

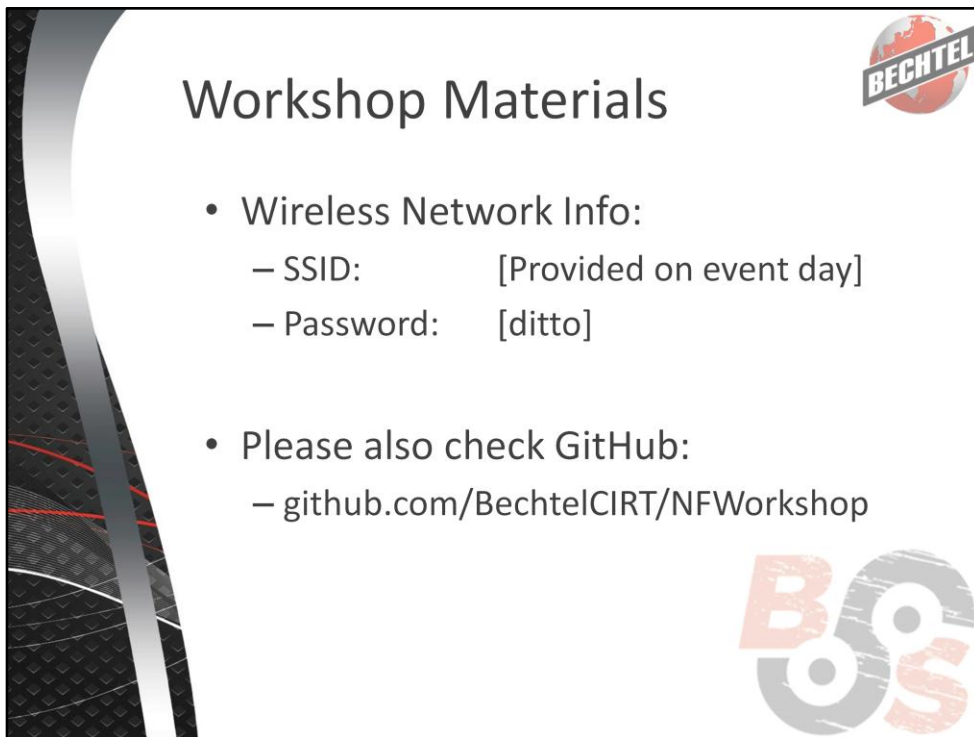


Network Forensics Fun: *Packet Pillaging Done Right, SON!*

In this workshop, I will walk attendees through how Bechtel's "Team DOFIR" took 1st place in LMG Security's Network Forensics Puzzle Contest (NFPC) at DefCon 22. Each year, LMG holds an awesome contest, and we are proud to show the tech that we used to complete last year's challenge.

To solve the sucker, we used tools such as Wireshark, tshark, tcpflow, bash, perl (regex one-liners baby!), Python, and others. I'll show how we put together some scripts and commands in order to streamline our methodology. My goal: Show off some cool network forensics tech and garner interest for this year's NFPC. We want some top-notch competition, so check out what we have to offer and be sure to get your game on this year!

Preferred: Participants show up with Kali Linux locked, loaded, and ready to rock. [Please install the bless hex editor: 'sudo apt-get install bless'.]



For this workshop, you will need a few different things (in addition to your Kali VM):

1) LMG's NFPC 2014 Files

This includes the PCAP files that are the focus of the workshop, not to mention the supporting materials that LMG provides. Without these files, we wouldn't have much of a workshop 😊.

2) Our workshop-specific materials


This includes our step-by-step instructions (.txt) and Python scripts.

To obtain all of the above, you will need to connect to the Wi-Fi network for the workshop.

Details will be provided the day of the event.

The workshop-specific materials will also be available on GitHub:

<https://github.com/BechtelCIRT/NFWorkshop>



Agenda

- Who's This Fool?
- The Challenge!
- Tools of the Trade
- Let's Do The Damn Thang!
 - Round 1, Fight!



This is what we'z gunna do, OK? OK!

Who's This Fool?

- Ryan J. Chapman (@rj_chap)
- Incident Response w/ 
- MS IA, BS CN
- GREM, GCIH, Linux+, LPIC-1, SUSE CLA, Sec+



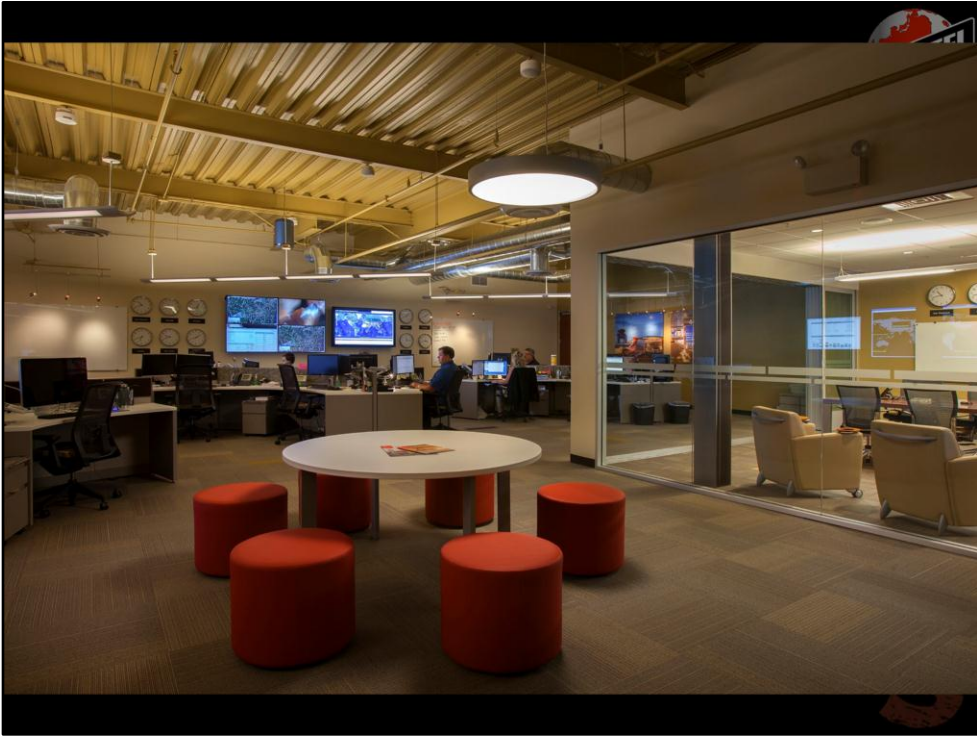
<https://www.linkedin.com/in/ryanjchapman>

I work for Bechtel Corporation. We have a SOC and CIRT, the two of whom work hand-in-hand to provide the security monitoring and security engineering, respectively, for one of the largest construction companies in the world. Check us out: <http://www.bechtel.com/>.

