

WINTRE: AN ADVERSARY EMULATION TOOL

Final Report

Student: Martin Earls / C00227207

Supervisor: Richard Butler

Contents

1	Abstract.....	3
2	Introduction	4
3	Final Product.....	5
3.1	Features.....	7
3.2	Features Not Achieved.....	8
4	Problems Encountered and Solutions.....	9
4.1	Reporting.....	9
4.2	Compilers	9
4.3	Storing Technique Details.....	10
4.4	Dynamically Adding New Techniques.....	10
4.5	File Structure.....	10
5	Future Design Considerations	11
6	Product Testing.....	11

1 ABSTRACT

This document outlines the final product that was produced i.e., "WINTRE" to perform adversary emulation, the process of developing the product, its features and an overview of the product.

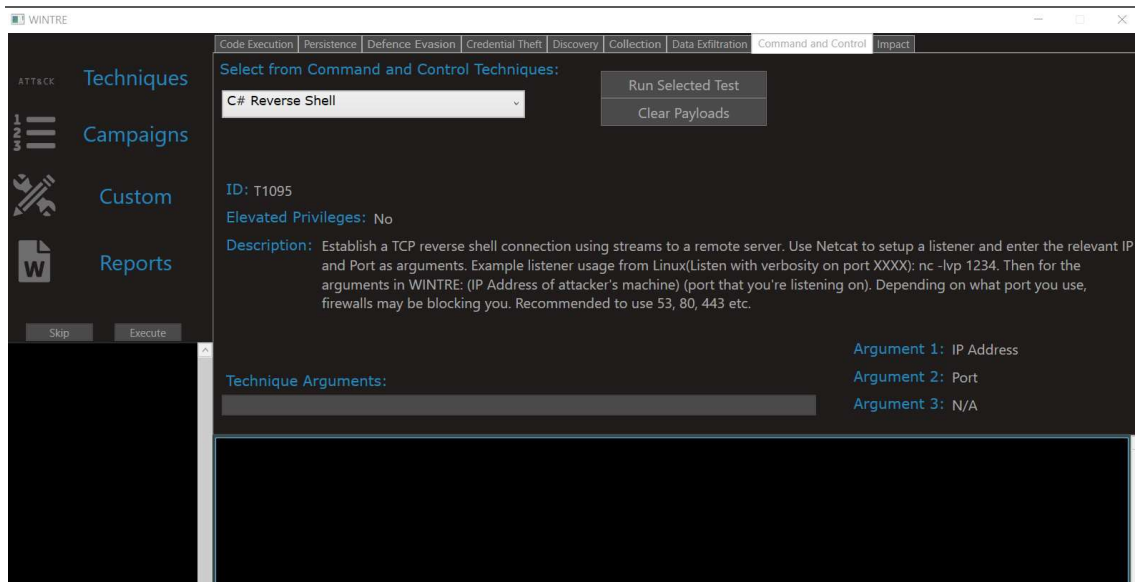
WINTRE involves a C# GUI application that can execute tactics, techniques and procedures (TTPs) based on popular methods utilised by threat actors and advanced persistent threat groups. This app can be used to run techniques and produce detailed logs to help an organisation test their detection analytics. 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 INTRODUCTION

