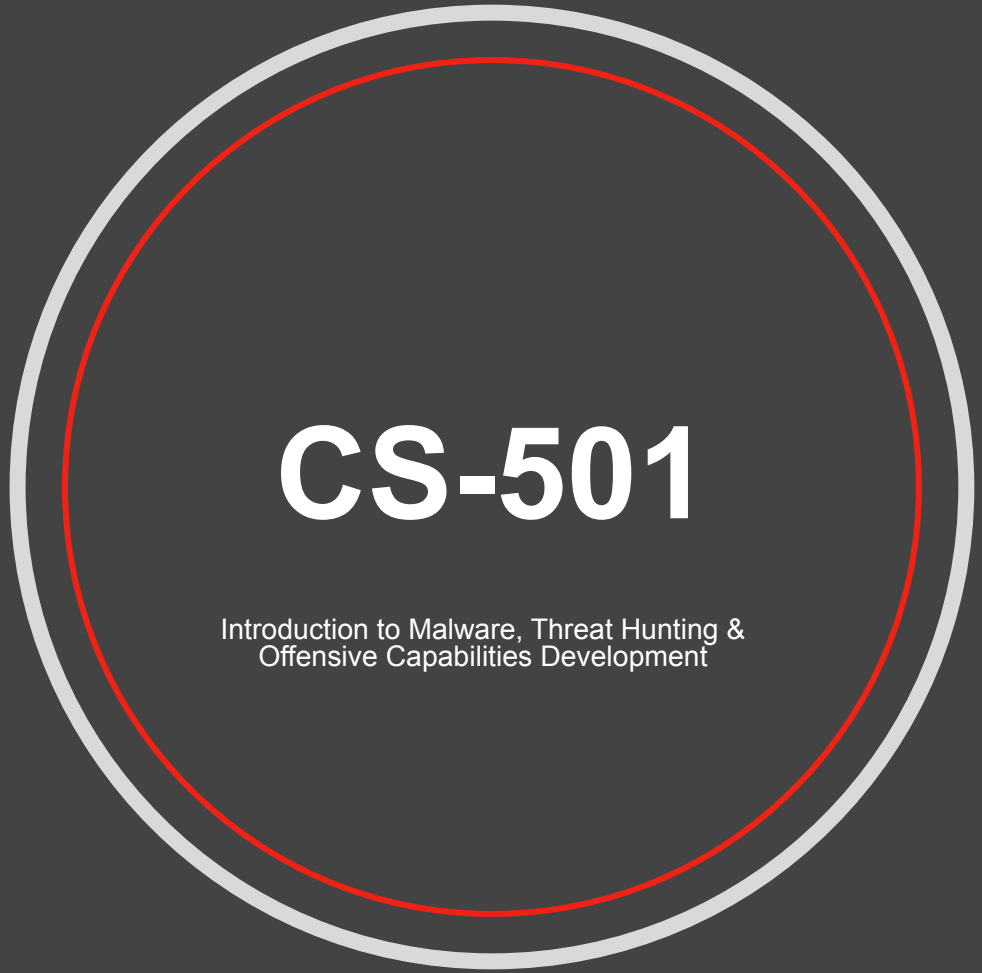


CS-501

Introduction to Malware, Threat Hunting &
Offensive Capabilities Development

Homework 1 Solutions Speedrun

General Advice

- Run the Binary and observe the strings
- Find references to those strings using Ghidra
- Use this to easily identify the real entry point
- Look at the decompiled assembly
- Use a Debugger. That will usually save you a LOT of time
- If something doesn’t work the way you would expect it to, ask a question!
- Collaborate with with your classmates. The discord has been absolutely silent.
    - Are y’all really doing all the homework solo after I made that wonder, impassioned speech about teamwork :-) ?