Shout out to my alma mater, Regis University. If anyone is looking for a good security program, check out Regis. At the very least, check out any university accredited as a Center of Academic Excellence (CAE) by the NSA and CSS: https://www.nsa.gov/ia/academic_outreach/nat_cae/institutions.shtml.

Sorry for the school tidbit, but I really want to pass this information along as much as possible. Anyone interested in security needs to know about the CAE program 😊.

Moving along...



I work here



I like these things



<http://forensicscontest.com/>
by LMG Security -- <http://lmgsecurity.com/>

These guys and gals freakin' rock. Follow them on Twitter!
@LMGSecurity

They also wrote a book that rocks:

Davidoff, S., & Ham, J. (2012, June). *Network forensics: Tracking hackers through cyberspace* (1st Ed.). New Jersey: Prentice Hall.



The Challenge!



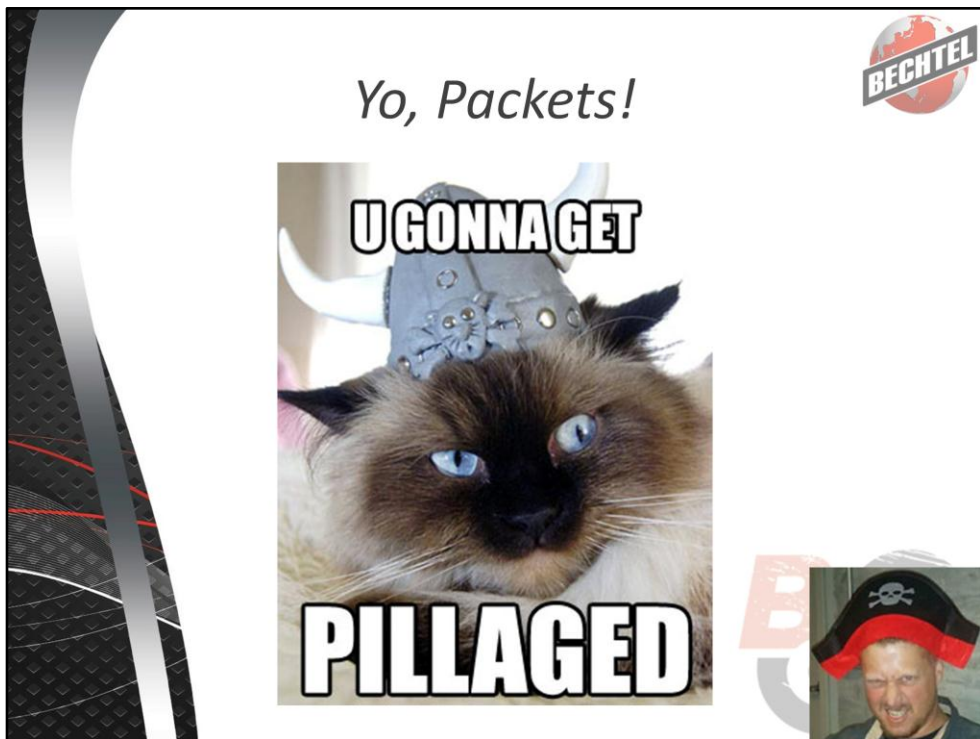
- NFPC Held @ DefCon 22 (2014)
- Hosted by 
- Bechtel Team DOFIR Took 1st!
 - 49 freakin' hours!!
 - We have < 2 hours, so let's get to it



Given our time constraints, we most likely will not make it through the entire challenge. That's OK! The slide notes were created for you to use at home/work/etc.

If you have any questions following the con, please ping me.

For today, my goal is to get through at least a few rounds, so let's get to it!



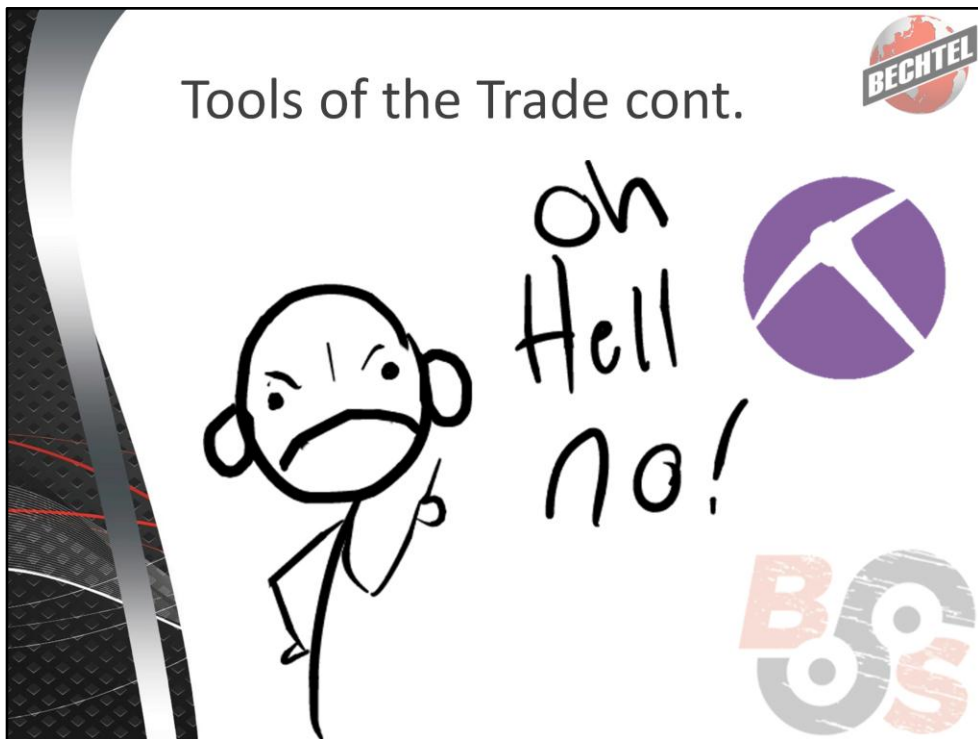
That's Steve in the corner. He's our resident pirate.

Tools of the Trade




RegEx






NetworkMiner is a useful tool, but I find the tool is often used as a crutch. In this workshop, we will not be using NetworkMiner.


Do not discount the tool though...



