an operating system. An example of a common hypervisor is Xen.[44] A type-1 hypervisor operates at a level below the operating system and could be designed with Rootkit functionality to hide its existence from the guest operating system.[45] A malicious hypervisor of this nature could be used to persist on systems through interruption.

Name	Tactics	Technical Description
Image File Execution Options Injection	Defense Evasion **Persistence** Privilege Escalation	Image File Execution Options (IFEO) enable a developer to attach a debugger to an application. When a process is created, any executable file present in an application's IFEO will be prepended to the application's name, effectively launching the new process under the debugger (e.g., "C:\dbg\ntsd.exe -g notepad.exe").[46] IFEOs can be set directly via the Registry or in Global Flags via the Gflags tool.[47] IFEOs are represented as Debugger Values in the Registry under `HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options/<executable>` and `HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<executable>` where `<executable>` is the binary on which the debugger is attached.[46] Similar to Process Injection, this value can be abused to obtain persistence and privilege escalation by causing a malicious executable to be loaded and run in the context of separate processes on the computer.[4] Installing IFEO mechanisms may also provide **Persistence** via continuous invocation. Malware may also use IFEO for Defense Evasion by registering invalid debuggers that redirect and effectively disable various system and security applications.[48][49]
LC_LOAD_DYLIB Addition	**Persistence**	Mach-O binaries have a series of headers that are used to perform certain operations when a binary is loaded. The LC_LOAD_DYLIB header in a Mach-O binary tells macOS and OS X which dynamic libraries (dylibs) to load during execution time. These can be added ad-hoc to the compiled binary as long adjustments are made to the rest of the fields and dependencies[29]. There are tools available to perform these changes. Any changes will invalidate digital signatures on binaries because the binary is being modified. Adversaries can remediate this issue by simply removing the LC_CODE_SIGNATURE command from the binary so that the signature isn't checked at load time[30].
LSASS Driver	Execution **Persistence**	The Windows security subsystem is a set of components that manage and enforce the security policy for a computer or domain. The Local Security Authority (LSA) is the main component responsible for local security policy and user authentication. The LSA includes multiple dynamic link libraries (DLLs) associated with various other security functions, all of which run in the context of the LSA Subsystem Service (LSASS) lsass.exe process.[50] Adversaries may target lsass.exe drivers to obtain execution and/or persistence. By either replacing or adding illegitimate drivers (e.g., DLL Side-Loading or DLL Search Order Hijacking), an adversary can achieve arbitrary code execution triggered by continuous LSA operations.

Name	Tactics	Technical Description
Launch Agent	**Persistence**	Per Apple’s developer documentation, when a user logs in, a per-user launchd process is started which loads the parameters for each launch-on-demand user agent from the property list (plist) files found in `/System/Library/LaunchAgents`, `/Library/LaunchAgents`, and `$HOME/Library/LaunchAgents`[51][52][35]. These launch agents have property list files which point to the executables that will be launched[53]. Adversaries may install a new launch agent that can be configured to execute at login by using launchd or launchctl to load a plist into the appropriate directories [34] [54]. The agent name may be disguised by using a name from a related operating system or benign software. Launch Agents are created with user level privileges and are executed with the privileges of the user when they log in[55] [56]. They can be set up to execute when a specific user logs in (in the specific user’s directory structure) or when any user logs in (which requires administrator privileges).
Launch Daemon	**Persistence** Privilege Escalation	Per Apple’s developer documentation, when macOS and OS X boot up, launchd is run to finish system initialization. This process loads the parameters for each launch-on-demand system-level daemon from the property list (plist) files found in `/System/Library/LaunchDaemons` and `/Library/LaunchDaemons`[51]. These LaunchDaemons have property list files which point to the executables that will be launched[54]. Adversaries may install a new launch daemon that can be configured to execute at startup by using launchd or launchctl to load a plist into the appropriate directories[55]. The daemon name may be disguised by using a name from a related operating system or benign software [36]. Launch Daemons may be created with administrator privileges, but are executed under root privileges, so an adversary may also use a service to escalate privileges from administrator to root. The plist file permissions must be root:wheel, but the script or program that it points to has no such requirement. So, it is possible for poor configurations to allow an adversary to modify a current Launch Daemon’s executable and gain persistence or Privilege Escalation.
Launchctl	Defense Evasion Execution **Persistence**	Launchctl controls the macOS launchd process which handles things like launch agents and launch daemons, but can execute other commands or programs itself. Launchctl supports taking subcommands on the command-line, interactively, or even redirected from standard input. By loading or reloading launch agents or launch daemons, adversaries can install persistence or execute changes they made [34]. Running a command from launchctl is as simple as `launchctl submit -l <labelName> -- /Path/to/thing/to/execute "arg" "arg" "arg"`. Loading, unloading, or reloading launch agents or launch daemons can require elevated privileges. Adversaries can abuse this functionality to execute code or even bypass whitelisting if launchctl is an allowed process.

