

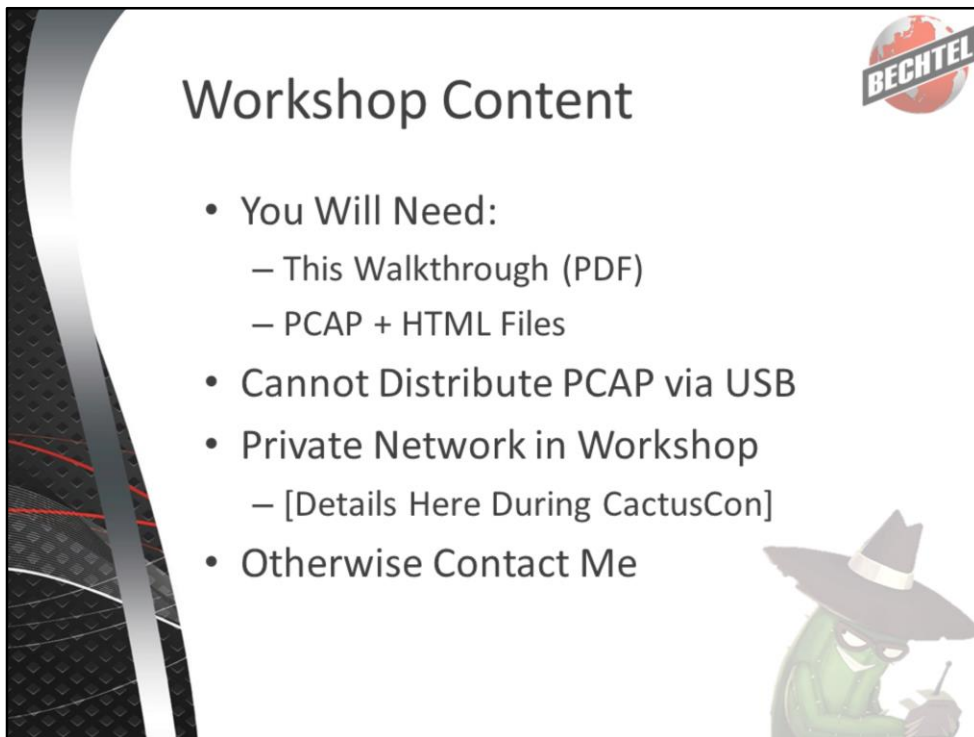


Network Forensics Workshop Deux: *Long Live Packet Pillaging!*
<http://www.cactuscon.com/ryan-chapman/>

The most up-to-date version of the workshop materials, including this PDF, can be found here:

<https://github.com/BechtelCIRT/NFWorkshop16>

Participants will walk through each step, learning about Wireshark, Python, Volatility, NetworkMiner, bash, and other tools as we cover the process to complete the challenge. The workshop will cover tips and tricks such as utilizing Wireshark filters, conversations, and the protocol hierarchy to expedite forensic analysis.



For workshop files and content, please see:

<https://github.com/BechtelCIRT/NFWorkshop16>

To prepare for the workshop, please install some required tools in Kali as such:

`sudo apt-get install bless audacity tcplay`

Bless – Great *nix-based hex editor

Audacity – Audio playing/editing program with the ability to play audio files in reverse (see Round 3)

TCPlay – Free and simple TrueCrypt Implementation based on dm-crypt (see Round 6)

If you received the competition files in the original TrueCrypt disk image format (.tc volumes), you will need the following passwords to decrypt the rounds:

Round 1: WhcFDjEQm9

Round 2: 4TWSDjtAeb

Round 3: jHfk4ykZBC

Round 4: 86BNnSn7Jp

Round 5: djawp7Tw6W

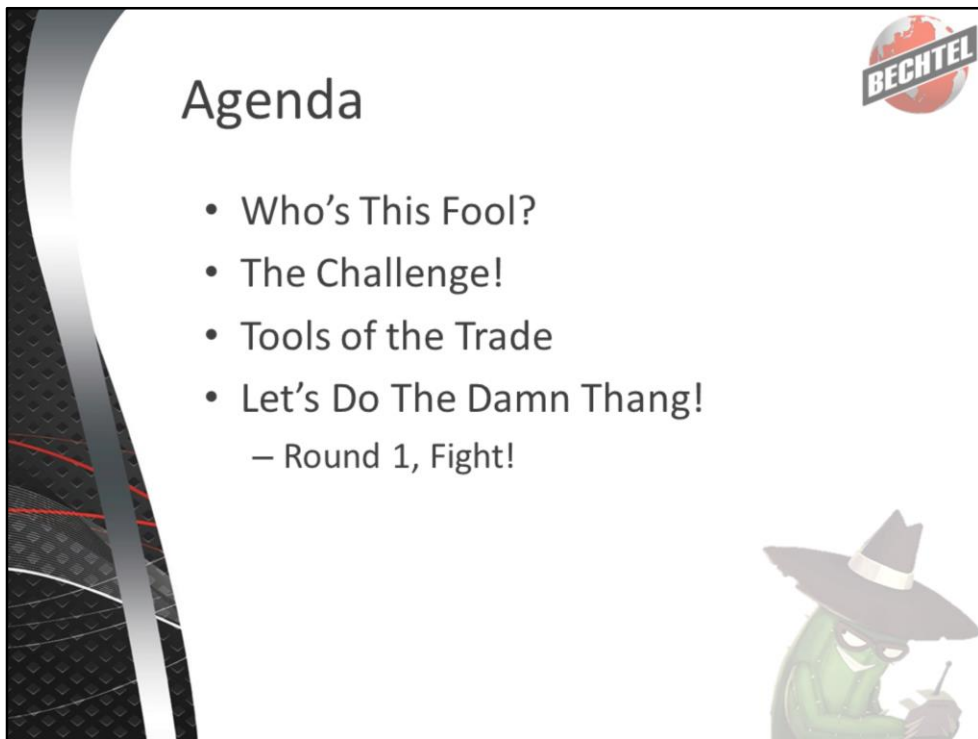
Round 6: hcdLwUKPTC

Round 7: zxEjEhCsVP

(This is a bonus round that we do not cover in this workshop, but I felt it appropriate to include the password anyway.)

During the competition, LMG slowly released hints for the competing teams. While I recommend you avoid them for this workshop (except for Round 6, dangit), you can find the hints on LMG's official competition blog:


<http://forensicscontest.com/2015/08/11/nfpc-2015-passwords-and-hints>




This is what we’z gunna do, OK? OK!

Please keep in mind that *we will not will finish the challenge during the workshop*. Given our time constraints, we may only finish a few rounds.



However, I have developed this workshop as a step-by-step guide to walk you through the challenge from start to finish. Not only will everyone be able to pick up where we left off, but anyone who did not attend the actual workshop will not miss anything other than me running my mouth and making silly and/or inappropriate jokes.



Ryan J. Chapman



- Incident Response Analyst
– Network Security Monitoring
- MS IA / BS CN
- GREM, GCIH, blah blah
- I Like To Run My Mouth
– *You'll See*



Ryan Chapman
@rj_chap

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.

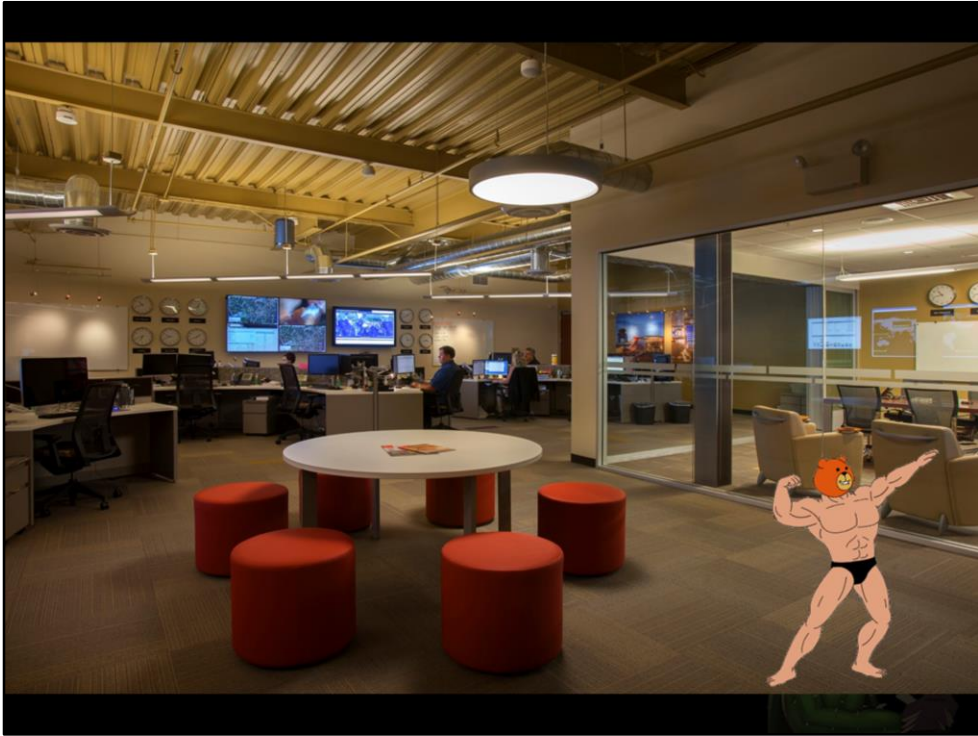
Check us out: <http://www.bechtel.com/>



I like these things:

- Retro gaming
- Stand-up comedy
- The Wheel of Time fantasy series
- Brazilian Jiu-Jitsu

Oh. And my family: Wife and daughter. They are cool too.



This is our Security Operations Center (SOC)

It Takes A Village

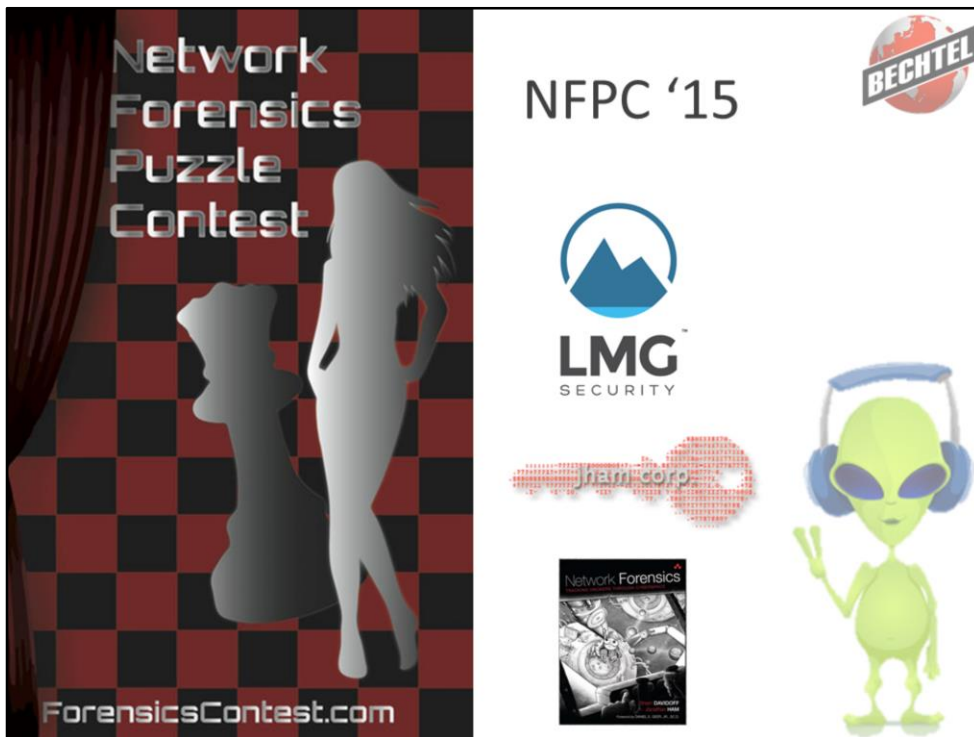


- Bechtel Team Threat Level Pancakes
- Here To Keep Things Kewl & Smooth
 - Just don't try to touch us in an inappropriate way
 - Well maybe Chris... He likes that sort of thing



Bechtel Team Threat Level Pancakes is in the house!