Methodology



- Read Round Dossier
- Discuss M.O.
- In WireShark, Always Check:
 - Protocol Hierarchy
 - Conversations
 - Endpoints

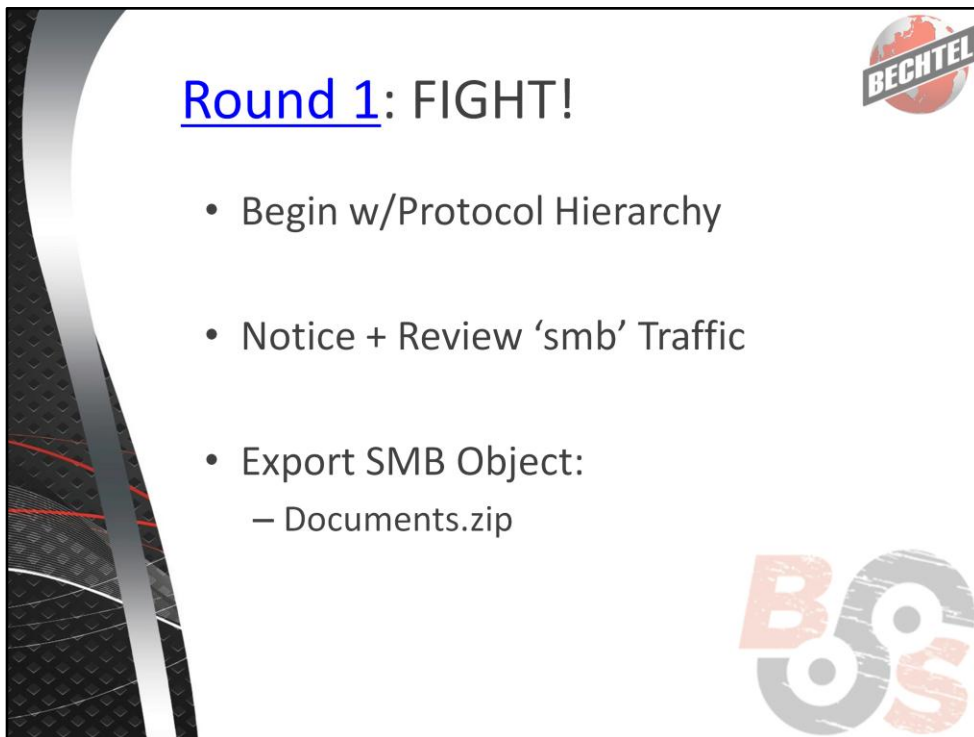


For each round, we will read the dossier together.

Then, we will cover our team's modus operandi. The goal is to answer the question: How did our team know where to look to solve this round?

Whenever loading a new PCAP in WireShark, make SURE to check the Protocol Hierarchy, along with the Endpoints and Conversations windows.

- These statistics-based, little windows of joy will make things much, much easier!




Round 1 Methodology:

We begin by creating a keyword list as usual, which is a great way to search through PCAPs for possible hits:

pyongyang, russian, comrade, document, snowden, information, bribery, chess, boxing, username... etc.


- 1) We noticed a decent byte count in 'smb' traffic in the Protocol Hierarchy, so we used the filter 'smb' to review SMB traffic
 - Scrolling through the smb traffic, we noticed some interesting share browsing, so we tried to export any SMB objects using the "easy method"
- 2) To export SMB objects, we used WireShark's 'File -> Export Objects -> SMB/SMB2' object export method
 - Doing so, we found that packet '24186' contained a file transfer for '\\DOG-WS\BLAH\Documents.zip'
 - We exported this object and then unzipped the file



Round 1 cont.

- Wu Tang SON!
 - Forget keywords for a second...
- HEY! Base64. Kewl.
- Decode Base64 → PROFIT (\$\$\$)

```
$ echo "[base64]" | base64 -d  
>>> base64.b64decode("[base64]")
```



Round 1 Methodology cont.:

4) In both 'track10.docx' and 'track6.docx', we find Base64-encoded text

5) We can decode Base64 a million different ways, but we prefer using bash or Python:

bash

```
echo "base64_here" | base64 -d | less
```


python

```
import base64  
base64.b64decode("base64_here")
```

The Mystery of Chess Boxing:
(usernames)


Mr. Method
Kim Ill-Song
Mr. Razor
...snipped...

<- This is the answer to Round 1 😊



Round 2: Begin!

- Ooooooh, IRC traffic!
- Found some PRIVMSGs
 - Not so private really
- Let's Extract the IRC Traffic
- More Base64-encoded ciphertext



Round 2 Methodology:

[Create keyword list!]

1) In the Protocol Hierarchy, we noticed IRC traffic, so we used the filter 'irc' to review

2) Looking through the traffic, we noticed some IRC private messages (PRIVMSGs)

3) We filtered the IRC traffic to view all the PRIVMSGs:


```
irc.response.command == PRIVMSG
```

3.5) We also used tshark (bundled with Wireshark) to check out which packets contained PRIVMSGs

```
tshark -r FIFA22.pcap -Y 'irc.response.command == PRIVMSG'
```


4) We then used tshark to export the IRC traffic to its own PCAP:


```
tshark -nr FIFA22.pcap -R irc -w irc.pcap
```



Round 2 cont.

- tcpflow fun
- Grab all Base64
 - Regex FTW
- Automation
 - Decoding w/Python
 - **DEMO**





Round 2 Methodology cont.:

5) Next, we used tcpflow (bundled with Wireshark) to reassemble the IRC chat sessions:

Copy irc.pcap to a new directory, cd to the new directory, and run:

```
tcpflow -r irc.pcap
```

6) Upon reviewing the tcpflow output files, we noticed that many of the PRIVMSGs contained Base64-encoded ciphertext. After decoding some of these bad boys, we noticed that some of them decoded to Base32-encoded ciphertext.

Pretty sneaky, sis!

7) We wrote a quick Python script to deobfuscate these twice-encoded (and other) messages. The script begins by looking for the 'PRIVMSG' string in each line. If found, the script extracts the content to the right of the colon, which delimits the message itself. Next, various deobfuscation attempts are performed:

ASCII hex w/in Base64	<code>base64.b64decode(strip_line).decode('hex')</code>
Base32 w/in Base64	<code>base64.b32decode(base64.b64decode(strip_line))</code>
Base64 w/in Base64	<code>base64.b64decode(base64.b64decode(strip_line))</code>
Base64	<code>base64.b64decode(strip_line)</code>

Please note the script is a no-frills POC. No argparse, no error-checking, etc. For that matter, we ran into some other encoding, but let's just ignore that for now ☺. Feel free to extend the script as you wish. **Python 2.7 required.**

