Noah Beckman
Fall 2022
Malware Analysis

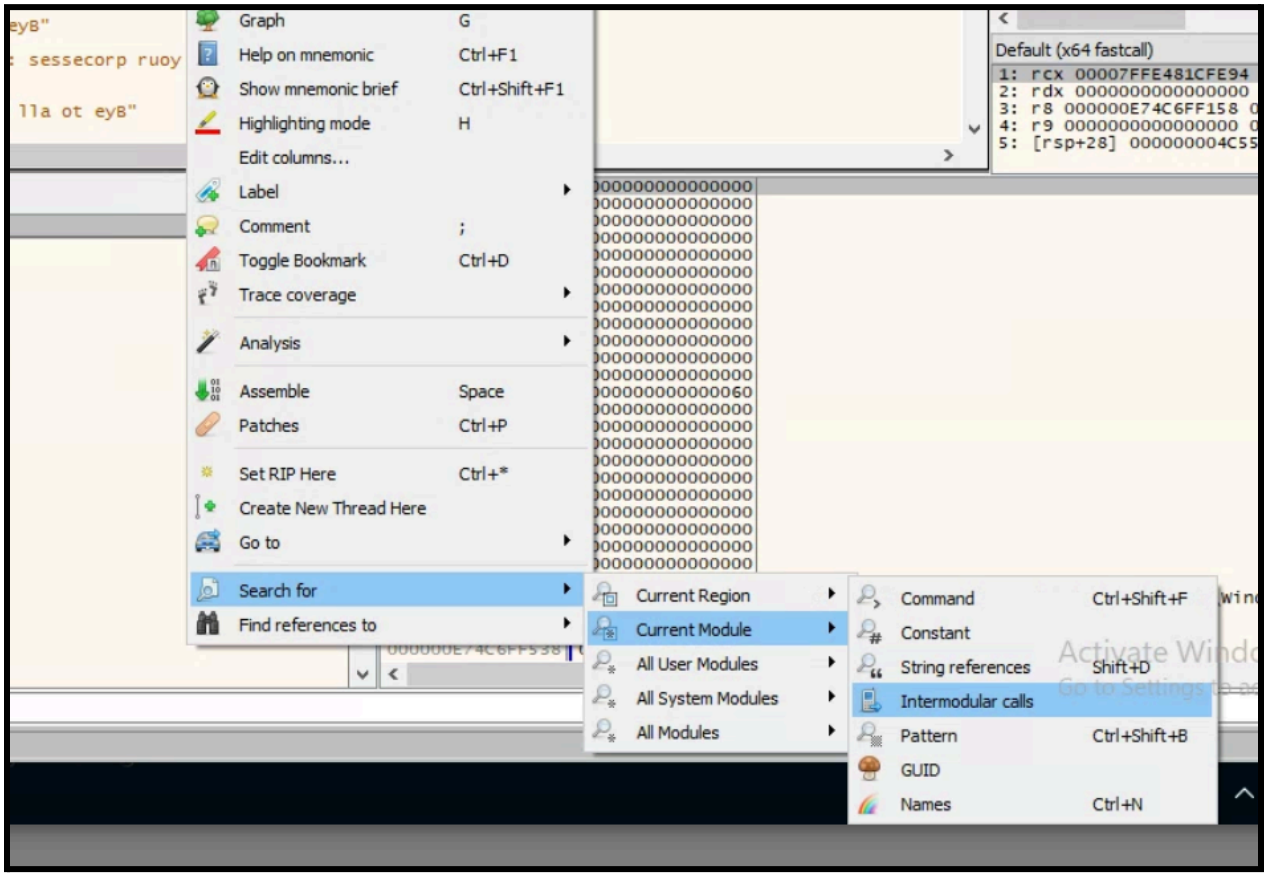
Overview

The purpose of this lab is to patch and analyze a sample of malware that interferes with a variety of our forensic tools and messes with the mouse. Once patched, a full analysis of the sample can be done to see how the sample works and be documented.

