



Channel Islands

CALIFORNIA STATE UNIVERSITY

Voyager: Tracking Via a Click

A Thesis Presented to

The Faculty of the Computer Science Department

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Student Name:
Samuel DECANIO

Advisor:
Dr. Soltys

October 2020

© 2020
Samuel Decanio
ALL RIGHTS RESERVED

APPROVED FOR MS IN COMPUTER SCIENCE

Advisor: Michael Soltys

Date

Name

Date

Name

Date

APPROVED FOR THE UNIVERSITY

Name

Date

Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Title of Item

3 to 5 keywords or phrases to describe the item

Author(s) Name (Print)

Author(s) Signature

Date

Voyager: Tracking Via a Click

Samuel Decanio

November 4, 2020

Abstract

Attribution, the ability to match events on the Internet to actors who caused them, is a difficult problem in Cybersecurity and Digital Forensics. The Internet was not designed to track behavior of users [8]. In fact, being an easy to access and open platform, it is often thought of as synonymous with anonymity. When actors attempt to take advantage of this anonymity to break the law, it is up to law enforcement to track them down, in a way that can potentially hold up in a court of law. This is no easy feat though, and the odds are stacked against them. In this paper a tool is presented, called Voyager, based on the idea of a *tracking pixel*, meant to help investigators with problem attribution.

Contents

1	Introduction	3
2	Background	7
2.1	AWS	7
2.1.1	S3	7
2.1.2	EC2	9
2.1.3	EBS	10
2.2	The OSI Model	10
2.3	The TCP/IP Model	11
2.4	HTTP	13
2.4.1	User Agent Strings	14
2.4.2	HTTP Methods	16
2.5	The Southern California High Technology Task Force	18
2.6	Combating Child Pornography	19
2.7	Mortgage Scams	20
2.8	Ransomware Attacks	21
3	Design and Implementation	25
3.1	Hosting Server	26
3.2	Provisioning	30

3.3	Web Interface	39
3.3.1	Front End	40
3.3.1.1	Home Screen	40
3.3.1.2	Photo Log	41
3.3.1.3	Photo Details	42
3.3.1.4	Event Log	44
3.3.1.5	Event Details	45
3.3.2	Back End and Server Side	47
3.4	Data Storage	49
3.5	Capturing Network Traffic	52
3.6	Analyzing The Traffic	55
4	Observations and Results	58
4.1	Real World Usage and Results	58
4.1.1	Case 1	58
4.1.2	Case 2	60
4.2	Exploratory Results	62
4.3	Observations	70
5	Conclusion and Future Work	74
5.1	Future Work	74
5.1.1	Pcap Files	74

5.1.2	Authentication	76
5.1.3	Leverage Additional AWS Resources	76
5.1.4	Addressing VPNs, Proxies, and TOR	77
5.1.5	User Agent String Analysis	78
5.1.6	Addressing HTTPS	79
5.2	Conclusion	80
References		85

List of Figures

1	The seven layers of the OSI Model. [1]	11
2	The four layers of the TCP/IP Model. [1]	11
3	The Internet Protocol (IP) header structure. [17]	13
4	User Agent strings showing the device type.	15
5	Example tags applied to Elastic Cloud Compute (EC2) instances. [3]	29
6	Diagram of the installation script process.	31
7	The Voyager directory structure.	34
8	The cron command to run the parser.py script.	35
9	Voyager's logging structure.	37
10	The command to configure the iptables port routing.	38
11	Voyager's home screen.	41
12	Voyager's photo log page.	42
13	Voyager's photo details page.	43
14	Voyager's event log page.	44
15	Voyager's event details page.	46
16	Voyager's event details page displaying geolocation data.	47
17	The Voyager work flow for file uploads.	49
18	Voyager's database schema.	50
19	The tcpdump command used to capture network traffic.	52

20	Breakdown of an example URL.	57
21	Example ransomware message, courtesy of the SCHTTF. . . .	61
22	Example email containing a Voyager link disguised as a thumb-nail, courtesy of the SCHTTF.	62
23	Using Google Chrome on a macbook pro connected to a home WiFi network. (Scenario 1)	64
24	Using the Safari browser on an iPhone 11 connected to a home WiFi network. (Scenario 2)	65
25	Using the Safari browser on an iPhone 11 with 4g cellular data. (Scenario 3)	67
26	Using Google Chrome to view a Gmail image attachment on a macbook pro connected to home WiFi. (Scenario 4)	68
27	Using Safari browser on an iPhone 11 from public WiFi. (Scenario 7)	70

Glossary

AMI Amazon Machine Image. 9, 27

API Application Programming Interface. 17, 32, 36, 37, 51, 52

AWS Amazon Web Services. 7–10, 26–28, 30, 72, 73, 75

CP Child Pornography. 5

EBS Elastic Block Store. 10, 28, 73

EC2 Elastic Cloud Compute. iv, 9, 10, 26–30, 36, 37, 39, 55, 72, 75

HTTP HyperText Transfer Protocol. 13–17, 29, 38, 51, 54, 56

HTTPS HyperText Transfer Protocol Secure. 14, 29, 54, 56

IP Internet Protocol. iv, 12, 13, 26–29, 45, 59, 61–66, 68, 69, 71, 78, 80

NVM Node Version Manager. 35

OSI Open Systems Interconnection Model. 10, 11

PM2 A daemon process manager used to run NodeJS applications. 32, 33,
38, 39

S3 Simple Storage Service. 7, 8, 32, 33, 38, 55, 73, 75

SCHTTF Southern California High Technology Task Force. 5, 18–21, 24,
25, 58, 59, 80, 81

TCP Transport Control Protocol. 12, 13, 29, 56

TOR The Onion Router. 77, 78

UAS User Agent String. 14

VPN Virtual Private Network. 59, 63, 70

1 Introduction

With the creation and subsequent adoption of the internet the world became more connected than it has ever been. Accompanying this came massive changes to our everyday lives. The way people communicate, conduct business, and learn are just a few examples of areas that have been forever changed by the Internet. But, with these advances came a pitfall as well. Just as in every other facet of the world there are those who wish to take advantage, the internet is no different. In fact, the internet has proven to be a proverbial playground for these individuals due to the somewhat anonymous nature as well as the Internet's inherent ability to blur lines between countries, continents, and all else. In addition to these factors, the world has steadily moved to making more available via the internet, ranging from personal and private information to services such as banking and other important infrastructure. For all these reasons many individuals looking to conduct nefarious activities have moved online too. These actors, along with any others using the internet with the intent of breaking the law, are to be referred to as cybercriminals in this thesis.

Just as in the regular world there are those working to bring criminals to justice, there are those working to stop cybercriminals as well. Sadly, these law enforcement investigators are often fighting an uphill battle. The sheer number of crimes committed online is enough to overwhelm law enforcement.

In 2019, the IC3 or Internet Crime Complaint Center, a division of the FBI received 467,361 complaints [10]. That is a 32.79% increase from 2018, and a 62.27% increase from 2015 [10]. It is clear that the frequency of cybercrimes committed is increasing, and shows no signs of slowing.

While defenders must discover every hole, bug, and flaw in their systems to keep attackers out, the criminals only need to find one to get in. Cybercriminals have a plethora of tools at their disposal to both help discover these security holes as well as cover their tracks and keep them from being discovered. Add to this that investigations often do not take place in real time but long after an incident has occurred, when much of the digital evidence is potentially gone, and you begin to paint a bleak picture for these investigators. It was with these facts in mind that it became clear law enforcement investigators need to look at adopting new approaches to combat cybercriminals.

In this thesis we propose a solution and tool to help combat the aforementioned problems, named Voyager. Voyager is loosely based on the idea of a tracking pixel, capable of automatically loading when a URL containing it is requested. The goal being that the requester of the page is none the wiser as this tracking pixel has no effect on the rendering of the page. Through this tracking pixel, information is then logged about the requested page as well as whoever requested it. This approach is often used in online marketing.

Through a similar approach used by companies in online marketing, Voyager was developed to be able to log a plethora of information about a suspect who accesses a specially crafted link that is created and served by law enforcement investigators. In a sense Voyager is capable of conducting a reverse phishing scheme on cybercriminals allowing law enforcement to identify, track, and apprehend these malicious actors.

Voyager was developed in direct support of the Southern California High Technology Task Force (SCHTTF). More specifically, Voyager began as an application to be used in combating online Child Pornography (CP). The premise behind the idea being that investigators would be able to upload files ranging from images to PDFs to Voyager. Then, Voyager would serve them at a specially crafted URL which investigators would then provide to suspects. Once the suspect opens the link, Voyager will record information about that request that law enforcement could then use in their investigation. However, due to the broad nature of Voyager’s implementation its quickly expanded beyond simply combating CP. Investigators have also leveraged Voyager in the same previously mentioned manner to entice suspects of other cybercrime such as mortgage scams and ransomware attacks, both of which are discussed in Section 2. It is worth noting that another California State University, Channel Islands student, Dhruv Pandya, worked on a solution to the same problem for his master’s thesis [15].

This project was first developed as an undergraduate capstone between

Vlad Synnes and myself. While the result of that capstone was a working application, Voyager 1.0, it was in a primitive stage. It was capable of capturing network traffic and displaying a limited amount of information. This thesis served to build off that foundation to develop a more comprehensive and fleshed out version of Voyager. This includes fixing bugs, adding additional functionality and features, adopting a microservice oriented project structure, as well as completely rewriting the project from the ground up using a more industry standard development stack.

As a last note, a shortened version of this thesis paper was presented at the KES 2020 International conference and subsequently published in volume 176 of the Procedia Computer Science journal [4].

2 Background

2.1 AWS

Amazon Web Services (AWS) is a cloud computing platform with a large number of services. Cloud computing is the on-demand availability of computing resources over the internet. These range from on demand virtual computing instances, long term data storage, and databases to some bleeding edge technologies such as quantum computing, machine learning capabilities, and virtual reality. It is leveraged by millions of companies and consumers; from startups to large enterprise corporations such as Netflix and Capital One, and even government agencies. AWS is the largest Cloud provider in the world.

2.1.1 S3

Simple Storage Service (S3) is a functionality offered by AWS that provides storage of objects while offering availability, security and performance. S3 has a plethora of use cases ranging from big data analytics to disaster recovery and data archival. S3 serves as a financially feasible method of storing large amounts of data for extended periods of time while still offering many useful security features. For instance, S3's standard storage costs \$0.023 per

gigabyte per month for the first 50 terabytes of storage [2]. It is worth noting these prices vary slightly depending on the AWS region selected. For the purposes of Voyager, S3 has two specific use cases.

The first use case is to store data required for the provisioning of Voyager, discussed in Section 3.2. This data is stored in an S3 bucket, which can be thought of as similar to a file folder. Buckets hold objects, which are defined as data and that data's associated metadata. Voyager then retrieves the necessary data from the S3 bucket during the provisioning process.

The second use case is for the storage of all network traffic that Voyager records. Network traffic captures can quickly grow to become an overwhelming amount of data, yet maintaining it is vital for the investigations Voyager aides in. S3 allows for this large amount of data to be moved off the Voyager instance to a more long term and cost effective storage medium.

Should any data need to be archived for an even longer period of time, in the event of a court case down the road, S3 offers Glacier storage as an additional option. Glacier is specifically for long term backups and archives and as such provides extremely inexpensive storage options, with the catch being the time required to access the data. Should you need to access data stored using Glacier, it can take anywhere from 1 minute to 12 hours to retrieve it. Pricing for Glacier storage is roughly \$0.004 per gigabyte per month [2]. Similarly to the standard S3 prices, these vary slightly based

upon the AWS region selected.

2.1.2 EC2

Elastic Cloud Compute (EC2) instances, are on demand computing instances that can be spun up and spun down at a moments notice, all hosted on Amazon's infrastructure and managed through AWS's command line interface, software development kit, API, or web interface. They are a secure, resizable, and highly configurable compute capacity in the cloud. The beauty of these EC2 instances is that they can be created and destroyed at anytime, from anywhere. Not only can they be created at a moment's notice but they are cost effective to use when compared to an in house hosting solution.