Crackme 0

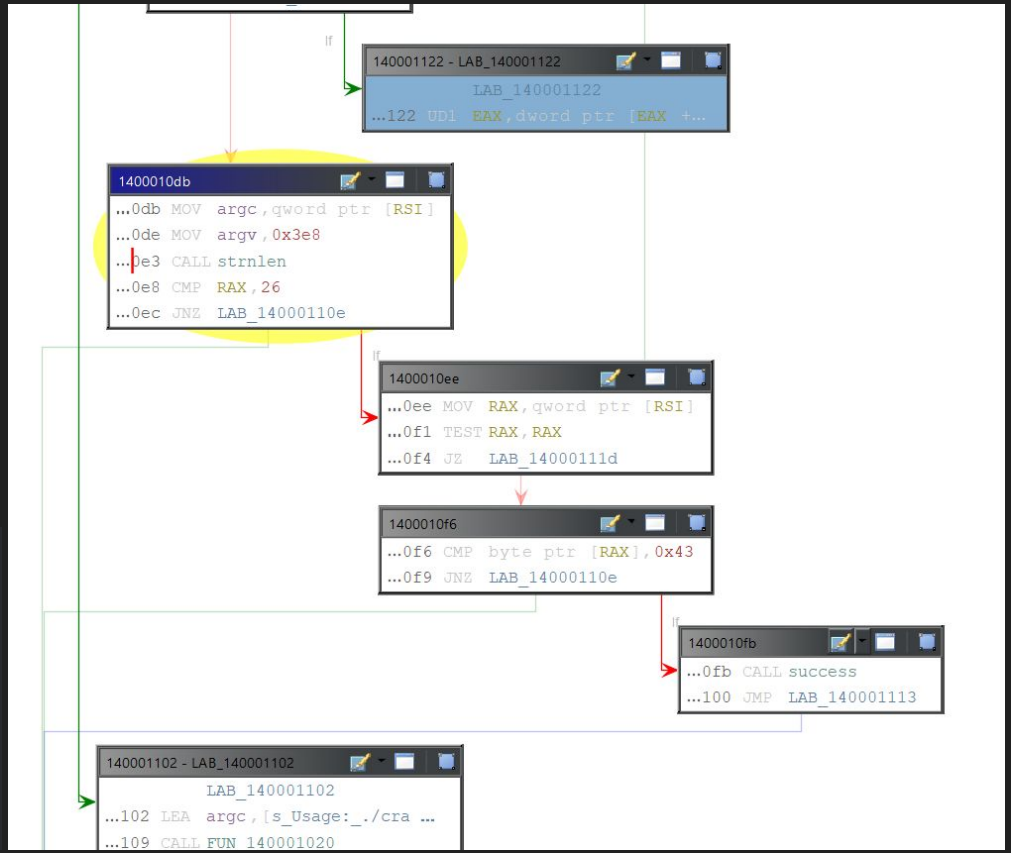
- Intended difficulty: trivial (so long as you know how to find the main entry point!)
- Run strings on the binary.
- Find main
- See that it is directly comparing your input to a static string
- GG

Crackme 1:

Intended Difficulty: Easy

The binary checks if the length of the input is 26, and if it starts with a C

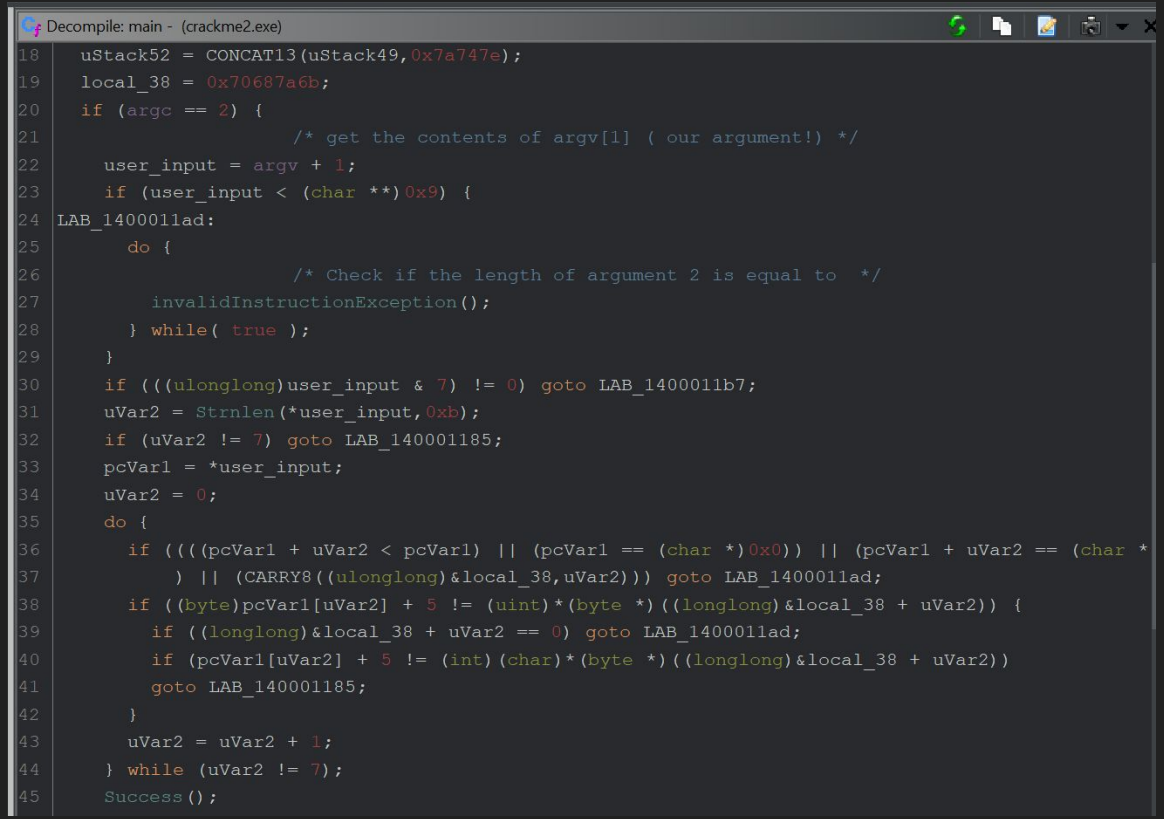
```
  uVar1 = strnlen(*ppcVar2,1000);
  if (uVar1 == 26) {
    if (*ppcVar2 == (char *)0x0) goto LAB_14000111d;
    if (**ppcVar2 == 'C') {
      success();
      return 0;
    }
  }
}
else {
```