Name	Tactics	Technical Description
Local Job Scheduling	**Persistence** Execution	On Linux and Apple systems, multiple methods are supported for creating pre-scheduled and periodic background jobs: cron,[57] at,[58] and launchd.[59] Unlike Scheduled Task on Windows systems, job scheduling on Linux-based systems cannot be done remotely unless used in conjunction within an established remote session, like secure shell (SSH). **cron** System-wide cron jobs are installed by modifying `/etc/crontab` file, `/etc/cron.d/` directory or other locations supported by the Cron daemon, while per-user cron jobs are installed using crontab with specifically formatted crontab files.[59] This works on Mac and Linux systems. Those methods allow for commands or scripts to be executed at specific, periodic intervals in the background without user interaction. An adversary may use job scheduling to execute programs at system startup or on a scheduled basis for **Persistence**,[60][54][30][61] to conduct Execution as part of Lateral Movement, to gain root privileges, or to run a process under the context of a specific account. **at** The at program is another means on Linux-based systems, including Mac, to schedule a program or script job for execution at a later date and/or time, which could also be used for the same purposes. **launchd** Each launchd job is described by a different configuration property list (plist) file similar to Launch Daemon or Launch Agent, except there is an additional key called `StartCalendarInterval` with a dictionary of time values.[59] This only works on macOS and OS X.
Login Item	**Persistence**	MacOS provides the option to list specific applications to run when a user logs in. These applications run under the logged in user's context, and will be started every time the user logs in. Login items installed using the Service Management Framework are not visible in the System Preferences and can only be removed by the application that created them[62]. Users have direct control over login items installed using a shared file list which are also visible in System Preferences[62]. These login items are stored in the user's `~/Library/Preferences/` directory in a plist file called `com.apple.loginitems.plist`[54]. Some of these applications can open visible dialogs to the user, but they don’t all have to since there is an option to ‘Hide’ the window. If an adversary can register their own login item or modified an existing one, then they can use it to execute their code for a persistence mechanism each time the user logs in[30][53].

Name	Tactics	Technical Description
Logon Scripts	Lateral Movement **Persistence**	===Windows=== Windows allows logon scripts to be run whenever a specific user or group of users log into a system.[63] The scripts can be used to perform administrative functions, which may often execute other programs or send information to an internal logging server. If adversaries can access these scripts, they may insert additional code into the logon script to execute their tools when a user logs in. This code can allow them to maintain persistence on a single system, if it is a local script, or to move laterally within a network, if the script is stored on a central server and pushed to many systems. Depending on the access configuration of the logon scripts, either local credentials or an administrator account may be necessary. **Mac** Mac allows login and logoff hooks to be run as root whenever a specific user logs into or out of a system. A login hook tells Mac OS X to execute a certain script when a user logs in, but unlike startup items, a login hook executes as root[64]. There can only be one login hook at a time though. If adversaries can access these scripts, they can insert additional code to the script to execute their tools when a user logs in.
Modify Existing Service	**Persistence**	Windows service configuration information, including the file path to the service's executable, is stored in the Registry. Service configurations can be modified using utilities such as sc.exe and Reg. Adversaries can modify an existing service to persist malware on a system by using system utilities or by using custom tools to interact with the Windows API. Use of existing services is a type of Masquerading that may make detection analysis more challenging. Modifying existing services may interrupt their functionality or may enable services that are disabled or otherwise not commonly used.
Netsh Helper DLL	**Persistence**	Netsh.exe (also referred to as Netshell) is a command-line scripting utility used to interact with the network configuration of a system. It contains functionality to add helper DLLs for extending functionality of the utility.[65] The paths to registered netsh.exe helper DLLs are entered into the Windows Registry at `HKLM\SOFTWARE\Microsoft\Netsh`. Adversaries can use netsh.exe with helper DLLs to proxy execution of arbitrary code in a persistent manner when netsh.exe is executed automatically with another **Persistence** technique or if other persistent software is present on the system that executes netsh.exe as part of its normal functionality. Examples include some VPN software that invoke netsh.exe.[66] Proof of concept code exists to load Cobalt Strike's payload using netsh.exe helper DLLs.[67]

Name	Tactics	Technical Description
New Service	**Persistence** Privilege Escalation	When operating systems boot up, they can start programs or applications called services that perform background system functions.[68] A service's configuration information, including the file path to the service's executable, is stored in the Windows Registry. Adversaries may install a new service that can be configured to execute at startup by using utilities to interact with services or by directly modifying the Registry. The service name may be disguised by using a name from a related operating system or benign software with Masquerading. Services may be created with administrator privileges but are executed under SYSTEM privileges, so an adversary may also use a service to escalate privileges from administrator to SYSTEM. Adversaries may also directly start services through Service Execution.

Name	Tactics	Technical Description
Office Application Startup	**Persistence**	Microsoft Office is a fairly common application suite on Windows-based operating systems within an enterprise network. There are multiple mechanisms that can be used with Office for persistence when an Office-based application is started. **Office Template Macros** Microsoft Office contains templates that are part of common Office applications and are used to customize styles. The base templates within the application are used each time an application starts.[69] Office Visual Basic for Applications (VBA) macros[70] can inserted into the base templated and used to execute code when the respective Office application starts in order to obtain persistence. Examples for both Word and Excel have been discovered and published. By default, Word has a Normal.dotm template created that can be modified to include a malicious macro. Excel does not have a template file created by default, but one can be added that will automatically be loaded.[71][72] Word Normal.dotm location:`C:\Users\(username)\AppData\Roaming\Microsoft\Templates\Normal.dotm` Excel Personal.xlsb location:`C:\Users\(username)\AppData\Roaming\Microsoft\Excel\XLSTART\PERSONAL.XLSB` An adversary may need to enable macros to execute unrestricted depending on the system or enterprise security policy on use of macros. **Office Test** A Registry location was found that when a DLL reference was placed within it the corresponding DLL pointed to by the binary path would be executed every time an Office application is started[73] `HKEY_CURRENT_USER\Software\Microsoft\Office test\Special\Perf` **Add-ins** Office add-ins can be used to add functionality to Office programs.[74] Add-ins can also be used to obtain persistence because they can be set to execute code when an Office application starts. There are different types of add-ins that can be used by the various Office products; including Word/Excel add-in Libraries (WLL/XLL), VBA add-ins, Office Component Object Model (COM) add-ins, automation add-ins, VBA Editor (VBE), and Visual Studio Tools for Office (VSTO) add-ins.[75]
Path Interception	**Persistence** Privilege Escalation	Path interception occurs when an executable is placed in a specific path so that it is executed by an application instead of the intended target. One example of this was the use of a copy of cmd in the current working directory of a vulnerable application that loads a CMD or BAT file with the CreateProcess function.[76]