[Roll call for folks helping out during workshop]



In last year's CactusCon workshop, we attacked the NFPC '14 from DefCon 22. This year, it's time for NFPC '15 from DefCon 23.

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


These guys and gals freakin' rock. Go follow them on Twitter:
@LMGSecurity


LMG also has a pretty darn good book:


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



NFPC '15 cont.



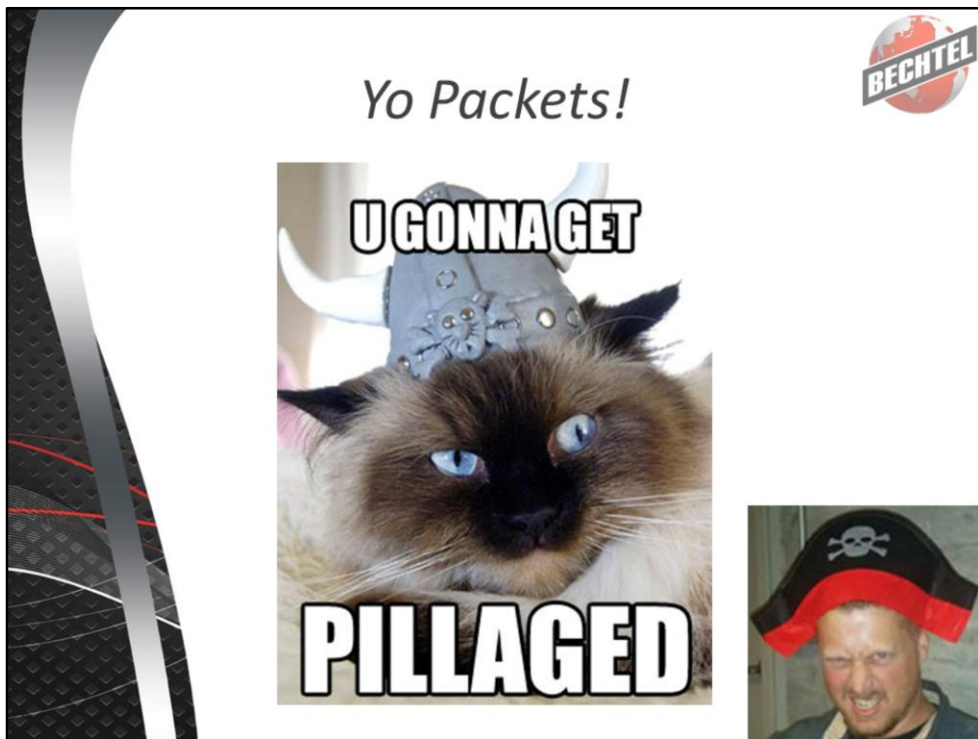
- NFPC Held @ DefCon 23 (2015)
- Yearly Competition by 
- Threat Level Pancakes Took 1st!
 - 2nd Year Running Baby!
 - Took 36 Freakin' hours!!
 - We have < 2 hours, so let's get to it



To re-iterate: 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, go ahead and ping me.

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



It's about time to PILLAGE SOME PACKETS!


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

Tools of the Trade




RegEx





Methodology

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



For each round, we will begin by reading the round 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: FIGHT!

- Begin w/Protocol Hierarchy
- IRC Traffic! We love IRC Traffic.
- PRIVMSGs FTW



Round 1 Methodology:

Q: What is the first and last name of the crazed patron?


We often gravitate toward Internet Relay Chat (IRC) traffic when we see it, so we jumped at the opportunity to review the IRC traffic in this capture.

1) Filter the network capture to review IRC traffic by using the following filter:
irc

2) Take note of the IRC private messages (labeled "PRIVMSG")


3) Filter the IRC traffic to view just the PRIVMSGs:
irc.response.command == PRIVMSG

4) Use **tshark**, a tool bundled with Wireshark, to export the IRC traffic to its own PCAP:
tshark -nr Round1.pcap -Y irc -w irc.pcap
(If the **tshark** command does not work, change directory into the Wireshark install directory and try again.)



Round 1 cont.

- **tcpflow** to the Rescue!
- Grab all PRIVMSGs
- Automation / Bulk Processing
 - Quick Python Script to Decode
 - **DEMO**



Round 1 Methodology cont.:

5) Use **tcpflow** (another tool bundled with Wireshark) to reassemble the IRC chat sessions. To do so:

Create a new directory called **irc**, copy **irc.pcap** to this new directory, to the new directory, and then run **tcpflow**:

```
mkdir irc
cp irc.pcap ./irc/
cd irc
tcpflow -r irc.pcap
```

6) While there are many ways to go about this, extract the private messages to a single file using the shell command:

```
cat 083* 162* 192* | grep 'PRIVMSG' > irc_privmsg.txt
```

7) Upon reviewing the the **irc_privmsg.txt** file, we noticed that the private messages look to be ROT13 encoded. How did we know this? Experience. When you have viewed enough ROT13-encoded messages, you just start to recognize them. We simply had a hunch, and it paid off when we began deobfuscation. Win!

Caesar would be proud!

8) Last year, we wrote a quick and dirty Python script to decode various IRC private messages (See here: <https://github.com/BechtelCIRT/NFWorkshop/blob/master/NFWorkshop-round2.py>). We modified this script to decode the ROT13 messages. Since ROT13 means “rotate 13 characters,” we can simply ROT13 the ROT13 messages to create a ROT26 situation, which just rotates all letters back to their original positions.

Please note that just like last year, this script is a no-frills POC. No argparse, no error-checking, etc. Feel free to extend the script as you wish. **Python 2.7 required.** (Maybe not, but Python 3 is horrible. YEAH I SAID IT.)

For example, to run the script from your Round1 directory, which contains the **irc** directory, you would run the following:

`./round1.py ./irc/irc_privmsg.txt ./irc/irc_decoded.txt`

```
#!/usr/bin/env python
import sys
import os
import re

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

output_file = sys.argv[2]

outputdata = []

with open(str(file), '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
ROT13: ' + str(strip_line).decode('rot13') + '\n'))
                continue
            except:
                pass
            else:
                outputdata.append(str(line))

with open(output_file, 'wb+') as f:
```




```
    for line in outputdata:
        f.write(line)

print "Done!  See %s." % (output_file)
```



Round 1 cont.

- Out of Order
 - Nature of **tcpflow**
- We Could Rectify, BUT ANSWERS!
- ANSWER:
Dimitri Bogomolovo



Round 1 Methodology cont.:


9) After running the script, you will find the following decoded PRIVMSG (most likely in **irc_decoded.txt**):

:becausealiens!~becauseal@162.219.72.250 PRIVMSG #meeh :Lbh zrna Qvzvgev
Obtbzbybib. Whfg orpnhfr ur'f ba gi qbra'g zrna ur'f nal yrff penml.^M
+++ Decoded ROT13: **You mean Dimitri Bogomolovo**. Just because he's on tv doesn't
mean he's any less crazy.

The answer to Round 1 is:


Dimitri Bogomolovo

Rock on. Let's check out Round 2.



Round 2: You 'Member?

- Run **imageinfo** & **kdbgscan**
- Suggested Profiles:
 - WinXPSP2x86, WinXPSP3x86
- Check the Build String:
 - 2600.xpsp_sp2_gdr.100216-1441
- Going forward:
 - **volatility -f Round2.mem --profile=WinXPSP2x86**



Round 2 Methodology:

Q: What is the MD5 sum of Dimitri's research file?

This round stepped away from network forensics and delved into the work of host-based forensics in the form of memory analysis. The provided memory image, **Round2.mem**, can be analyzed with a bevy of tools. The two powerhouses in the memory analysis realm are Volatility (<http://www.volatilityfoundation.org/>) and Rekall (<http://www.rekall-forensic.com/>). Technically, we used both during the competition, but we are going to focus on Volatility for this walkthrough. Aside from being personally partial to Volatility, v2.4 is installed by default on Kali 2.0. As a note, Volatility 2.5 introduced initial Windows 10 support. Since we don't need this feature, we can use the default Kali install.

When using Volatility, your initial task is to identify and configure the proper profile for the memory image. Volatility uses profiles to designate the operating system (OS) that was running on the host from which the memory image came. If you take the memory image yourself or have the forensic analyst notes for the image, you will most likely know exactly which profile to select. However, if you are unsure, you can use either the **imageinfo** or **kdbgscan** plugins to determine the profile.

- For more information on profiles, see here:
<https://github.com/volatilityfoundation/volatility/wiki/Volatility%20Usage#selecting>

-a-profile

- For info pertaining to image identification modules, see here:
<https://github.com/volatilityfoundation/volatility/wiki/Command-Reference#image-identification>

1) To identify the proper Volatility profile, run the following command:

volatility -f Round2.mem imageinfo

(The -f argument is used to provide the input filename for volatility)

You will most likely receive results similar to the following:

```
Volatility Foundation Volatility Framework 2.4
Determining profile based on KDBG search...

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated
with WinXPSP2x86)
      AS Layer1 : IA32PagedMemory (Kernel AS)
      AS Layer2 : FileAddressSpace
(/mnt/hgfs/Public/NFPC2015/Round2/Round2.mem)
      PAE type : No PAE
      DTB : 0x39000L
      KDBG : 0x8054c2e0
Number of Processors : 1
Image Type (Service Pack) : 2
      KPCR for CPU 0 : 0xffdff000
      KUSER_SHARED_DATA : 0xffdf0000
Image date and time : 2015-07-14 18:54:11 UTC+0000
Image local date and time : 2015-07-14 12:54:11 -0600
```

You will notice that the module provides a recommendation for two different profiles:

Suggested Profile(s) : **WinXPSP2x86, WinXPSP3x86** (Instantiated with WinXPSP2x86)

2) To identify the proper profile further, you can run the **kdbgscan** module as such:

volatility -f Round2.mem kdbgscan

In this output, you will see the following snippets:


```
Profile suggestion (KDBGHeader): WinXPSP3x86
Build string (NtBuildLab)       : 2600.xpsp_sp2_gdr.100216-
1441
```

```
Profile suggestion (KDBGHeader): WinXPSP2x86
Build string (NtBuildLab)       : 2600.xpsp_sp2_gdr.100216-
1441
```

Again, we see that both XP SP2 and XP SP3 profiles are recommended. You will also notice that the build string in both instances specifically designates SP2. Let's go with that. Target Profile = **WinXPSP2x86**.

Now that we have our profile, any subsequent Volatility commands that we run will start with:


```
volatility -f Round2.mem --profile=WinXPSP2x86
```



Round 2 cont.

- Commands to Run:
 - **clipboard**
 - **notepad**
 - **mftparser --dump-dir='./mft/'**
- Grep the MFT Dump
 - **grep -ir 'research' .**
- Generate MD5 Sum
- ANSWER:

19bebeab4457def688c9520b28464157



Round 2 Methodology cont.:

We spent a decent amount of time toying with various Volatility commands before we found the answer to this round. For a list of Volatility commands (up to the current version), see here: <https://github.com/volatilityfoundation/volatility/wiki/Command-Reference>. For workshop purposes, we are going to run the commands that net vital information.

Hoping for an easy win, we ran the **clipboard** and **notepad** modules right away. Both commands provided critical details that we needed for later rounds.