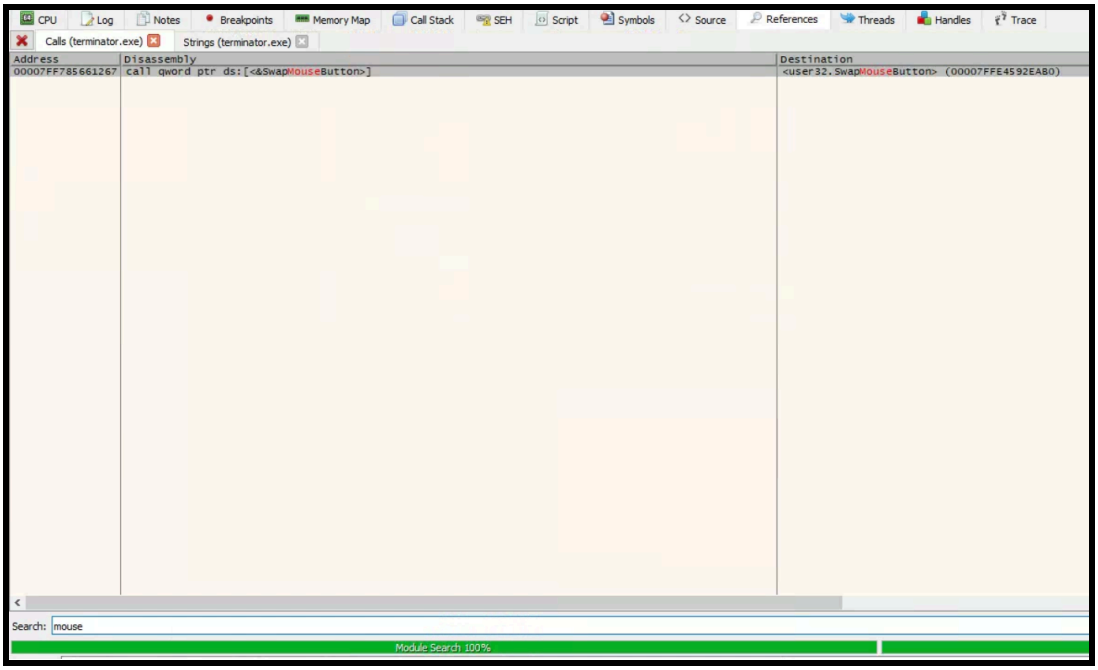
Deliverable 1 - Patching the Sample

Patching the Mouse Button Switching

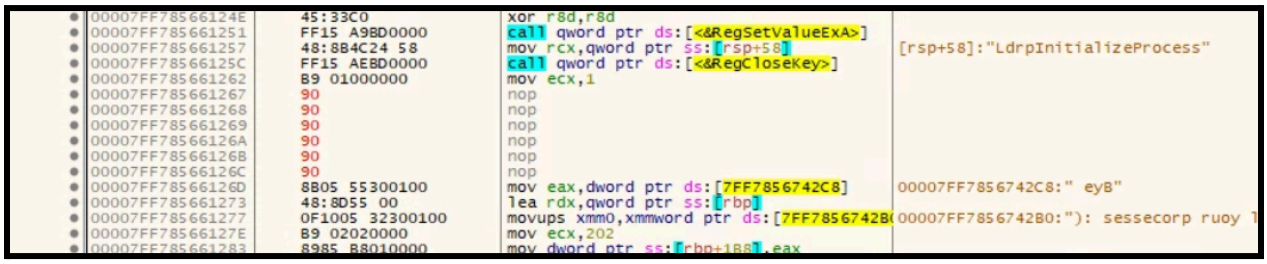
In order to patch the mouse button switch, you need to find the function within the sample code that switches the button. To find the function call right click, search for, current module, intermodule cells, then filter the results at the bottom with the word mouse. This can be seen in the screenshot below.

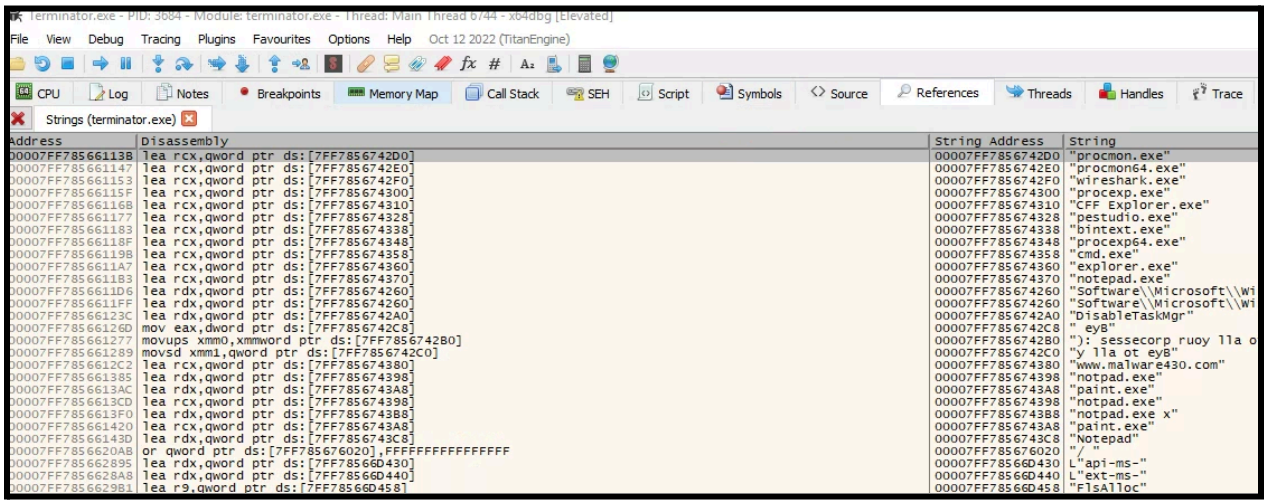


Clicking this button allows us to view all the functions / calls within x65 debug. The bottom of the window has a filter we can use to find the specific mouse button swap function.

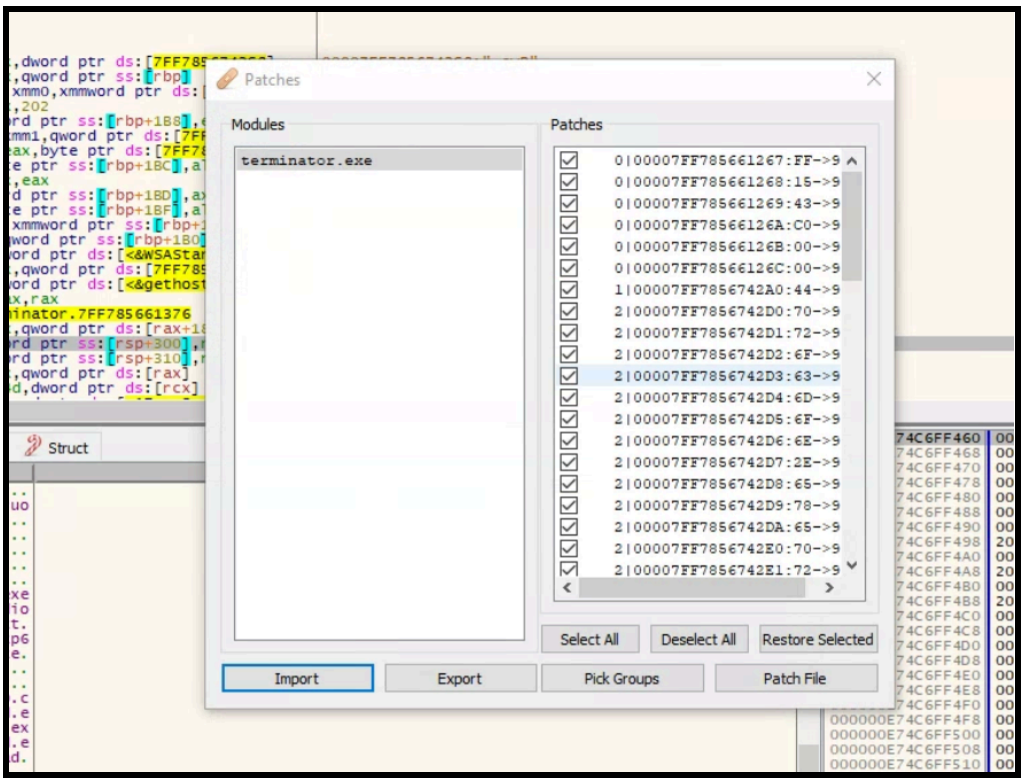


The filter and function can be seen in the screenshot above. We can then follow the address and nop (no operation) out the function. This can be seen in the screenshot below





This screenshot shows the strings of the sample, we can then right click and follow in disassembler where it says procmon.exe in the actual code. Once in the code we can nop out the first letter of procmon to patch by placing in the value 90. The end result can be seen below.