Their creation is simplified by the ability to select a pre-configured image, referred to as an Amazon Machine Image (AMI). These AMIs come in many different flavors. The main differences between them is their operating systems as well as the packages and services that come already installed. Leveraging AMIs allows for a much easier provisioning process than building the server from the ground up.

2.1.3 EBS

Elastic Block Store, commonly referred to as EBS, is storage that can be attached to AWS EC2 instances. EBS offers persistent storage for EC2 instances when otherwise all data on the instance would be lost at shutdown. They also provide an easy means of encrypting whatever data is stored on the instance, providing another layer of security. Upon creating an EC2 instance using the standard options, an EBS will also be created for that instance. Further configuration can be done to increase capacity however the standard configuration has proven more than enough for all uses of Voyager thus far.

2.2 The OSI Model

The OSI or Open Systems Interconnection Model is a conceptual model that serves to define and standardize communication functions of computer systems. It does so with no thought to the underlying structure a computer system uses to implement these functions. This communication model is necessary to define a set of common ground rules that ensures all computers can communicate with each other.

The OSI model consists seven distinct layers, shown in Figure 1.

Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Figure 1: The seven layers of the OSI Model. [1]

2.3 The TCP/IP Model

Equally important as the OSI model is the TCP/IP model. This model consists of only four layers, as opposed to the seven of the OSI model. While the OSI model is responsible for defining and standardizing communication functions of a computer system, the TCP/IP model is a more protocol oriented standard. The four layers of the TCP/IP model are shown in Figure 2.

Application Layer
Transport Layer
Internetwork Layer
Network Access Layer

Figure 2: The four layers of the TCP/IP Model. [1]

There are two main protocols used on the modern Internet to transmit data, both of which operate on the Transport layer of the TCP/IP model. The first of which is TCP or Transmission Control Protocol. The second, which we are not concerned with for the context of this paper, is UDP or User Datagram Protocol. TCP, which was developed by ARPA (otherwise known as DARPA) [1], was designed to be a “highly reliable host-to-host protocol” [18]. This reliability was accomplished via the implementation of sequence numbers and acknowledgements. This means each packet of data that is transmitted between two hosts is assigned a sequence number and each host must acknowledge the receipt of any data. TCP is also responsible for handling the continuous flow of data between two sources and resolving any associated problems such as congestion.

In order to work over a network, such as the internet, TCP works in association with IP or the Internet Protocol. IP, which operates at the Internetwork layer of the TCP/IP model [1], defines how computers send packets of data to each other, essentially the routing between two destinations. This is accomplished by attaching a header that contains metadata such as addressing and control information. Shown in Figure 3 is the structure of the header that gets attached to data as part of the IP. This header contains information that is very useful to Voyager such as the source and destination addresses. These denote where the data originated and where it is being sent to.

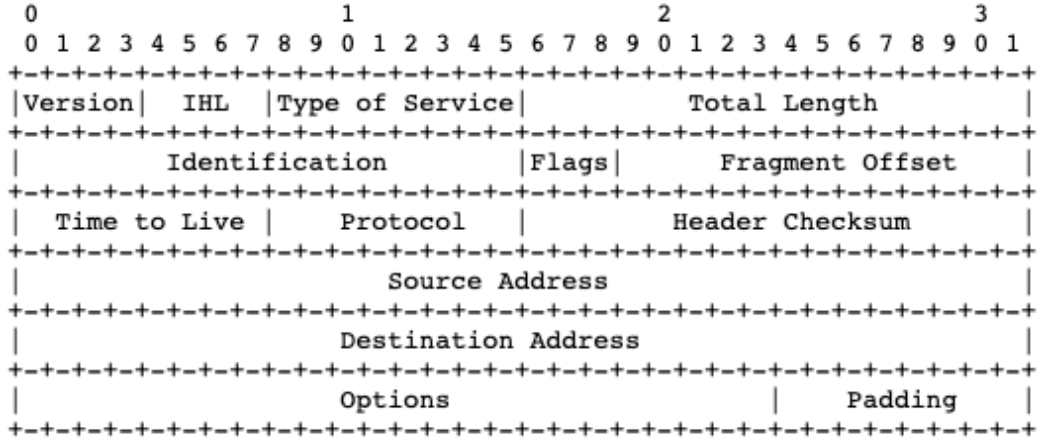


Figure 3: The IP header structure. [17]

TCP/IP is relevant to Voyager because all the network traffic the server captures and parses is TCP traffic transmitted over the Internet using the Internet Protocol. In order to parse the traffic received it is important to understand the structure of the TCP/IP model, and the protocols associated with it.

2.4 HTTP

HTTP or HyperText Transfer Protocol is a protocol whose primary purpose is serving hypermedia documents such as HTML or Hypertext Markup Language. While HTTP can be leveraged over any reliable transport layer protocol, though most commonly TCP, HTTP itself operates at the Application Layer [1]. The protocol implements communication between client

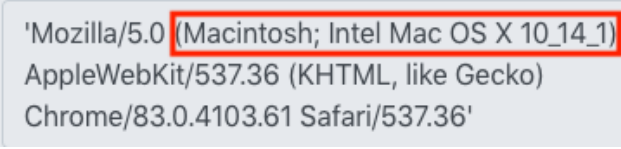
and server computers via the transmission of HTTP requests and HTTP responses. In any instance where the URL of a website is prepended with HTTP or even HTTPS then that website is utilizing the HyperText Transfer Protocol. HTTP serves a crucial role in Voyager as it is both the protocol that investigators use to interact with the web application as well as how a suspect will access a file from a Voyager link. When this happens, HTTP contains much of the useful information Voyager extracts and logs from these requests.

2.4.1 User Agent Strings

The User Agent string (UAS), is a field included in all HTTP requests. It is a string used to “convey client system configuration details to ensure that content returned by a server is appropriate for the requesting host” [12]. The purpose of a UAS is “to provide sufficient detail about a client system to enable a server to transmit content in the appropriate format and for debugging interoperability problems” [12].

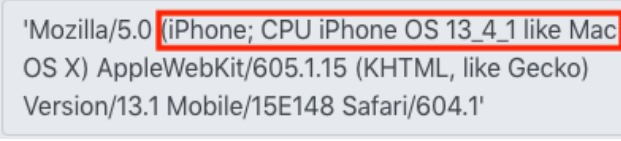
Websites will use the User Agent string to determine the format of the content to return to the user [13]. You can see an example of this yourself when you visit websites from your mobile device. Often times you will be automatically redirected to a mobile friendly version of that website. These mobile friendly versions are built to be viewed and used on the smaller screens

of devices like cell phones and tablets. Programmers can determine which version of the website to serve a user by examining the User Agent string. If the string contains information stating that the device that sent the HTTP request is a mobile device, then the user is redirected. This can be observed in Figure 4. Displayed in the figure are two different User Agent strings. The first was from a HTTP request sent from a desktop computer, while the second was from a HTTP request via a mobile device. The highlighted portions show the parts of the User Agent strings that can be examined to determine the device types.



```
'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_1)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/83.0.4103.61 Safari/537.36'
```

((a)) An example desktop computer user agent string.



```
'Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac
OS X) AppleWebKit/605.1.15 (KHTML, like Gecko)
Version/13.1 Mobile/15E148 Safari/604.1'
```

((b)) An example mobile device user agent string.

Figure 4: User Agent strings showing the device type.

In the case of Voyager, these User Agent strings can be extracted and used to find out additional information about whomever made the request. These strings can contain details such as the computer's operating system and the browser used to make the request even down to specific version numbers. An important note is that User Agent strings are public information sent in all HTTP traffic, and thus do not require a search warrant to collect or inspect.

This makes them an excellent means of open source intelligence gathering for law enforcement.

2.4.2 HTTP Methods

Within HTTP there are a total of 8 different methods that can be leveraged in HTTP 1.1, the most popular version of the protocol currently in use on the internet. These methods define specific actions to be performed by a resource. The eight methods are: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, and CONNECT [7]. When it comes to traffic capture, Voyager is mainly concerned with the GET method implemented in HTTP. However, the portion of Voyager that investigators use leverages GET, POST, PUT, PATCH, and DELETE.

When an HTTP message is sent that uses the GET method, it is often referred to as a GET request. This is because the GET method is only used in HTTP requests and not in HTTP responses.

GET requests serve as a means to “retrieve whatever information is identified by the Request-URI” [7]. Whenever a client is requesting a resource from a web server, a GET request is sent to the server. This prompts the server to respond with the specific resource that was requested. These GET requests to the server are the traffic that Voyager captures. Voyager is then able to extract meta-data from these requests, as discussed in detail later.

The POST method is used in HTTP for the purpose of telling the server to accept the enclosed entity in the request as a new instance of whatever resource is denoted by the request-URI. This simply means POST requests are sent from the client to the server to create a new instance of some resource on the server. Many web applications, including Voyager, leverage POST requests to allow for the creation of new records via an API. In Voyager's case, POST requests are leveraged to both create new records within the database as well as to upload files to the server.

PUT is often used in a similar manner to the POST method. The main difference is that PUT is used to update a specified resource denoted by the request-URI if it already exists, instead of creating it. This allows for the updating of records that already exist on the server. This differs from a simple POST request, which is merely responsible for creating new records. However, if a PUT request is sent and the resource does not already exist on the server then it will be created. This behavior mimics how a POST request works. Within Voyager, PUT requests are used to update an existing record on the server, such as a database record or an uploaded file.

The PATCH method loosely resembles the PUT method. It is also used to update or modify resources that already exist on the server. The difference between PATCH and PUT is that when using the PATCH method you need only include the parts of the resource that you wish to be updated, as opposed to the entire new version of the resource. This is especially useful when

updating a single field in a large record, as you only need to include the single field instead of the entire record in the request.

2.5 The Southern California High Technology Task Force

The Southern California High Technology Task Force or SCHTTF is one of five task forces regional to California that was developed as part of the High Technology Theft Apprehension and Prosecution Program (HTTAP). The HTTAP was established in 1998 as a result of Senate Bill 1734 in order to “help combat computer-related crimes such as network intrusions, computer hacking, counterfeiting and piracy, theft of trade secrets, theft of high tech related equipment, and telecommunications fraud”[19]. According to a Ventura County Sheriff’s Office funding document, the SCHTTF’s jurisdiction spans the counties of Los Angeles, Orange, and Ventura.

Part of the work that the SCHTTF does is partnering with other entities to help prevent, detect, and respond to computer-related crimes. This allows them to leverage these partnerships to better respond to the growing amount of cybercrime taking place. One such partnership is with California State University, Channel Islands (CSUCI). This partnership affords students the opportunity to do meaningful work as well as gain experience through real world applications of their knowledge, while keeping the task force abreast of the latest in cybersecurity related academic developments. Voyager’s in-

ception and implementation is one such result of this partnership.

2.6 Combating Child Pornography

Criminals looking to exploit people for financial gain were not the only ones to transition to using the internet for their nefarious actions. Unfortunately many other types of crime and criminals have developed a presence on the internet, but perhaps none more despicable than pedophiles. Similarly to criminals using the internet to anonymously steal data and money, pedophiles have begun to leverage that same anonymity to protect themselves as well. This proves to be a difficult problem for law enforcement to address when attempting to bring these criminals to justice.

The initial inception of Voyager was originally oriented towards fighting online child pornography. Unfortunately, the SCHTTF had seen many cases of it and hoped to develop some way of helping to put a stop to it by identifying these individuals. Thus, the idea of Voyager was conceived, an application that would allow them to upload fake images of children as bait for these predators. While Voyager served this purpose it became apparent that it could be used in other scenarios as well. The SCHTTF then began using it in other types of investigations which has helped to steer its further refinement and feature development.

2.7 Mortgage Scams

One of the prevalent types of online crime that the SCHTTF deals with is mortgage scams. In this relatively simple scam, a criminal typically sends the victim, who is in the process of purchasing a home, instructions containing details on where to wire the down payment for their new home. The problem is that the account they are told to wire the money to is not the bank's but instead controlled by the criminal, typically through many proxies or mules. Mules are, knowing or unknowing, participants in the scam. Their bank accounts are used to store the stolen money, before it is transferred to the real criminals account. By using many mules and transfers the criminal is able to obscure the true path of the money, making it harder to law enforcement to find.