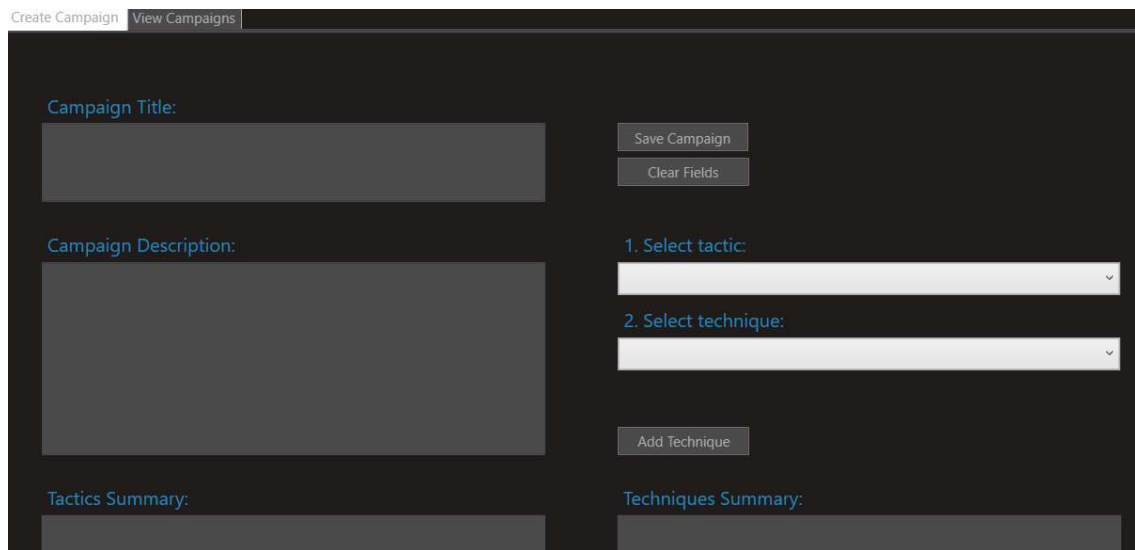
WINTRE focuses on providing a solution for endpoint security testing, based on adversary simulation, i.e., the process of running techniques used by attackers or malware in cyber-attacks in a controlled environment in order to assess whether or not the current endpoint protections are sufficient and if detection analytics are working as intended.

An organisation ideally needs to be able to block and detect attacks that may occur. WINTRE simplifies validation of this process, with 40+ pre-built techniques, custom technique implementation and automated reporting while covering the majority of tactics used in the post-exploitation phase of an attack. WINTRE can also act as a major cost saving measure by allowing an organisation to evaluate and validate the cost of their existing endpoint security controls.

3 FINAL PRODUCT



Screenshot of the main techniques page.



Create Campaign Page.

Create Campaign View Campaigns

Double click the campaign you'd like to run to load its techniques into the queue:

Title	Description	Tactics	Techniques
123	1234	["Credential Theft"]	["SAM SYSTEM Reg exe"]
Assumed Breach	test	["Discovery", "Code Execution"]	["Get Registered AV", "BitsAdmin Execute CMD", "PowerShell Run Exe"]
C++ Test campaign	testing C++ technique (win API cred theft)	["Credential Theft"]	["WinAPI Credential Prompt"]
Code Execution testing	testing testing testing testing testing testing	["Code Execution"]	["Explorer", "FTP", "PowerShell Run Exe", "Scheduled Task", "WMIC"]

Load campaigns page.

Technique Name: e.g. Local Account Discovery

MITRE ATT&CK ID: e.g. T1087

Elevated privileges:

Category/Tactic:

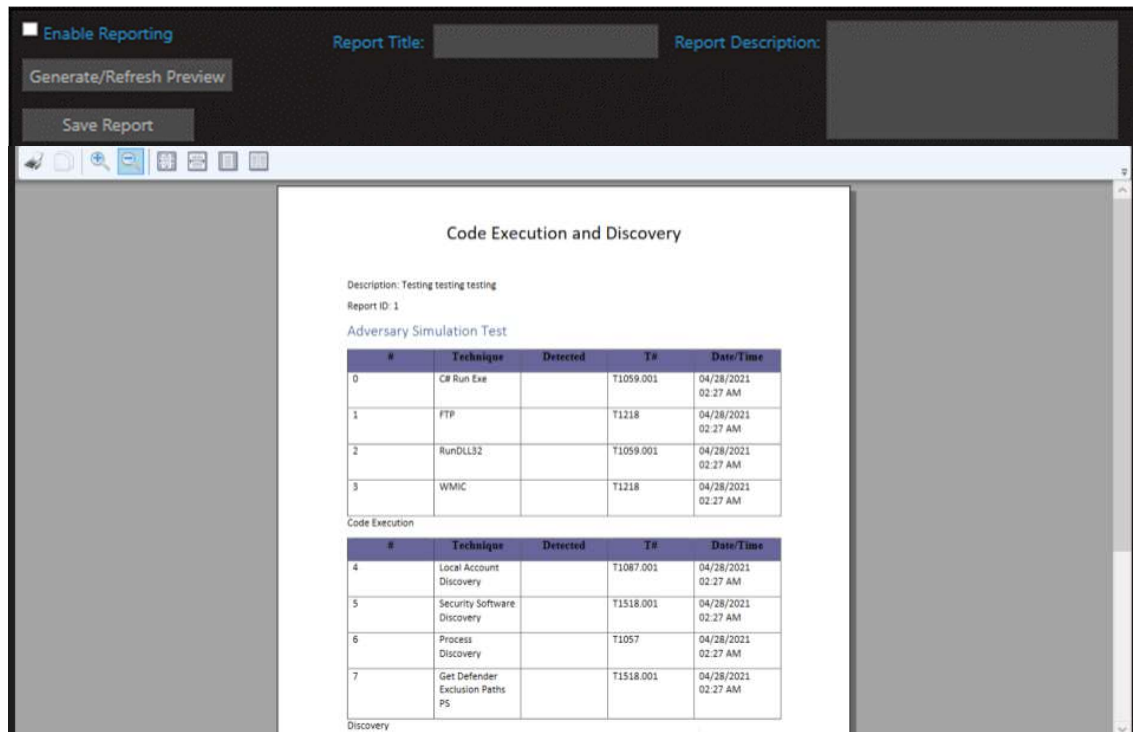
Selected Template:

Command(s): Make sure your syntax is correct! Refer to manual for further information.

e.g. net user

Description: e.g. Monitor for the usage of "net user", this can be achieved using Sysmon and filtering eventData based on CommandLine values

Custom Techniques Page.



Automated Reports page.

3.1 FEATURES

- Perform adversary simulation on Windows endpoints.
- Locally compiles each simulation test into a separate executable (in-case of anti-virus quarantining).
- Support for dynamic C# and C++ compilation (improved technique coverage).
- Campaigns feature allowing you to keep track of which techniques you've ran, enabling the ease of re-testing by automatically loading those techniques into a queue when re-testing.
- Highly extendible, custom technique creation, allowing the user to define new techniques based on command line scripting (via Command Prompt or PowerShell).
- Windows API based techniques can also be added easily by directly adding the relevant source code files.
- Automated reporting via Microsoft Word generating tables automatically that include all the relevant details needed to document the testing process.
- Techniques covering code execution, discovery, persistence, command and control, defence evasion, collection, impact, data exfiltration and credential theft.
- Allows the simulation of spyware, assumed breach scenarios, ransomware attacks and password stealers.

3.2 FEATURES NOT ACHIEVED

Some features that were researched during the initial development phase were not completed to time constraints and complexity involved:

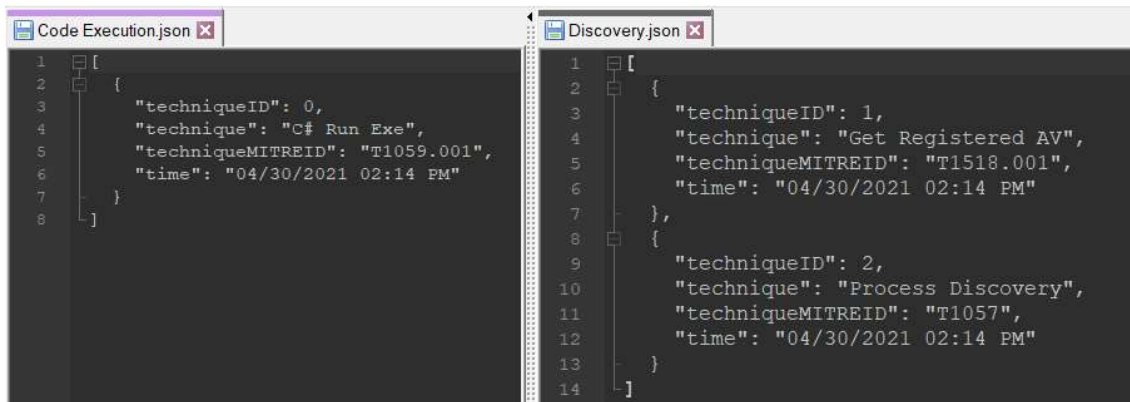
- Privilege escalation techniques - The act of privilege escalation itself is generally more difficult to emulate than other techniques such as code execution ones. UAC bypasses were considered and researched as well as token impersonation, a well-known privilege escalation technique. Ultimately prototypes were developed for both as well as utilising existing open-source research such as UACMe (repository of UAC bypasses) but were not suitable for the production in the given time necessary.
- Lateral movement techniques - Initially techniques similar to those utilised in Microsoft's SysInternals tool PsExec which are used in lateral movement were considered and researched. After further researching the time investment required to truly understand and code these techniques from the ground up it was decided to focus more on the user experience and other technique categories.
- Defense evasion techniques - I researched and began developing various defense evasion techniques such as process injection that utilise Windows API functions. Due to time constraints, I was unable to fully complete and include these techniques in the final solution.