[round2.py]

```
import sys
import os
import re
import base64

# Directory containing tcpflow output files (include trailing backslash)
directory = sys.argv[1]

output_file = sys.argv[2]

outputdata = []

for filename in os.listdir(directory):

    with open(str(directory + filename), 'r') as flow:

        for line in flow.readlines():
            if re.search('PRIVMSG', line):
                parsed_line = line.split(':')
                strip_line = parsed_line[-1].strip()


                try:
                    outputdata.append(str(line + '+++ Decoded ascii hex inside base64: ' +
                                           str(base64.b64decode(strip_line).decode('hex')) + '\n'))
                    continue
                except:
                    pass
                try:
                    outputdata.append(str(line + '+++ Decoded base32 inside base64: ' +
                                           str(base64.b32decode(base64.b64decode(strip_line)))) + '\n'))
                    continue
                except:
                    pass
                try:
                    outputdata.append(str(line + '+++ Decoded double base64: ' +
                                           str(base64.b64decode(base64.b64decode(strip_line)))) + '\n'))
```

```

        continue
    except:
        pass
    try:
        outputdata.append(str(line + '+++ Decoded base64: ' +
                                str(base64.b64decode(strip_line)) + '\n'))
        continue
    except:
        pass
    else:
        outputdata.append(str(line))

with open(output_file, 'wb+') as f:
    for line in outputdata:
        f.write(line)


```



Round 2 cont.

- Is That an MD5?!
 - Yup!
- Google is Your Friend
- Verify:

```
echo -n "Caracas" | md5sum  
C9fa5b8cb3b197ae5ce4baf8415a375b
```



Round 2 Methodology cont.:

8) In our output, we find the following decoded PRIVMSG:

PRIVMSG:

```
SkVRR0dZTE9FQ1JHS01ESk5ZUUDHT0xHTUUYV0VPRERNsvpXRU1KWkc1UV  
dLTkxETVUyR0VZTEdIQTJEQ05MQkdNM1RLWVJBTzVVWEkyREpOWVFISTJE  
RkVCM1dLWkxMR1k9PT09PT0=
```

Base64 Decoded:

```
JEQGGYLOEBRGKIDJNYQGGOLGME2WEODDMIZWEMJZG5QWKNLDMU2GEYLGHA  
2DCNLBGM3TKYRAO5UXI2DJNYQHI2DFEB3WKZLLFY=====
```

Base32 Decoded:

I can be in **c9fa5b8cb3b197ae5ce4baf8415a375b** within the week.

Check that out... does that remind you of anything? That little bugger right up there, the “c9fa5b8cb3b197ae5ce4baf8415a375b” string. YUP! It’s an MD5!


9) Google “c9fa5b8cb3b197ae5ce4baf8415a375b” and we find it’s the MD5 for the word “Caracas”, which is the answer to Round 2.

BTW, we can verify the md5sum of a string by using this method in bash:

```
echo -n "Caracas" | md5sum  
c9fa5b8cb3b197ae5ce4baf8415a375b
```


NOTE 1: You MUST use the “-n” option with echo, as this suppresses the trailing newline character. Otherwise, the string you would be running md5sum against would be ‘Caracas\n’, which would be a completely different hash.

NOTE 2: Many *nix distros include ‘md5sum’, but OSX uses ‘md5’.



Round 3: Get Some!

- Protocol Hierarchy
 - FTP traffic
- Conversations
 - Host-to-Host traffic (FTP)
- “Super Secret Server” @ 172.29.1.23
 - ZIP file accessed



Round 3 Methodology:


Two of us found the same data very quickly, using two different methods:

1A) Looking at the Conversations window, we sorted the contents by ‘Bytes’, which revealed two obvious endpoints: A Dell and a Gigabyte device communicating through a Cisco device.

The third item down in this sorted view shows host-to-host communication between these two devices. We filtered on this communication by right-clicking and choosing:
Apply as Filter → Selected → A<->B


1B) In the Protocol Hierarchy, we noticed FTP traffic, so we filtered the traffic using ‘ftp’.

Both of these methods revealed traffic to the “Super Secret Server” FTP server @ 172.29.1.23. We then noticed the file “ojd34.zip” being accessed.



Round 3 cont.

- Found 5 Total ZIP Files
 - tcp contains ".zip"



Round 3 Methodology cont.:

2) Wanting to find any additional .zip files, we used the filter 'tcp contains ".zip"'. This yielded a packet list that ended up providing us the five (5) total zip files for this level. NOTE: When working as a team, you should always assign the task of "look for more" to someone. Sure, we found five .zip files, but who's to say there were no additional files of importance? In this round, some moved on, while others keep looking for possible files.

Zip File 1 & 2: 34jdsioj.zip & breaking_bad_season_6.zip

Packet '2622', which is reassembled in frame '26661' -> Check MIME -> export raw data

Zip File 3 & 4: sandofwhich.zip & ojd34.zip

sandofwhich.zip - Packet '5891' starts FTP transfer

ojd34.zip – Packet '5983' starts FTP transfer

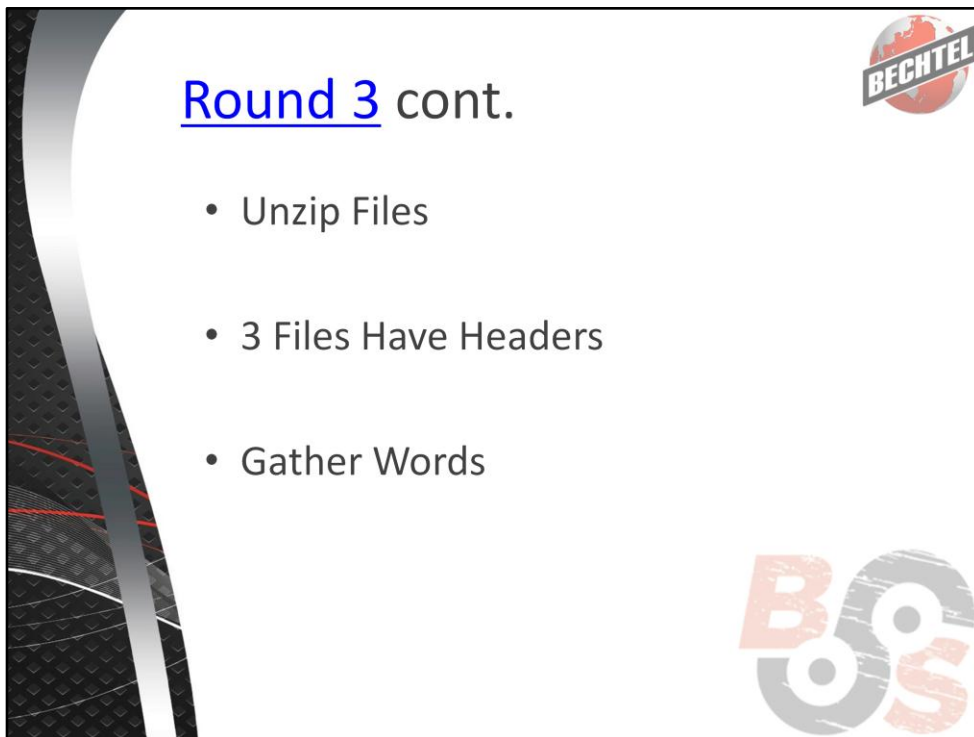
FTP uses port 21 for control and 20 for data, and WireShark uses 'ftp' for control and 'ftp-data' for data, so we filtered using 'ftp-data'.