These emails are typically sent from one of two types of email addresses. The first, is from an email address that is made to look legitimate, but is not. This is commonly achieved by using similar spelling to the name of the bank but replacing, adding, or removing letter that may be harder to notice. The problem with this method is that an attentive victim may notice these inconsistencies and realize it is a scam. The second and ideal (in the mind of the criminal) way, is to hack the email account of a bank employee and then use that account to send these fraudulent emails. These emails can easily fool even security oriented individuals since they are coming from

a legitimate email address at the bank that the victim is more likely to recognize and trust.

In either scenario the recommended action is to directly contact the bank via a different communication medium, such as in person or over the phone, to confirm any such emails. An important note is to independently look up the phone number for the bank, rather than using the one included in the email. Often times these criminals will include a fake phone number that people will call and be reassured of the legitimacy of the email.

Worse yet, it is not an easy feat to recover the money that is lost due to this scam. Often times it is long gone, lost in a series of transfers between banks around the world, never to be recovered. While these mortgage scams are relatively simple and do not necessarily require advanced technical knowledge, they are effective. Sadly, many people in the SCHTTF's jurisdiction alone have fallen prey to this scam.

2.8 Ransomware Attacks

In today's technology driven world, people live online. This has caused an exodus of personal information and data onto devices such as our smartphones, personal computers and even our televisions. On these devices people store photos, business correspondence, financial information, conversations with loved ones, and all other forms of sensitive data. But, what happens when

the access to that data, which we all take for granted, is threatened or even taken away? Ransomware is a family of malware that exploits this fear. It “locks the victims’ computers until they make a payment to re-gain access to their data”[11]. This type of malware is especially scary for those who do not have backups of their data or facilities, such as hospitals and power plants, that cannot afford to lose their access to it for any period of time.

There are many different subcategories of ransomware but some of the most prevalent ones are scareware, screen lockers, and encrypting ransomware.

- **Scareware** is perhaps the most benign type of ransomware. Typically, scareware involves receiving many intruding pop-ups or notifications on your screen alerting you to the fact there is some sort of malware present on your system that can only be removed by paying. More often than not there is no actual malware, other than the scareware itself, on the system. Additionally, scareware does not typically destroy, modify, or block files on the system, meaning it is essentially safe; albeit rather annoying until removed.
- **Screen Lockers** do exactly what the name describes: they lock your screen. This prevents you from being able to use the system at all. This is achieved by overlaying the screen with a full size window that cannot be closed and contains some sort of message stating you must pay to have it removed. Often times the window will be made to look

official and claim it was put in place by the FBI, Police, Department of Justice, or some other law enforcement entity that detected illegal activity on the system. While screen lockers are often more tricky to get rid of than scareware, they also usually do not modify the actual files on the system. This makes them more annoying than dangerous as your data is relatively safe.

- **Encrypting Ransomware** is the most infamous and dangerous category of ransomware. This ransomware infects a computer and proceeds to encrypt all the files on it, preventing the user from being able to access them. This form of malware is especially nasty because the only way to decrypt the files is using the key, which the hacker possesses. More often than not, the hacker demands a payment in some form of hard to trace currency, such as bitcoin, to be made in order to decrypt the victim's files. Unfortunately, even after paying the ransom, there is no guarantee that the hacker will decrypt the files. Often times the hacker will simply disappear and leave the victim high and dry, no money and no decrypted files. If the prospect of potentially paying the hacker and having them disappear with your money and leaving the files lock was not scary enough; there are instances such as the Ryuk ransomware. Due to a bug in the malware's code, even when victims paid the ransom any files larger than 54.4MB [6] could not be decrypted successfully.

Unfortunately, instances of ransomware infecting unsuspecting victims occur frequently. In the case that the victim is a nearby business or other local entity that is specifically targeted and is being blackmailed for a ransom, the SCHTTF gets involved. In these cases it is important to consider that a hacker capable of this sort of attack has advanced technological knowledge and capability, often implementing many steps and protections to stay anonymous. The SCHTTF has deployed the first iteration of Voyager during some of these investigations in an attempt to reveal more information about the hacker that could potentially aid in their apprehension or recovery of the encrypted data.

3 Design and Implementation

Voyager's inception stems from the SCHTTF's need for additional tools to effectively combat the cybercrime they face on a day to day basis. Thus, it was developed based on feedback from their team as well as from Professor Soltys. Voyager consists of multiple distinct parts that come together to grant the aforementioned functionality. These distinct parts are as follows:

- Hosting Server
- Provisioning Script
- Web Interface
 - Front End
 - Back end
- Database Storage
- Network Traffic Capture
- Network Traffic Parser

3.1 Hosting Server

Hosting Voyager on a virtual machine in the cloud provides many advantages such as a degree of anonymity as well as ease of use. More specifically, Voyager is hosted on AWS using EC2 instances.

The functionality discussed in Section 2.1.2 along with ease of use and availability influenced the decision of how to host Voyager. The use of cloud computing, and more specifically AWS, lends Voyager the ability to be running independently on many instances at once. In particular, this allows Voyager to be used for multiple separate investigations without risk of data spillage between them because all the instances are siloed from one another. Additionally, the lack of overhead required to use EC2 instances, or AWS as a whole, means that law enforcement does not have to invest thousands into servers and all their associated infrastructure. Throughout development and testing, I have had multiple instances of Voyager continuously running. While these versions of Voyager are running on micro instances, without much computing power, it has cost no more than \$10 per month. Even scaling the instances up for real world use to handle more traffic and have better performance, the savings would still be drastic. Furthermore, if law enforcement were to use its own servers, those IP addresses would quickly become known, and that would render Voyager useless. But on AWS, investigators are able to open and close instances programmatically with new “unburned”

IP addresses.

Using the AWS website to create a new EC2 instance for Voyager is very straightforward. The process is as follows:

1. **Choosing an AMI.** Leveraging these pre-configured images allows the user to absolve themselves of the responsibility of need to completely build the server from the ground up. Additionally, it insures that Voyager is running on a server that is properly configured, using an image that has been thoroughly tested. This serves to reduce the likelihood of encountering any such errors related to the server’s creation or setup. Voyager was created to run on the “Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type”. The main reason why Voyager had to be created for a specific AMI is due to the difference in package managers across different AMIs. While it is likely that everything would work on another AMI that also uses the “yum” package manager, this was outside of the project’s scope and has not been thoroughly tested.
2. **Choosing an instance type.** Once the AMI has been chosen, the next step is to select an instance type. The different instance types dictate the hardware that the EC2 instance will run on and thus its performance. The user has control over this by selecting an instance with the appropriate amount of CPUs, memory, and level of network performance desired. Voyager is capable of running on any general pur-

pose instance type and the decision of which to use should be influenced by the amount of expected traffic to the instance.

3. **Setting configuration details.** This step has a plethora of options, which may be leveraged or ignored at the discretion of the user. In this section the user can configure many details of the instance being created, such as the IP address, IAM (Identity and Access Management) roles for the instance, as well as shutdown behavior. The most important option in this step is the “User Data” section under advanced details. This is where the provisioning script, which is responsible for setting up Voyager on the new instance, is included. This can be achieved by either copy and pasting the script’s text or uploading the file directly.
4. **Configuring storage and tags.** This step can be skipped, in which case AWS will create a default EBS setup for us and attach it to our instance. However, if the user wishes to modify the storage used on the EC2 instance then this is the step in which they have the option to do so. Additionally, the user can add any desired tags to the EC2 instance at this point. Tags are used to help categorize your AWS resources and consist of a key and an optional value as shown in Figure 5.

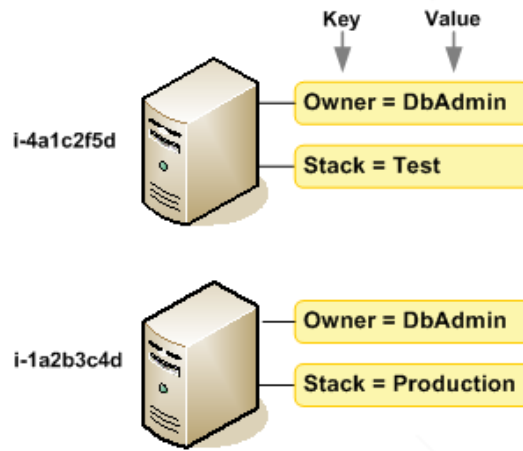


Figure 5: Example tags applied to EC2 instances. [3]

5. **Choosing security groups.** This step is required in order to allow network traffic to reach your EC2 instance. Each security group is essentially a set of firewall rules that control **inbound** traffic for your instance. It is important to remember that these rules only apply to inbound traffic and have no bearing on any traffic coming out of the instance. In the case of Voyager the default security group is left and two additional groups are added. One of the two additional groups allows HTTP and HTTPS traffic on ports 80 and 443 respectively. The other group allows any TCP traffic on port 9000 to reach the Voyager front end interface. The applied inbound network traffic rules are shown in Table 1. Note the last rule, which allows TCP traffic on port 9000, should have the source IP address restricted to only allow traffic from the investigator’s network. Since port 9000 is where the front end

web application lives, it should only be accessible by investigators and nobody else.

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	Allows user to SSH in to instance.
HTTP	TCP	80	0.0.0.0/0	Allows traffic to the uploaded artifacts.
HTTPS	TCP	443	0.0.0.0/0	Allows traffic to the uploaded artifacts.
Custom TCP Rule	TCP	9000	Investigators' Network	Allows user to access the Voyager front end.

Table 1: Inbound traffic rules.

3.2 Provisioning

In order for each EC2 instance to be quickly provisioned with Voyager, an installation script was created. It began as a simple bash script that was manually run by the user, via SSHing into the instance, but has since evolved to be more complex in efforts to make set up easier when spinning up new instances of Voyager. The current implementation of the installation script was created with the purpose of being run as a User Data script during the creation of an EC2 instance from the AWS console. The user simply includes it during the third step of the instance creation process by either uploading the file or copying the text and then completing the launch of a new EC2 instance, allowing AWS to take care of running the script.