**** Required Items for Future Rounds ****

3) Let's take a look at the clipboard contents using the **clipboard** module:

volatility -f Round2.mem --profile=WinXPSP2x86 clipboard

```
Volatility Foundation Volatility Framework 2.4
Session      WindowStation Format                Handle Object      Data
-----
0 WinSta0    CF_UNICODETEXT    0x2a003f 0xe294d738
acturians first contact with stegosauruses
```

0	WinSta0	CF_LOCALE	0x60105	0xe274b4e0
0	WinSta0	CF_TEXT	0x1	-----
0	WinSta0	CF_OEMTEXT	0x1	-----

The WinSta0 WindowStation contains a Unicode string: **“acturians first contact with stegosauruses”**. This little sucker is a hint for Round 4. However, the hint and the answer do not quite match. You’ll see what I mean. For now, keep this string in your pocket and we’ll come back to it later.

Note: If you are not familiar with the concept of a Window Station, check out the following MSDN article: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms687096\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687096(v=vs.85).aspx) (or just Google “msdn window station”).

4) We can review the contents of any open instance of **notepad.exe** using the **notepad** command:

volatility -f Round2.mem --profile=WinXPSP2x86 notepad

```
Volatility Foundation Volatility Framework 2.4
Process: 2320
```

```
...
Text:
Open: 3,9
Closed: 4
Open: 25,1,22
Closed: 9,12,19,21
Open: 11,18
Closed: 2
Open: 10,16
Closed: 13
```

```
Process: 2376
```

```
...
Text:
8dagrl+7EiLkJER4JmBEy6owEZdPEQBt90S2QJ0DvgRVEE81AQgAnuVdCSsKR0O1
0sdaF3lYLZeluBdvC9/VR6MJtO72HFyxDEJ3iD46GptJC8ePvK+fUD/lc+s3gQUP
flPQy7iv651KcdVNvUKBoNUQtU5b97me2Egj4R7YnWbBk024G8qFRk/4if0TUCiZ
aOZRUYtTa0IOksya4WupFzRQMq61pKUGz4XTlilLrN+c88AG9fQ4/+jvS/RRMEIZ
lkJDDsNomuuZZFqdtSORvsDv4eeZ918NB/e7hizWBz1CA/F1luxlXt86/RVcdI0P
C4N2N3P3frRrjAQvx06PIEw5gm6mQxjCXBdcVe74mzGQwFordcJ+rF9nV6O0ZOUl
yooPIucQcQARAQAB/gMDAhDdgA/GJMA+0B8Y4lOIBo3dlKMdQuNJCqUS5JjRKF+a
T/ij2aWKleX/7xeEe/28Oz7oYy/KyalRyP9NXRG8+YGHUVH/BMUKYbom3Mhis/Dr
aETbcaVYlyiko9eK1blBG3FOCO/aLFxQMrHT1gN9CF2JK2iUX86NwhU1Eq5OYptu
J9yw8190zP3qnKhvbwV0kkawK8a0wwofb6eAt9H3y9YM3YIsNRKRVRUeWzntp6dK
IHGnbVN0bU6olvUueTusGbrRbuATx6V2A6NJAR+wStQyapdFOBVGNU8Fqxq3GwQW/
PXvXapjrFXvv1Rf+oSxboiOgoq4J3Erfd6GVda1ZjpVoaAPt+z+XHK8SJTa0VwWy
HZ0P5WlhM+oIix20uYEYnPOPsXqw5/laqoBDNF39RVt1bnHz9113WpYa3IC7uJgT
```

o6lv5+z3Edj0z/rmG69gAS0oEaCv++lIoDIFi5LyRvPrf89R2i6f51dmuyCAsHto
qrwKSkza+xbKJaGSwQHKBI2qHbAjDvLlRekRaLzBFYyZHD+rmrzfJnioloBVtGTx
HmfEY9HXAMNWDi9xypYArXS10oswe8nUmvMn+gqYnjSwOadaOsTAT6sJUnxLV2/U
in4/tb1JIG0Rn0LMKOpj2BCnMee9vA14FRhk3PJ5qNUObX7n3R5f11PYc4q2sQzQ
BD5p+h7gNNd9VPp3zcLJhToVWgA8FDMff6EUabHJMokmdXI6I3hfp5XbJPeHyRZv
hGN4M7SuVMsmHaem86eVTp1V+mSYDemMxnNQ1uw36hQMPPwzbIoAbbUSNYhIb1Qm
mJ4fiMJSAMPmAiKu2dY+xZOlmWEnOYU2wHeN8cyHCElh+rVZmXYxdmLS5F5SRC7c
EOhwInqkJYDnn6Hpe8PL1+u/rB2GDyph1+Bnrj+JASUEGAECAA8FA1V4TzUCGwwF
CQeGH4AACgkQOUxGFasYh551yAf9EARlLhREdp/w7GjUroIZSMZ1lJiCtU3AJ8Vb+
lge/ZU5/nSkD0rPSrSp/nnrBhhpNgcMmaTPxr+RHK0bbK0JXeuHvFgmJjtBo8xSY
jdet6IxV2eR0J32yAlmsRSJhmzpsQvD+n60l5qTwbT/DgBMe+dXnHc+OcdDZgQdH
+or0d8lS1ZEGZj/NBPA+kr7vimanyybqIT/WHhvuS5KrCi4rKleKcHG/oelk8chT
+QfHMLfTL/aTLt9Tupb7vazZIdjF65RmWldvLD8bg3yamb7Yblv36XMnvB5yy8Tq
BOPWRGIFr14P0/6RRVh5hAdtV4vHp/jvYQPOUF8CiholcgH6lg==
=l4pC
-----END PGP PRIVATE KEY BLOCK-----

Process: 2432

...

Text:

-----BEGIN PGP PRIVATE KEY BLOCK-----

Version: GnuPG/MacGPG2 v2

lQO+BFV4TzUBCAD4CAngutPnU0fcvIxVSdKM8l/tGHSWlOg9bltmH+CRh197t+0z
W4nu3nefyFjXKRWunH90mRSSm/7lDeKCnQw9EuNRVAiwh1I1bigcZwkLZCKoX3xn
HbD6WLMQ2EAaCIyHyZwvbqo9lsQRCcsyyUMJnLttTFGaWjw3omV0CQ4ZHthcpsDd
ME+Bb+i0D9jJc+aTbflo7y/IPM4yXSwwlyG4x1XBL4TmDdnLQxNQPEPGiy+/kgIpm
58EGsyuzB1UNb9UWfdLpHgemKVzrpGGVRUM7aawPzlnjRFKEjRa4Ap9j8L+874nU
jtkiSpwPZskNuSa+0Fwk9Gjmbey8sug23WPHABEBAAH+AwMCEN2AD8Ykwd7QXn1E
TUAfbyJkNbIdPYoZXXnqAwycQ3J0rC2hDgVrbzm0JGDlLtxSZ6ZUBjXRzWdwLONo
lhaKCaS6zp73Wm7E9a+J088WT+3vcZb0t8Hc//WPXE1mz0qEwkBTnfpmUOPsNVav
Uarf7Vjp9VRUZ5X2PmQ67WhxnZnmTKOv6wGW97F2dAg+T5D35GPkklLcguIeBlNv
vx+ZuGFOP4T/KZiDtaBrriyGX7DPI/oQzK8s2Kd+jcf4TTaLWnAHCilMYIVTs+MD
Ea9eyaXFHUna1jPyzcDgnJEghAvLg/d/OaCBN2AuW8Lh8tVajdyfEd5qjlcpjvLV
JkMdiua6IFei5mSmufADFFJx5vaGZBNJ8/9j6OoOUp6Vf4BOoXvPw25oB7ROCgqP
w0xjYZ9gX7qZ0j/20qHAFYQ7pNB1C7/GBT3iIrGQ1litCIxM3AizcAthjPnkbaFy
dqCKXLWg6B6cg6ll7buqW3QUtBvrANa1igONjDok/CvdN39yn+vVztZPFjkJ252f
i1JLRmDSZG0yJ61ldOpPfSmJBVFysulKKNT1nwTLZrMdY7plUoPo0rgK+BD9L6EJ
mlDmikQBS9Ncyk6mvu9I928tFCCOHd/+0eIzjNO8Rf0Utf7PIOTTUODfQce95N7U
Sok50NbtEwzw92P2sIEwwRR9GQyM9wJIoZE2sqaN4gb7n2qplBqhUG+ZGBYQnobB
H2n+whibMc/QGuAon+6lc9Kv9WjyN3YUqceaj17ZzKQ+Su+TCGgI/bvE3stFJhIJ
maYoTJBdW+EJffAjvLZm3Ax4lkoIk7ZKbs2VUsp4cyNswmcBwpDVZJjZi48hTZ3
AJdJLHm8VY0uAgUFj6vvGvOCvmvnwggjLB0inzoLJjwxjoa0nHaZefCNTro7Cumz
gLQoQWN0dXJpYW5zIDxicnlhbnRoZXNjaWVuY2VndXlAZ21haWwuY29tPokBPgQT
AQIAKAUCVXhPNQIbAwUJB4YfgAYLCQgHAwIGFQgCCQoLBbYCAwECHgECF4AACgkQ
OUxGFasYh56PHAgAx1C5eRtpQVVfRR7nekjekXW8xQk5zavgarVgGmial8at3n9K
xPPMJclFNCIreA6rry3NUhR3in0U/TV0j0+5NQDhprI5OeUg740/xSce72pJBYRl
TIJZM7zyb9CxmGoD0E36FSD0YUQaCD+UvH98nGAK+dJ5E3WRGN4gdp84gwATJeVH
JN+jmL0tOXXTOPNEZQ/V3H4pYnwIG+CBfzruRNv3eSqRlauj5eheTBMkAXtDEaCY
XQBJdbA/mF+6RrZPemyBjxQ06fFaJb0XeTH4R5Rq30ghIRLCuBqbMPBQLVGxsEY7

The notepad command provides two critical pieces of information, both of which are required to finish Round 6. We have a complete Pretty Good Privacy (PGP) key along with an open/closed list. For now, stick these in your notes. We will return to them in Round 6.

5) Let's review the **cmdscan** module, which searches the memory of **csrss.exe** for commands entered into the console:

volatility -f Round2.mem --profile=WinXPSP2x86 cmdscan

```
Volatility Foundation Volatility Framework 2.4
*****
CommandProcess: csrcss.exe Pid: 616
CommandHistory: 0x12386f8 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 5 LastAdded: 4 LastDisplayed: 4
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x758
Cmd #0 @ 0x12447f0: SUPER_DUPER_SECRET1.txt + SUPER_DUPER_SECRET2.txt >
The_Key
Cmd #1 @ 0x12448f0: I have hidden my research
Cmd #2 @ 0x1244930: I hope the aliens don't figure out I am on to them
Cmd #3 @ 0x12449a0: I am sure they implanted something in me to monitor
me
Cmd #4 @ 0x1244a18: if you are human, good luck the world is counting
on you
```

See the “SUPER_DUPER_SECRET1.txt + SUPER_DUPER_SECRET2.txt > The_Key” bit? We took that to be a hint to combine the two portions of the obvious PGP key that we found in the **notepad** command above to form a proper private key. We will need this key in Round 6.

**** Round 2 Answer ****

The NTFS file system stores information pertaining to files, directories, and metadata in a file called the Master File Table (MFT). Files smaller than 500 bytes are stored directly in the MFT rather than elsewhere on the disk. This data is referred to as “resident data”. See here: <https://whereismydata.wordpress.com/2009/03/31/forensics-resident-data/>.