Name	Tactics	Technical Description
		There are multiple distinct weaknesses or misconfigurations that adversaries may take advantage of when performing path interception: unquoted paths, path environment variable misconfigurations, and search order hijacking. The first vulnerability deals with full program paths, while the second and third occur when program paths are not specified. These techniques can be used for persistence if executables are called on a regular basis, as well as privilege escalation if intercepted executables are started by a higher privileged process. **Unquoted Paths** Service paths (stored in Windows Registry keys)[77] and shortcut paths are vulnerable to path interception if the path has one or more spaces and is not surrounded by quotation marks (e.g., `C:\unsafe path with space\program.exe` vs. `"C:\safe path with space\program.exe"`).[78] An adversary can place an executable in a higher level directory of the path, and Windows will resolve that executable instead of the intended executable. For example, if the path in a shortcut is `C:\program files\myapp.exe`, an adversary may create a program at `C:\program.exe` that will be run instead of the intended program. **PATH Environment Variable Misconfiguration** The PATH environment variable contains a list of directories. Certain methods of executing a program (namely using cmd.exe or the command-line) rely solely on the PATH environment variable to determine the locations that are searched for a program when the path for the program is not given. If any directories are listed in the PATH environment variable before the Windows directory, `%SystemRoot%\system32` (e.g., `C:\Windows\system32`), a program may be placed in the preceding directory that is named the same as a Windows program (such as cmd, PowerShell, or Python), which will be executed when that command is executed from a script or command-line. For example, if `C:\example path` precedes `C:\Windows\system32` is in the PATH environment variable, a program that is named net.exe and placed in `C:\example path` will be called instead of the Windows system "net" when "net" is executed from the command-line. **Search Order Hijacking** Search order hijacking occurs when an adversary abuses the order in which Windows searches for programs that are not given a path. The search order differs depending on the method that is used to execute the program.[79][80][81] However, it is common for Windows to search in the directory of the initiating program before searching through the Windows system directory. An adversary who finds a program vulnerable to search order hijacking (i.e., a program that does not specify the path to an executable) may take advantage of this vulnerability by creating a program named after the improperly specified program and placing it within the initiating program's directory.

Name	Tactics	Technical Description
		For example, "example.exe" runs "cmd.exe" with the command-line argument `net user`. An adversary may place a program called "net.exe" within the same directory as example.exe, "net.exe" will be run instead of the Windows system utility net. In addition, if an adversary places a program called "net.com" in the same directory as "net.exe", then `cmd.exe /C net user` will execute "net.com" instead of "net.exe" due to the order of executable extensions defined under PATHEXT.[82] Search order hijacking is also a common practice for hijacking DLL loads and is covered in DLL Search Order Hijacking.
Plist Modification	Defense Evasion **Persistence** Privilege Escalation	Property list (plist) files contain all of the information that macOS and OS X uses to configure applications and services. These files are UT-8 encoded and formatted like XML documents via a series of keys surrounded by < >. They detail when programs should execute, file paths to the executables, program arguments, required OS permissions, and many others. plists are located in certain locations depending on their purpose such as `/Library/Preferences` (which execute with elevated privileges) and `~/Library/Preferences` (which execute with a user's privileges). Adversaries can modify these plist files to point to their own code, can use them to execute their code in the context of another user, bypass whitelisting procedures, or even use them as a persistence mechanism.[34]
Port Monitors	**Persistence** Privilege Escalation	A port monitor can be set through the AddMonitor API call to set a DLL to be loaded at startup.[83] This DLL can be located in `C:\Windows\System32` and will be loaded by the print spooler service, spoolsv.exe, on boot. The spoolsv.exe process also runs under SYSTEM level permissions.[84] Alternatively, an arbitrary DLL can be loaded if permissions allow writing a fully-qualified pathname for that DLL to `HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors`. The Registry key contains entries for the following: ▪ Local Port ▪ Standard TCP/IP Port ▪ USB Monitor ▪ WSD Port Adversaries can use this technique to load malicious code at startup that will persist on system reboot and execute as SYSTEM.
Rc.common	**Persistence**	During the boot process, macOS and Linux both execute `source /etc/rc.common`, which is a shell script containing various utility functions. This file also defines routines for processing command-line arguments and for gathering system settings, and is thus recommended to include in the start of Startup Item Scripts[85]. In macOS and OS X, this is now a deprecated technique in favor of launch agents and launch daemons, but is currently still used. Adversaries can use the rc.common file as a way to hide code for persistence that will execute on each reboot as the root user[54].