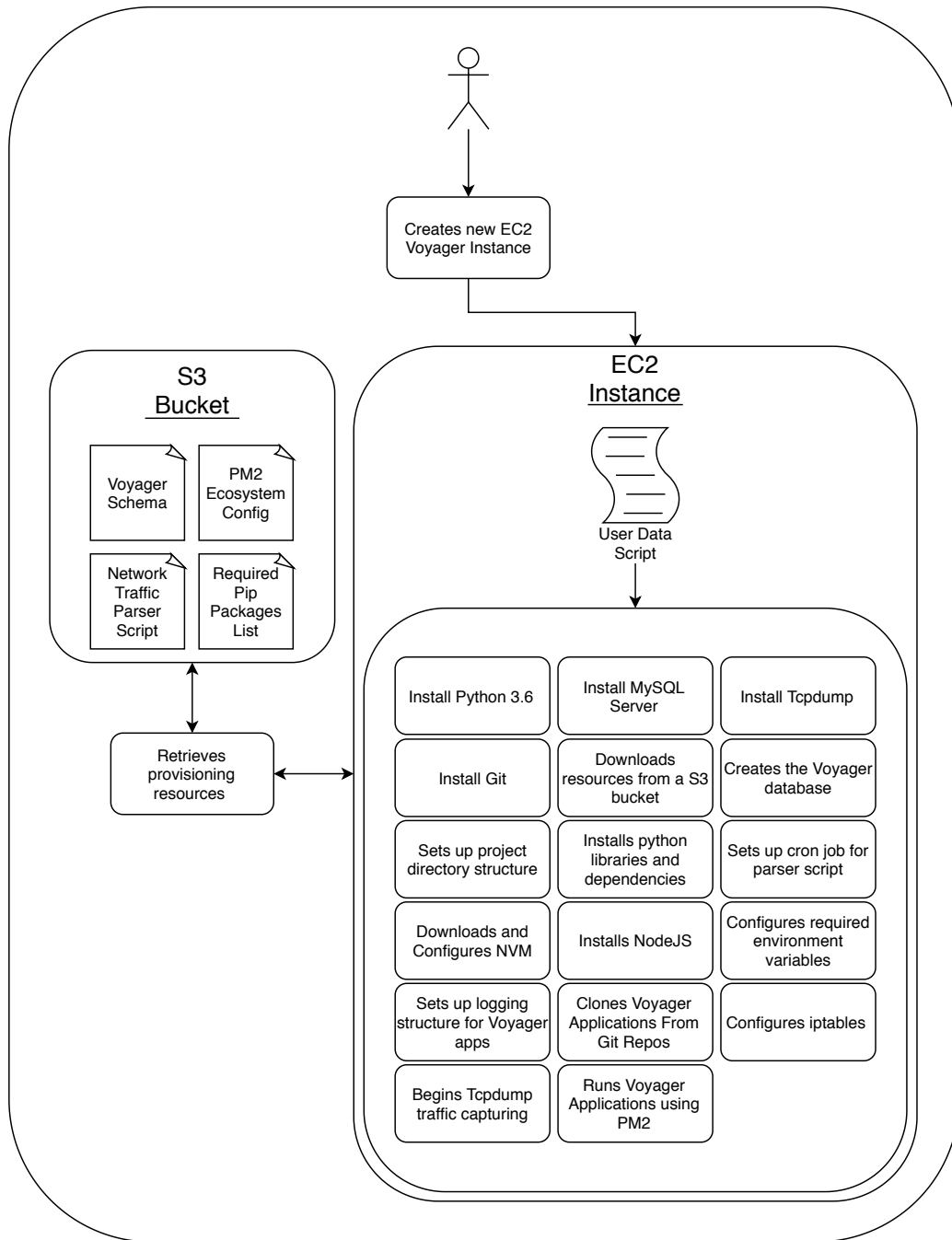


Figure 6: Diagram of the installation script process.

This provisioning script consists of multiple components, each responsible for setting up and provisioning a different part of the Voyager project. These steps, illustrated in Figure 6, are as follows:

1. **Installation of required packages** via the yum package manager.

These packages include Python 3, MySQL server, the tcpdump utility, and Git.

2. **Downloading resources from S3** that are required during the provisioning process. There are four files that are downloaded from a remote S3 bucket, the address of which can be easily changed inside the provisioning script.

- (a) **voyager.sql**: This SQL file contains the database schema definition. It also creates two MySQL users, one for the API and one for the python parsing script. The creation of these two users is done in an effort to increase the security of Voyager in the event of an attack such as SQL Injection. By using distinct users for each part of the application this supports the implementation of the principle of least privilege as well as increases audibility for forensics should an attack occur.

- (b) **ecosystem.config.js**: This JavaScript file is the configuration file used by PM2 ¹, a Node.JS process manager used to keep Voyager's

¹<https://pm2.keymetrics.io/>

front end and server running. Within it is defined how PM2 is supposed to run the web applications as well as specify options such as the location of error and output logging files. Lastly, also included are environment variables which will be used by the applications upon launch. These variables are used to instruct the application which environment type to run in, be it production or development, as well as which database and database user to use.

- (c) **parser.py**: This python file is responsible for parsing the network traffic, indentifying events, and subsequently creating entries in the database. It is copied onto the system for later use. This script is discussed in detail in section 3.6.
- (d) **pip_packages.txt**: This text file is used for installing all the python libraries and dependencies that are required for the `parser.py` script to run.

3. **Creation of the Voyager database** using the `voyager.sql` file previously downloaded from a S3 bucket.
4. **Set up of the project directory structure** is required in order for Voyager to operate correctly. An outline of the required directory structure is pictured in Figure 7.

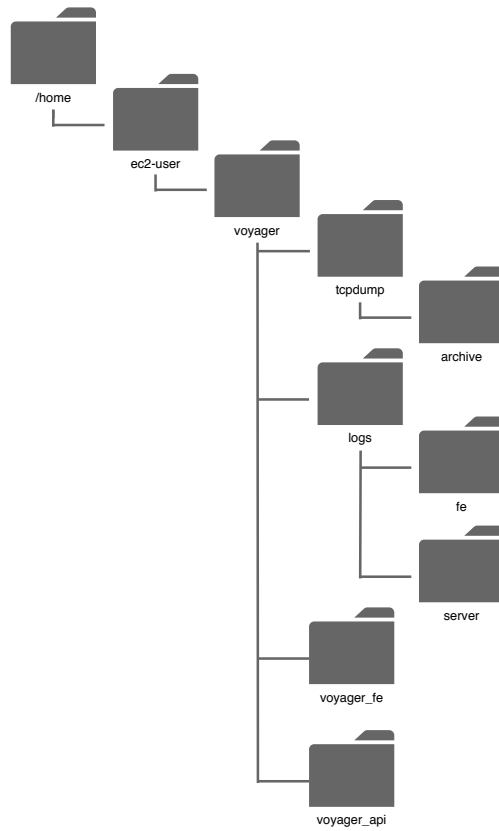


Figure 7: The Voyager directory structure.

5. **Installation of python libraries and dependencies** in order to allow the `parser.py` script to run. Specifically, the Python 3.6 development tools as well as the mysql development tools are installed using the yum package manager. Then, the `pip_packages.txt` file is used with pip, the standard Python package manager, to install all other

dependencies. The critical packages listed in `pip_packages.txt` and subsequently installed with `pip` are depicted in Table 2.

Package Name	Version	Purpose
mysql-connector-python	8.0.19	Used to connect to the Voyager MySQL database.
mysqlclient	1.4.6	Used to easily create MySQL queries.
requests	2.23.0	Used to query the IP Geolocation API.
scapy	2.4.3	Used to dissect and analyze pcap files containing network traffic.

Table 2: Relevant `pip_packages.txt` packages.

- 6. Setting up the cron job of the parser script.** This is to ensure that the parser script runs on a regular time interval to clear the network traffic capture files. The time between runs can be modified based on the amount of expected traffic however it is given a default of every 2 minutes. This can be adjusted to a longer time between runs if you expect less traffic, or more often if large amounts of traffic are expected.

The command used to set up the cronjob is displayed in Figure 8.

```
(crontab -l 2>/dev/null; echo '*/*2 * * * * /usr/bin/python3
/home/ec2-user/voyager/parser.py') | sort - | uniq - | crontab -
```

Figure 8: The cron command to run the `parser.py` script.

- 7. Installing NVM (Node Version Manager) and NodeJS.** NVM is a version manager for the NodeJS programming language that works in any POSIX compliant shell. It is required because the web interface as well as the API are written using NodeJS. Once NVM is installed

successfully, it is used to install NodeJS.

8. **Environment variable configuration** is required in order for all applications to function properly. These variables are used when launching the Voyager web application and can be changed to vary the port the front end and API run on. It is important to keep in mind that changing the port that the front end runs on requires updating the inbound network traffic rules that were set up during the EC2 instance creation. Otherwise the front end application will not be accessible at all.
9. **Setting up logging directories and files** is necessary before running the applications. To provide for easy debugging as well as general logging there are two separate logs per application. There is a general log which contains all generic output messages and an error log which contains system errors as well as more serious output messages pertaining to anything that interrupts the running of the application. This configuration is depicted in Figure 9.

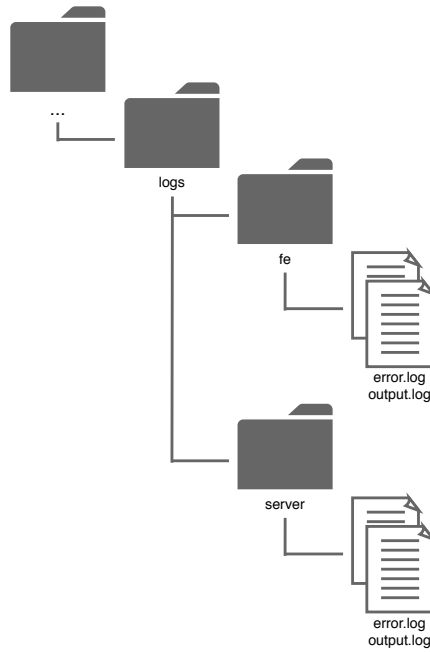


Figure 9: Voyager’s logging structure.

10. **Cloning the Voyager project** from the remote Git repositories is the next step. There are two separate repositories, one hosting the front end, and the other hosting the API or back end. While it is possible to combine these two pieces into a single repository they are purposely left apart. One reason for this is that it allows for the potential to run each one on separate EC2 instances without any extra work of decoupling them. Additionally, by following a microservice pattern this allows for updates and changes to one piece without having any effect on the other or forcing everything to be redeployed. Similarly,

any errors encountered will not crash the entire application but just one piece of it, providing greater durability of the project as a whole.

11. **Configuring iptables** is done after downloading all packages and repositories. This step is responsible for directing all traffic from port 80 into the application. Doing so allows the applications to run on any desired port while still directing incoming traffic from the standard HTTP port to them. Specifically, this traffic is directed to the part of the application responsible for hosting the uploaded content. This is accomplished via the command in Figure 10.

```
sudo iptables -A PREROUTING -t nat -i eth0 -p tcp
--dport 80 -j REDIRECT --to-port 9000
```

Figure 10: The command to configure the iptables port routing.

12. **Starting tcpdump** allows for the capture of network traffic from this point forward. This is saved as one of the last steps to avoid capturing any traffic pertaining to the provisioning of the instance. The specific use of the tcpdump command is covered in much greater detail in Section 3.5.
13. **Running the Voyager applications**, the last step of provisioning, is handled via PM2. Using the configuration contained in the `ecosystem.config.js` downloaded from S3 earlier, PM2 starts both the front end and back end applications.

Upon successful completion of the provisioning script, the EC2 instance is now fully configured and running Voyager. Should any errors occur, EC2 instances output logs from the user data script into `/var/log/cloud-init-output.log`.

3.3 Web Interface

The web interface refers to the interface that law enforcement officials use to interact with Voyager as well as where files that have been uploaded are hosted. Originally, it was created using basic PHP and very simple HTML. This has however been refactored in an attempt to adopt more industry standard web development techniques. Presently, it consists of two separate components, the front end application and the server side or backend application. Both the front end and back end servers are run using PM2, which is a process manager for the JavaScript runtime Node.js. PM2 not only provides a simple and easy to use interface for running both applications but grants other useful functionality. For example, it keeps applications running, even after signing out of the EC2 instance and will automatically restart either application should it crash. With a simple command the application is configured to run with PM2 while also setting up logging for both error logs and standard output logs. This proves useful for simply checking the status of the application as well as debugging should any crashes or errors occur.

3.3.1 Front End

The front end user interface portion of the application is created using the Node.js programming language, more specifically the React library. React was chosen because it allows for the easy creation of interactive user interfaces via encapsulated components each of which can contain their own state. Furthermore, it offers “greater flexibility in terms of how [I] structure and display data” [9], making the application easier to both understand and use. Additionally, other libraries and frameworks were leveraged to create the user interface such as bootstrap. The front end provides interfaces for important tasks such as allowing investigators to upload photos or other types of files they wish to use as well as viewing important information stored in the database. This data consists of but is not limited to all uploaded photos, and the logged events corresponding to those photos.

3.3.1.1 Home Screen

Shown in Figure 11 is the home page of the Voyager web application. It is intentionally kept minimal in an effort to be simple and easy to use. From the home screen the user can access the three different functionalities that the web application provides: the photo log, the event log, and the file upload page.