6) We can parse the MFT and dump associated data using the **mftparser** command. Make sure to create a new directory in your working directory called **mft** first:

mkdir mft

volatility -f Round2.mem --profile=WinXPSP2x86 mftparser --dump-dir='./mft/'

7) We knew that we are looking for Dimitri's research file, so we **grep'd** the MFT dump

for the string “research”:

cd mft

grep -ir 'research' . (make sure to run this from the **mft** directory!)

```
./file.0x1b618400.data0.dmp:This is my research.  
Binary file ./file.0x6ac4400.data0.dmp matches  
Binary file ./file.0x7542800.data0.dmp matches  
./file.0x7689000.data0.dmp:This is my research.
```

8) Hey! Looks like we might have found our boy’s research. Let’s review the contents of the first identified file:

cat file.0x1b618400.data0.dmp

```
This is my research.  
I believe that I have a newly implanted chip in my head.  
The doctors scanned me and said there was nothing there.  
I think they're lying.
```

Per the extracted document, we’ve found our research file. Kind of odd, since the file doesn’t contain much data.


9) Let’s go ahead and grab the MD5 sum for this bad boy:

md5sum file.0x1b618400.data0.dmp

```
19bebeab4457def688c9520b28464157  file.0x1b618400.data0.dmp
```

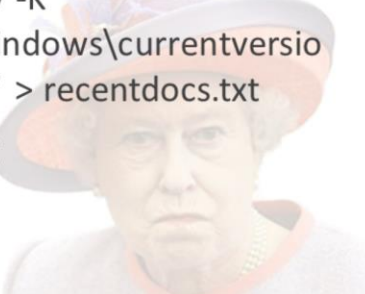
That’s it! We now have the answer to Round 2:

19bebeab4457def688c9520b28464157



Round 2 Alt. Method

- Run **hivelist**
 - Queen_Elizabeth\NTUSER.DAT
 - Virtual Address: 0xe25a6b60
- Run **printkey**
 - o 0xe25a6b60 printkey -K
"software\microsoft\windows\currentversion\explorer\recentdocs" > recentdocs.txt
- Check recentdocs.txt
 - **Line 204**



Round 2 Alternate Methodology:

We also found the answer to Round 2 Most Recently Used (MRU) lists from the image. To learn more about MRUs, see these resources:

<http://www.forensicmag.com/articles/2012/04/windows-7-registry-forensics-part-4>

<https://digital-forensics.sans.org/summit-archives/2009/8-harlan-carvey-registry-analysis.pdf>

In particular, the

HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDocs registry key maintains a list of commands entered into the Run dialogue box. Let's review it. First, we'll want to find the current user's NTUSER.DAT file:

volatility -f Round2.mem --profile=WinXPSP2x86 hivelist

```
Volatility Foundation Volatility Framework 2.4
Virtual    Physical    Name
-----
...
0xe25a6b60 0x11d7ab60 \Device\HarddiskVolume1\Documents and
Settings\Queen_Elizabeth\NTUSER.DAT
```

Now that we have the virtual address for the Queen_Elizabeth user's NTUSER.DAT file,

we can run the following to redirect the target MRU data to a new file called **recentdocs.txt**:

```
volatility -f Round2.mem --profile=WinXPSP2x86 -o 0xe25a6b60 printkey -K  
"software\microsoft\windows\currentversion\explorer\recentdocs" > recentdocs.txt
```

Let's review the contents of this new file:

```
less -N recentdocs.txt
```


(The -N argument enables line numbering.)

Take a look at line 204, which can be done by typing: 204g (type the number "204", then press the "g" key for "go to line" in **less**). You'll see the following:

```
204 REG_BINARY      22          : (S)
205 0x00000000 31 00 39 00 62 00 65 00 62 00 65 00 61 00 62 00 1.9.b.e.b.e.a.b.
206 0x00000010 34 00 34 00 35 00 37 00 64 00 65 00 66 00 36 00 4.4.5.7.d.e.f.6.
207 0x00000020 38 00 38 00 63 00 39 00 35 00 32 00 30 00 62 00 8.8.c.9.5.2.0.b.
208 0x00000030 32 00 38 00 34 00 36 00 34 00 31 00 35 00 37 00 2.8.4.6.4.1.5.7.
209 0x00000040 2e 00 74 00 78 00 74 00 00 00 94 00 32 00 00 00 ..t.x.t.....2...
210 0x00000050 00 00 00 00 00 00 00 00 00 00 31 39 62 65 62 65 61 62 .....19bebeab
211 0x00000060 34 34 35 37 64 65 66 36 38 38 63 39 35 32 30 62 4457def688c9520b
212 0x00000070 32 38 34 36 34 31 35 37 2e 6c 6e 6b 00 00 60 00 28464157.lnk..`.
213 0x00000080 03 00 04 00 ef be 00 00 00 00 00 00 00 00 14 00 .....
214 0x00000090 00 00 31 00 39 00 62 00 65 00 62 00 65 00 61 00 ..1.9.b.e.b.e.a.
215 0x000000a0 62 00 34 00 34 00 35 00 37 00 64 00 65 00 66 00 b.4.4.5.7.d.e.f.
216 0x000000b0 36 00 38 00 38 00 63 00 39 00 35 00 32 00 30 00 6.8.8.c.9.5.2.0.
217 0x000000c0 62 00 32 00 38 00 34 00 36 00 34 00 31 00 35 00 b.2.8.4.6.4.1.5.
218 0x000000d0 37 00 2e 00 6c 00 6e 00 6b 00 00 00 34 00 00 00 7...l.n.k...4...
```


Check that out! We are not sure if this was supposed to be included in the memory image. However, we can clearly see that an .lnk file was created and named with the proper MD5 sum for this round. Keep in mind that this entry *does not* indicate that this is the correct research file, but it could serve as a hint.

On to Round 3!



Round 3: “Blank” Files

- Begin w/Protocol Hierarchy
- Review Yahoo Messenger Traffic
- Filter on Traffic
 - **ymsg**
- Frame 16899 = File Transfer Begins
 - File: **blank.mp3**
 - Find the MP3!



Round 3 Methodology:

Q: What word was used to describe the Earth?

When checking the Protocol Hierarchy, we noticed Yahoo Messenger Traffic. Upon filtering on this traffic, we noticed a file transfer. Let's follow this process now.


1) Filter the Yahoo Messenger traffic in Wireshark. We can simply use the filter **ymsg**. Or, if you filter from the Protocol Hierarchy dialogue, you will get the following filter (both results in 15 displayed frames, so we're good):

ymsg and tcp and ip and eth and frame

Once you've filtered this traffic, you'll notice a File Transfer being setup starting in frame 16889.


2) Click frame 16889 to highlight the sucker, then expand the “Yahoo YMSG Messenger Protocol (Y7 File Transfer)” protocol in the Packet Details pane (middle pane). I recommend simply right-clicking on this protocol and then selecting “Expand Subtrees”.

Under the “Content” subkey, take a look at **Key 27**, which has a value of “**blank.mp3**”. Yup! An MP3 file is being transferred. Let's find that bad boy.



Round 3 cont.

- Many Ways to Identify
 - We Used the **MP3 File Signature**
 - Hex: 49 44 33
 - ASCII: ID3
- **frame contains 49:44:33**
- First Result: Frame **17003**




Round 3 Methodology cont.:

We can identify the data blob belonging to the Y7 protocol file transfer multiple ways. One of the ways that we used involved searching the packet capture for the file signature of an MP3 file. For more information on file signatures, see here: http://www.garykessler.net/library/file_sigs.html.

- 3) Filter the traffic containing the MP3 file signature:
frame contains 49:44:33


This filters out any frames that contain this hexadecimal string, which happens to be the file signature for an MP3 file. The filter results in six (6) displayed frames.

- 4) Click the first frame in the results, 17003, to highlight the bad boy.



Round 3 cont.

- Check Packet Details Pane
 - Expand “Transmission Control Protocol”
 - *Clear the Wireshark Filter*
 - Double-click:
 - **Reassembled PDU in frame: 19521**
 - Click “HTML Form URL Encoded... blah...”
 - Right-Click + Export Data
- **BOOM! We has an MP3!**



Round 3 Methodology cont.:

5) In the Packet Details pane, expand “Transmission Control Protocol”, which will be the fourth protocol listed.

6) Scroll to the bottom of this section. Notice the hyperlink that reads “Reassembled PDU in frame: 19521”. We want to click this link, but we must first remove the Wireshark filter. If you do not remove the filter, you will receive the error “*Packet number 19521 isn’t displayed.*” To remove the filter, simply click the Clear button to the right of the filter window.

7) Now that we have cleared the display filter, **double-click the “Reassembled PDU in frame: 19521” link**, which will take you to frame 19521. *Alternatively, simply press Control+G to open the “Go To Packet” window, enter “19521”, and click “Jump To” (this method also requires you to clear the filter).*


8) With frame 19521 selected, scroll down to the bottom of the Packet Details pane. At the bottom, find and click the item labeled “**HTML Form URL Encoded: application/x-www-form-urlencoded**”, which is just below “Hypertext Transfer Protocol”.

9) Look at the bottom pane in Wireshark, which is the Packet Bytes pane. You will notice that the content highlighted begins with the ASCII text “**ID3**”. This is our MP3

file! To export the file, right-click the “HTML Form URL Encoded: application/x-www-form-urlencoded” item and select “Export Selected Packet Bytes...”


10) In the Export Selected Packet Bytes window, label the data you are exporting “**blank.mp3**” and click the “Save” button.

BOOM! We have our MP3!



Round 3 cont.

- blank.mp3
 - Aliens Sounds and Stuff!
- Run **foremost**
 - File Carving Tool
- Identify Two Extracted MP4 Files
- Audio is Reversed



Round 3 Methodology cont.:

If you try to play the **blank.mp3** file in an audio player, you will hear ~32 seconds of what *might* supposed to be an alien spaceship sound. Maybe some sort of alien communication sound. Dunno. What we do know is that ***we need to go deeper***. In these competitions, simple carrier files (such as an MP3 file) are not always what they seem.

11) In a Terminal (**gnome-terminal** if using Kali w/Gnome), run **foremost** against the MP3 file. You can do so as such:

foremost blank.mp3

This command will create a new directory in your working directory named **output**. Inside the **output** directory, you'll find a text file called **audit.txt**, the contents of which will look similar to this:

```
Foremost version 1.5.7 by Jesse Kornblum, Kris Kendall, and Nick Mikus
Audit File
```

```
Foremost started at Sat Apr 23 18:41:10 2016
Invocation: foremost blank.mp3
Output directory: /mnt/hgfs/Public/NFPC2015/Round3/output
Configuration file: /etc/foremost.conf
```

```
-----  
File: blank.mp3  
Start: Sat Apr 23 18:41:10 2016  
Length: 1 MB (1167071 bytes)
```

Num	Name (bs=512)	Size	File Offset	Comment
0:	00000465.mp4	305 KB	238169	
1:	00001182.mp4	307 KB	605224	

```
Finish: Sat Apr 23 18:41:10 2016
```

```
2 FILES EXTRACTED
```

```
mp4:= 2  
-----
```

```
Foremost finished at Sat Apr 23 18:41:10 2016
```


Notice that **foremost** found two different MP4 files within the MP3 file. Oooooohhhh. Let's take a gander.

12) Open the **mp4** directory and review the two MP4 files. Note that the default media player in Kali 2.0, **Videos**, will not play these files. Kali comes with VLC installed, but VLC does not like to run as root. Just find some way to play the videos! Copy them out of your VM, whatever.

(If Kali 2.0 doesn't actually come with vlc, just run **sudo apt-get install vlc** to install the bugger. I don't recall and I'm too lazy to look it up, but I'm not lazy enough to type this message. Go figure.)


Tip: You can run the following to enable VLC to run under the root account on Kali:
sudo sed -i 's/geteuid/getppid/' /usr/bin/vlc

Whatever you find to play the two files, you'll notice that they don't sound quite right. Why? ***Because the audio is reversed!*** We simply need to play these files backwards... but how?



Round 3 cont.