The screenshot above shows the patching of the application closes within the malware. This can be done by finding the reference in the strings and rewriting the strings with 90.

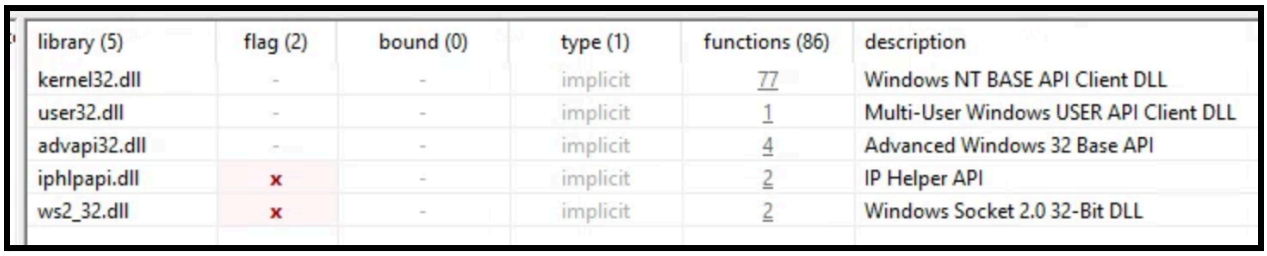
Now that the malware has been patched static and dynamic analysis can be done.

Deliverable 2 - Static Analysis

Metadata

Sample Name: Terminator.exe
SHA 256 File Hash:
3ECC3DE52A8BECF12965DAC256B02D7995B653FCE805359D27452B6DF8790F25
Compiler Stamp: Fri Sep 27 12:08:09 2019 | UTC
Subsystem Type: Console

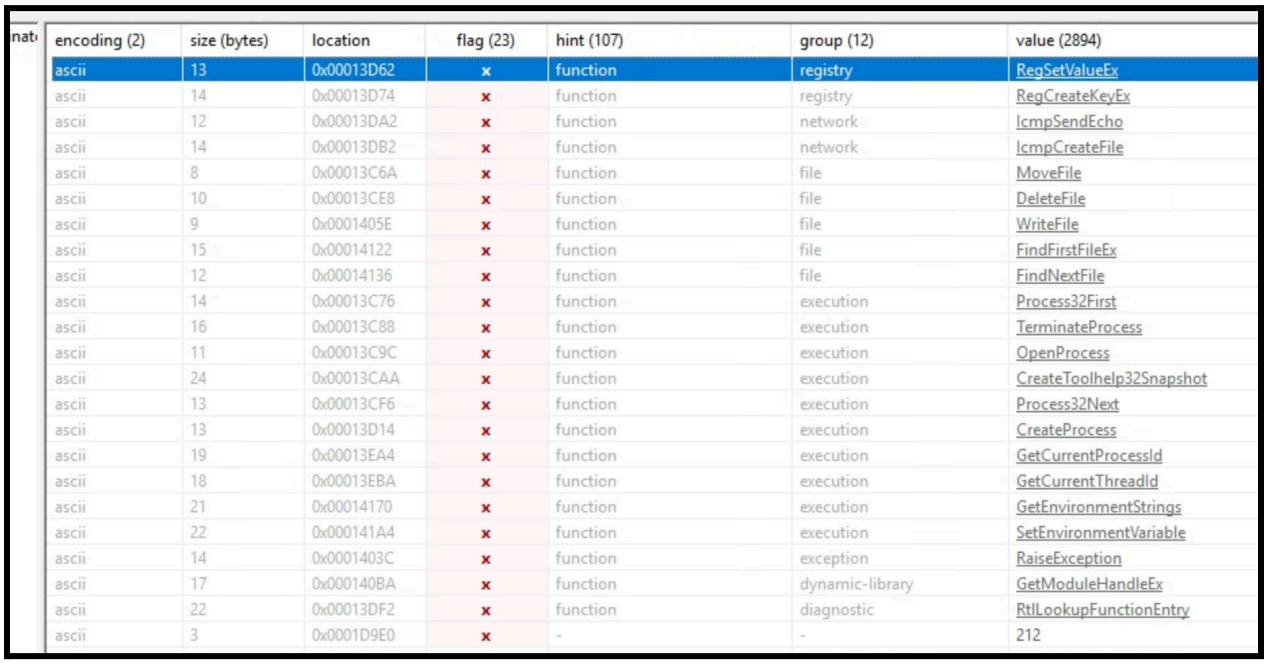
Libraries / DLLS



library (5)	flag (2)	bound (0)	type (1)	functions (86)	description
kernel32.dll	-	-	implicit	77	Windows NT BASE API Client DLL
user32.dll	-	-	implicit	1	Multi-User Windows USER API Client DLL
advapi32.dll	-	-	implicit	4	Advanced Windows 32 Base API
iphlpapi.dll	x	-	implicit	2	IP Helper API
ws2_32.dll	x	-	implicit	2	Windows Socket 2.0 32-Bit DLL

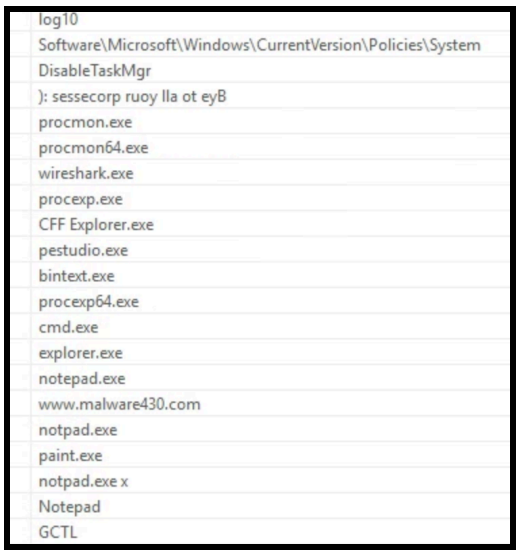
This screenshot shows the different imported dlls within the sample. The iphlpapi.dll and ws2_32.dll libraries are a bit suspicious as they both have to do with IP and creating windows sockets. The functions from these should be looked into.