Crackme 2 (Again, I am really sorry)

Intended Difficulty: Moderate

Question to answer: WTF is happening inside of that do-while loop?



Crackme 2 (Again, I am really sorry)

- A stack variable is declared, with values inside of the ascii range (HINT)
- We iterate through values in argv[1]
- Copy that character into a buffer
- Copy the char from the stack var into a buffer
- Checks to see if the the input char + 5 is the same as the stack variable
- If not, Fail. If yes, continue
- Once we are done checking all the characters in the stack string with no errors, Success

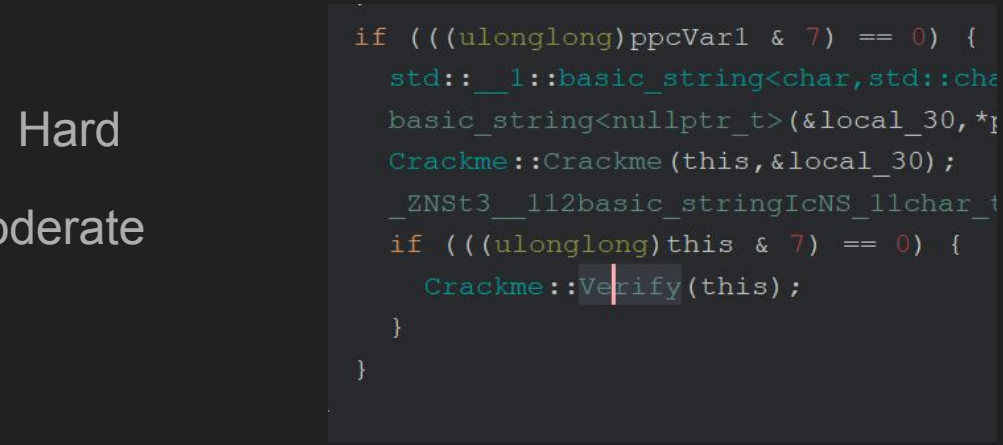
Crackme 2 (Again, I am really sorry)

```
In [38]: x = 0x70687a6b.to_bytes(4, "little")

In [39]: y = 0x7a747e.to_bytes(3, "little")

In [40]: z = x + y

In [41]: zz = bytearray()

In [42]: for i in z:
    ...:     zz.append(i - 5)
    ...:

In [43]: zz
Out[43]: bytearray(b'[illegible]')
```

Crackme 3

Intended Difficulty without PDB: Hard

Intended difficulty with PDB: Moderate



Crackme 3

We can trace the execution to see a reference to a variable called alphabet

There is a some function that is outputting integer sequences that are used as lookup values

We need only trace through all those indexes and compare them to the lookups in the table

The easiest way to do this is to use the debugger to change the values in the verify function to always true, and observe the full sequence of indexes

Crackme 4:

Intended Difficulty: Easy if you know the trick, **computationally intractable otherwise**