Name	Tactics	Technical Description
Re-opened Applications	**Persistence**	Starting in Mac OS X 10.7 (Lion), users can specify certain applications to be re-opened when a user reboots their machine. While this is usually done via a Graphical User Interface (GUI) on an app-by-app basis, there are property list files (plist) that contain this information as well located at `~/Library/Preferences/com.apple.loginwindow.plist` and `~/Library/Preferences/ByHost/com.apple.loginwindow.*.plist`. An adversary can modify one of these files directly to include a link to their malicious executable to provide a persistence mechanism each time the user reboots their machine[54].
Redundant Access	Defense Evasion **Persistence**	Adversaries may use more than one remote access tool with varying command and control protocols as a hedge against detection. If one type of tool is detected and blocked or removed as a response but the organization did not gain a full understanding of the adversary's tools and access, then the adversary will be able to retain access to the network. Adversaries may also attempt to gain access to Valid Accounts to use External Remote Services such as external VPNs as a way to maintain access despite interruptions to remote access tools deployed within a target network.[86] Use of a Web Shell is one such way to maintain access to a network through an externally accessible Web server.
Registry Run Keys / Start Folder	**Persistence**	Adding an entry to the "run keys" in the Registry or startup folder will cause the program referenced to be executed when a user logs in.[87] The program will be executed under the context of the user and will have the account's associated permissions level. Adversaries can use these configuration locations to execute malware, such as remote access tools, to maintain persistence through system reboots. Adversaries may also use Masquerading to make the Registry entries look as if they are associated with legitimate programs.
Scheduled Task	Execution **Persistence** Privilege Escalation	Utilities such as at and schtasks, along with the Windows Task Scheduler, can be used to schedule programs or scripts to be executed at a date and time. A task can also be scheduled on a remote system, provided the proper authentication is met to use RPC and file and printer sharing is turned on. Scheduling a task on a remote system typically required being a member of the Administrators group on the the remote system.[88] An adversary may use task scheduling to execute programs at system startup or on a scheduled basis for persistence, to conduct remote Execution as part of Lateral Movement, to gain SYSTEM privileges, or to run a process under the context of a specified account.
Screensaver	**Persistence**	Screensavers are programs that execute after a configurable time of user inactivity and consist of Portable Executable (PE) files with a .scr file extension.[89] The Windows screensaver application scrnsave.exe is located in `C:\Windows\System32\` along with screensavers included with base Windows installations. The following screensaver settings are stored in the Registry (`HKCU\Control Panel\Desktop\`) and could be manipulated to achieve persistence: ▪ `SCRNSAVE.exe` - set to malicious PE path ▪ `ScreenSaveActive` - set to '1' to enable the screensaver ▪ `ScreenSaverIsSecure` - set to '0' to not require a password to unlock ▪ `ScreenSaverTimeout` - sets user inactivity timeout before screensaver is executed Adversaries can use screensaver settings to maintain persistence by setting the screensaver to run malware after a certain timeframe of user inactivity.[90]

Name	Tactics	Technical Description
Security Support Provider	**Persistence**	Windows Security Support Provider (SSP) DLLs are loaded into the Local Security Authority (LSA) process at system start. Once loaded into the LSA, SSP DLLs have access to encrypted and plaintext passwords that are stored in Windows, such as any logged-on user's Domain password or smart card PINs. The SSP configuration is stored in two Registry keys: `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages` and `HKLM\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\Security Packages`. An adversary may modify these Registry keys to add new SSPs, which will be loaded the next time the system boots, or when the AddSecurityPackage Windows API function is called. [91]
Service Registry Permissions Weakness	**Persistence** Privilege Escalation	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 permissions.[92] If the permissions for users and groups are not properly set and allow access to the Registry keys for a service, then adversaries can change the service binPath/ImagePath to point to a different executable under their control. When the service starts or is restarted, then the adversary-controlled program will execute, allowing the adversary to gain persistence and/or privilege escalation to the account context the service is set to execute under (local/domain account, SYSTEM, LocalService, or NetworkService).
Shortcut Modification	**Persistence**	Shortcuts or symbolic links are ways of referencing other files or programs that will be opened or executed when the shortcut is clicked or executed by a system startup process. Adversaries could use shortcuts to execute their tools for persistence. They may create a new shortcut as a means of indirection that may use Masquerading to look like a legitimate program. Adversaries could also edit the target path or entirely replace an existing shortcut so their tools will be executed instead of the intended legitimate program.
Startup Items	**Persistence** Privilege Escalation	Per Apple's documentation, startup items execute during the final phase of the boot process and contain shell scripts or other executable files along with configuration information used by the system to determine the execution order for all startup items[85]. This is technically a deprecated version (superseded by Launch Daemons), and thus the appropriate folder, `/Library/StartupItems` isn't guaranteed to exist on the system by default, but does appear to exist by default on macOS Sierra. A startup item is a directory whose executable and configuration property list (plist), `StartupParameters.plist`, reside in the top-level directory. An adversary can create the appropriate folders/files in the StartupItems directory to register their own persistence mechanism[54]. Additionally, since StartupItems run during the bootup phase of macOS, they will run as root. If an adversary is able to modify an existing Startup Item, then they will be able to Privilege Escalate as well.
System Firmware	**Persistence**	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.[93][94][95] System firmware like BIOS and (U)EFI underly the functionality of a computer and may be modified by an adversary to perform or assist in malicious activity. Capabilities exist to overwrite the system firmware, which may give sophisticated adversaries a means to install malicious firmware updates as a means of persistence on a system that may be difficult to detect.

Name	Tactics	Technical Description
Trap	Execution **Persistence**	The `trap` command allows programs and shells to specify commands that will be executed upon receiving interrupt signals. A common situation is a script allowing for graceful termination and handling of common keyboard interrupts like `ctrl+c` and `ctrl+d`. Adversaries can use this to register code to be executed when the shell encounters specific interrupts either to gain execution or as a persistence mechanism. Trap commands are of the following format `trap 'command list' signals` where "command list" will be executed when "signals" are received.
Valid Accounts	Defense Evasion **Persistence** Privilege Escalation	Adversaries may steal the credentials of a specific user or service account using Credential Access techniques. Compromised credentials may be used to bypass access controls placed on various resources on hosts and within the network and may even be used for persistent access to remote systems. 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. Adversaries may also create accounts, sometimes using pre-defined account names and passwords, as a means for persistence through backup access in case other means are unsuccessful. The overlap of credentials and permissions across a network of systems is of concern because the adversary may be able to pivot across accounts and systems to reach a high level of access (i.e., domain or enterprise administrator) to bypass access controls set within the enterprise.[96]
Web Shell	**Persistence** Privilege Escalation	A Web shell is a Web script that is placed on an openly accessible Web server to allow an adversary to use the Web server as a gateway into a network. A Web shell may provide a set of functions to execute or a command-line interface on the system that hosts the Web server. In addition to a server-side script, a Web shell may have a client interface program that is used to talk to the Web server (see, for example, China Chopper Web shell client).[97] Web shells may serve as Redundant Access or as a persistence mechanism in case an adversary's primary access methods are detected and removed.
Windows Management Instrumentation Event Subscription	**Persistence**	Windows Management Instrumentation (WMI) can be used to install event filters, providers, consumers, and bindings that execute code when a defined event occurs. Adversaries may use the capabilities of WMI to subscribe to an event and execute arbitrary code when that event occurs, providing persistence on a system. Adversaries may attempt to evade detection of this technique by compiling WMI scripts.[98] Examples of events that may be subscribed to are the wall clock time or the computer's uptime.[99] Several threat groups have reportedly used this technique to maintain persistence.[100]
... further results		