Strings



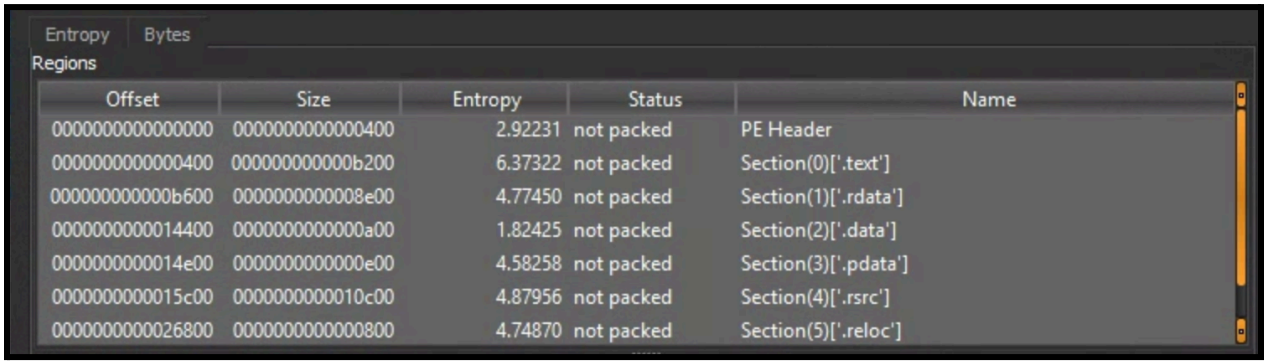
encoding (2)	size (bytes)	location	flag (23)	hint (107)	group (12)	value (2894)
ascii	13	0x00013D62	x	function	registry	RegSetValueEx
ascii	14	0x00013D74	x	function	registry	RegCreateKeyEx
ascii	12	0x00013DA2	x	function	network	IcmpSendEcho
ascii	14	0x00013DB2	x	function	network	IcmpCreateFile
ascii	8	0x00013C6A	x	function	file	MoveFile
ascii	10	0x00013CE8	x	function	file	DeleteFile
ascii	9	0x0001405E	x	function	file	WriteFile
ascii	15	0x00014122	x	function	file	FindFirstFileEx
ascii	12	0x00014136	x	function	file	FindNextFile
ascii	14	0x00013C76	x	function	execution	Process32First
ascii	16	0x00013C88	x	function	execution	TerminateProcess
ascii	11	0x00013C9C	x	function	execution	OpenProcess
ascii	24	0x00013CAA	x	function	execution	CreateToolhelp32Snapshot
ascii	13	0x00013CF6	x	function	execution	Process32Next
ascii	13	0x00013D14	x	function	execution	CreateProcess
ascii	19	0x00013EA4	x	function	execution	GetCurrentProcessId
ascii	18	0x00013EBA	x	function	execution	GetCurrentThreadId
ascii	21	0x00014170	x	function	execution	GetEnvironmentStrings
ascii	22	0x000141A4	x	function	execution	SetEnvironmentVariable
ascii	14	0x0001403C	x	function	exception	RaiseException
ascii	17	0x000140BA	x	function	dynamic-library	GetModuleHandleEx
ascii	22	0x00013DF2	x	function	diagnostic	RtlLookupFunctionEntry
ascii	3	0x0001D9E0	x	-	-	212

Some notable strings are RegSetVale, CreateKey, ICMP send Echo, ICMP Create File, Moving, writing, and deleting file, etc. As mentioned before we can see some of the different functions from the suspicious dlls within the strings. ICMPsendEcho is from the iphlpapi.dll. (This function is explained in deliverable 3)



Some more strings can be seen above. A registry key location, disabling task manager and a list of many forensic programs. These values are good to investigate. Since we patched the sample we know their purpose was to close the active processes with these in it.

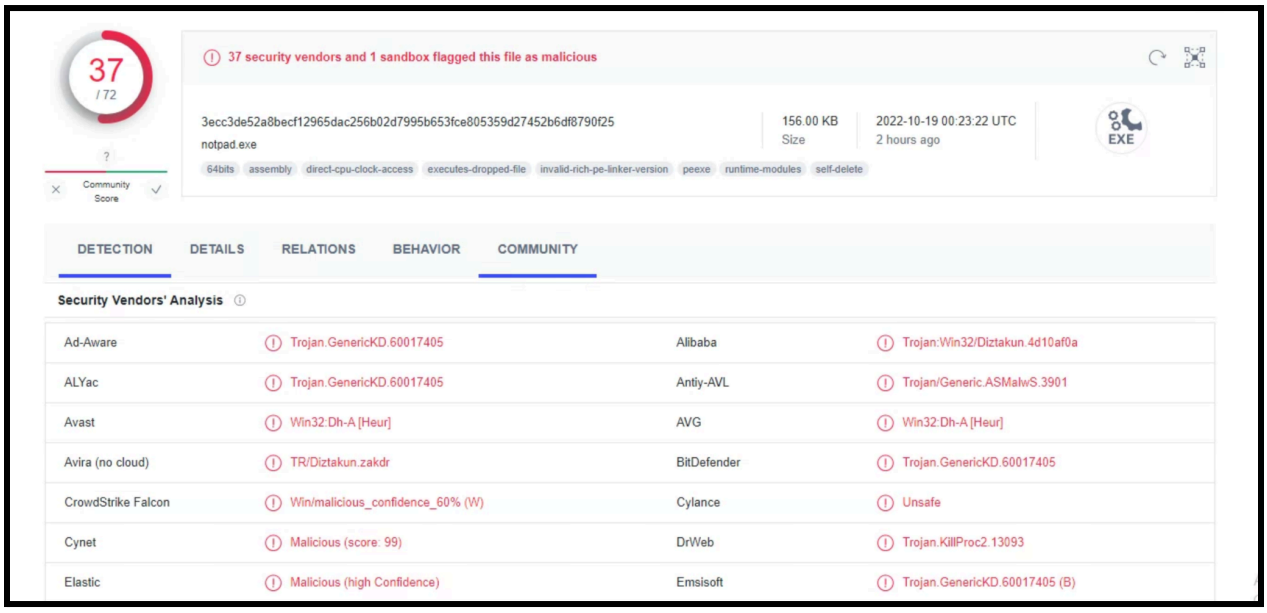
Entropy and Packing



Offset	Size	Entropy	Status	Name
0000000000000000	0000000000000400	2.92231	not packed	PE Header
0000000000000400	000000000000b200	6.37322	not packed	Section(0)['.text']
000000000000b600	0000000000008e00	4.77450	not packed	Section(1)['.rdata']
0000000000014400	0000000000000a00	1.82425	not packed	Section(2)['.data']
0000000000014e00	0000000000000e00	4.58258	not packed	Section(3)['.pdata']
0000000000015c00	0000000000010c00	4.87956	not packed	Section(4)['.rsrc']
0000000000026800	0000000000000800	4.74870	not packed	Section(5)['.reloc']

Using Detect It Easy, we can see that there are 5 different sections within the sample. They use standard names and the entropy suggests the sample is not packed.