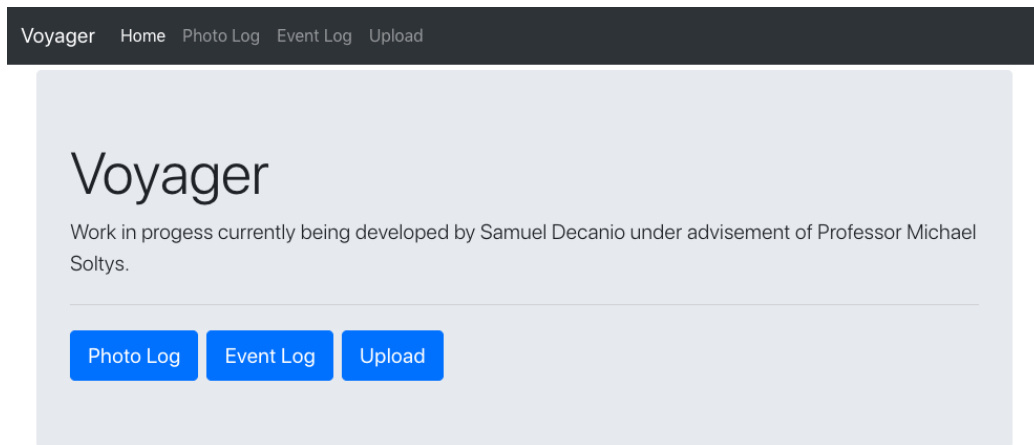


Figure 11: Voyager's home screen.

3.3.1.2 Photo Log

The photo log page is responsible for displaying a table that contains entries for all the files that have been uploaded to Voyager. This table can be seen in Figure 12. The table is built to allow investigators to quickly find a previously uploaded file. Sorting is made available on the relevant rows, making finding records of interest a quicker endeavour. When a record of interest has been found, the user can click on that specific row to open the photo details page.

Voyager	Home	Photo Log	Event Log	Upload
---------	------	-----------	-----------	--------

Photo Log

Click on a row to open the details view.

ID ↑↓	Photo Hash ↑↓	Link Hash	Upload Date ↑↓	File Extension ↑↓	Notes
33	7a105a03219445b8a911a0c9ae6661a1	7a105a03219445b8a911a0c9ae6661a1	2020-04-26T19:51:27.000Z	undefined	updated notes
34	fd78c09a67635b759680a149463f25c3	fd78c09a67635b759680a149463f25c3	2020-04-26T20:32:04.000Z	test	test notes.
35	2181431966fd23338459c95242ae8f30	2181431966fd23338459c95242ae8f30	2020-04-26T20:35:42.000Z	.jpg	update
36	test	a69e7b520cbbf561adfdc35e7576155f	2020-04-26T23:56:33.000Z	.jpg	
37	4ed47905a973a4ad36bc9559ae1d489c	1a40d2aab90665af6b5865dd58d46fc8	2020-04-28T22:33:25.000Z	.jpg	
38	8cccd993e3787d61d9f7fec1c0c6988	4177fdcb90a5303a8ed2a6d1523fcc00	2020-08-30T19:07:54.000Z	.jpg	

Figure 12: Voyager's photo log page.

3.3.1.3 Photo Details

The photo details page, shown in Figure 13, provides a more in depth look at an individual file that was uploaded to Voyager. On this page, information about that specific file upload can be found, such as the file's *link_hash*. There is two editable fields on this page, the notes text box and the file extension

input field. The notes box allows investigators to log and information relevant to that specific file upload to its record. The file extension input allows for the user to change the file type that will be appended when accessing the file. Additionally, this page has a button that allows the user to preview the file that was uploaded.

[Voyager](#) [Home](#) [Photo Log](#) [Event Log](#) [Upload](#)

Photo Log Entry: 39

Photo Id	<input type="text" value="39"/>
Photo Hash	<input type="text" value="641827a284687bf150f394209ae5b1ab"/>
Link Hash	<input type="text" value="a177e6c0b1769aedfd16c107cb293ec6"/>
Upload Date	<input type="text" value="2020-08-30T19:34:36.000Z"/>
File Extension	<input type="text" value=".jpg"/>
Notes	<input type="text" value="CSUCI Logo"/>

Save Changes

Toggle Image Preview

Figure 13: Voyager’s photo details page.

3.3.1.4 Event Log

Accessible via the home screen or the banner that runs across the top of every page is the event log. As seen in Figure 14, this screen is similar to the photo log. The difference between the two is the information displayed in the tables. In this case, the information pertains to specific events that Voyager has recorded. This table also allows sorting based on relevant columns. Since many of the fields recorded for an event contain large amounts of text, they are not able to easily be displayed or read from a table like view. In order to combat this, each row is clickable and redirects the user to the event details screen.

Voyager

Home

Photo Log

Event Log

Upload

Event Log

Click on a row to open the details view.

Event ID ↑↓	Photo ID ↑↓	Event Date ↑↓	Source IP ↑↓	IP Geo Info	User Agent String	Get Request	Raw Data	Notes
14	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	
15	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	
16	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	
17	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	
18	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	
19	36	2020-04...	47.157.18...	{"ip": "47...	'Mozilla/5...	b'GET /a...	###[Eth...	Updating...

Figure 14: Voyager's event log page.

3.3.1.5 Event Details

Once on the event details screen, the user is able to see a plethora of information about one specific event. This page contains the *photo_id* to which the event corresponds as well as other fields containing details about the event. Shown in Figure 15 are all the fields that are recorded about an event. Furthermore, by clicking on the “Toggle Geo Info” button, a new field appears. This field, as seen in Figure 16, displays all the geolocation information that was able to be gathered for the given source IP address. The “Get Request” and “Raw Data” text areas can be expanded to more easily read all of the data stored in them. Lastly, similar to the photo details page, the event details page has an editable field for notes. This allows investigators to save relevant information associated with that specific event.

Event Log Entry: 19

Event Id 19

Photo Id 36

Event Date 2020-04-26T23:47:47.000Z

Source IP 47.157.188.186 [Toggle Geo Info](#)

User Agent String 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132

Get Request b'GET /a69e7b520cbbf561adfc35e7576155f.jpg HTTP/1.1\r\nHost: ec2-3-81-220-

Raw Data ###[Ethernet]###
dst = 0e:4b:87:5e:12:71
src = 0e:b6:83:43:28:3e

Notes Updating the notes field!

[Save Changes](#)

Figure 15: Voyager's event details page.

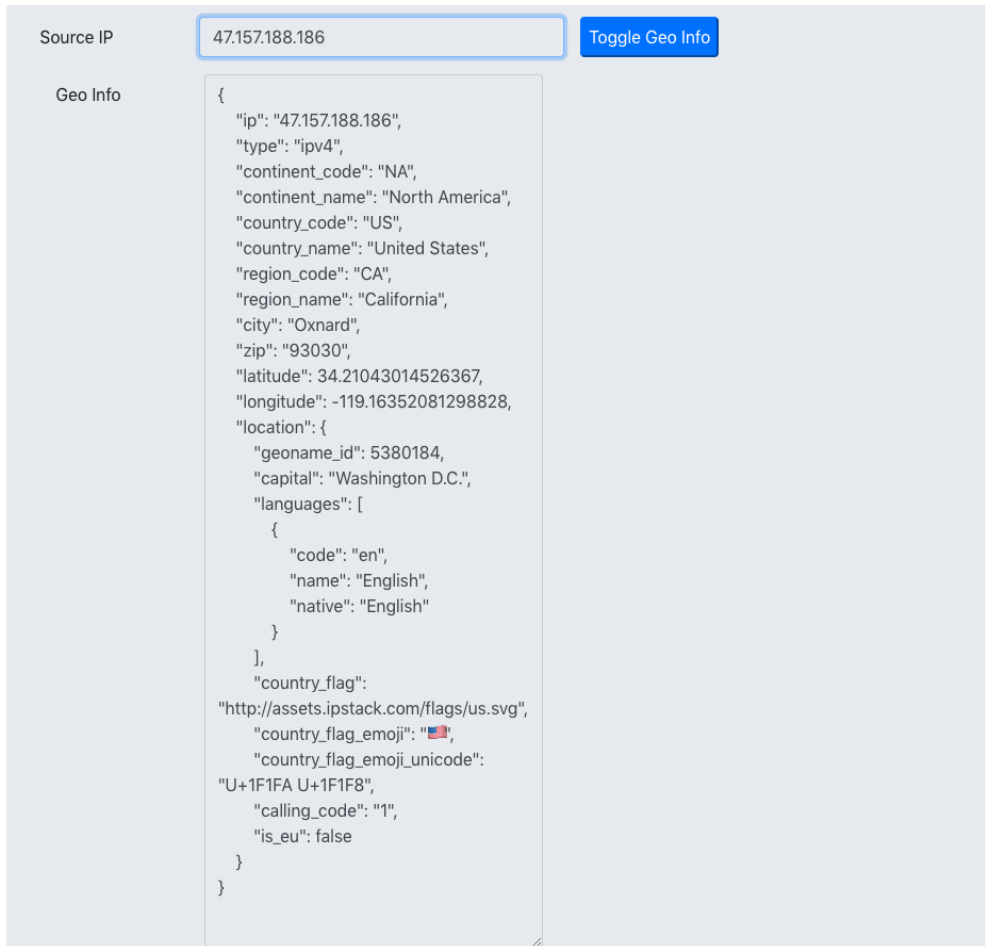


Figure 16: Voyager's event details page displaying geolocation data.

3.3.2 Back End and Server Side

The server side of the application is a simple API that interfaces with the MySQL database, discussed in Section 3.4. It receives requests from the front end and then creates, manipulates, or returns data based on those

requests. As shown in Figure 17, when a new artifact is uploaded via the user interface it is sent to the server where it is given a hash corresponding to the artifact itself combined with the time of upload. This hash along with other information is entered into the database and the photo is then moved to a directory that is accessible by the world wide web. The server side is developed using Express, a Node.js framework and the de facto standard for server frameworks. Express is especially useful for handling tasks such as routing requests and simplifying the creation of APIs.

In addition to processing and returning data, the server is also responsible for hosting the artifacts that have been uploaded by investigators. It accomplishes this by storing them in a separate folder from the rest of the application that has been made publicly accessible. Upon receiving a new upload the server renames the file to the *link_hash*, discussed in depth during Section 3.4, that it has generated for that artifact and then moves it into the public folder. Once this has been completed that artifact is ready to be used by investigators.

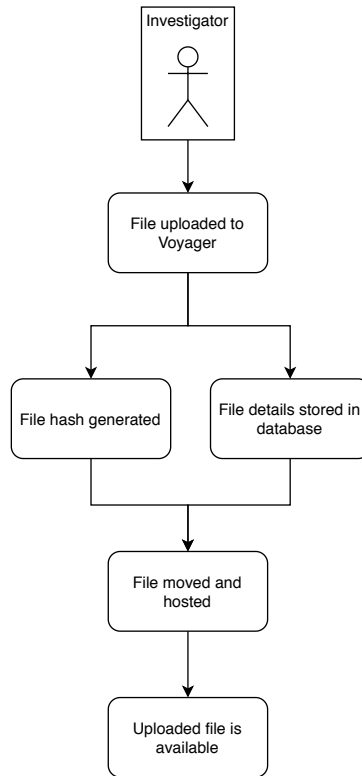


Figure 17: The Voyager work flow for file uploads.

3.4 Data Storage

The backbone of the Voyager tool is the MySQL database, containing two tables of note, displayed in Figure 18. The first is the *photo_log* table which is responsible for storing information associated with an artifact that has been uploaded by an investigator. The table has a primary key, *photo_id*, which is a standard auto increment field. Additionally it contains a field

named *photo_hash* that is created by hashing the raw bytes of whatever artifact is uploaded. This allows for easy discovery of instances of the same artifact being uploaded for different uses, potentially with different file names. Similarly, the table contains a field called *link_hash* which is a hash created from the raw bytes of the photo appended with a timestamp of the upload date and time, creating a different hash than the *photo_hash*. This serves as a completely unique identifier for each artifact uploaded, even if it is the exact same artifact, as the upload time would differ. This *link_hash* is also used in the URL for which that particular resource will be made available at. Both the *photo_hash* as well as the *link_hash* are created using the md5 hashing algorithm.

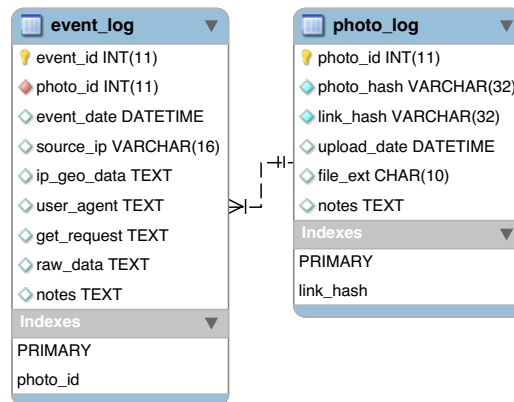


Figure 18: Voyager's database schema.