Boot up Ghidra, find the main function, and take a look at the stack data

```
FUN_140002070();
local_30 = DAT_14000b050 ^ (ulonglong)local_128;
local_58 = 0xdb21592e;
uStack84 = 0xf35baedf;
uStack80 = 0x35cc0ede;
uStack76 = 0x635874b6;
uStack72 = 0x3a17bab8;
uStack68 = 0x999a9686;
uStack64 = 0xfc3d504e;
uStack60 = 0xf2c35b83;
local_38 = 0;
```

Crackme 4

Decompile: main - (crackme4.exe)

```
46    }
47    if (((ulonglong)our_guess & 7) != 0) {
48      do {
49        invalidInstructionException();
50      } while( true );
51    }
52    vzeroupper_avx();
53    uVar3 = FUN_1400028e0(*our_guess,1000);
54    FUN_140001230(local_168);
55    FUN_1400012d0((ulonglong)local_168);
56    FUN_140001300((ulonglong)local_168);
57    uVar4 = 0;
```

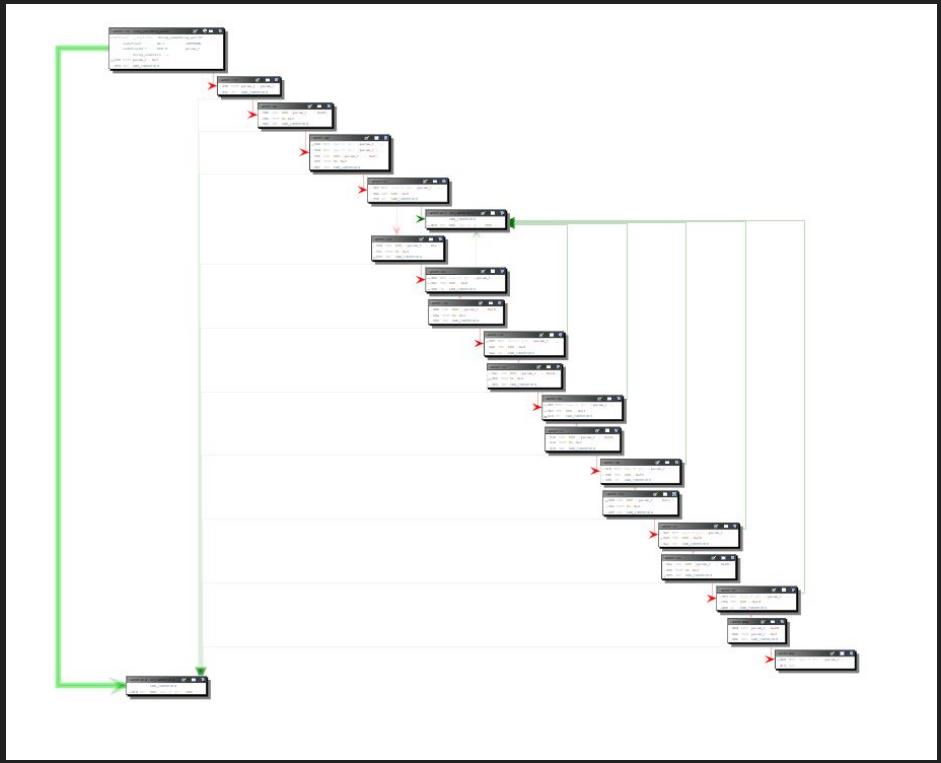
Crackme 4

```
void mystery_function(undefined8 *param_1)

{
  int iVar1;

  if ((((((ulonglong)param_1 & 7) == 0) && (param_1 != (undefined8 *)0x0)) &&
       (iVar1 = (int)param_1, (iVar1 + 0x98U & 7) == 0)) &&
      (((param_1[0x13] = 0x40, (iVar1 + 0xa0U & 7) == 0 &&
        (param_1[0x14] = 0x20, (iVar1 + 0xa8U & 7) == 0)) &&
       ((param_1[0x15] = doing_something_weird?, (iVar1 + 0xb0U & 7) == 0 &&
        (param_1[0x16] = &LAB_140001420, (iVar1 + 0xb8U & 7) == 0)))))) {
    param_1[0x17] = &LAB_140001550;
    doing_something_weird?(param_1);
    return;
  }
  do {
    invalidInstructionException();
  } while( true );
}
```

Crackme 4

When you see graphs like this, you are dealing with some sort of repeated function or round function.

For malware, it is usually crypto.