Virus Total

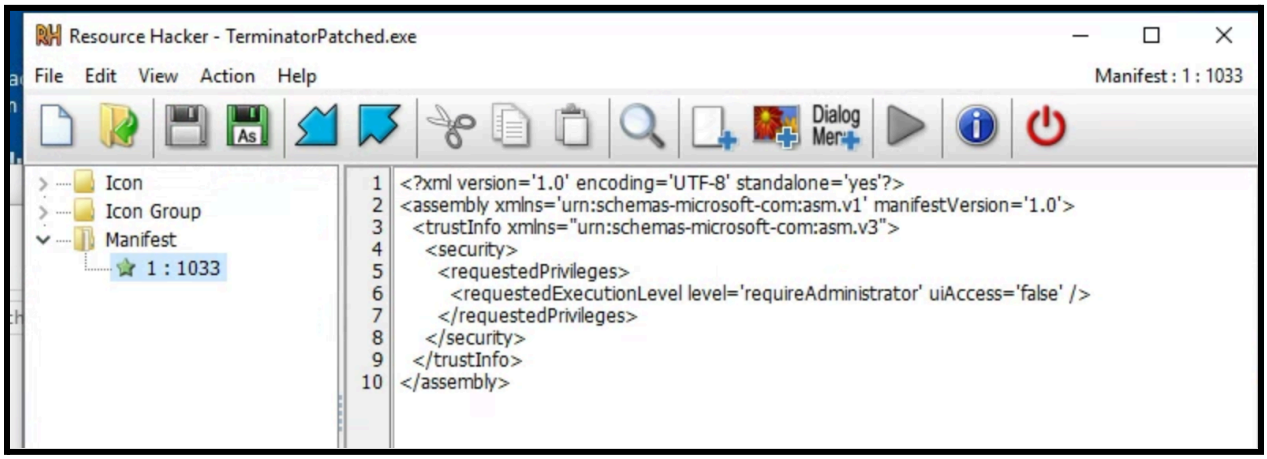


The sample has a 37 out of 72 score on virus total. It is thought to be a form of trojan. We know from analyzing the assembly that it closes any process with the names of forensic programs, switches the mouse button, and adds persistence registry keys.

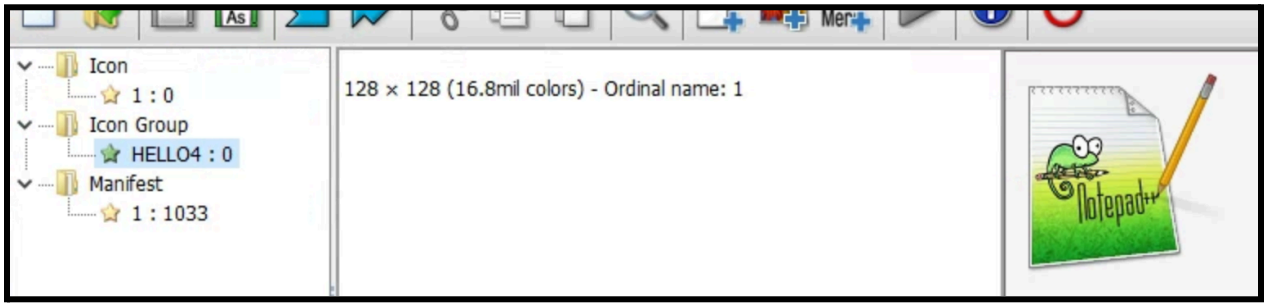
This screenshot shows some of the Mitre Att&ck Techniques that the sample uses such as process injection, masquerading, sandbox evasion, and more.

Resource Hacker

Resource hacker is helpful when looking at the manifest of the file. This gives information such as the schema used, the execution levels, and sometimes more. The manifest for the sample can be seen in the screenshot below.



The next screenshot is the image file for the samples icon. It is a notepad++ icon. This lets us know that the program is trying to disguise itself as notepad.exe. This is further correlated in the fact that it creates a new process called notepad once it is run.