- Audacity to the Rescue!
- Open **00000465.mp4** in Audacity
 - “Effect” Menu → “Reverse”
 - Play the Audio... Meh
- Reverse **00001182.mp4** Audio & Play
- ANSWER:
Precious



Round 3 Methodology cont.:

13) **Audacity** to the rescue! If you do not have **audacity** installed in Kali yet, you can run **sudo apt-get install audacity** to install the bad boy. Once installed, open the first MP4 file, **00000465.mp4**, in **audacity**.

NOTE: If you are in Kali 2.0 (possibly 1.x also), you *might* need to define the proper playback device to hear audio files. In Audacity, select the “Edit” menu and choose “Preferences...” On the default tab, which is “Device”, change the Playback Device to “Ensoniq AudioPCI: ES1371 DAC1 (hw:0,1)”. Save your settings by clicking the “OK” button at the bottom right of the Preferences window. *The name of your audio device may differ*, but this is the name of my audio device when running Audacity in Kali Linux 2.0 in VMWare Fusion.

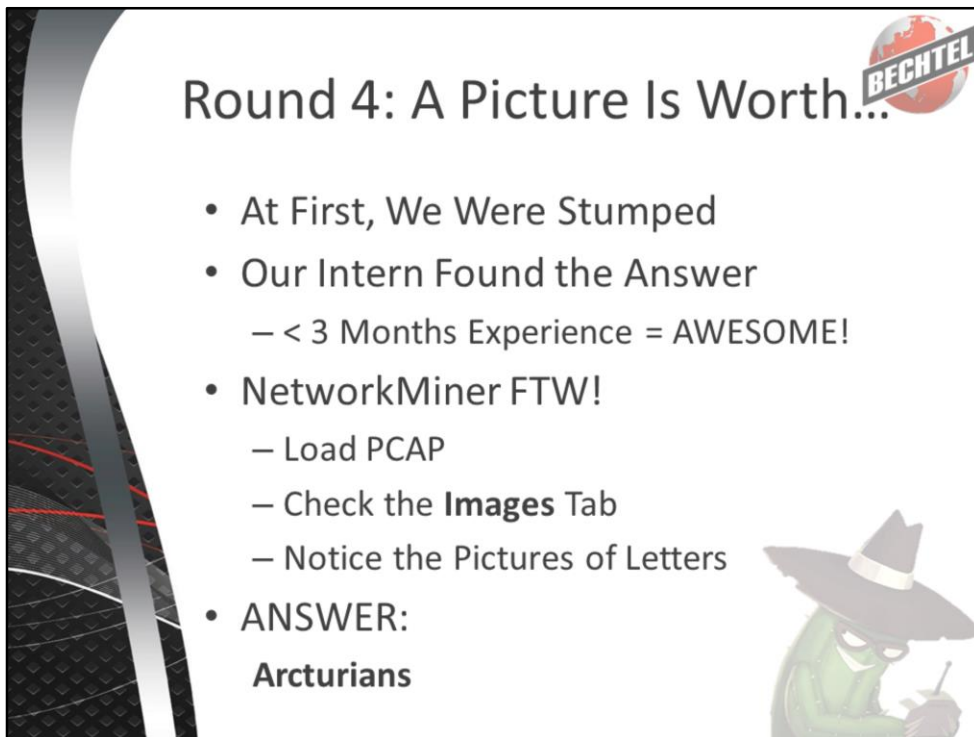
14) To reverse the audio, select the “**Effect**” menu and choose “**Reverse**”. Once done, play the audio once again. Hey!! There we go! Did you hear that? “*They are coming for you yadda yadda blah blah*”

15) Open the second MP4 file, **00001182.mp4**, in Audacity. Once again, reverse the audio by choosing Effect → Reverser. Play this reversed audio.

Did you hear it?! DID YOU?! “We are coming for your **precious** earth.”

Yup! That's it! The answer to Round 3 is:
precious

Let's move on to Round 4.



The slide features a dark, textured background on the left side with a white curved line. In the top right corner, there is a red circular logo with the word 'BECHTEL' in white. The main title 'Round 4: A Picture Is Worth...' is in a large, bold, black font. Below the title is a bulleted list of points. At the bottom right, there is a cartoon illustration of a green alien wearing a black hat and a mask, holding a small object.

Round 4: A Picture Is Worth...

- At First, We Were Stumped
- Our Intern Found the Answer
 - < 3 Months Experience = AWESOME!
- NetworkMiner FTW!
 - Load PCAP
 - Check the **Images** Tab
 - Notice the Pictures of Letters
- **ANSWER:**
Arcturians

Round 4 Methodology:

Q: What alien race contacted him?

We were quite lost in this round. Eventually, our intern said, "Do you guys think these letters have anything to do with the answer?" Our intern, who had < 3 months experience in security at the time, was reviewing the Round4.pcap file in NetworkMiner. She was reviewing the content of the "Images" tab, which... well, take a look:

1) Launch **NetworkMiner** in either Windows or in Linux via **mono**.

2) Load the **Round4.pcap** network capture in **NetworkMiner**.

NOTE: NetworkMiner takes a while to parse the **Round3.pcap** capture. Parsing the file under NetworkMiner 2.0 via **mono** on Kali 2.0 took just under 15 minutes for me. You may have better luck running NetworkMiner natively under Windows. For info on running NetworkMiner in Kali, see here:

<http://www.netresec.com/?page=Blog&month=2011-12&post=No-more-Wine---NetworkMiner-in-Linux-with-Mono>.

3) After parsing completes, check the "Images" tab.

4) Notice the pictures of single letters? Put them in order and dedupe them, and they will form the answer.

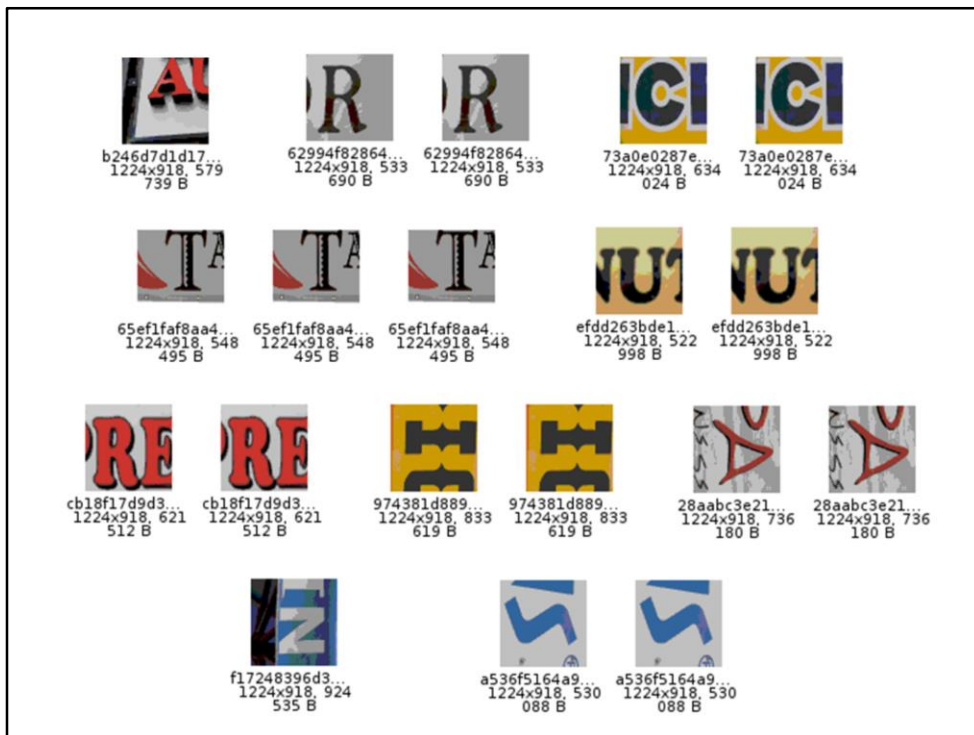
Awwwww snap! The answer to Round 4 is:

Arcturians

Speaking of which, do you remember the clipboard contents we found in Round 2? If not, here's the string we found in Step 3 from Round 2:

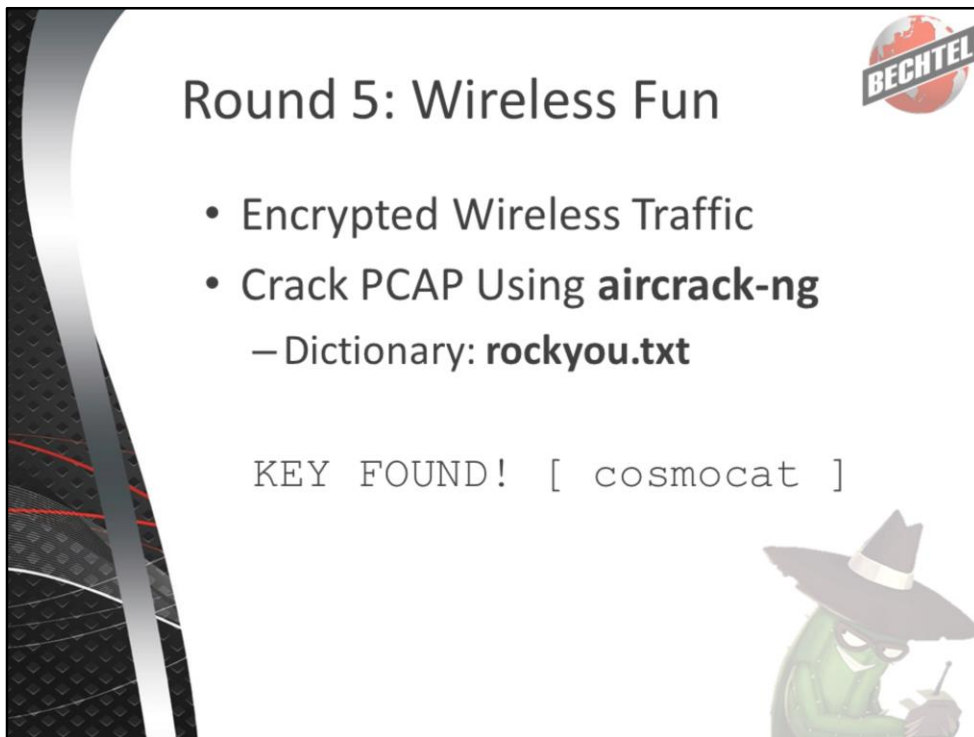
acturians first contact with stegosauruses

While these words for the alien race are similar, one of them includes an extra letter. The actual answer includes an "r" after the first letter ("ARCTurians"). The clipboard version includes basically the same word, but is missing the "r" ("ACturians"). Just a heads-up, as this is still a valid hint.



The various letters from NetworkMiner's Images tab. De-dupe to get the answer.

ARCTURIANS



Round 5 Methodology:

Q: What technological instrument are they going to use to destroy the Earth?

When we first load the **Round5.pcap** file, we see that the traffic contained within includes encrypted wireless traffic.

1) Use **aircrack-ng** to brute force the WPA password and decrypt the packet capture. To perform our brute force cracking, we will be using a wordlist provided with Kali called **rockyou.txt**. In some versions of Kali Linux, you will need to unzip/untar/unwhatever the rockyou file first. Please check **/usr/share/wordlists/** to see if you have rockyou.txt or a zipped/tarred/somehow compacted version. If you do, unzip/untar/whatever.

2) Once you have ensured that rockyou.txt is in plaintext, proceed by running the following:

aircrack-ng -w /usr/share/wordlists/rockyou.txt Round5.pcap

Let this command run for as long as it takes to find the wireless passphrase (shouldn't take longer than 5-10 minutes). Once run, this will provide output similar to:

```
Opening Round5.pcap
Read 75627 packets.
```


#	BSSID	ESSID	Encryption
1	04:A1:51:AD:52:5A	TheMothership	WPA (1 handshake)

Choosing first network as target.