4 PROBLEMS ENCOUNTERED AND SOLUTIONS

4.1 REPORTING

Automated reporting had several complications during the development process. Firstly, the automated tracking of techniques was required in order to dynamically update the report in the background whilst allowing the user to generate a preview of the report.

Initially, I was unable to update the report properly due to file lock issues. To solve this issue, I began dynamically tracking which techniques had been ran using list arrays per category of technique. These lists would then be serialized as JSON. The JSON, acting as temporary files per session could then be parsed and the table generated on the deserialized data.

The image shows two side-by-side windows of a code editor. The left window is titled 'Code Execution.json' and contains a single JSON object with the following content:

```
1 [
2   {
3     "techniqueID": 0,
4     "technique": "C# Run Exe",
5     "techniqueMITREID": "T1059.001",
6     "time": "04/30/2021 02:14 PM"
7   }
8 ]
```

The right window is titled 'Discovery.json' and contains a JSON array of two objects:

```
1 [
2   {
3     "techniqueID": 1,
4     "technique": "Get Registered AV",
5     "techniqueMITREID": "T1518.001",
6     "time": "04/30/2021 02:14 PM"
7   },
8   {
9     "techniqueID": 2,
10    "technique": "Process Discovery",
11    "techniqueMITREID": "T1057",
12    "time": "04/30/2021 02:14 PM"
13  }
14 ]
```

Example JSON structure of report in progress (generated automatically using JSON.NET).

Generating a report preview also initially proved challenging as Windows Presentation Foundation does not have any native components capable of presenting a word document. To solve this, I was able to convert the word document in progress as an XPS document, which could then be loaded in a document viewer component.

This introduced another issue when generating multiple previews in the same session as the XPS file used as the preview would become locked as well. In order to mitigate this, it was necessary to manually access the XpsPackage handle and allow for the file lock to be lifted, removing the loaded preview file and re-generating the report to load a new XPS document.

