Beacon Command Behavior and OPSEC Considerations

A good operator knows their tools and has an idea of how the tool is accomplishing its objectives on their behalf. This document surveys Beacon's commands and provides background on which commands inject into remote processes, which commands spawn jobs, and which commands rely on cmd.exe or powershell.exe.

API-only

The following commands are built into Beacon and rely on Win32 APIs to meet their objectives:

- cd
- cp
- connect
- download
- drives
- exit
- getprivs
- getuid
- inline-execute
- jobkill
- kill
- link
- ls

make_token
mkdir
mv
ps
pwd
rev2self
rm
rportfwd
rportfwd_local
setenv
socks
steal_token
token_store
unlink
upload

House-keeping Commands

The following commands are built into Beacon and exist to configure Beacon or perform house-keeping actions. Some of these commands (e.g., clear, downloads, help, mode, note) do not generate a task for Beacon to execute.

argue
beacon_config
beacon_gate
blockdlls
cancel
checkin
clear
data-store
downloads
file_browser
help
history
jobs
mode dns
mode dns-txt

mode dns6
note
powershell-import
ppid
process_browser
sleep
socks stop
spawnto
sycall-method
windows_error_code
! (run a command from history)

Inline Execute (BOF)

The following commands are implemented as internal Beacon Object Files. A Beacon Object File is a compiled C program, written to a certain convention, that executes within a Beacon session. The capability is cleaned up after it finishes running.

clipboard
dllload
elevate svc-exe
elevate uac-token-duplication
getsystem
jump psexec
jump psexec64
jump psexec_psh
kerberos_ccache_use
kerberos_ticket_purge
kerberos_ticket_use
net domain
reg query
reg queryv
remote-exec psexec
remote-exec wmi
runasadmin uac-cmstplua
runasadmin uac-token-duplication
timestomp

The network interface resolution within both the portscan and covertvpn dialogs uses a Beacon Object File as well.

OPSEC Advice

The memory for Beacon Object Files is controlled with settings from the Malleable C2's process-inject block.

Post-Exploitation Jobs (Fork&Run)

Many Beacon post-exploitation features spawn a process and inject a capability into that process. Some people call this pattern fork&run. Beacon does this for a number of reasons: (i) this protects the agent if the capability crashes. (ii) historically, this scheme makes it seamless for an x86 Beacon to launch x64 post-exploitation tasks. This was critical as Beacon didn't have an x64 build until 2016. (iii) Some features can target a specific remote process. This allows the post-ex action to occur within different contexts without the need to migrate or spawn a payload in that other context. And (iv) this design decision keeps a lot of clutter (threads, suspicious content) generated by your post-ex action out of your Beacon process space. Here are the features that use this pattern:

Fork&Run Only

- covertvpn
- execute-assembly
- powerpick

Target Explicit Process Only

- browserpivot
- psinject

Fork&Run or Target Explicit Process

- chromedump
- dcsync
- desktop
- hashdump
- keylogger
- logonpasswords
- mimikatz
- net *
- portscan
- printscreen
- pth
- screenshot
- screenwatch
- ssh
- ssh-key

OPSEC Advice

Use the **spawnto** command to change the process Beacon will launch for its post-exploitation jobs. The default is rundll32.exe (you probably don't want that). The **ppid** command will change the parent process these jobs are run under as well. The **blockdlls** command will stop userland hooking for some security products. Malleable C2's process-inject block gives a lot of control over the process injection process. Malleable C2's post-ex block has several OPSEC options for these post-ex DLLs themselves. For features that have an explicit injection option, consider injecting into your current Beacon process. Cobalt Strike detects and acts on self-injection different from remote injection.

Explicit injection will not cleanup any memory after the post-exploitation job has completed. The recommendation is to inject into a process that can be safely terminated by you to cleanup in-memory artifacts.

Process Execution

These commands spawn a new process:

- execute
- run

runas
runu

OPSEC Advice

The **ppid** command will change the parent process of commands run by execute. The ppid command does not affect runas or runu.

Process Execution (cmd.exe)

The **shell** command depends on cmd.exe. Use **run** to run a command and get output without cmd.exe

The **pth** command relies on cmd.exe to pass a token to Beacon via a named pipe. The command pattern to pass this token is an indicator some host-based security products look for. Read How to Pass-the-Hash with Mimikatz for instructions on how to do this manually.

Process Execution (powershell.exe)

The following commands launch powershell.exe to perform some task on your behalf.

jump
winrm
jump winrm64
powershell
remote-exec winrm

OPSEC Advice

Use the ppid command to change the parent process powershell.exe is run under. Use the POWERSHELL_COMMAND Aggressor Script hook to change the format of the PowerShell command and its arguments. The **jump winrm**, **jump winrm64**, and powershell [when a script is imported] commands deal with PowerShell content that is too large to fit in a single command-line. To get around this, these features host a script on a self-contained web server within your Beacon session. Use the

POWERSHELL_DOWNLOAD_CRADLE Aggressor Script hook to shape the download cradle used to download these scripts.

Process Injection (Remote)

The post-exploitation job commands (previously mentioned) rely on process injection too. The other commands that inject into a remote process are:

- dllinject
- dllload
- execute-dll <pid>
- inject
- shinject

OPSEC Advice

Malleable C2's process-inject block block gives a lot of control over the process injection process. When beacon exits an injected process it will not clean itself from memory and will no longer be masked when the stage.sleep_mask is set to true. With the 4.5 release most of the heap memory will be cleared and released. Recommendation is to not exit beacon if you do not want to leave memory artifacts unmasked during your engagement. When your engagement is done it is recommended to reboot all of the targeted systems to remove any lingering in-memory artifacts.

Process Injection (Spawn&Inject)

These commands spawn a temporary process and inject a payload or shellcode into it:

- elevate uac-token-duplication
- execute-dll
- shspawn
- spawn
- spawnas
- spawnu
- spunnel
- spunnel_local

OPSEC Advice

Use the **spawnto** command to set the temporary process to use. The **ppid** command sets a parent process for most of these commands. The **blockdlls** command will block userland hooks from some security products. Malleable C2's process-inject block gives a lot of control over the process injection process. Malleable C2's post-ex block provides options to adjust Beacon's in-memory evasion options.

Service Creation

The following internal Beacon commands create a service (either on the current host or a remote target) to run a command. These commands use Win32 APIs to create and manipulate services.

elevate svc-exe
jump psexec
jump psexec64
jump psexec_psh
remote-exec psexec

OPSEC Advice

These commands use a service name that consists of random letters and numbers by default. The Aggressor Script PSEXEC_SERVICE hook allows you to change this behavior. Each of these commands (excepting jump psexec_psh and remote-exec psexec) generate a service EXE and upload it to the target. Cobalt Strike's built-in service EXE spawns rundll32.exe [with no arguments], injects a payload into it, and exits. This is done to allow immediate cleanup of the executable. Use the Artifact Kit to change the content and behaviors of the generated EXE.

RELATED TOPICS

4.10 | 202407160902 | July 2024