Opening Round5.pcap

Reading packets, please wait...

Aircrack-ng 1.2 rc3

[00:00:56] 88788 keys tested (1695.58 k/s)

KEY FOUND! [**cosmocat**]

```


Master Key      : 2B 26 CB 25 64 DB BC 83 B8 95 87 6A B0 32 0E A7
                  AC B0 8E 33 25 28 D6 07 F2 22 CD D8 DF 07 1B 3F

Transient Key   : 0E 27 69 29 6D 9B F1 01 FD 58 9A 17 8F 27 3A 94
                  30 6A 20 BD 5F 76 C0 0B DD 20 D2 71 03 B9 3A 38
                  40 59 E1 36 79 83 13 FF 3D 42 DA 52 5F C8 C4 14
                  B2 09 F2 A4 01 5C 91 A1 E5 A6 5E 9E 37 2E F5 E5

EAPOL HMAC     : 3C 1A 97 A8 7A B9 80 6A E0 79 9A D3 82 37 74 32


```

Hey! We have a passphrase. Notice that **aircrack-ng** identified the encryption method as Wi-Fi Protected Access (WPA). That's useful. Time to decrypt the traffic.



Round 5 cont.

- Decrypt PCAP in Wireshark
 - Edit → Preferences
 - IEEE 802.11
 - Check “**Enable Decryption**”
 - Add **wpa-pwd** Decryption Key



Round 5 Methodology cont.:


3) Decrypt the PCAP in Wireshark. With Roud5.pcap open in Wireshark, select the “Edit” menu and choose “Preferences...” Expand the “Protocols” section in the left sidebar of the Preferences window. Scroll down to and select “IEEE 802.11”.

4) **MAKE SURE that the checkbox to the right of “Enable decryption” is CHECKED.** Otherwise, no dice.

5) Input the WPA key for decryption. To do so, click the “Edit...” button next to “Decryption Keys”. Click the “New” button in the left sidebar of the “WEP and WPA Decryption Keys” dialogue window. Click the “Key type” drop-down menu and select “wpa-pwd”. In the “Key” field, enter our WPA password: **cosmocat**, and then click the “OK” button.


To save your changes, click the “OK” button at the bottom right of the decryption keys dialogue. Finally, close the Preferences window by clicking the “OK” button at the bottom right of this window also.

We now have a decrypted packet. We can verify this by going to the Protocol Hierarchy, at which point we will see all the decrypted traffic. This decrypted traffic happens to include IRC traffic, so let’s get nosy.



Round 5 cont.

- *Oooh! A Piece of [IRC] Candy!*
 - Filter IRC Traffic: **irc**
- Follow TCP Flow
- Check the PRIVMSGs
 - Answer is Encoded Using Hashes
 - Google Them Bad Boys!
- **ANSWER:**
Mega Death Ray 5102



Round 5 Methodology cont.:

6) Filter on the IRC traffic via the Protocol Hierarchy or by simply using the filter:
irc

Oh look, some PRIVMSGs!

7) Choose any of the frames shown in the **irc** filter and right-click to select **Follow TCP Flow**. We don't need any advanced extraction methods, as this conversation is short:

```
PRIVMSG #Mothership :These humans are so dumb they will never figure
out all the clues.
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :I know as time ticks
down to the earth's destruction I grow happier.
PRIVMSG #Mothership :You seem particularly happy about it. Any reason
why?
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :Of course we just
finished a new weapon of massive planetary distruction.
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :Destruction*
PRIVMSG #Mothership :I didn't think that was finished. What's it called
again.
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :Oh yes it's finished
and we are going to try it out on earth first.
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership
```

```

:fde5020a5a97322bf5a7aee8174abbd8
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership
:d8c7d877ba6139c4872450e3847613a50c79f4e2
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership
:9406e3c325bfc9873426e5eda4ba6e18
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership
:0519a3b8d19f6d01501da1960c19385b5e938f86
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :I think it sounds
powerful.
PRIVMSG #Mothership :Yeah I would agree.
PRIVMSG #Mothership :Have fun working on it. I wish I was assigned
there.
:vork!~bryan@162.219.72.250 PRIVMSG #Mothership :Well it is pretty
awesome. But no rest for us. Talk to you later.
PING :morgan.freenode.net
PONG :morgan.freenode.net
PING :TIMEOUTCHECK
:morgan.freenode.net PONG morgan.freenode.net :TIMEOUTCHECK

```


In this conversation, we see that the weapon of mass destruction's (WMD's) names has been encoded. If you guessed that these are hashes, you'd be correct!

7) Hashes are one-way algorithms, so we cannot outright "reverse" these bad boys. However, you can ***simply Google the various hashes to find the name of the WMD.*** Site such as **md5cracker.org** have lookup and rainbow tables available that will provide the answers that we need.

fde5020a5a97322bf5a7aee8174abbd8	= Mega
d8c7d877ba6139c4872450e3847613a50c79f4e2	= Death
9406e3c325bfc9873426e5eda4ba6e18	= Ray
0519a3b8d19f6d01501da1960c19385b5e938f86	= 5102


Thus, we have our answer for Round 5:
Mega Death Ray 5102

OMG GINA! Time for Round 6!



Round 6: TC Knockin'

- Review Conversations
 - Sort Descending by Bytes
 - Giga-Byt_e4:8b:d3 → Compalln_31:be:25
 - eth.addr==50:e5:49:e4:8b:d3 && eth.addr==f0:76:1c:31:be:25
 - Follow TCP Stream: tcp.stream eq 949
- Save As → “round6.bin”
 - TrueCrypt Volume!
 - Password = ???



Round 6 Methodology:

Q: What time must the message be sent? Remember, time is running out! We need to know the exact time down to the very second!

This round was ridiculously frustrating for us. We found what we believed to be a TrueCrypt (TC) volume within a few minutes, but we were not able to “put 2 and 2 together” to mount the volume. Funny enough, the method we used to identify the TC volume also identified the rest of the information that we needed... we just didn’t know it until LMG released the official hint for this round.

1) Upon loading the Round6.pcap in Wireshark, check the captured conversations by choosing the **“Statistics” menu and selecting “Conversations”**. Once the Conversations window is open, sort the results by **Bytes in descending order**, which will most likely be the fourth (4th) column from the left.

Upon sorting, you will notice that the second result shows a conversation between **Giga-Byt_e4:8b:d3 -> Compalln_31:be:25**. This conversation consists of 8,941 packets and denotes a transfer size of 8,983,336 bytes.

2) Filter on this traffic by right-clicking and selecting **“Apply as Filter → Selected → A ↔ B”** (the first option). You can alternatively use the filter that this will generate:

eth.addr==50:e5:49:e4:8b:d3 && eth.addr==f0:76:1c:31:be:25

Once this filter has been applied, we see a Secure Shell (SSH) session begin in frame 81589. *We also found quite a few SYN packets both before and after the SSH session, but we did not know what they were all about (dangit).* Following the last part of the SSHv2 handshake in frame 83093, you will notice traffic from 192.168.1.6:50953 to 192.168.1.5:1234 begin in frame 83094. The thing that we noticed right off the bat was the use of destination port 1234.

3) Right-click frame 83094 and select “Follow TCP Stream”. This will result in the following filter:

tcp.stream eq 949

If you attempt to sort the flow, you’ll see that all traffic is egressing from 192.168.1.6. This hints toward this being a file transfer, which is what it is! Let’s save this bad boy.

4) Click the “Save As” button and label the file: **round6.tc**

The file we have just saved is a TrueCrypt volume. When we ran a Shannon entropy calculation on the file, we were convinced (TC volumes usually have high entropy). For that matter, the file size is 8,388,608, which when divided by 1,024 yields 8,192, exactly 8MB.

If you’re interested, we ran across a Python-based Shannon entropy calculation script here:

<http://code.activestate.com/recipes/577476-shannon-entropy-calculation/#c3>

python file_entropy.py round6.tc


```
File size in bytes:
8388608
```

```
Shannon entropy (min bits per byte-character):
7.99997779995
```

```
Min possible file size assuming max theoretical compression efficiency:
67108677.7725 in bits
8388584.72156 in bytes
```

Check that out! This file has a min bits per byte-character value of nearly eight (8), which means the sucker has insanely high entropy.


To mount the volume, we need the password. This is where we became lost during the competition.



Round 6 cont.

- Waited for LMG's Round 6 Hint
 - “Knock, knock. Who’s there? A TrueCrypt password!”
- **Port Knocking** in Use!
- Re-Filter: See Step 2 from This Round
 - Also Filter Out the ARP Protocol

```
eth.addr==50:e5:49:e4:8b:d3 &&  
eth.addr==f0:76:1c:31:be:25 &&  
!arp
```



Round 6 Methodology cont.:

We did not know how to proceed until LMG released the official hint for Round 6:
“Knock, knock. Who’s there? A TrueCrypt password!”

As soon as we saw the hint, we knew that the SYN packets we found in Step 2 involved port knocking. Port knocking involves a service opening and/or closing only after certain circumstances are met, such as a secret sequence of ports being hit. More info here: <http://www.portknocking.org/>. For numerous port knocking tools, see here: <http://www.portknocking.org/view/implementations>.

In this case, the SSH service was opened and closed with a series of “knocks” to specific ports. Let’s review.


5) Re-filter the capture using the filter that we used in Step 2 from this round:
eth.addr==50:e5:49:e4:8b:d3 && eth.addr==f0:76:1c:31:be:25

Once filtered, scroll to the top in the Packet List pane (top pane). Notice how the 192.168.1.5 host is sending numerous SYN packets to the 192.168.1.6 host? That’s the port knocking in progress. The port knocking begins in frame 6892 and completes in frame 80722. Notice that the next displayed frame is 81587, which has the initiating host hit port 22. This indicates that the port knock sequence was correct and that port

22 is now open.


The SSH session is used to setup the file transfer, but due to the protocol's inherent encryption, we cannot see the actual setup. However, we know that the TrueCrypt volume is transferred from 192.168.1.6:50953 to 192.168.1.6:1234, as we found this in Step 3 of this round.

At this point, we know that the password for the transferred TrueCrypt volume is related to the port knocking sequence, but how?



Round 6 cont.

- Head Back to Step X of Round 2
 - Results of **notepad** Volatility Module
- Open/Closed = SYN Packet #s
 - Pertinent Value = Destination Port #s
- Example:
 - Open: 3,9 = Dest Ports from 3rd/9th SYN
 - SYN #3 (frame 9651) = Dest Port 9488
 - SYN #9 (frame 24099) = Dest Port 9488
 - Password = 94889488...




Round 6 Methodology cont.:

The answer lies in data that we obtained back in Round 2. Remember the results of the **notepad** module *from Step 4 of Round 2*? If not, here's a snippet of the data:


```
Process: 2320
...
Text:
Open: 3,9
Closed: 4
Open: 25,1,22
Closed: 9,12,19,21
Open: 11,18
Closed: 2
Open: 10,16
Closed: 13
```

The trick here is identifying that the terms “Open” and “Closed” refer to the port knocks required to open and close the SSH service. The crucial part of the port knocks involved the destination ports that were hit. Thus, the TrueCrypt password is a long decimal-based string (a number if you will) that is a combination of the destination ports for the SYN packets identified.



Round 6 cont.