While the table is called *photo_log* there is no requirement in the application that the uploaded content be a photo. Since the table contains the extension of the file that is uploaded this allows Voyager to handle virtually any file type that a user wishes to upload. Voyager accomplishes this by simply appending the file type saved at time of upload to the web hosted version of the content whenever that resource is requested.

The second table is the *event_log* table, which is responsible for holding details about captured network traffic. In particular, the Python script, which is discussed in detail in Section 3.6, populates this table with data about HTTP requests that are made to the specific addresses of the uploaded artifacts. These are the requests arising from clicking the bait link. The data saved is robust in that certain parts of request are parsed out and saved individually, such as the source ip address of the request and the user agent string. In addition to these fields, the entire packet is saved as well, allowing for deep investigation of the traffic at a later time. This provides a means for the data Voyager extracted from the request to be validated. The events are associated with the uploaded image or artifact requested via a foreign key, *photo_id*, referencing the primary key in the *photo_log* table.

In addition to data extracted from these requests the *event_log* table also stores data regarding the believed geolocation of the ip address that made the request. This is stored as JSON in the form that the leveraged API returns. This allows for the potential of future processing of this data, either

from within Voyager or another application that externally accesses the data via the API.

Both tables consist of fields that allow for the investigators using Voyager to create notes associated with either an uploaded artifact or a specific event that is logged. These fields have seen use as a place to list notes about the specific case being worked on as well as a field to list which investigator uploaded which artifact.

3.5 Capturing Network Traffic

In order to log the events, or clicks, on a specific artifact a tool needed to exist that was capable of capturing all the incoming traffic on the server. The tool selected for this job was tcpdump. Tcpdump is a command line tool for packet analysis and is used in conjunction with libpcap, which allows for the capturing of network traffic. Both of these are used on the web server in order to capture incoming TCP requests. This traffic is logged in the form of a .pcap file which is analyzed at a later time by the Python script.

```
tcpdump -w /home/ec2-user/voyager/tcpdump/tcpdump_%F_%H:%M:%S.pcap  
-G 60 -n -Z root 'dst $IP_ADDRESS and (port 80 or 443 or 8080)'
```

Figure 19: The tcpdump command used to capture network traffic.

The tcpdump command used, shown in Figure 19, leverages several options that allows for high level filtering of the traffic recorded. This is use-

ful because the ability to eliminate a large majority of the irrelevant traffic greatly reduces the size of the .pcap files, making the later processing much easier as well as more efficient. Additionally options are used to allow for a more optimal organization of the traffic that is recorded, rather than simply saving it all to a nondescript .pcap file. The specific tcpdump options leveraged are as follows:

- **-w**: Allows the user to specify the output file path and name. Then, strftime formatting options are used to specify the output file name. %F outputs the date in YYYY-MM-DD format. %H specifies the hours in a 24 hour format. %M specifies the minutes, ranging from 0 to 59. %S specifies seconds, again ranging from 0 to 59. Appending the date and time to the output file is useful for reference if the specific traffic needs to be examined at a later date. It can also be cross referenced with the date times recorded in the event log to easily find the file containing the traffic that yielded a specific event record.
- **-G**: Used to configure output file rotation. In this case the process will be creating a new .pcap file every 60 seconds; this rotation frequency keeps the files from becoming too large. Rotation every minute also allows for easier reference should they be needed at a later time, as mentioned above.
- **-n**: To avoid DNS lookups; investigators want to keep the IP addresses

of the incoming traffic rather than associating them with a host name. Associating with a hostname can be done later on IPs of interest if it is determined to be relevant.

- **-Z:** By using this option with the argument of root this ensures that the tcpdump command is running at root privilege level. This is needed to allow for all traffic to be captured on the web server.

The last argument, '`dst $IP_ADDRESS and (port 80 or 443)`', is the filter that used to narrow down the traffic that is recorded to the output files. The '`dst`' denotes the destination address that should be filtered for, in this case it is the address of the web server. This is read in from an environment variable that is set up during provisioning of the instance. We are then further filtering the traffic to look specifically for anything that is using the common HTTP port: 80. Additionally, we are capturing traffic for port 443 as well, which is used for HTTPS. While we cannot read this traffic due to its encryption, we wanted to capture and archive it for potential future use. By filtering out any traffic that is not on these specific ports it avoids recording excess traffic that is not relevant. For example, any interactions by the investigators with the front end interface will not be recorded because it takes place over port 9000.

Saving the network traffic in easy to organize and find .pcap files is extremely important in this specific scenario. After the creation and processing

of these .pcap files, they are archived indefinitely. This is because they may be needed down the road to submit as evidence in court. Thus, investigators need to be able to find the files of interest easily while sifting through potentially thousands of archived traffic capture files. For more long term storage these files can also be moved off the EC2 instance into an S3 bucket or even Glacier storage. This provides a more cost effective way to store large amounts of network traffic for extended periods of time.

3.6 Analyzing The Traffic

After the network traffic is received, filtered and recorded to pcap files by tcpdump it will then be parsed by the Python script `parser.py`. This script runs at scheduled intervals via the Cron tool. While the Cron tool allows for easy modification of how often the script should be run it is initially set to every other minute. This timing can be adjusted based on how much traffic the instance is receiving.

The `parser.py` script is responsible for taking the pcap files, processing them, inserting necessary data into the database and then archiving them. It is important to note that it will process the total number of pcap files in the directory minus one. This is because the tcpdump is constantly running, and thus writing to pcap output files, even as the parser is processing them. By sorting the files in the directory via time of creation in ascending order the

script is able to process the older files first, leaving the newest one as the last file, which remains unprocessed. This avoids conflicts with both processes attempting to access a file at the same time.

The parser leverages a Python library named Scapy² to read the pcap files and examine the traffic captured in them. Should it come across any HTTP GET requests for artifacts that exist in the database, that traffic is noted and has an entry created for it in the event log table of the database. Furthermore, Scapy is used to extract the specific details of interest to investigators out of the entire request, including but not limited to: the IP address that made the request, the specific URL that was requested and the user agent string.

The task of finding packets of relevance is accomplished through a process of elimination via progressive filtering. The tcpdump command has already filtered out any traffic that is not TCP traffic on ports 80 or 443. This means the majority of the traffic will be HTTP or HTTPS. This process begins by checking if the packet is a GET request. This will filter out HTTPS traffic as the parser is unable to see inside the packet to check if it is a GET due to the encryption. This also filters out anything that is not HTTP as it will obviously not be a GET request. Once the script is sure it is working with a GET request then it uses string manipulation to filter out the path of the URL. As shown in Figure 20, the path is the last part of the URL that specifies which file on the server is being requested. More specifically,

²<https://scapy.net/>

only the last element of the path, which is referred to as the link hash, is of interest in this case. This link hash is, by design, unique to each artifact uploaded to Voyager. Once it is confirmed the URL in question has a link hash it is extracted. Then it is compared to all link hashes stored in the database to find which artifact this specific request corresponds to. Once the script has found the correct file, confirming the request was for an artifact uploaded to the server, the script begins extracting additional information from the packet. This information is then inserted into the event_log table in the database.

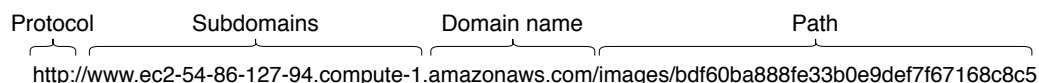


Figure 20: Breakdown of an example URL.

In addition to just extracting information from the requests to artifacts of concern, the script goes a step further. It also extracts the source IP address from the request and use it to query the IPStack³ Geolocation API. This generally returns additional information regarding the believed geographical location of the IP address as well as other useful information. All this is then stored in the database as well to give investigators even more information to work with during their investigations.

³<https://ipstack.com/>

4 Observations and Results

4.1 Real World Usage and Results

Due to the first iteration of Voyager being deployed to the SCHTTF for some period of time, there have been multiple instances of its use in real investigations and cases. While it would be simple to boast the cases where Voyager worked perfectly and led to the identification of a cybercriminal with no complications, this is not always the reality. Many of these cybercriminals are very savvy in protecting themselves, using a plethora of tools to do so. Some of these are discussed in Section 5. Two more realistic examples of the deployment of Voyager’s first iteration that are especially interesting are described below.

4.1.1 Case 1

The first case consisted of a cybercriminal illegally accessing a government health care agency’s computer network and compromising the email account of an employee. Using this email account the criminal sent out emails, pretending to be the government employee, attempting to steal \$18,650 through wire transfers. The transfers were sent to a co-conspirator’s Chase Bank account. This is a typical scenario referred to as Business Email Compromise

or BEC. Business email compromise as a popular and still growing scam. According to the 2019 Internet Crime Report [10], published by the IC3; there was 23,775 complaints with over \$1.7 billion in losses due to BEC in just 2019 alone.

Working with the CISO of the government agency, the SCHTTF deployed Voyager in an attempt to learn more about the cybercriminal in this case. An email was sent to the suspect which included an attachment that linked to Voyager. This link was disguised as a thumbnail containing an image of the online banking portal for Chase Bank. Later that day, Voyager logged a click on the bait that was sent to the suspect, capturing the suspect's IP address along with other information.

Researching the captured IP address revealed that it was in a block owned by the internet service provider M247.com. Following this trail, M247.com revealed that the IP address was subleased to a Canadian company, windscribe.com. Windscribe.com is an online company that provides services such as adblockers, VPNs, and other anonymizing features. While this is a dead end for Voyager, the investigators were able to learn that the suspect's user account was made with an email created with a German IP address and a Ghana phone number.

In this particular instance, Voyager did not lead to successful identification of the criminal. However, it did open a path of investigation that

otherwise would not have existed as well as provided valuable insight into the operational security or OPSEC of the criminal. All the aforementioned information led the task force to come to the conclusion this cybercrime was not the product of a local small time criminal who gained access to the victim's email using dumpster diving or simply guessing credentials. Instead, it was deemed very likely that this attack came from a sophisticated ring of cybercriminals, likely carrying out such attacks from a foreign country. It is also thought that this government health care agency was just one victim in a larger campaign of these types of attacks from the same group.

4.1.2 Case 2

The second case was one in which Voyager was deployed against a suspect in a ransomware case. The victim came in to work one morning to find Figure 21 displayed on their monitor.

Upon further inspection, the victim discovered all the company's data had been encrypted. In this particular instance the company elected to pay the ransom that the cybercriminal was demanding, in hopes of their data being returned. While some data was unencrypted as a result, the rest was ransomed again. Unfortunately, this is not particularly uncommon in these crimes. It was at this point that the company agreed to send the suspect an email. Similar to the first case, the email contained a link to Voyager

disguised as a thumbnail image, shown in Figure 22.

Voyager recorded two unique IP addresses accessing the link. The first was quickly dismissed as an automated web crawler. The second however was discovered to be within an range of addresses owned by Dropbox. Unfortunately they were unable to receive any response from Dropbox, so this is where this avenue of investigation dead ends as the significance of this finding is undetermined.

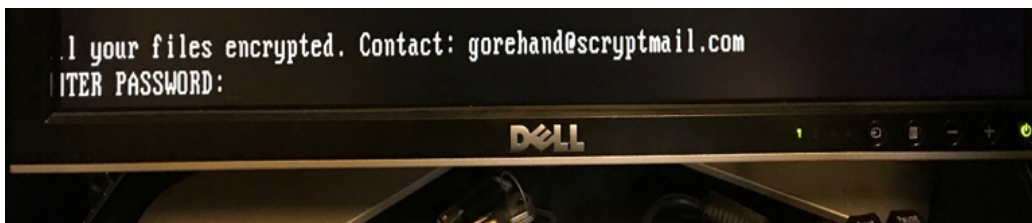


Figure 21: Example ransomware message, courtesy of the SCHTTF.