References

1. ^ ↑ Glyer, C., Kazanciyan, R. (2012, August 20). THE “HIKIT” ROOTKIT: ADVANCED AND PERSISTENT ATTACK TECHNIQUES (PART 1). Retrieved June 6, 2016. (https://www.fireeye.com/blog/threat-research/2012/08/hikit-rootkit-advanced-persistent-attack-techniques-part-1.html)

2. *a b* ↑ Maldonado, D., McGuffin, T. (2016, August 6). Sticky Keys to the Kingdom. Retrieved July 5, 2017. (https://www.slideshare.net/DennisMaldonado5/sticky-keys-to-the-kingdom)
3. *a b* ↑ Tilbury, C. (2014, August 28). Registry Analysis with CrowdResponse. Retrieved November 12, 2014. (http://blog.crowdstrike.com/registry-analysis-with-crowdresponse/)
4. *a b c d e f g* ↑ Hosseini, A. (2017, July 18). Ten Process Injection Techniques: A Technical Survey Of Common And Trending Process Injection Techniques. Retrieved December 7, 2017. (https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process)
5. ^ ↑ Microsoft. (2006, October). Working with the AppInit_DLLs registry value. Retrieved July 15, 2015. (https://support.microsoft.com/en-us/kb/197571)
6. ^ ↑ Microsoft. (n.d.). AppInit DLLs and Secure Boot. Retrieved July 15, 2015. (https://msdn.microsoft.com/en-us/library/dn280412)
7. ^ ↑ Microsoft. (n.d.). Authentication Packages. Retrieved March 1, 2017. (https://msdn.microsoft.com/library/windows/desktop/aa374733.aspx)
8. ^ ↑ Mandiant. (2016, February). M-Trends 2016. Retrieved January 4, 2017. (https://www.fireeye.com/content/dam/fireeye-www/regional/fr_FR/offers/pdfs/ig-mtrends-2016.pdf)
9. ^ ↑ Lau, H. (2011, August 8). Are MBR Infections Back in Fashion? (Infographic). Retrieved November 13, 2014. (http://www.symantec.com/connect/blogs/are-mbr-infections-back-fashion)
10. ^ ↑ Wikipedia. (2017, October 8). Browser Extension. Retrieved January 11, 2018. (https://en.wikipedia.org/wiki/Browser_extension)
11. ^ ↑ Chrome. (n.d.). What are Extensions?. Retrieved November 16, 2017. (https://developer.chrome.com/extensions)
12. ^ ↑ Jagpal, N., et al. (2015, August). Trends and Lessons from Three Years Fighting Malicious Extensions. Retrieved November 17, 2017. (https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43824.pdf)
13. ^ ↑ Brinkmann, M. (2017, September 19). First Chrome extension with JavaScript Crypto Miner detected. Retrieved November 16, 2017. (https://www.ghacks.net/2017/09/19/first-chrome-extension-with-javascript-crypto-miner-detected/)
14. ^ ↑ De Tore, M., Warner, J. (2018, January 15). MALICIOUS CHROME EXTENSIONS ENABLE CRIMINALS TO IMPACT OVER HALF A MILLION USERS AND GLOBAL BUSINESSES. Retrieved January 17, 2018. (https://www.icebrg.io/blog/malicious-chrome-extensions-enable-criminals-to-impact-over-half-a-million-users-and-global-businesses)
15. ^ ↑ Marinho, R. (n.d.). (Banker(GoogleChromeExtension)).targeting. Retrieved November 18, 2017. (https://isc.sans.edu/forums/diary/BankerGoogleChromeExtensiontargetingBrazil/22722/)
16. ^ ↑ Marinho, R. (n.d.). "Catch-All" Google Chrome Malicious Extension Steals All Posted Data. Retrieved November 16, 2017. (https://isc.sans.edu/forums/diary/CatchAll+Google+Chrome+Malicious+Extension+Steals+All+Posted+Data/22976/https:/threatpost.com/malicious-chrome-extension-steals-data-posted-to-any-website/128680/))
17. ^ ↑ Vachon, F., Faou, M. (2017, July 20). Stantinko: A massive adware campaign operating covertly since 2012. Retrieved November 16, 2017. (https://www.welivesecurity.com/2017/07/20/stantinko-massive-adware-campaign-operating-covertly-since-2012/)
18. ^ ↑ Kjaer, M. (2016, July 18). Malware in the browser: how you might get hacked by a Chrome extension. Retrieved November 22, 2017. (https://kjaer.io/extension-malware/)