- Continue the Decode Process
94889488423292072372006423840394882
920583871095838695030065838
- Mount the TC Volume
- Change Directory to Mountpoint
- We Now Have Two Files:
 - **btsnoop_hci.log**
 - **information.txt.gpg**



Round 6 Methodology cont.:

6) Use the key to decode the TrueCrypt password as such:

Open: 3,9	Frame 9651, 23099	= 94889488
Closed: 4	Frame 111825	= 423
Open: 25,1,22	Frame 61602, 6892, 55219	= 292072372006
Closed: 9,12,19,21	Frame 138862, 140189, 140915, 141753	= 423840394882920
Open: 11,18	Frame 32207, 45585	= 58387109
Closed: 2	Frame 92899	= 5838
Open: 10,16	Frame 29204, 42023	= 69503006
Closed: 13	Frame 140300	= 5838

Thus, the password is:

94889488423292072372006423840394882920583871095838695030065838

7) Find a tool to mount the TrueCrypt volume. **Unfortunately, TrueCrypt is no longer bundled with Kali as of v2.0.** If you have TrueCrypt installed in Kali already (not in the standard Kali repo!), go ahead and use it. If not, I recommend using **tcplay** to mount the volume. You'll have to install this bad boy. If you didn't already, go ahead and run:

sudo apt-get install tcplay

8) Mount the TC volume! To mount the volume using **tcplay**, do the following:

```
losetup /dev/loop0 ./round6.tc
tcplay -m round6.tc -d /dev/loop0
mkdir /root/tc
mount -o nodev,nosuid,uid=1000,gid=100 /dev/mapper/round6.tc /root/tc/
ls /root/tc
```

When prompted for the "Passphrase", enter:


94889488423292072372006423840394882920583871095838695030065838

- The **losetup** command is used to create a new loopback device. If this command fails, increment the **loop0** to **loop1** and try again.
- The **tcplay** command mounts the **round6.tc** volume to the loopback device we just created
- We then make a new directory called **tc** in the **/root** directory. NOTE: This will work fine if you are running Kali as root. If not, **sudo** the bad boy or make the directory in your current user's home directory (/home/whatever/).
- Finally, the **mount** command is used to mount the loopback device to a folder of our choosing, which is **/root/tc/**

9) List the contents of the new mount point using **ls** to find:


```
total 6663
-rwxr-xr-x 1 1000 users 6821056 Jul  2  2015 btsnoop_hci.log
-rwxr-xr-x 1 1000 users    560 Jul  2  2015 information.txt.gpg
```

Getting Warmer!



Round 6 cont.

- Re-Compile PGP Key
 - From **notepad** Volatility Results
 - Save As **round6.key**
- Decrypt **information.txt.gpg**
 - **gpg --import round6.key**
 - **gpg --decrypt information.txt.gpg**
- Ruh Roh! Passphrase Req.



Round 6 Methodology cont.:

We gravitated toward the **information.txt.gpg** file, as we recalled finding a Pretty Good Privacy (PGP) key in Round 2. Do you remember that? DO YOU?! YOU BETTER! Seriously though, take a look back at Step 4 from Round 2, in which we ran the **notepad** module in Volatility. Then:

10) Re-compile the PGP key that we found in Step 4 of Round 2. To do so, copy the contents from process 2432 above the contents from process 2376, as such:

```
-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: GnuPG/MacGPG2 v2
```

```
lQO+BFV4TzUBCAD4CAngutPnU0fcvIxVsdkM8l/tGHSWlOg9bltmH+CRh197t+0z
W4nu3nefyFjXKRWunH90mRSsm/7lDeKCnQw9EuNRVAiwh1I1bigcZwkLZCKoX3xn
HbD6WLMQ2EAaCIyHyZwvbqo9lsQRCcsyyUMJnLttTFGaWjw3omV0CQ4ZHthcpsDd
ME+Bb+i0D9jJc+aTbflo7y/IPM4yXSwlyG4x1XBL4TmDdnLQxNQPEPGiy+/kgIpm
58EGsyuzB1UNb9UWFdLpHgcmKVzrpGGVRUm7aawPzlnjRFKEjRa4Ap9j8L+874nU
jtkiSpwPZskNuSa+0Fwk9Gjmbey8sug23WPHABEBAAH+AwMCEN2AD8YkwD7QXn1E
TUAfbyJkNbidPYoZXXnqAwycQ3J0rC2hDgVrbzm0JGD1LtxSZ6ZUBjXRzWdwLONo
lhaKCaS6zp73Wm7E9a+J088WT+3vcZb0t8Hc//WPXE1mz0qEwkBTnfpmpUOPsNVav
Uarf7Vjp9VRUz5X2PmQ67WhxnZnmTKOv6wGW97F2dAg+T5D35GPkkfLcguIeBlnV
vx+ZuGFOP4T/KZiDtaBrriyyGX7DPI/oQzK8s2Kd+jcf4TTaLWnAHCilmYIVTs+MD
Ea9eyaXFHUna1jPyzcDgnJEghAvLg/d/OaCBN2AuW8Lh8tVajdyfEd5qjlcpcjvLV
```

```

JkMdiua6IFei5mSmufADffJx5vaGZBNJ8/9j6OoOUp6Vf4B0oXvPw25oB7ROCgqP
w0xjYZ9gX7qZ0j/20qHAFYQ7pNB1C7/GBT3iIrGQ1litCIxM3AizcAthjPnkbaFy
dqCKXLWg6B6c6g61l7buqW3QUtBvrANa1igONjD0k/CvdN39yn+vVZtZPFjkJ252f
i1JLRmDSZG0yJ611dOpPfSmJBVFysulKKNT1nwTLZrMdY7plUoPo0rgK+BD9L6EJ
mlDmikQBS9Ncyk6mvu9I928tFCCOHd/+0eIzjNO8Rf0Utf7PIOTTUOdFQce95N7U
Sok50NbtEwzw92P2sIEwRR9GQyM9wJioZE2sqaN4gb7n2qplBqhUG+ZGBYQnobB
H2n+whibMc/QGuAon+6lc9Kv9WjyN3YUqcea17ZzKQ+Su+TCGgL/bvE3stFJhIJ
maYoTJBdW+EJffAjvlZm3Ax4lkoIk7ZKBS2VUsp4cyNswmcBwpDVZJjZi48hTZ3
AJdJLHm8VY0uAgUFj6vvGvOCvmvnmwggjLB0inzoLJjwxjoa0nHaZefCNTRo7Cumz
gLQoQWN0dXJpYW5zIDxicnlhbnRoZXNjaWVuY2VndXlAZ21haWwuY29tPokBPgQT
AQIAKAUCVXhPNQIbAwUJB4YfgAYLCQgHAwIGFQgCCQoLBbYCAwECHgECF4AACgkQ
OUxGFasYh56PHAgAx1C5eRtpQVVfRR7nekjekXW8xQk5zavgarVgGmial8at3n9K
xPPMJclFNCIreA6rry3NUhR3in0U/TV0j0+5NQDhprI5OeUg740/xSce72pJBYRl
TIJZM7zyb9CxmG0D0E36FSD0YUQaCD+UvH98nGAK+dJ5E3WRGN4gdp84gwATJeVH
JN+jmL0tOXXTOPNEZQ/V3H4pYnwIG+CBfzruRNv3eSqrLauj5eheTBmkAXtDEaCY
XQBJdbA/mF+6RrZPemyBjxQ06fFaJb0XeTH4R5Rq30ghIRLCuBqbMPBQLVGxsEY7
8dagr1+7EiLkJER4JmBEy6owEZdPEQBt90S2QJ0DvgRVE81AQgAnuVdCSsKR0O1
0sdaF3lYLZeluBdvC9/VR6MJtO72HFyxDEJ3iD46GptJC8ePvK+fUD/lc+s3gQUP
flPQy7iv651KcdVNvUKBoNUQtU5b97me2Egj4R7YnWbBk024G8qFRk/4if0TUCiZ
aOZRUYtTa0IOksya4WupFzRQMq61pKUGz4XTlilLrN+c88AG9fQ4/+jvS/RRMEIZ
lkJDDsNomuuZZFqdtSORvsDv4eeZ918NB/e7hizWBzlCA/FlluxlXt86/RVcdI0P
C4N2N3P3frRrjAQvx06PIEw5gm6mQxjCXBDCVe74mzGQwFordcJ+rF9nV600ZOUl
yooPIucQcQARAQAB/gMDAhDdgA/GJMA+0B8Y4l0IBo3dlKMDQuNJCqUS5JjRKF+a
T/ij2aWKleX/7xeEe/28Oz7oYy/KyalRyP9NXRG8+YGHUVH/BMUKYbom3Mhis/Dr
aETbcaVYlyiko9eK1blBG3FOCO/aLFxQmRHT1gN9CF2JK2iUX86NwhU1Eq5OYptu
J9yw8190zP3qnKhvbwV0kkawK8a0wwofb6eAt9H3y9YM3YIsNRKRVRUeWzntp6dK
IHGnbVN0bU6olvUueTusGbrRBUATx6V2A6NJAR+wStQyapdFOBVGnu8Fqx3GwQW/
PXvXapjrFXvv1Rf+oSxboiOgoq4J3ErfD6GVda1ZjpVoaAPt+z+XHK8SJTa0VwWY
HZ0P5WlhM+oIix20uYEYnPOPsXqw5/laqoBDNF39RVt1bnHz9l13WpYa3IC7uJgT
o6lv5+z3Edj0z/rmG69gAS0oEaCv++lIoDIFI5LyRvPrf89R2i6f51dmuyCAsHto
qrwKSkza+xbKJaGSWQHKBi2qHbAjDvLlRekRaLzBFYyZHD+rmrzfJnioloBVtGTx
HmFEY9HXAMNWDi9xypYArXS10oswe8nUmvMn+gqYnjSwOadaOsTAT6sJUnxLV2/U
in4/tb1JIG0Rn0LMKOpj2BCnMee9vA14FRhk3PJ5qNUObX7n3R5f11PYc4q2sQzQ
BD5p+h7gNNd9VPp3zcLJhToVWgA8FDMff6EUabHJMokmdXI6I3hfp5XbJPeHyRZv
hGN4M7SuVMsmHaem86eVTp1V+mSYDemMxnNQ1uw36hQMPPwzbIoAbbUSNYhIb1Qm
mJ4fimJSAMPmAiKu2dY+xZOlWEnOYU2wHeN8cyHCElh+rVZmXYxdmLS5F5SRC7c
EOhwInqkJYDnn6Hpe8PL1+u/rB2GDyph1+Bnrj+JASUEGAECaa8FA1V4TzUCGwwF
CQeGH4AACgkQOUxGFasYh551yAf9EARlLhREDp/w7GjUroIZSMZ1lJicU3AJ8Vb+
lge/ZU5/nSkD0rPSrSp/nnrBhhpNgcMmaTPxr+RHK0bbK0JXeuHvFgmJjtBo8xSY
jdet6IxV2eR0J32yAlmsRSJhmzpsQvD+n60l5qTwbT/DgBMe+dXnHc+OcdDZgQdH
+or0d8lS1ZEGZj/NBPA+kr7vimanyybqIT/WHhvuS5KrCi4rKleKcHG/oelk8chT
+QfHMLFtL/aTLt9Tupb7vazZIdjF65RmWldvLD8bg3yamb7Yblv36XMnVB5yy8Tq
BOPWRGIFr14P0/6RRVh5hAdtV4vHp/jvYQPOUF8CiholcgH6lg==
=14pC
-----END PGP PRIVATE KEY BLOCK-----

```

- 11) Save the compiled PGP key to a new file (using any text editor) and label it: **round6.key**

12) Decrypt the **information.txt.gpg** file using Gnu Privacy Guard (GPG). To do so, you must first import the private key you re-compiled into the GPG keyring. *If you do not do this, you will receive the following error:*

```
gpg: encrypted with RSA key, ID 68B1386B
gpg: decryption failed: secret key not available
```

That's silly. We don't want that error. To avoid that error:

13) Import the private key you re-compiled into GPG using the following command:

gpg --import round6.key

```
gpg: key AB18879E: secret key imported
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key AB18879E: public key "Acturians
<bryanthesienceguy@gmail.com>" imported
gpg: Total number processed: 1
gpg:             imported: 1   (RSA: 1)
gpg:       secret keys read: 1
gpg:       secret keys imported: 1
```

The command results show that our key has been imported properly. Good! Now we can:


14) Begin the decryption process of the GPG-encrypted file as such:

gpg --decrypt information.txt.gpg

```
You need a passphrase to unlock the secret key for
user: "Acturians <bryanthesienceguy@gmail.com>"
2048-bit RSA key, ID 68B1386B, created 2015-06-10 (main key ID
AB18879E)
```


Enter passphrase:

The passphrase was found back in Round 1. I didn't want to burden everyone so early in Round 1 with retrieving the sucker, but let's find the passphrase now.