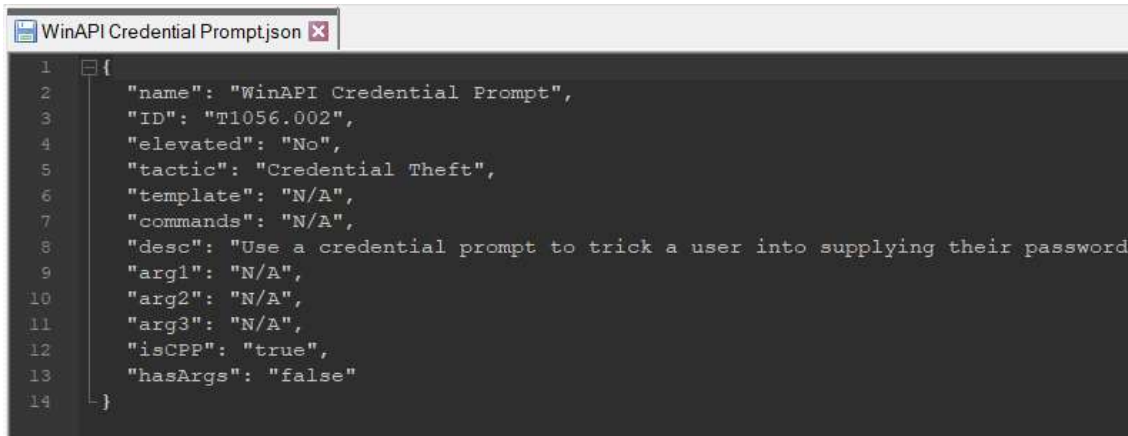
4.2 COMPILERS

In order to support both C# and C++ techniques it was necessary to figure out what command arguments would work for the majority of use cases to allow dynamic compilation of either C# or C++ source code files. The application compiles each technique that is ran into a separate executable to prevent quarantining of that file. I considered using GCC or other compilers and ended up opting for the Visual Studio compiler using cl.exe due to it being built into my development environment and being familiar with it.

It made testing easier but getting the correct arguments to prevent compiler errors took time to test out. Now with C# and C++ compilation and execution working the product has the ability to be extended using command line techniques and more complex Windows API based functions using C# and C++.

4.3 STORING TECHNIQUE DETAILS

I had to consider a storage mechanism for the technique's details. MySQL was considered but I wanted the infrastructure cost of the tool to be as low as possible. Having used JSON.NET for the reporting feature, which was fast and efficient to code, I decided to save each techniques information in its own JSON file that would be loaded when selecting that technique from the menu.



```
1 {
2   "name": "WinAPI Credential Prompt",
3   "ID": "T1056.002",
4   "elevated": "No",
5   "tactic": "Credential Theft",
6   "template": "N/A",
7   "commands": "N/A",
8   "desc": "Use a credential prompt to trick a user into supplying their password",
9   "arg1": "N/A",
10  "arg2": "N/A",
11  "arg3": "N/A",
12  "isCPP": "true",
13  "hasArgs": "false"
14 }
```

Sample JSON file storing technique information.

4.4 DYNAMICALLY ADDING NEW TECHNIQUES

I wanted the ability for users to easily extend the product themselves using command line techniques. To accomplish this, I created templates of source code files that would execute any command based on which template was chosen. The biggest challenge for this feature was learning how to properly escape commands that contained special characters, serializing the escaped commands and placing them in the source code templates.

Command Prompt:

```
cmd /c "your command"
```

PowerShell:

```
powershell -command "your command"
```

The relevant source code template is then saved as a new technique, which simply uses `Process.Start()` to launch the new technique, giving users the choice to add it to a campaign or run the technique manually.

4.5 FILE STRUCTURE

In order to load, run or log techniques that have been ran, filenames of the technique's source code were used. This brought additional challenges as it would cause exceptions in certain use cases, e.g., where a user tries to create a custom technique using special characters.

This required additional handling and validating of user input, using regular expressions to check the technique titles and improving the responsiveness of the user interface by creating relevant message pop ups informing the user that only alphanumeric characters were accepted for new techniques.

5 FUTURE DESIGN CONSIDERATIONS

For future design considerations, there are several things I would change given what I learned during the development process. Firstly, I would design it as a platform rather than a tool involving a web interface. This web interface would communicate with a server. The server would then communicate with GUI-less agents for Windows and Linux threat emulation.

As well as this I would like to have developed more Windows API based techniques, specifically around process injection and defense evasion techniques aimed at evading enterprise security solutions, such as API unhooking which would blind anti-virus/Endpoint Detection and Response software. I would also consider developing a shellcode generator similar to Metasploit's or at least creating various options for shellcode wrappers.

I would like the future design to have a great focus on testing Data Loss Prevention solutions, given how serious the consequences can be for an organisation if a data breach occurs.

In order to achieve these ideal features a variety of new languages would need to be learned and more low-level programming such as for shellcode would be necessary as well as new research into developing techniques for Linux systems.

6 PRODUCT TESTING

In order to test the final product, there were two major stages of testing performed. The first involved testing the techniques locally on a default Windows 10 virtual machine. Testing was done to ensure tests executed as expected, testing for exceptions, testing UI elements and in order to make improvements to the user experience.

The second stage of testing was done in collaboration with IT Carlow Computing Services. This testing followed an assumed breach methodology and the results of this testing will not be published. Based on the results I was able to recommend various mitigations to Computing Services to help improve the overall security posture of IT Carlow.