sandofwhich.zip – File xfer starts in packet '5892' -> Follow TCP stream -> export tcp.stream eq 158

ojd34.zip – File xfer starts in packet '5938' -> Follow TCP stream -> export
tcp.stream eq 159

Zip File 5: canc3l.zip

AOL mail traffic in packet '8156', which is reassembled in packet '8190' -> Check
MIME -> export raw data



Round 3 Methodology cont.:

3) Upon extracting the .zip files, we found that they all contained filenames that seemed to form some type of phrase/paragraph/speech/etc. A quick Google search for a few of the words together led us to various articles regarding Snowden's famous quote.

We also noticed that three of the extracted files contained a file signature indicating they were a JPG, but only these three files contained headers. We figured that all files put together into a pool needed to be combined into three separate pictures, which was the case.

By unzipping all the zips at the same time, we can copy and regex the individual words from the archives:

```
unzip '*.zip'
```

Make sure to include the single quotes, or unzip won't process the files.

Copy the text output from the unzip command (all the filenames) to "files.txt" and then run:

```
perl -ne 's/.*\/(.*)\.jpg/\1/g; print' files.txt > words.txt
```



Round 3 cont.

- Search Google for Words
- Snowden Quote
<http://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance>
- Use Quote to Assemble Image

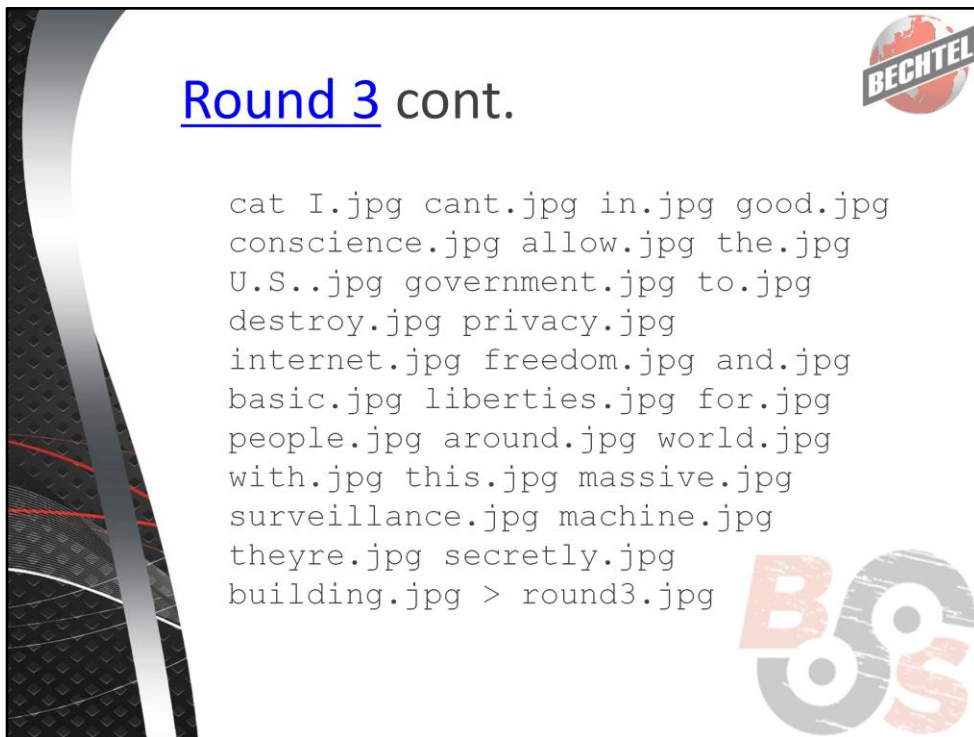


Round 3 Methodology cont.:

4) Searching Google for some of these words, we find the following quote from Snowden:

"I'm willing to sacrifice all of that because I can't in good conscience allow the US government to destroy privacy, internet freedom and basic liberties for people around the world with this massive surveillance machine they're secretly building."

<http://www.theguardian.com/world/2013/jun/09/edward-snowden-nsa-whistleblower-surveillance>



Round 3 Methodology cont.:

5) Make a new directory called "images" and copy all images to the directory:

```
cp */*.jpg ./images/
```

6) Re-Assemble the proper image:


```
cat I.jpg cant.jpg in.jpg good.jpg conscience.jpg allow.jpg the.jpg  
U.S..jpg government.jpg to.jpg destroy.jpg privacy.jpg internet.jpg  
freedom.jpg and.jpg basic.jpg liberties.jpg for.jpg people.jpg  
around.jpg world.jpg with.jpg this.jpg massive.jpg surveillance.jpg  
machine.jpg theyre.jpg secretly.jpg building.jpg > round3.jpg
```

7) Open round3.jpg. It's a chess set. An expensive chess set. We uploaded the picture to Google Images Search and found the following:

<http://www.chrisbathgate.com/#!sculpture-works/c5u>.


The chess set was a special commission art piece, made of metal by artist Chris Bathgate. We submitted "Chris Bathgate's Commissioned Chess Set," but the answer was just "**chess set.**" ☺

Note: We reassembled the two other images over time, which ended up being the Korean dictator and Voltron. Hehe, Voltron. Awesome.



Round 4: Get it Gurl!

- The Easy Round 😊
- What the Heck is “STUN” Traffic?
 - Not that kind of stun
- Scrollin’ Scrollin’ Scrollin’
 - Or Protocol Hierarchy 😊



Round 4 Methodology:


1) Right off the bat, we noticed “Session Traversal Utilities for NAT” (STUN) traffic in the Protocol Hierarchy. We reviewed this traffic using the ‘stun’ filter.

STUN traffic shows up in packet ‘6540’


2) Using the “Follow **UDP** Stream” option, we happened across the hostname.
To find it quickly, search for “@127.0.0.1” in the stream

Yeah, that was easy – **drpoppins-735**


(More info: <https://www.wireshark.org/docs/dfref/s/stun.html>)



Round 5: Stayin' Alive!