To: gorehand@cryptmail.com
Subject: Re: Security Report

Hello,

Our IT company is still waiting for the security report you promised. They think they have located the security flaw and would be willing to pay a very small amount additional Bitcoin for confirmation, even if you are no longer willing to provide the full report. Please advise if you would like to negotiate.

[Screen shot of Palo Alto firewall.](#)



Figure 22: Example email containing a Voyager link disguised as a thumbnail, courtesy of the SCHTTF.

4.2 Exploratory Results

The second case described above led to an interesting finding, an IP address belonging to Dropbox being logged in an undetermined manner. As a result of this, testing was conducted in order to observe what effects occurred when accessing Voyager’s links from different mediums. Setup included provisioning a Voyager instance in the exact same manner that it would actually be deployed, by using the provisioning script discussed in Section 3.2.

Once the instance was finished provisioning itself, images were uploaded via the Voyager web interface. In order to keep all results completely segre-

gated, an independent image and thus link was used for each different medium of access. This allows for an easy distinction when viewing the event log page as each image was only accessed by one simulated scenario.

For this testing 7 different scenarios were explored. The selected scenarios were used as they were the most likely cases to resemble the real world use of Voyager. Each scenario along with its results are outlined below.

1. **Standard WiFi, Macbook Pro, Google Chrome.** Standard WiFi refers to the network most people have at home. Specifically, there was no user firewalls, VPN or other protections set up. The computer used was a Macbook Pro. Google Chrome refers to the web browser used to access the link.

The results of visiting the Voyager link are displayed in Figure 23. As can be seen from the highlighted fields, Voyager correctly recorded the IP address of the request and subsequently used that to identify the city in which the computer is located. Additionally, the user agent string was extracted from the request as well. From that string it can be observed that the computer making the request was a Mac running OS X version 10.14.1 and used the Google Chrome browser.

Event Date	2020-06-21T17:11:51.000Z	
Source IP	47.155.17.188	<button>Toggle Geo Info</button>
Geo Info	<pre>{ "ip": "47.155.17.188", "type": "ipv4", "continent_code": "NA", "continent_name": "North America", "country_code": "US", "country_name": "United States", "region_code": "CA", "region_name": "California", "city": "Oxnard", "zip": "93030", }</pre>	
User Agent String	<pre>'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61 Safari/537.36'</pre>	

Figure 23: Using Google Chrome on a macbook pro connected to a home WiFi network. (Scenario 1)

2. **Standard WiFi, iPhone 11, Safari.** Similar to the previous, a standard WiFi network was used, but this time via a mobile device. Safari is the standard iOS web browser that is on all of Apple's devices. Unsurprisingly, the results of this scenario were similar to the previous one. Voyager was able to correctly identify the IP address and further use it to geolocate the device. The main difference lies in the user agent string field that was collected. Examining the event logged, shown in Figure 24 can see that this time it identifies the device which made the request as an iPhone using the Safari browser.

Event Date	2020-06-21T17:22:55.000Z	
Source IP	47.155.17.188	Toggle Geo Info
Geo Info	<pre>{ "ip": "47.155.17.188", "type": "ipv4", "continent_code": "NA", "continent_name": "North America", "country_code": "US", "country_name": "United States", "region_code": "CA", "region_name": "California", "city": "Oxnard", "zip": "93030", }</pre>	
User Agent String	<div>Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Mobile/15E148 Safari/604.1</div>	

Figure 24: Using the Safari browser on an iPhone 11 connected to a home WiFi network. (Scenario 2)

3. **4g Cellular Data, iPhone 11, Safari.** This scenario was the first to not be connected to the same standard WiFi network. Instead, a 4g cellular data connection was used to access the link, via the Safari browser on an iPhone 11.

With this scenario came very interesting results. As with the last scenario, the user agent string correctly identified the device as an iPhone using the Safari browser. However, when resolving the geolocation of the recorded IP address, it is identified as being from Temple City, CA even though the actual device was located in the same location as

the previous scenarios. This can be seen in geo info displayed in Figure 25. This seems to align with the findings detailed in [20] in which the authors examined the geolocation of ip addresses that were used in cellular data networks. They concluded that the margin of error in geolocation of these IP addresses was at least 100km (which is roughly 62 miles) for about 70% of there dataset. My result falls in this 70% figure as the distance from Oxnard to Temple City is roughly 128 km (80 miles).

This result is likely due to the use of NAT or Network Address Translation that is implemented by the mobile provider. Use of NAT by mobile providers is increasingly common with the scarcity of space in the IPv4 address range.

Event Date	2020-06-21T17:27:25.000Z	
Source IP	166.137.8.126	Toggle Geo Info
Geo Info	<pre>{ "ip": "166.137.8.126", "type": "ipv4", "continent_code": "NA", "continent_name": "North America", "country_code": "US", "country_name": "United States", "region_code": "CA", "region_name": "California", "city": "Temple City", "zip": "91780", }</pre>	
User Agent String	'Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Mobile/15E148 Safari/604.1'	

Figure 25: Using the Safari browser on an iPhone 11 with 4g cellular data. (Scenario 3)

4. **Standard WiFi, Macbook Pro, Google Chrome, Gmail Image Attachment.** This scenario is the first where the link was not directly accessed in a browser. Rather than simulating a suspect clicking the link, the file was included in an email using the “Attach Image” feature in Gmail and inputting the URL.

Unlike the previous scenarios, this one generated two separate events in Voyager’s event log. The first event logged was at the time of inserting the image into the email. The second event was at the time of the recipient opening the email which proceeded to load the included image.

Unfortunately, both events failed to log the actual IP address of the user. Instead they logged the IP address and subsequently the location of Google’s servers running their Gmail Image Proxy. These results can be observed in Figure 26. Additionally, it can be seen that the user agent string also collaborates this.

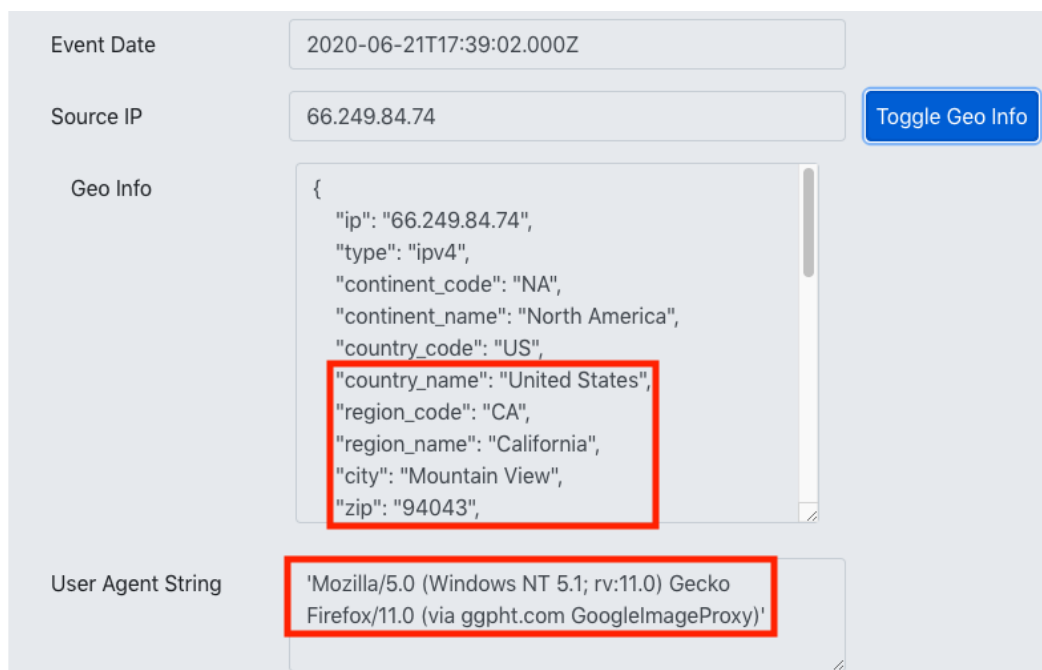


Figure 26: Using Google Chrome to view a Gmail image attachment on a macbook pro connected to home WiFi. (Scenario 4)

5. **Standard WiFi, Macbook Pro, Google Chrome, Gmail Hyperlink.** Similar to the previous scenario this one also involved using email, more specifically Gmail. This time, the Voyager link was set as a hyperlink inside the body of the email.

Unlike the previous scenario however, there was only one event generated this time. This event occurred when the recipient clicked on the hyperlink, directing them to the image hosted on Voyager. As with scenarios 1 and 2, the IP address of the device along with its geolocation were correctly captured by Voyager. This makes sense as they effectively accessed the link as anyone typing it into their browser would. However, it is notable that Google does nothing to screen hyperlinks in their emails.

6. Standard WiFi, Macbook Pro, Google Chrome, Dropbox URL.

While this scenario uses the same hardware as scenario 1, there is one crucial difference. This scenario has the Voyager link being accessed via a shortcut created via Dropbox. To access it, the shortcut was clicked. Surprisingly, this scenario logged roughly the same information as scenario 1. Unfortunately this leaves the question unanswered as to how an IP address within Dropbox's range was logged in Case 2 described above.

- 7. Public WiFi, iPhone 11, Safari.** This last scenario simulates a criminal operating out of a cafe or other location with publicly accessible WiFi. In this case I observed the effects of visiting a Voyager link on an iPhone 11 that was connected to the free WiFi of a local Starbucks. As shown in Figure 27, the IP address resolves to a location that is consistent with the location of the Starbucks. Additionally, the

user agent string is no different then when visiting the Voyager link over home WiFi or cellular data. This shows that even when visiting a Voyager link from a public source without any other protections such as a VPN or proxy, the collected information can still be of tremendous use to investigators.

Event Date	2020-06-28T19:59:23.000Z	
Source IP	47.156.72.192	Toggle Geo Info
Geo Info	<pre>{ "ip": "47.156.72.192", "type": "ipv4", "continent_code": "NA", "continent_name": "North America", "country_code": "US", "country_name": "United States", "region_code": "CA", "region_name": "California", "city": "Oxnard", "zip": "93030", }</pre>	
User Agent String	<pre>'Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.1 Mobile/15E148 Safari/604.1'</pre>	

Figure 27: Using Safari browser on an iPhone 11 from public WiFi. (Scenario 7)

4.3 Observations

Examining the results of the previously described scenarios, both real world and staged, we can observe many instances in which Voyager can be leveraged

to provide investigators another tool in their battle against cybercriminals. In the majority of tested scenarios Voyager is able to record the actual IP address of a suspect, allowing for easy geolocating of them as well as additional follow up in the investigation. However, there are also multiple scenarios where Voyager is not capable of identifying the sole IP address of a suspect. While this is unfortunate, it may not be as detrimental as originally thought. Often times the IP address logged can be identified as belonging to some sort of organization, be it an ISP, mobile service provider, or even a Starbucks. It is not always the case, but sometimes these organizations are often able to provide investigators with useful information that can help lead to the apprehension of the suspect.

Even though throughout the scenarios that were tested I was unable to deduce how exactly the IP address described in Case 2 was logged, I also eliminated multiple possibilities. It is worth noting though that is it entirely possible the event logged was correct and in fact whoever accessed the link was using a network that belongs to Dropbox. This is possible if they were using a wireless network at a Dropbox office, or even accessed the link from a server owned by Dropbox somehow. Without additional information from the investigators as to their exact actions, as well as the exact results, it is unlikely I will be able to determine the exact circumstances that led to Voyager logging an IP address in Dropbox's IP space.

When it comes to observations about this tool there is more to examine

than merely the output. A vital part of the project was creating an easily usable and deploying system. Even a system with all the answers in the world is likely to garner a groan of contention from its users if it is unreliable, slow, or just too complicated to understand. With this in mind, Voyager was designed to be as fast, simple, and easy to use as possible from the perspective of the end user. A large amount of time was devoted to implementing this design philosophy in regards to the creation process of new Voyager instance.