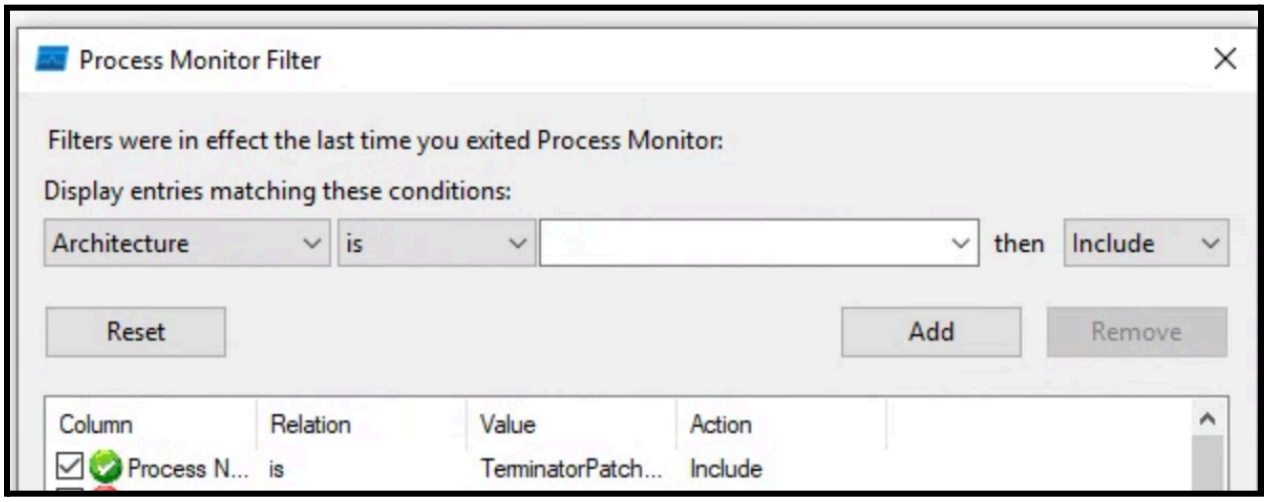


Deliverable 2 - Dynamic Analysis

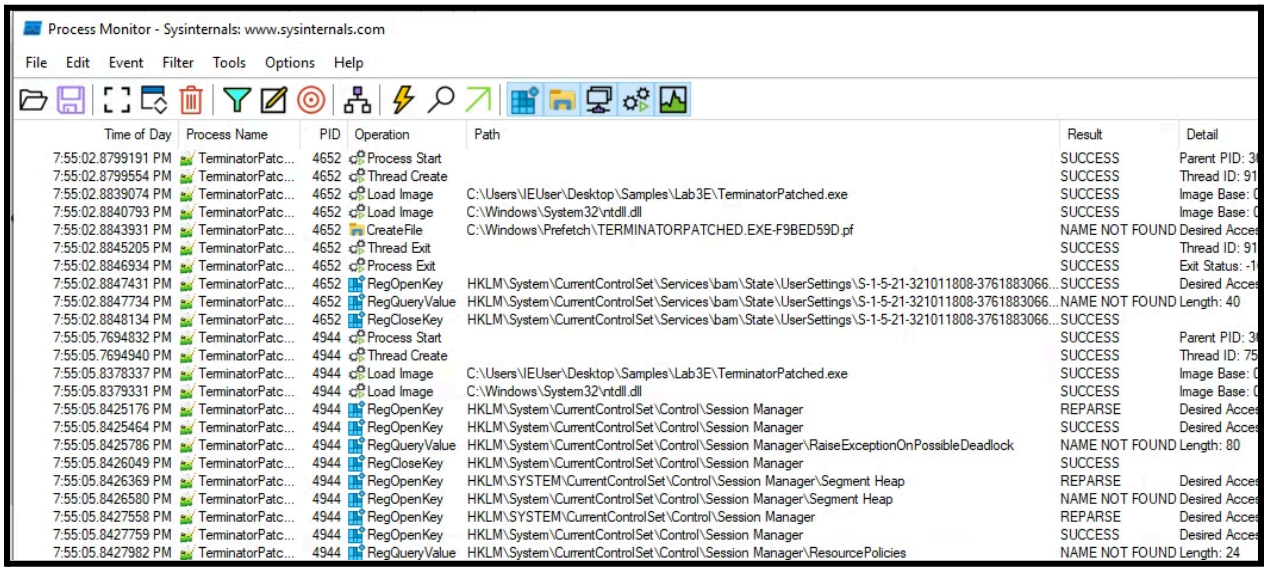
The first step to dynamic analysis is to run the samples with various tools that can capture information regarding the samples. One tool that does this is Process Monitor. Process Monitor can capture artifacts like registry activity, file system activity, network activity, and process activity.

Process Monitor

In order to set up Process Monitor to capture the samples activity, a filter with samples process name should be applied. This can be seen in the screenshot below next to the green check mark. A filter with the process name TerminatorPatched.exe has been applied.



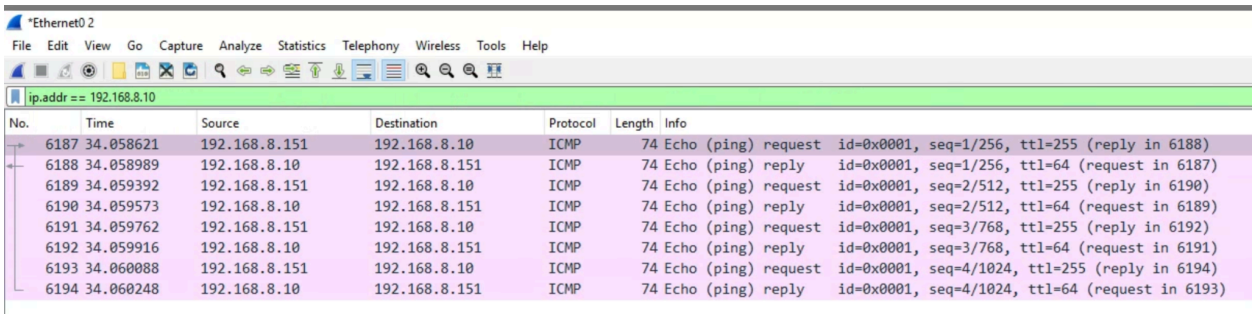
Next, analysis of the activity should take place. This can be seen in the screenshot below.



Instead of parsing through each piece of information individually, another program called ProcDot will be used to visualize the execution steps of the sample. Before Procdot can be run, another two tools need to be run first.

Wireshark

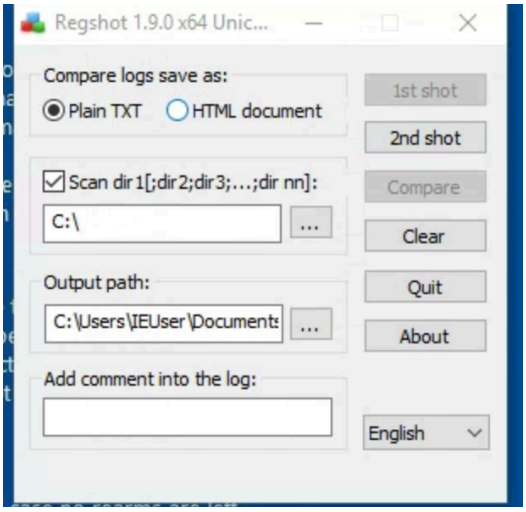
Many forms of malware will attempt to reach out to foreign network addresses to pull files, create reverse connections, and more. In order to capture this traffic we can use a tool called Wireshark. This tool allows for the capturing of network packets and the screenshot below shows the important ones for this sample.