- Keyword Hit: “meet”
 - Keyword lists FTW!
- Found JSON Objects from Pinger
- GSP Coordinates?
 - Possible side channel in use



Round 5 Methodology:

1) From our initial keywords list, we found a hit for the string “meet”, which pointed us to packet ‘5473’. You can do a basic search or just use the filter: tcp contains “meet”. This packet contains a JSON object. In this object, we see messages being sent using a messaging service called Pinger.

Specifically, we see the following message in this packet:

From: Ann – “still we should be careful. Pay attention.

I want to meet in September at 5PM.”

This provided the month and time of the meeting, but we still needed the day of the month.

2) We used the filter ‘json’ to view all JSON objects and looked at the packets following our initial hit (after packet ‘5473’).

Starting in packet ‘6491’, we find some JSON objects with MapQuest data. In packet ‘7113’, we noticed a series of JSON objects being returned with latitude and longitude coordinates. Although we did not compete in the 2013 NFPC at DefCon 21, a few of us went through the challenge in order to prepare for this challenge. In that challenge, GPS coordinates were used to spell a password on a map. Realizing that we most likely had the same technique being used (side channel), we decided to review the raw packet details to get a better feel for how we could parse the data.

Round 5 cont.

- JSON Object in Packet 7113
 - tcp.stream eq 167
 - **SECRETS!**
- Regex ftw!
- Map Online or Cr
 - Python to the res
 - **DEMO**



Round 5 Methodology cont.:

3) We reviewed the first JSON object packet with GPS coordinates in packet '7113', which happens to be 'tcp.stream eq 167'. In this stream, we see:

```
{
  "info": {
    "statusCode": 0,
    "copyright": {
      "text": "\u00A9 2014 MapQuest, Inc.",
      "imageUrl": "http://api.mqcdn.com/res/mqlogo.gif",
      "imageAltText": "\u00A9 2014 MapQuest, Inc."
    },
    "messages": [],
    "options": {
      "maxResults": 1,
      "thumbMaps": true,
      "ignoreLatLngInput": false
    },
    "results": [
      {
        "providedLocation": {
          "latLng": {
            "lat": 46.85661315917969,
            "lng": -114.01860809326172
          },
          "locations": [
            {
              "street": "S Russell St",
              "adminArea6": "",
              "adminArea6Type": "Neighborhood",
              "adminArea5": "Missoula",
              "adminArea5Type": "City",
              "adminArea4": "Missoula",
              "adminArea4Type": "County",
              "adminArea3": "MT",
              "adminArea3Type": "State",
              "adminArea1": "US",
              "adminArea1Type": "Country",
              "postalCode": "59801",
              "geocodeQualityCode": "B1A AA",
              "geocodeQuality": "STREET",
              "dragPoint": false,
              "sideOfStreet": "R",
              "linkId": "0",
              "unknownInput": "",
              "type": "s",
              "latLng": {
                "lat": 46.856622,
                "lng": -114.018573
              },
              "displayLatLng": {
                "lat": 46.856622,
                "lng": -114.018573
              },
              "mapUrl": "http://mob.mapquestapi.com/staticmap/v4/getmap?key=Cmjtd|luaa2qu2nd,b5=o5-gzb0&type=map&size=225,160&pois=purple-1,46.856622,-114.018573,0,0,|&center=46.856622,-114.018573&zoom=15&rand=493008068"
            }
          ]
        }
      }
    ]
  }
}
```

The part we care about is the following:

```
"providedLocation":{"latLng":{"lat":46.85661315917969,
"lng":-114.01860809326172}}
```

4) We used a little perl one-liner with some regex to extract the coordinates from these JSON objects directly into a text file:

```
perl -ne 'print if
s/.*{"lat":(.*),"lng":(.*)}}.*/$1,$2/' RomanticDate.pcap >
latlng.txt
```

5) Next, we plotted them using online tools, **which yielded the answer of “17”**

<http://www.darrinward.com/lat-long/>

<http://www.hamstermap.com/quickmap.php>

Optional) To have some fun with Google Earth, we created a KML file using these coordinates. Since the KML file structure requires long,lat rather than lat,long, we simply changed the order of our capture groups in our regex:

```
perl -ne 'print if
s/.*{"lat":(.*),"lng":(.*)}}.*/$2,$1/' RomanticDate.pcap >
lnglat.txt
```

NOTE: Not only did we swap the capture groups (\$2,\$1), but we also named the output file “lnglat.txt” to reflect the different order.

We then wrote a quick and dirty Python script to create the actual KML file.

[See next slide]

```
import sys
import simplekml

kml = simplekml.Kml()

coordinates = open(sys.argv[1], 'r').read().splitlines()

point_count = 0

for point in coordinates:
    point_count += 1
    kml.newpoint(name="point %i" % point_count,
coords=[tuple(point.split(','))
    ])


kml.save("round5.kml")
```




To begin, we instantiate the KML object. Next, we set the 'coordinates' variable to a list consisting of each line from the input file (our 'lng,lat' formatted file). Then, we loop through each coordinate pairing, setting a new point in the output KML file for each pair. Finally, we save the result to a file.

NOTE: This is a quick and dirty script that we threw together while in the heat of competition. No, there's no error checking. This bad boy does what it needs to do and that's all. You can extend this if you'd like, but I felt the need to leave it as-is.


[Open the .kml file in Google Earth, or upload to Google Drive and simply click it.]



Round 6: Goin' Mobile!





- SNAP! An iPod Image!
 - iOS forensics?! Ruh Roh!
- Found “[iphonebackupbrowser](#)”
- iPod Backup Requires Decryption
 - Need the password ☹



Woah. An iPod backup, eh? That's different ☺.


One tool we found along the way was “iphonebackupbrowser”, which is useful when it comes to reviewing iOS backups. This tool confirmed that the iOS backup was encrypted, which was inferred due to the high level of entropy in the files.

Iphonebackupbrowser is an open source Windows tool. Given our time constraints, we'll skip reviewing this program, but you should check it out nonetheless.



Round 6 cont.

- Go Back to Round 1
 - Documents.zip -> More Documents
- Open “NorthKorea.jpg” in Hex Editor
 - Carve out the ZIP file -> broken.py.zip
- Fix the Script = Get iPod Password
 - But first, we need something else...