Round 6 cont.

- Go Back to Round1.pcap
- Filter Yahoo Messenger Messages
 - **ymsg.service == message**
- Find the Value for Key 17 in Content
- Copy All Four into **ymsg.txt**
- Now to Decode!



Round 6 Methodology cont.:

15) Re-Open the Round1.pcap file in Wireshark

16) Filter on the Yahoo Messenger traffic using the following filter:

ymsg


17) We then noticed messages being sent, so let's filter further using this bad boy:

ymsg.service == message

When applying this filter, we find eight (8) packets displayed, with a total of four (4) unique messages.

18) Copy the individual messages into a new text file. In our case, we labeled our text file "**ymsg.txt**".

To copy the messages, expand the "Yahoo YMSG Messenger Protocol (Message)" protocol in the Packet Details pane. Scroll down to **Content** → **Key 14**. You will find the actual messages in the Value field for this key. Right-click the Value field and select **Copy** → **Value**.




Round 6 cont.

- DECODE!!
- Substitution Cipher:

 abcdefghijklmnopqrstuvwxyz

 zyxwvutsrqponmlkjihgfedcba
- See Provided Python Script
 - “yeah i guess sincd [sic] it's just **ilikealiens**”
- Whoops. It's Actually “**ilovealiens**”



Round 6 Methodology cont.:

At this time, you should have a file (possibly labeled **ymsg.txt**) with the following content:

```
Wrw blf ivnvnyvi gl hsldevi zmw kfg lm xovzm fmwvidvzi uli WvuXlm?
Nln sld wrw blf tvvg lm Ofpv'h xlnkfgvi?...zmw bvh.
Nln'h szez vbvh rm gsv yzxp lu gsvri svzwh...zmw z gdvoev bvzi lowh
kzhhdliw rh vzhb gl tfvhh.
Bvzs R tfvhh hrmxw rg'h qfhg rorpvzorvmh.
```

19) Decode the ciphertext: Use a manual replacement method or the provided Python script

These four lines are encoded using a substitution cipher. We lucked out in that we have an awesome cryptogram fella on our team. Through manual analysis, he found the substitution pattern. In fact, the pattern is simply the alphabet backwards. Hah!

```
abcdefghijklmnopqrstuvwxyz
zyxwvutsrqponmlkjihgfedcba
```

During the challenge, he decoded the strings manually. You can do so if you wish, but I've also included a short Python script to do the work for you. *Please note that this is a*

simple proof of concept script. No error checking, no try/catch, blah. But it works:

```
#!/usr/bin/env python

def reverse(plaintext):

    alphabet = {'a': 'z',
                'b': 'y',
                'c': 'x',
                'd': 'w',
                'e': 'v',
                'f': 'u',
                'g': 't',
                'h': 's',
                'i': 'r',
                'j': 'q',
                'k': 'p',
                'l': 'o',
                'm': 'n',
                'n': 'm',
                'o': 'l',
                'p': 'k',
                'q': 'j',
                'r': 'i',
                's': 'h',
                't': 'g',
                'u': 'f',
                'v': 'e',
                'w': 'd',
                'x': 'c',
                'y': 'b',
                'z': 'a'}

    decoded = ""

    #Convert each letter to corresponding letter in the reverse
    alphabet
    for character in plaintext.lower():
        if character in alphabet:
            character = alphabet[character]
            decoded += character

    return decoded

decode_us = ["Wrw blf ivnvnyvi gl hsldvi zmw kfg lm xovzm fmwvidvzi uli
WvuXlm?",
             "Nln sld wrw blf tvg lm Ofpv'h xlnkfgvi?...zmw
bvh.",
             "Nln'h szez vbv h rm gsv yzxp lu gsvri svzwh...zmw z
gdvoev bvzi lowh kzhhdliw rh vzhb gl tfvhh.",
             "Bvzs R tfvhh hrmxw rg'h qfhg rorpvzorvmh."]
```

```
for decode_me in decode_us:
    print decode_me
    print reverse(decode_me) + "\n"
```

20) Review the decoded results:

Wrw blf ivnvnyvi gl hsl dvi zmw kfg lm xovzm fmwvidvzi uli WvuXlm?
did you remember to shower and put on clean underwear for defcon?


Nln sld wrw blf tvg lm Ofpv'h xlnkfgvi?...zmw bvh.
mom how did you get on luke's computer?...and yes.

Nln'h szez vbv h rm gsv yzxp lu gsvri svzwh...zmw z gdvoev bvzi lowh kzhhdliw rh vzhb gl
tfvhh.
mom's have eyes in the back of their heads...and a twelve year olds **password** is easy to
guess.

Bvzs R tfvhh hrmxw rg'h qfhg rorpvzorvmh.
yeah i guess sincd it's just **ilikealiens**.


Boom! We just found a password: **ilikealiens**

NOTE: Although the password we found is **ilikealiens**, the PGP private passphrase is actually **ilovealiens**. Some of the teams struggled with this bit, but in all honesty, we simply made an error and entered the “wrong” password, and to our surprise it worked. I guess you can say that the kid is more fond of aliens than one would first think.



Round 6 cont.

- Decrypt GPG-encrypted File
- `gpg --decrypt information.txt.gpg`
- Password: **ilovealiens**



Round 6 Methodology cont.:

21) Decrypt the GPG-encrypted file following the same process from Step 14, but this time use the passphrase: **ilovealiens**

`gpg --decrypt information.txt.gpg`

You need a passphrase to unlock the secret key for

user: "Acturians <bryanthescienceguy@gmail.com>"

2048-bit RSA key, ID 68B1386B, created 2015-06-10 (main key ID AB18879E)

gpg: encrypted with 2048-bit RSA key, ID 68B1386B, created 2015-06-10

"Acturians <bryanthescienceguy@gmail.com>"

Cyper

08/21/15 16:37:54

DATE: 2015-07-01

Weight:3

Minutes Awake:2

Gives Month

DATE: 2015-05-21

Calories Burned:1

Steps:2
Gives Day

DATE: 2015-05-17
Floors:2
Minutes Sedentary:4
Gives Year

DATE: 2015-06-03
Steps:2
Minutes Lightly Active:3
Gives Hour

DATE: 2015-06-09
Steps:1
Calories:2
Gives Minutes


DATE: 2015-06-27
Activity Calories:1
Distnace:3
Gives Seconds

Just about done!



Round 6 cont.

- Open **btsnoop_hci.log** in Wireshark
- Filter on OBEX Traffic: **btobex**
- Filter Header Data:
 - **btobex.header.value.unicode**
- Export All Five CSVs



Round 6 Methodology cont.:

We recognized the **btsnoop_hci.log** file to be a Bluetooth capture. That got us thinking: The first place prize for the challenge was a FitBit. This coupled with the recently decrypted GPG file that includes terms such as “Calories Burned”, “Minutes Awake”, etc. led us to realize that the captured Bluetooth traffic was most likely from a Fitbit.

Since we seemed to have found a Rosetta stone involving Fitbit data, we knew that the data in the Fitbit capture must contain the final answer. Luckily, Wireshark has a decoder for **btsnoop** captures. We ended up finding the proper protocol on which to focus from trial and error along with reading online documentation regarding how files are transferred via Bluetooth (simple enough: https://en.wikipedia.org/wiki/Object_EXchange).

22) Load the **btsnoop_hci.log** file in Wireshark

23) Filter the Object Exchange (OBEX) traffic by using the following filter:
btobex

24) Once filtered, you will notice a total of 214 frames displayed. Of these frames, the third one is frame 2630:

```
OBEX 315 Sent Put continue "fitbit_export_20150701-3.csv"
"text/comma-separated-values"
```

Oh look! A file transfer. The content transferred looks to be a comma separated value (CSV) file containing Fitbit data. Perfect! After scrolling through the OBEX traffic, we noticed many CSV files, so we opted to grab them all at once.


25) Filter the traffic further to review the transferred files:

btobex.header.value.unicode

26) Export all five (5) CSVs. To begin, select a frame that contains a .csv file, such as frame 2889. In the Packet Details pane, expand the **"Bluetooth OBEX Protocol"**. Scroll all the way down so that you are looking at the **Header → Body → Value** section. Right-click this section and select "Export Selected Packet Bytes..." Let's be lazy and label the files **1.csv** through **5.csv**.


NOTE: The 3.csv and 4.csv files are the exact same. This isn't something that you did incorrectly when gathering the files.

Let's finish this bad boy!



Round 6 cont.

- Use Key to Decode the Answer
- Key Format:
 - DATE
 - Item:Index
- ANSWER:
08/21/15 13:37:54



Round 6 Methodology cont.:

To use the key, take note of the date provided along with the x:y notation. The x value is the item in question, whereas the y value is the index in the field value. For example, for **DATE: 2015-07-01 Weight:3**, we need to find the date 2015-07-01 in one of our CSVs. Next, we find the Weight value. Let's say that the weight value is 142. The index value is 3, which means we take the third number in this value, providing the number 2.

27) Using the **1.csv** through **5.csv** files along with the decrypted key file, find the Round 6 answer.

DATE: 2015-07-01 Weight:3 Minutes Awake:2 Gives **Month = ??**

Unfortunately, the 2015-07-01 date is not available in any of our CSV files. This was a slight error in the contest. No biggie, stay tuned.

DATE: 2015-05-21 Calories Burned:1 Steps:2 Gives **Day = 21**

5.csv

Calories Burned = 2,030 / Index 1 = **2**

Steps = 6,109 / Index 2 = **1**

DATE: 2015-05-17 Floors:2 Minutes Sedentary:4 Gives **Year = 51**

5.csv

Floors = 15 / Index 2 = **5**

Minutes Sedentary = 1,091 / Index 4 = **1**

DATE: 2015-06-03 Steps:2 Minutes Lightly Active:3 Gives **Hour = 13**

3.csv

Steps = 6,144 / Index 2 = **1**

Minutes Lightly Active = 233 / Index 3 = **3**

DATE: 2015-06-09 Steps:1 Calories:2 Gives **Minutes = 37**

2.csv

Steps = 3,885 / Index 1 = **3**

Calories = 1,793 / Index 2 = **7**

DATE: 2015-06-27 Activity Calories:1 Distance:3 Gives **Seconds = 54**

1.csv

Activity Calories = 577 / Index 1 = **5**

Distance = 1.74 / Index 3 = **4**

So far, we have: **??/21/51 13:37:54** [MM/DD/YY hh:mm:ss]

Unfortunately, the **2015-07-01** date does not exist in our CSV files, so we do not have a month. This was a slight error with the contest. However, we were able to recover from this quite easily.

Notice that our decoded date is not far from the date at the top of the decrypted GPG file: **08/21/15 16:37:54**. Since the dates were so similar, we substituted the month 8 (August) into our answer, thus providing: **08/21/51 13:37:54**. Also take note that our decoded year was 51, but the date at the top of the decoded file is 15. Since it was 2015, we figured this might have been a slight mistake, so we changed out year to 15. And that was it!

The answer to Round 6 and to finish the challenge is:

08/21/15 13:37:54

Woot! DONE BABY!



That's it gang!

We welcome any comments, suggestions, or questions. Have a better 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 hang out and "talk shop."

Much love to LMG for putting so much time and effort into these challenges. We love them!

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

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

Thanks!!