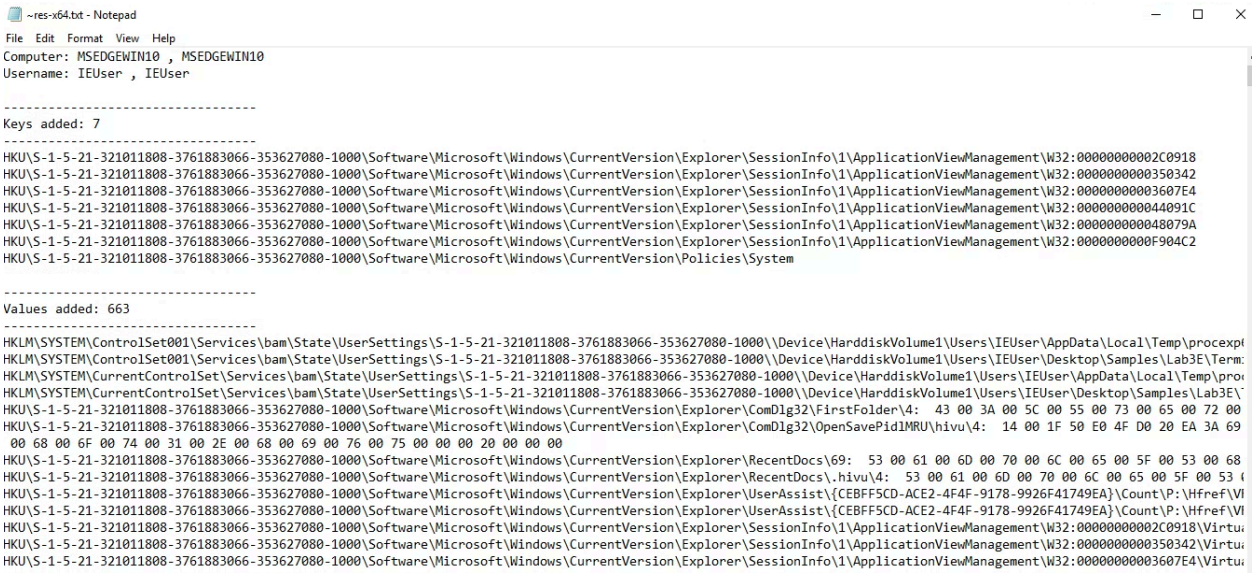
The screenshot above has an IP filter on it for the malware430 domain. In an actual case, to find this information for a filter it could be found in strings or slowly filtering the packet capture. This information can be saved as a pcap file and loaded into procdot to visualize what the sample might be doing.

RegShot

The next thing that needs to be done is taking a RegShot image before and after running the sample and compare it. This is helpful later on when we know more specifics on how the program works and can look at the specific keys that were changed.



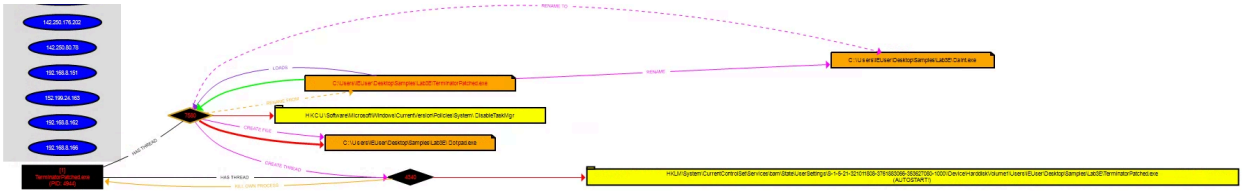
The screenshot above shows the menu for regshot. The screenshot below shows a piece of the compared registry image files output from regshot.



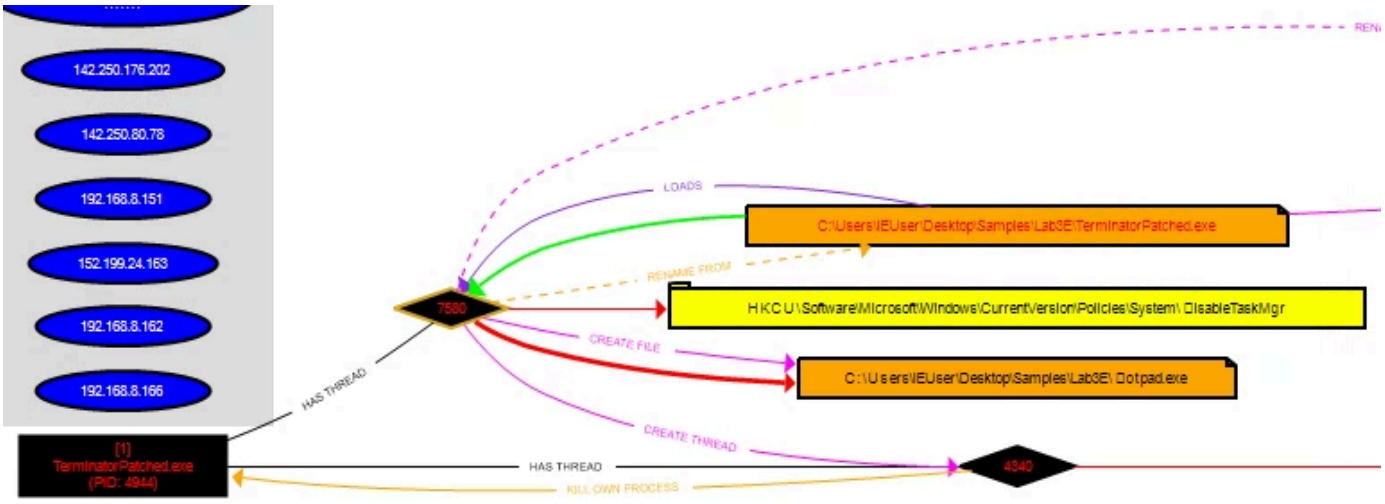
We will come back to this screenshot later when looking at the files.

ProcDot

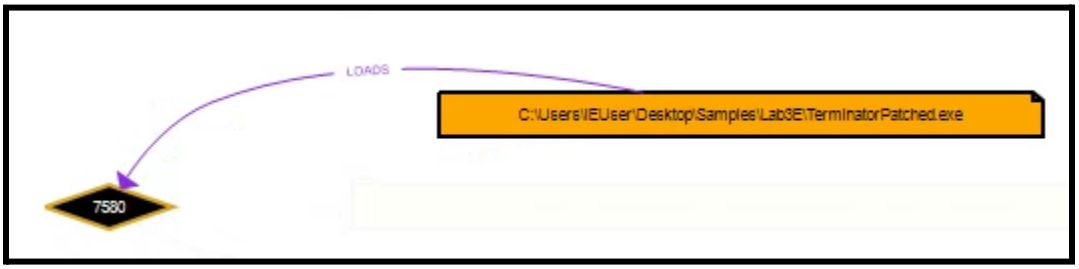
ProcDot is a great tool for showing how samples work behind the scenes by connecting their processes, registry activity, and more from the previously talked about tools into a visual graph and timeline