Round 6 Methodology:

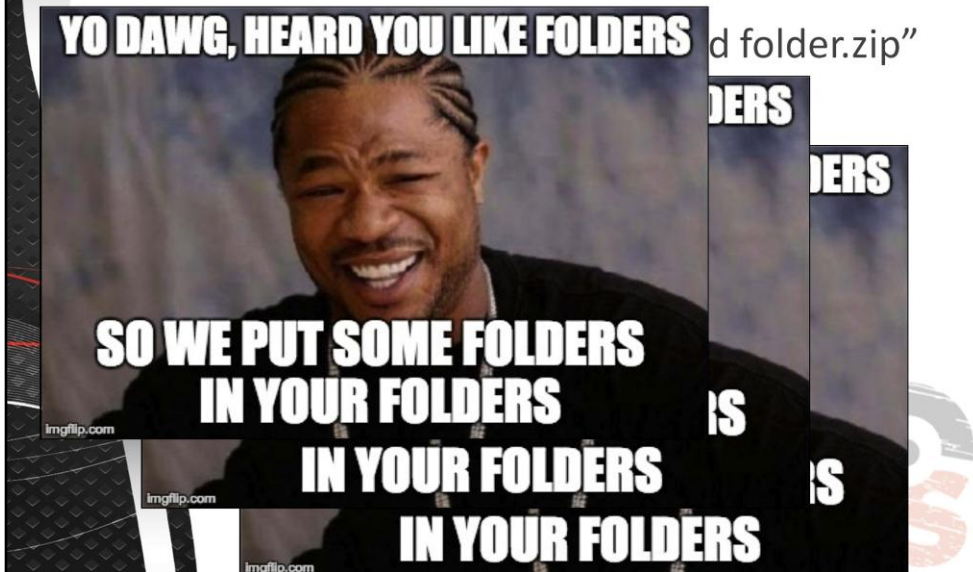
1) Carve the “broken.py.zip” file from the “Documents\More Documents\NorthKorea.jpg” file. Save the carved file as “broken.py.zip”.

Offsets: 0xd7d – 0x10bc

2) Unzip “broken.py.zip” and review the extracted data, which takes the form of a Python script in “/untitled/broken.py”.

We need to fix this script, but first we need something else...


Round 6 cont.



Round 6 Methodology cont.:


3) Going back to the "Documents/" directory, take a look at the "untitled folder.zip" file. Extracting this folder, we find a "Yo dawg!" effect that would garner Xzibit's personal approval. Eventually, you come to a final directory named "SilentEye". The directory is empty, but a quick Google search of this term brings up the SilentEye stego tool. STEGO!!!

NOTE: I won't lie. We put every darn image under the sun through SilentEye. We even pulled random graphics out of every PCAP up until this round, just to see what we would find. However, the answer was in the first image we popped into the darn thing: The chess set JPG file from round 3 (the one we had to reassemble).



Round 6 cont.

- Run SilentEye on ALL THE THINGS!
 - Chess board JPG from Round 3 FTW
 - Default options:
 - Luminance int: 5
 - Header position: bottom
 - Passphrase: SilentEye
- `i2454 2497d2496n2502 2470 2500 2507o2436s2452 2500s2503n2502l2487e2456 2497 2500h2485l2487 2470b2490e2491a2501m2466 2483a2501a2501e2505 2497 2500a2486`




Round 6 Methodology cont.:

4) Run SilentEye (Windows/Linux, doesn't matter) on the reassembled chess set JPG from Round 3. Leave the default options set! The output will be a secret ciphertext:


```
i2454 2497d2496n2502 2470 2500 2507o2436s2452
2500s2503n2502l2487e2456 2497 2500h2485l2487
2470b2490e2491a2501m2466 2483a2501a2501e2505 2497
2500a2486
```

Now that we have this encoded ciphertext, we can work to fix the “broken.py” script.



Round 6 cont.

- OK, Time to Fix the Script!
 - DEMO
- And Now We Decrypt the Backup
 - Master tool: [iphone-dataprotection](#)
 - DEMO
- Review “com.apple.wifi.plist”



Round 6 Methodology cont.:

5) Follow these steps to fix the script:

Optional) Replace the tabs with spaces and delete the extra lines – This just bothers me ☺. Also, take note:

We will be focusing on `indexInASCII()`, which is the decode function. We know this, because the return has a very nice comment for us:

```
#returns decoded message
```

5A) Close the parenthesis at the end of line 17 and 24:

```
17     sums+=ord(x) ← derp
24     indices.append(ASCIIArray.index(chrs)+sumName(name)*2) ← derp
```

This resolves the two major syntax errors, but the sucker isn't fixed yet. In fact, the script doesn't call any declared function(s).

5B) Look at the `encode()` function. Notice that it gives away what should be the variable 'name':

```
#convert file associated with name to a string
bill = fileToString("./%s.txt"%name)
```

Here, we see that 'name' is the name of a text file. Looking in "Documents\More Documents\", where we found the script, we see "BillofRights.txt".

Add the following to bottom of the script:

```
secret = indexInASCII('BillofRights')
print secret
```

This will set 'secret' by passing the string "BillofRights" to the indexInASCII() function, and finally print the decoded secret.

5C) When we try to run the script, we receive an error regarding line 10 in the ASCII() function. Looking at the function, we see the comment, "#number of ASCII characters" followed by "NumOfASCII == 0". Yup, another broken part of the script. The ASCII character set consists of 128 characters. For that matter, we don't have an assignment equals sign, which should have been used. Thus, let's change line 10 to reflect the proper # of ASCII characters:

```
NumOfASCII = 128
```

5D) Whoops! The script is trying to use the 're' module, but it's not imported. Add the following line to the *very* top of the script:

```
import re
```

5E) Now we see that line 33 is having issues resolving the variable 'encoded.' This is where the encoded value we found hidden in the chess board JPG from Round3 comes into play.

Add the following just above the call to indexInASCII('BillofRights'):

```
encoded = 'i2454 2497d2496n2502 2470 2500
2507o2436s2452 2500s2503n2502l2487e2456 2497 2500h2485l2487
2470b2490e2491a2501m2466 2483a2501a2501e2505 2497
2500a2486'
```

6) OK. The script is fixed. Run the sucker! – python broken.py. Results:

DontTry2BruteForceThisPassword

BOOM! Now we have the password to iPod backup. Sweet.

Let's use this password to decrypt the encrypted iPod backup:

7) We are going to use the mother of all iPhone backup/decryption tools:

[iphone-dataprotection](https://code.google.com/p/iphone-dataprotection/) – <https://code.google.com/p/iphone-dataprotection/>

NOTE: During DefCon 22, the functionality we will need from this tool to solve Round 7 was not yet part of the official code base. However, we found some patch notes here: <https://code.google.com/p/iphone-dataprotection/issues/detail?id=115>.

That's right, in Issue 115 of the Google Code base, we found what we needed and patched the program.

At the time, the patch was non-functional as provided and required some tweaks in the error-checking to get things running. These days, we have a patched version available:

<https://code.google.com/p/iphone-dataprotection/source/detail?r=44cf8b8dcb78>.

NICE! Download the updated code: "iphone-dataprotection-44cf8b8dcb78.zip".