Being able to provision a blank EC2 instance to a fully operational Voyager system through the use of one script is no small task. However, it pays dividends in making Voyager easy to set up and pain free for the end user. In addition to the script simplifying the provisioning process, it also greatly speeds it up. During the testing of Voyager it took roughly 30 seconds to create a new EC2 instance, including attaching the provisioning script, when using the AWS console. Once the instance was created, and using a t2.micro EC2 instance, it took roughly 3 minutes for the instance to be fully provisioned. This 3 minutes includes all the set up that AWS does to create the instance, along with the running of the provisioning script. Once complete, the user is left with a fully running Voyager system, in under 5 minutes total.

In addition to a quick and easy method of standing up Voyager instances, AWS EC2 instances provide cost effective computing resources for law enforcement. Using AWS's pricing calculator ⁴ anybody is able to estimate the

⁴<https://calculator.aws/>

cost of running Voyager. Using 5 t2.medium instances (which each consist of 2 virtual CPUs, 4GB of memory, and low to moderate network performance) along with 100GB of storage on an EBS the total comes to roughly \$117.50 per month. Adding 5GB of standard S3 storage as well as 100GB of Glacier storage (for previously processed .pcap files) only adds roughly \$0.50. This brings the total cost to about \$118 per month to run 5 separate instances of Voyager as well as store all process .pcap files in the cloud. Running an in house server capable of the same performance would cost at least a couple thousand dollars, for the server alone. Add to that all the potential expenses, such as electricity, and the cost continues to rise. When considering these costs coupled with all the other overhead that hosting a server in house requires, it quickly becomes a much less feasible set up. In contrast, AWS allows Voyager to be hosted for a fraction of the cost.

5 Conclusion and Future Work

5.1 Future Work

In the future there are multiple features that could be implemented to increase the functionality and further refine the Voyager application. Additionally, through further use of the tool the existing functionality and implementation can be refactored in order to better meet the needs of the investigators who are using the tool. Some of the potential changes envisioned for Voyager are discussed below.

5.1.1 Pcap Files

One potential problem with this application is that an attempt to scale it to ingest large amounts of traffic, the pcap files recording the traffic grow in size very quickly. While this has not been a problem during my testing there is the potential for a problem to occur where the Python parser script will not finish parsing all the pcap files in the directory because another instance of it is spawned 2 minutes later. This creates the potential for multiple Python parsers to attempt to access the same files, which could result in unexplored error cases.

A possible solution to this problem would be to move away from tcpdump

and pcap files as a whole. One way to do this would be to leverage the Scapy library being used in the Python parser. Scapy offers the ability to monitor traffic on the network via its sniffer module. This approach would monitor the traffic in real time and generate the same results of the Python parser but on a packet by packet basis as opposed to parsing a pcap file full of traffic on a set time interval. Thus, this would eliminate the need for tcpdump and pcap files, resolving any problems they may create during periods of heavy traffic. This however raises concerns over archiving and future review of data as user will only have records of what Scapy saves. This makes it difficult if user wanted to latter analyze all the traffic to see if any was missed or for another reason.

Another potential approach would be to change the architecture to be more microservice oriented. One way to go about this would be to automatically store all recorded pcap files in an AWS S3 bucket. Then you would use a λ function, which is trigger via new pcap files being added to the bucket, to add those files to an AWS Simple Queue Services queue. Lastly, the parser would then pull pcap files from the queue to process. This approach has several possible advantages. One such being that you could have multiple parsers running across multiple EC2 instances, all pulling from the queue without any worry of potential conflicts. Another advantage is the extreme cost effectiveness of storing all the pcap files on S3.

5.1.2 Authentication

Perhaps one of the most straight forward ways in which to increase the security of this application would be to implement some sort of login or user authentication for accessing the non-artifact pages of the application (such as the file upload and log pages). Currently these pages are restricted using the Apache htaccess feature (login and password required to enter the site).

5.1.3 Leverage Additional AWS Resources

An additional feature that would increase the overall security of the application would be splitting it up onto multiple EC2 instances. Serving the application that allows for file upload and event viewing from the artifact hosting would server to sandbox the parts and help to avoid any overlap. With the current implementation there is the possibility that someone who goes to view an artifact could potentially find their way to other pages hosted on that domain, such as the file upload or log pages. However, if these pieces were separated, along with the database, one could leverage AWS features such as the Virtual Private Cloud to ensure sensitive resources are not accessible by everybody.

An additional benefit gained from using separate EC2 instances is that it would allow for the use of HTTPS on the instance hosting the database

as well as the upload and log pages. This would mean that the traffic when using those pages would be encrypted. At the same time, since they are on separate instances the files that are hosted would be able to remain using HTTP allowing the traffic to be recorded using tcpdump.

Furthermore AWS offers a robust toolkit of cloud computing functionality, featuring many tools that could potentially be used to expand Voyager. More research would need to be done to determine which tools to leverage and how to integrate them into the current project.

5.1.4 Addressing VPNs, Proxies, and TOR

With the current implementation of Voyager there is no way to distinguish if somebody is using a VPN or a proxy. In these cases, Voyager will only be able to log the IP address of the exit node for their traffic rather than their real IP address. Adding techniques for detecting the presence of proxies described in [16] would potentially allow for the detection of proxy but would not allow for the detection of any information about the user behind that proxy unless they were not using a fully anonymized one.

In a similar vein, use of TOR or The Onion Router by the client would serve to stop identification of a suspect. TOR is “a distributed overlay network designed to anonymize TCP-based application” [5]. It functions by essentially routing your traffic through a number of intermediary servers be-

fore it reaches its real destination. These intermediary servers only know the previous and the next destination, and are thus unaware of where the traffic originated or where its final destination is. However, much like user agent string analysis, discussed in the next section, there is a large amount of research being done to investigate how secure TOR really is. Many approaches have been developed to deanonymize traffic on the TOR network. One such example that could be leveraged is a basic website fingerprinting attack, detailed in [14]. Many other more advanced techniques become feasible with the computing resources and state of the art algorithms of a nation state or similar entity.

5.1.5 User Agent String Analysis

While Voyager does extract the user agent string from requests, it does not perform any sort of analysis on the extracted string. However, these strings are potentially capable of yielding large dividends if analyzed correctly. The information contained in the user agent string has been leveraged before in attempts to track users across attempts to change or mask their IP address. Performing analysis on the user agent strings in the web traffic captured would provide an interesting addition to Voyager’s toolkit and make it even more robust in its ability to provide investigators with as much information as possible. However, “the analysis of these strings is a complex endeavor” [12] as they vary greatly in format. Additional information about this technique

along with other more advanced ones such as checking installed fonts and plugins can be found at the Panopticlick⁵ project.

5.1.6 Addressing HTTPS

HTTPS or Hypertext Transfer Protocol Secure is a more secure version of HTTP which offers encryption of network traffic using TLS or SSL. HTTPS's use and adoption has grown greatly in recent years as companies and users alike have become more security conscious. While this is undoubtedly a positive change in the internet it presents a problem for Voyager. Many users may shy away from accessing sites that do not support HTTPS, as it means the traffic between them is in plaintext and can be intercepted. In Voyager's case, we must not leverage HTTPS or we would be unable to read the traffic that is recorded by tcpdump into the pcap files. One potential solution would be moving away from tcpdump to some tool or functionality that allows Voyager to log the network requests after they have been decrypted on the server. Further investigation is required but this may be accomplished by recording the requests inside the server side application as once it receives the requests they have already been decrypted.

⁵<https://panopticlick.eff.org/>

5.2 Conclusion

As described by Kimo Hildreth, a member of the SCHTTF, “Voyager is a tool in the arsenal of an investigator”. Voyager is not capable of solving cases on its own. While it sometimes works extremely successfully, identifying the IP address of an attack and allowing investigators to obtain a search warrant, this is not always the case. In most situations Voyager can be used to guide investigation efforts and reveal new paths of investigation that may have otherwise been unknown.

Furthermore, Voyager demonstrates the ability of a partnership between law enforcement and academia to be a symbiotic relationship. Students gain the ability to develop tools that will not only see use but also serve to make a difference in the world. While law enforcement can gain insight into new techniques and methods they can leverage in their investigations.

It is evident that the struggle between law enforcement and cybercriminals is not going to be slowing down any time soon. As such, law enforcement must adapt in order to stay relevant and capable of bringing these criminals to justice. One such method to do so is by constantly developing new tools, techniques, and procedures to combat the crime they investigate. While it is impossible for a single tool to be the end all answer to the problem these investigators face it is my belief that by implementing some of the future work mentioned previously, along with other refinements, it is entirely possible for

Voyager to become an even more useful tool in the arsenal of the SCHTTF.

References

- [1] Mohammed M. Alani. *Guide to OSI and TCP/IP Models*. Springer International Publishing, Cham, 2014.
- [2] Amazon Web Services Inc. Amazon s3 simple storage service pricing, 2020. <https://aws.amazon.com/s3/pricing/>, Last accessed on 2020-08-10.
- [3] Amazon Web Services Inc. Tagging your amazon ec2 resources, 2020. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html?icmpid=docs_ec2_console, Last accessed on 2020-07-12.
- [4] Samuel Decanio, Michael Soltys, and Kimo Hildreth. Voyager: Tracking with a click. *Procedia Computer Science*, 176:98 – 107, 2020. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020.
- [5] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, USA, 2004. USENIX Association.
- [6] Emsisoft Malware Lab. Caution! Ryuk Ransomware decryptor damages larger files, even if you pay: Emsisoft: Secu-

- rity Blog, Dec 2019. <https://blog.emsisoft.com/en/35023/bug-in-latest-ryuk-decryptor-may-cause-data-loss/>, Last accessed on 2020-07-12.
- [7] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. <http://www.rfc-editor.org/rfc/rfc2616.txt>.
- [8] Rajesh Kumar Goutam. The problem of attribution in cyber security. *International Journal of Computer Applications*, 131(7), December 2015.
- [9] Pete Hunt, Paul O’Shannessy, Dave Smith, and Terry Coatta. React: Facebook’s functional turn on writing javascript. *Queue*, 14(4):96–112, August 2016.
- [10] Internet Crime Complaint Center. 2019 internet crime report, Feb 2020. https://pdf.ic3.gov/2019_IC3Report.pdf, Last accessed on 2020-07-12.
- [11] Amin Kharraz, William Robertson, Davide Balzarotti, Leyla Bilge, and Engin Kirda. Cutting the gordian knot: A look under the hood of ransomware attacks. In Magnus Almgren, Vincenzo Gulisano, and Federico Maggi, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24, Cham, 2015. Springer International Publishing.

- [12] Jeff Kline, Paul Barford, Aaron Cahn, and Joel Sommers. On the structure and characteristics of user agent string. In *Proceedings of the 2017 Internet Measurement Conference, IMC '17*, pages 184–190, New York, NY, USA, 2017. Association for Computing Machinery.
- [13] T. Laakko and T. Hiltunen. Adapting web content to mobile user agents. *IEEE Internet Computing*, 9(2):46–53, 2005.
- [14] Aleksandr Lazarenko and Sergey Avdoshin. Anonymity of tor: Myth and reality. In *Proceedings of the 12th Central and Eastern European Software Engineering Conference in Russia, CEE-SECR '16*, New York, NY, USA, 2016. Association for Computing Machinery.
- [15] Dhruv Pandya. Voyager: Identifying IPs from Online Clicks . Master’s thesis, California State University, Channel Islands, 2017.
- [16] M. Pannu, B. Gill, R. Bird, K. Yang, and B. Farrel. Exploring proxy detection methodology. In *2016 IEEE International Conference on Cybercrime and Computer Forensic (ICCCF)*, pages 1–6, June 2016.
- [17] Jon Postel. Internet protocol. STD 5, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc791.txt>.
- [18] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. <http://www.rfc-editor.org/rfc/rfc793.txt>.
- [19] State of California - Department of Justice - Office of the Attorney General. High technology theft apprehension and prosecution (httap) pro-

gram, Nov 2015. <https://oag.ca.gov/ecrime/httap>, Last accessed on 2020-07-12.

- [20] Sipat Triukose, Sebastien Ardon, Anirban Mahanti, and Aaditeshwar Seth. Geolocating ip addresses in cellular data networks. In Nina Taft and Fabio Ricciato, editors, *Passive and Active Measurement*, pages 158–167, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.