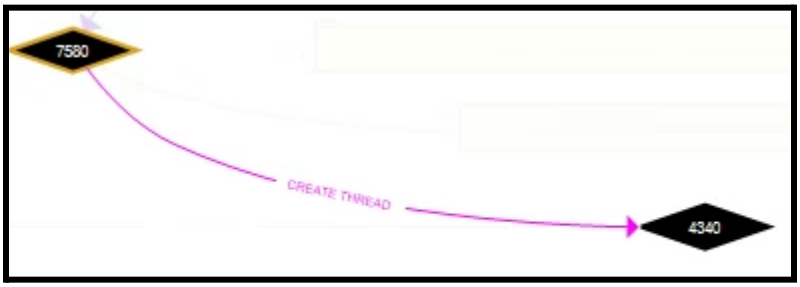
This screenshot shows the whole flow of the sample. It is hard to see so it is broken down below.



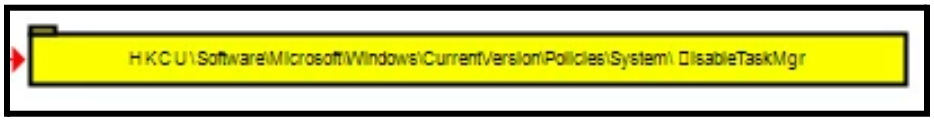
This screenshot shows many of the steps the sample took. We will look at what it does step by step next.



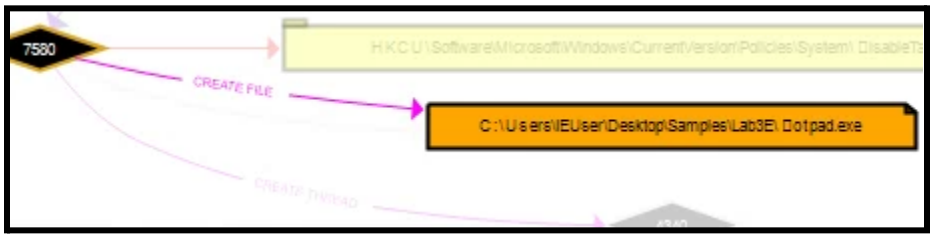
First the sample TerminatorPatched.exe loads itself into thread 7580.



The exe then creates a new thread 4340



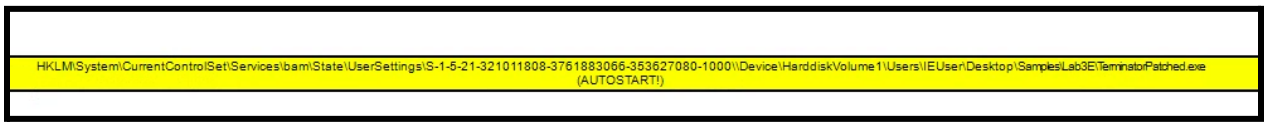
Once the new thread is created, it then sets the registry key to DisableTaskMgr. However, because we patched the sample, it searches for ~\HKCU\<path>\isableTaskManager. Without the first letter. This results in the reg key not changing as it cannot find the file.



After it sets the registry key, it creates a file called notepad.exe. This is because we removed the first letter of notepad.exe when patching. The thread then reads data from the patched exe and places it in the new file it created.

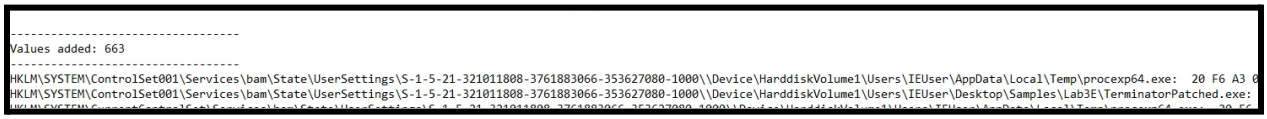
The sample then renames the file terminatorPatched.exe to aint.exe. This is similar to the new file above, but because we nulled out the first letter when patching it doesn't say paint.exe.

After reading a bit more information from the TerminatorPatched.exe file, it then kills its own process.



The new thread it created earlier then sets an auto start registry key under:
HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-321011808-3761883066-353627080-1000\\Device\HarddiskVolume1\Users\IEUser\Desktop\Samples\Lab3E\TerminatorPatched.exe
This allows TerminatorPatched.exe to start

This is a perfect time to look at our regshot registry image data. We can see in the screenshot below the registry key mentioned above and that it had a value added.



Deliverable 3 - Function Descriptions

Check at least 5 imported functions on MSDN that are important to the behavior of this given sample and explain what they are used for. Use the table below to add your response. An example is listed below.

API (function)	DLL	Explanation	Reference
RegSetValueExA	advapi32.dll	This function is used to set the data and type of the specified value in / under a registry key. This function has a few parameters: hkey, valuename, reserved, type, and data. The only one that is not optional is the value name.	Link
RegCreateKeyExA	advapi32.dll	This function is used to create a specified registry key. If the key already exists, the key is then opened instead. This function has 9 different	Link

		parameters. Hkey, subkey, reserved, class, options, desired sam, security attributes, result, and disposition.	
ICMPSendEcho	iphlpapi.dll	This function sends out an ICMP IPv4 echo request and will return any response replies. This function takes in 8 parameters ICMP handle (the open handle that is returned), Destination address, request data (a pointer to a buffer that contains data to send in the request), request size (the size in bytes of the request data buffer pointed to), reply buffer, reply size and timeout.	Link
ICMPCreateFile	iphlpapi.dll	This function opens a handle on which ICMP IPv4 echo requests can be issued. On success, it returns an open handle on success. On failure, the function returns INVALID_HANDLE_VALUE.	Link
WriteFileA	kernal32.dll	This functions writes data to the specified file or input/output device. It is designed for both synchronous and asynchronous operation. It has a few parameters hfile handle, lpbuffer, number of bytes to write, number of bytes written, and overlapped. The first three are the only required ones.	Link

Deliverable 4 - Reflection

This lab for some reason took me the longest of any lab so far. I decided to focus a bit more on my formatting and explanations. I had a lot of fun patching the sample and decided to challenge myself and patch the mouse button swapping. Hopefully I went into enough detail about the different artifacts and tools. Overall, fun lab!