19. ^ ↑ Microsoft. (n.d.). Change which programs Windows 7 uses by default. Retrieved July 26, 2016. (https://support.microsoft.com/en-us/help/18539/windows-7-change-default-programs)
20. ^ ↑ Microsoft. (n.d.). Specifying File Handlers for File Name Extensions. Retrieved November 13, 2014. (http://msdn.microsoft.com/en-us/library/bb166549.aspx)
21. ^ ↑ Microsoft. (n.d.). The Component Object Model. Retrieved August 18, 2016. (https://msdn.microsoft.com/library/ms694363.aspx)
22. ^ ↑ G DATA. (2014, October). COM Object hijacking: the discreet way of persistence. Retrieved August 13, 2016. (https://blog.gdatasoftware.com/2014/10/23941-com-object-hijacking-the-discreet-way-of-persistence)
23. ^ ↑ Microsoft. (n.d.). Dynamic-Link Library Search Order. Retrieved November 30, 2014. (http://msdn.microsoft.com/en-US/library/ms682586)
24. ^ ↑ OWASP. (2013, January 30). Binary planting. Retrieved June 7, 2016. (https://www.owasp.org/index.php/Binary_planting)
25. ^ ↑ Microsoft. (2010, August 22). Microsoft Security Advisory 2269637 Released. Retrieved December 5, 2014. (http://blogs.technet.com/b/msrc/archive/2010/08/21/microsoft-security-advisory-2269637-released.aspx)
26. ^ ↑ Microsoft. (n.d.). Dynamic-Link Library Redirection. Retrieved December 5, 2014. (http://msdn.microsoft.com/en-US/library/ms682600)
27. ^ ↑ Microsoft. (n.d.). Manifests. Retrieved December 5, 2014. (https://msdn.microsoft.com/en-US/library/aa375365)
28. ^ ↑ Mandiant. (2010, August 31). DLL Search Order Hijacking Revisited. Retrieved December 5, 2014. (https://www.mandiant.com/blog/dll-search-order-hijacking-revisited/)
29. *a b* ↑ Patrick Wardle. (2015). Writing Bad @$$ Malware for OS X. Retrieved July 10, 2017. (https://www.blackhat.com/docs/us-15/materials/us-15-Wardle-Writing-Bad-A-Malware-For-OS-X.pdf)
30. *a b c d* ↑ Patrick Wardle. (2015). Malware Persistence on OS X Yosemite. Retrieved July 10, 2017. (https://www.rsaconference.com/writable/presentations/file_upload/ht-r03-malware-persistence-on-os-x-yosemite_final.pdf)
31. ^ ↑ Adair, S. (2015, October 7). Virtual Private Keylogging: Cisco Web VPNs Leveraged for Access and Persistence. Retrieved March 20, 2017. (https://www.volexity.com/blog/2015/10/07/virtual-private-keylogging-cisco-web-vpns-leveraged-for-access-and-persistence/)
32. ^ ↑ Kugler, R. (2012, November 20). Mozilla Foundation Security Advisory 2012-98. Retrieved March 10, 2017. (https://www.mozilla.org/en-US/security/advisories/mfsa2012-98/)
33. ^ ↑ Kanthak, S. (2015, December 8). Executable installers are vulnerable^WEVIL (case 7): 7z*.exe allows remote code execution with escalation of privilege. Retrieved March 10, 2017. (http://seclists.org/fulldisclosure/2015/Dec/34)
34. *a b c d* ↑ Dani Creus, Tyler Halfpop, Robert Falcone. (2016, September 26). Sofacy's 'Komplex' OS X Trojan. Retrieved July 8, 2017. (https://researchcenter.paloaltonetworks.com/2016/09/unit42-sofacys-komplex-os-x-trojan/)
35. *a b* ↑ Thomas Reed. (2017, January 18). New Mac backdoor using antiquated code. Retrieved July 5, 2017. (https://blog.malwarebytes.com/threat-analysis/2017/01/new-mac-backdoor-using-antiquated-code/)
36. *a b* ↑ Claud Xiao. (n.d.). WireLurker: A New Era in iOS and OS X Malware. Retrieved July 10, 2017. (https://www.paloaltonetworks.com/content/dam/pan/en_US/assets/pdf/reports/Unit_42/unit42-wirelurker.pdf)

37. ^ ↑ Microsoft. (n.d.). Hooks Overview. Retrieved December 12, 2017. (https://msdn.microsoft.com/library/windows/desktop/ms644959.aspx)
38. ^ ↑ Tigzy. (2014, October 15). Userland Rootkits: Part 1, IAT hooks. Retrieved December 12, 2017. (https://www.adlice.com/userland-rootkits-part-1-iat-hooks/)
39. *a b* ↑ Hillman, M. (2015, August 8). Dynamic Hooking Techniques: User Mode. Retrieved December 20, 2017. (https://www.mwrinfosecurity.com/our-thinking/dynamic-hooking-techniques-user-mode/)
40. ^ ↑ Mariani, B. (2011, September 6). Inline Hooking in Windows. Retrieved December 12, 2017. (https://www.exploit-db.com/docs/17802.pdf)
41. ^ ↑ Microsoft. (2017, September 15). TrojanSpy:Win32/Ursnif.gen!I. Retrieved December 18, 2017. (https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanSpy%3AWin32%2FUrsnif.gen!I)
42. ^ ↑ Symantec. (n.d.). Windows Rootkit Overview. Retrieved December 21, 2017. (https://www.symantec.com/avcenter/reference/windows.rootkit.overview.pdf)
43. ^ ↑ Wikipedia. (2016, May 23). Hypervisor. Retrieved June 11, 2016. (https://en.wikipedia.org/wiki/Hypervisor)
44. ^ ↑ Xen. (n.d.). In Wikipedia. Retrieved November 13, 2014. (http://en.wikipedia.org/wiki/Xen)
45. ^ ↑ Myers, M., and Youndt, S. (2007). An Introduction to Hardware-Assisted Virtual Machine (HVM) Rootkits. Retrieved November 13, 2014. (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.8832&rep=rep1&type=pdf)
46. *a b* ↑ Shanbhag, M. (2010, March 24). Image File Execution Options (IFEO). Retrieved December 18, 2017. (https://blogs.msdn.microsoft.com/mithuns/2010/03/24/image-file-execution-options-ifeo/)
47. ^ ↑ Microsoft. (2017, May 23). GFlags Overview. Retrieved December 18, 2017. (https://docs.microsoft.com/windows-hardware/drivers/debugger/gflags-overview)
48. ^ ↑ FSecure. (n.d.). Backdoor - W32/Hupigon.EMV - Threat Description. Retrieved December 18, 2017. (https://www.f-secure.com/v-descs/backdoor_w32_hupigon_emv.shtml)
49. ^ ↑ Symantec. (2008, June 28). Trojan.Ushedix. Retrieved December 18, 2017. (https://www.symantec.com/security_response/writeup.jsp?docid=2008-062807-2501-99&tabid=2)
50. ^ ↑ Microsoft. (n.d.). Security Subsystem Architecture. Retrieved November 27, 2017. (https://technet.microsoft.com/library/cc961760.aspx)
51. *a b* ↑ Apple. (n.d.). Creating Launch Daemons and Agents. Retrieved July 10, 2017. (https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLaunchdJobs.html)
52. ^ ↑ Marc-Etienne M.Leveille. (2016, July 6). New OSX/Keydnap malware is hungry for credentials. Retrieved July 3, 2017. (https://www.welivesecurity.com/2016/07/06/new-osxkeydnap-malware-hungry-credentials/)
53. *a b* ↑ Thomas Reed. (2017, July 7). New OSX.Dok malware intercepts web traffic. Retrieved July 10, 2017. (https://blog.malwarebytes.com/threat-analysis/2017/04/new-osx-dok-malware-intercepts-web-traffic/)
54. *a b c d e f g* ↑ Patrick Wardle. (2014, September). Methods of Malware Persistence on Mac OS X. Retrieved July 5, 2017. (https://www.virusbulletin.com/uploads/pdf/conference/vb2014/VB2014-Wardle.pdf)
55. *a b* ↑ Patrick Wardle. (2016, February 29). Let's Play Doctor: Practical OS X Malware Detection & Analysis. Retrieved July 10, 2017. (https://www.synack.com/wp-content/uploads/2016/03/RSA_OSX_Malware.pdf)