8) Decrypt the backup as such:

Note) iphone-dataprotection has quite a few dependencies, such as:

```
sudo pip install protobuf
```

8A) Extract iphone-dataprotection and copy the "iphone-dataprotection-44cf8b8dcb78" folder to a new directory called "round6".

8B) Copy the encrypted backup folder, c7837a1b219513667089ec628044cfb9f8db2d14, to the "round6" directory.

8C) Make a new directory called "decrypted" in the "round6/" directory [?] "round6/decrypted/"

8D) cd into the "round6" directory and run the following:

```
python ./iphone-dataprotection-44cf8b8dcb78/python_scripts/backup_tool.py  
./c7837a1b219513667089ec628044cfb9f8db2d14/ ./decrypted
```

Choose "y" to extract the backup, then enter the backup password:

DontTry2BruteForceThisPassword

ROCK ON! We now have a decrypted copy of the backup in "round6/decrypted". We'll need this going forward, so don't delete it anytime soon.

9) Look for the following file in the decrypted data folder:

"SystemPreferences Domain -> System Configuration -> com.apple.wifi.plist"

NOTE: The resulting file is a binary plist:

<https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man5/plist.5.html>.

Open the .plist file in Xcode on OSX or check the following for Windows:

http://www.forensicswiki.org/wiki/Converting_Binary_Plists.

Check "Root -> List of know networks -> Item 0 -> SSID_STR", which is "LeakingSecrets".

BOOOOOM! The SSID is "LeakingSecrets". This is the answer to Round 6. Moving to the FINAL round!!

Round 7: Almost There!



- Amazon FireTV Forensics?!
 - Not quite
- Two PCAPs:
`/usr/share/networkFiles/`
- Check “trekking.pcap”
 - Search “.zip”



Round 7 Methodology:

After extracting the “firetv” directory, we noticed two (2) PCAP files in the “firetv/usr/share/networkFiles/” directory. Let’s review.

trekking.pcap


1) Search for .zip files using the filter: tcp contains “.zip”. We find packet ‘463’ (tcp.stream eq 34), which is reassembled in packet ‘18653’. Go to this packet and check the decoded MIME-encoded message:

```
form-data;name=\"file0\";filename=\"jumbled.mp3.zip\"
```

2) Extract and then unzip the “jumbled.mp3.zip” file. Play the MP3 file in an audio player. You will hear a woman’s voice spelling out “True HOOHA”. Take note of this, as we’ll need it later.


You might notice a weird “blip” in the audio. You might also notice that the short LAME-encoded MP3 file is > 10MB. That seems odd for such a short audio clip. You may even notice that media players (such as VLC) state that the recording is around 11 minutes long. Weird...

Next slide ☺



Round 7 cont.

- Check amazon.pcap
 - Encrypted wireless traffic!
- Find “LeakingSecrets” WPA Key
 - Patch required: [Issue 115](#)
- Decrypt amazon.pcap
 - Filter on “synergy”



Round 7 Methodology cont.:

amazon.pcap

The “amazon.pcap” packet capture contains encrypted wireless traffic. We can see the beacon frames throughout, but if we filter on “eapol”, we see the key handshake. In the previous round, we analyzed the decrypted “com.apple.wifi.plist” file. When doing so, we noticed that the network “LeakingSecrets” network was secured with WPA.

Since we have encrypted packets from the “LeakingSecrets” network in the “amazon.pcap” file, we need to find the WPA key for this network to decrypt the traffic.

1) From the “round6” directory we made in the last round, run the following:

```
python ./iphone-dataprotection-44cf8b8dcb78/python_scripts/keychain_tool.py -d  
./decrypted/KeychainDomain/keychain-backup.plist  
./decrypted/Manifest.plist > keychain.txt
```

Review the resulting keychain.txt file. You’ll find the password to the LeakingSecrets network:

UncrackableNetwork75

2) In WireShark, with "amazon.pcap" open, choose "Edit -> Preferences -> Protocols -> IEEE 802.11". Select the "Enable decryption" checkbox (important!). Then, click the "Edit..." button at the bottom, next to "Decryption Keys". Click the "New" button. Change the "Key type" to "wpa-pwd".

Enter "**UncrackableNetwork75:LeakingSecrets**" as the Key. Press "OK" and then "OK" again, at which point the traffic will decrypt.

Finally, close the Preferences window by pressing "OK" once more.

3) Pull up the Protocol Hierarchy. Take a gander at the "Synergy" traffic. Seem odd/different/interesting? Synergy is a mouse and keyboard (think software KVM) sharing program: <http://synergy-project.org/>



Filter on the Synergy traffic by using the filter 'synergy'.

4) The first packet that shows under this filter is '3817'. Follow the TCP stream, which ends up being "tcp.stream eq 3". Scroll through this traffic.

Note that "DKDN" refers to "key down", which provides the key(s) being pressed over the Synergy session.


You'll find the message "cats and bears and stuff," but that's not important. In the section following this message, you'll find the following message:

"carve 0x250c6" followed by "cave [sic] 0xa250c5"



Round 7 cont.

- Carve “jumbled.mp3” @ Offsets
- What’s the resulting blob/bin?
- TrueCrypt Archive!
 - What’s the password?
 - Try “True HOOHA”
 - Watch “Ann_love_letter.mov”



Round 7 Methodology cont.:

5) Go back to the “jumbled.mp3” file. Carve the file using a hex editor at the locations provided, 0x250c6 - 0xa250c5, and save the result as “carved.bin”.

6) Review the “carved.bin” file in a hex editor. See any file signature? NO. Notice anything about the file’s entropy? VERY HIGH.

What kind of file has very high entropy and no file signature? TrueCrypt volume ☺.

7) Try to mount the “carved.bin” file in TrueCrypt. When prompted for a password, use “True HOOHAH”.

8) Watch the “Ann_love_letter.mov” movie.

\$\$\$ - The movie contains the answer to Round 7: **New Jersey**

Achievement Unlocked! Level: Jedi Knight!



That's it gang! Thanks for fartin' around with me!

We welcome any comments, suggestions, or questions. Have a cooler or alternative way of doing something we covered? Let us know!

If you want to give me a holler, hit me up **@rj_chap**. I love to discuss network forensics, reverse engineering... whatever!

I would like to give a HUGE shout out to the Bechtel team, as this was the epitome of a team-based event. Rock on team!

If you are interested in a career w/Bechtel, check out our Careers page:
<http://jobs.bechtel.com>

Of course, I wouldn't have a workshop to present if it wasn't for **LMG Security** and their awesome NFPC. Thank you guys and gals for everything you do. **Your event deserves to be a black badge event at DefCon!**