56. ^ ↑ Eddie Lee. (2016, February 17). OceanLotus for OS X - an Application Bundle Pretending to be an Adobe Flash Update. Retrieved July 5, 2017. (https://www.alienvault.com/blogs/labs-research/oceanlotus-for-os-x-an-application-bundle-pretending-to-be-an-adobe-flash-update)
57. ^ ↑ Paul Vixie. (n.d.). crontab(5) - Linux man page. Retrieved December 19, 2017. (https://linux.die.net/man/5/crontab)
58. ^ ↑ Thomas Koenig. (n.d.). at(1) - Linux man page. Retrieved December 19, 2017. (https://linux.die.net/man/1/at)
59. *a b c* ↑ Apple. (n.d.). Retrieved July 17, 2017. (https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/ScheduledJobs.html)
60. ^ ↑ Thomas. (2013, July 15). New signed malware called Janicab. Retrieved July 17, 2017. (http://www.thesafemac.com/new-signed-malware-called-janicab/)
61. ^ ↑ Threat Intelligence Team. (2015, January 6). Linux DDoS Trojan hiding itself with an embedded rootkit. Retrieved January 8, 2018. (https://blog.avast.com/2015/01/06/linux-ddos-trojan-hiding-itself-with-an-embedded-rootkit/)
62. *a b* ↑ Apple. (2016, September 13). Adding Login Items. Retrieved July 11, 2017. (https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingLoginItems.html)
63. ^ ↑ Microsoft. (2005, January 21). Creating logon scripts. Retrieved April 27, 2016. (https://technet.microsoft.com/en-us/library/cc758918(v=ws.10).aspx)
64. ^ ↑ Apple. (2011, June 1). Mac OS X: Creating a login hook. Retrieved July 17, 2017. (https://support.apple.com/de-at/HT2420)
65. ^ ↑ Microsoft. (n.d.). Using Netsh. Retrieved February 13, 2017. (https://technet.microsoft.com/library/bb490939.aspx)
66. ^ ↑ Demaske, M. (2016, September 23). USING NETSHELL TO EXECUTE EVIL DLLS AND PERSIST ON A HOST. Retrieved April 8, 2017. (https://htmlpreview.github.io/?https://github.com/MatthewDemaske/blogbackup/blob/master/netshell.html)
67. ^ ↑ Smeets, M. (2016, September 26). NetshHelperBeacon. Retrieved February 13, 2017. (https://github.com/outflankbv/NetshHelperBeacon)
68. ^ ↑ Microsoft. (n.d.). Services. Retrieved June 7, 2016. (https://technet.microsoft.com/en-us/library/cc772408.aspx)
69. ^ ↑ Microsoft. (n.d.). Change the Normal template (Normal.dotm). Retrieved July 3, 2017. (https://support.office.com/article/Change-the-Normal-template-Normal-dotm-06de294b-d216-47f6-ab77-ccb5166f98ea)
70. ^ ↑ Austin, J. (2017, June 6). Getting Started with VBA in Office. Retrieved July 3, 2017. (https://msdn.microsoft.com/en-us/vba/office-shared-vba/articles/getting-started-with-vba-in-office)
71. ^ ↑ Nelson, M. (2014, January 23). Maintaining Access with normal.dotm. Retrieved July 3, 2017. (https://enigma0x3.net/2014/01/23/maintaining-access-with-normal-dotm/comment-page-1/)
72. ^ ↑ Hexacorn. (2017, April 17). Beyond good ol' Run key, Part 62. Retrieved July 3, 2017. (http://www.hexacorn.com/blog/2017/04/19/beyond-good-ol-run-key-part-62/)
73. ^ ↑ Hexacorn. (2014, April 16). Beyond good ol' Run key, Part 10. Retrieved July 3, 2017. (http://www.hexacorn.com/blog/2014/04/16/beyond-good-ol-run-key-part-10/)
74. ^ ↑ Microsoft. (n.d.). Add or remove add-ins. Retrieved July 3, 2017. (https://support.office.com/article/Add-or-remove-add-ins-0af570c4-5cf3-4fa9-9b88-403625a0b460)

75. ^ ↑ Knowles, W. (2017, April 21). Add-In Opportunities for Office Persistence. Retrieved July 3, 2017. (https://labs.mwrinfosecurity.com/blog/add-in-opportunities-for-office-persistence/)
76. ^ ↑ Nagaraju, S. (2014, April 8). MS14-019 – Fixing a binary hijacking via .cmd or .bat file. Retrieved July 25, 2016. (https://blogs.technet.microsoft.com/srd/2014/04/08/ms14-019-fixing-a-binary-hijacking-via-cmd-or-bat-file/)
77. ^ ↑ Microsoft. (n.d.). CurrentControlSet\Services Subkey Entries. Retrieved November 30, 2014. (http://support.microsoft.com/KB/103000)
78. ^ ↑ Baggett, M. (2012, November 8). Help eliminate unquoted path vulnerabilities. Retrieved December 4, 2014. (https://isc.sans.edu/diary/Help+eliminate+unquoted+path+vulnerabilities/14464)
79. ^ ↑ Microsoft. (n.d.). CreateProcess function. Retrieved December 5, 2014. (http://msdn.microsoft.com/en-us/library/ms682425)
80. ^ ↑ Hill, T. (n.d.). Windows NT Command Shell. Retrieved December 5, 2014. (http://technet.microsoft.com/en-us/library/cc723564.aspx#XSLTsection127121120120)
81. ^ ↑ Microsoft. (n.d.). WinExec function. Retrieved December 5, 2014. (http://msdn.microsoft.com/en-us/library/ms687393)
82. ^ ↑ Microsoft. (n.d.). Environment Property. Retrieved July 27, 2016. (https://msdn.microsoft.com/en-us/library/fd7hxfdd.aspx)
83. ^ ↑ Microsoft. (n.d.). AddMonitor function. Retrieved November 12, 2014. (http://msdn.microsoft.com/en-us/library/dd183341)
84. ^ ↑ Bloxham, B. (n.d.). Getting Windows to Play with Itself [PowerPoint slides]. Retrieved November 12, 2014. (https://www.defcon.org/images/defcon-22/dc-22-presentations/Bloxham/DEFCON-22-Brady-Bloxham-Windows-API-Abuse-UPDATED.pdf)
85. *a b* ↑ Apple. (2016, September 13). Startup Items. Retrieved July 11, 2017. (https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/StartupItems.html)
86. ^ ↑ Mandiant. (n.d.). APT1 Exposing One of China's Cyber Espionage Units. Retrieved July 18, 2016. (https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf)
87. ^ ↑ Microsoft. (n.d.). Run and RunOnce Registry Keys. Retrieved November 12, 2014. (http://msdn.microsoft.com/en-us/library/aa376977)
88. ^ ↑ Microsoft. (2005, January 21). Task Scheduler and security. Retrieved June 8, 2016. (https://technet.microsoft.com/en-us/library/cc785125.aspx)
89. ^ ↑ Wikipedia. (2017, November 22). Screensaver. Retrieved December 5, 2017. (https://en.wikipedia.org/wiki/Screensaver)
90. ^ ↑ ESET. (2017, August). Gazing at Gazer: Turla's new second stage backdoor. Retrieved September 14, 2017. (https://www.welivesecurity.com/wp-content/uploads/2017/08/eset-gazer.pdf)
91. ^ ↑ Graeber, M. (2014, October). Analysis of Malicious Security Support Provider DLLs. Retrieved March 1, 2017. (http://docplayer.net/20839173-Analysis-of-malicious-security-support-provider-dlls.html)
92. ^ ↑ Microsoft. (n.d.). Registry Key Security and Access Rights. Retrieved March 16, 2017. (https://msdn.microsoft.com/library/windows/desktop/ms724878.aspx)
93. ^ ↑ Wikipedia. (n.d.). BIOS. Retrieved January 5, 2016. (https://en.wikipedia.org/wiki/BIOS)
94. ^ ↑ Wikipedia. (2017, July 10). Unified Extensible Firmware Interface. Retrieved July 11, 2017. (https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface)
95. ^ ↑ UEFI Forum. (n.d.). About UEFI Forum. Retrieved January 5, 2016. (http://www.uefi.org/about)

96. ^ ↑ Microsoft. (2016, April 15). Attractive Accounts for Credential Theft. Retrieved June 3, 2016. (https://technet.microsoft.com/en-us/library/dn535501.aspx)
97. ^ ↑ Lee, T., Hanzlik, D., Ahl, I. (2013, August 7). Breaking Down the China Chopper Web Shell - Part I. Retrieved March 27, 2015. (https://www.fireeye.com/blog/threat-research/2013/08/breaking-down-the-china-chopper-web-shell-part-i.html)
98. ^ ↑ Dell SecureWorks Counter Threat Unit™ (CTU) Research Team. (2016, March 28). A Novel WMI Persistence Implementation. Retrieved March 30, 2016. (https://www.secureworks.com/blog/wmi-persistence)
99. ^ ↑ Kazanciyan, R. & Hastings, M. (2014). Defcon 22 Presentation. Investigating PowerShell Attacks [slides]. Retrieved November 3, 2014. (https://www.defcon.org/images/defcon-22/dc-22-presentations/Kazanciyan-Hastings/DEFCON-22-Ryan-Kazanciyan-Matt-Hastings-Investigating-Powershell-Attacks.pdf)
100. ^ ↑ Mandiant. (2015, February 24). M-Trends 2015: A View from the Front Lines. Retrieved May 18, 2016. (https://www2.fireeye.com/rs/fireeye/images/rpt-m-trends-2015.pdf)

Retrieved from "https://attack.mitre.org/w/index.php?title=Persistence&oldid=1364"

Category: Tactic