Crackme 4

Some weird constants

0x6a09e66 ?

```
void doing_something_weird?(undefined8 *param_1)

{
  undefined8 *puVar1;

  if (((((ulonglong)param_1 & 7) == 0) && (param_1 != (undefined8 *)0x0)) &&
     (((int)param_1 + 0x28U & 7) == 0)) {
    *(undefined4 *)(param_1 + 5) = 0;
    *param_1 = 0;
    puVar1 = param_1 + 1;
    if (((ulonglong)puVar1 & 7) == 0) {
      *(undefined4 *)(param_1 + 1) = 0x6a09e667;
      if ((undefined8 *)0xfffffffffffffffb < puVar1) {
LAB_140001419:
        do {
          invalidInstructionException();
        } while( true );
      }
      if (((longlong)param_1 + 0xcU & 3) == 0) {
        *(undefined4 *)((longlong)param_1 + 0xc) = 0xbb67ae85;
        if ((undefined8 *)0xfffffffffffffff7 < puVar1) goto LAB_140001419;
        if (((ulonglong)(param_1 + 2) & 7) == 0) {
          *(undefined4 *)(param_1 + 2) = 0x3c6ef372;
          if ((undefined8 *)0xfffffffffffffff3 < puVar1) goto LAB_140001419;
          if (((longlong)param_1 + 0x14U & 3) == 0) {
            *(undefined4 *)((longlong)param_1 + 0x14) = 0xa54ff53a;
            if ((undefined8 *)0xffffffffffffffef < puVar1) goto LAB_140001419;
            if (((ulonglong)(param_1 + 3) & 7) == 0) {
```

Crackme 4

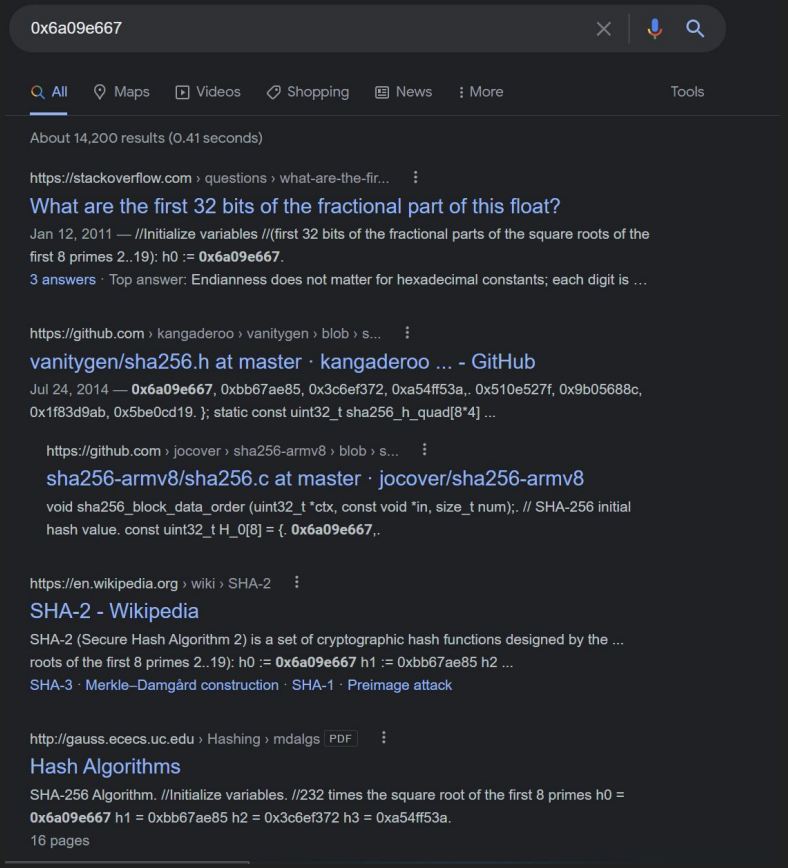
Some weird constants

0x6a09e66 ?

```
void doing_something_weird?(undefined8 *param_1)

{
  undefined8 *puVar1;

  if (((((ulonglong)param_1 & 7) == 0) && (param_1 != (undefined8 *)0x0)) &&
     (((int)param_1 + 0x28U & 7) == 0)) {
    *(undefined4 *)(param_1 + 5) = 0;
    *param_1 = 0;
    puVar1 = param_1 + 1;
    if (((ulonglong)puVar1 & 7) == 0) {
      *(undefined4 *)(param_1 + 1) = 0x6a09e667;
      if ((undefined8 *)0xfffffffffffffffb < puVar1) {
LAB_140001419:
        do {
          invalidInstructionException();
        } while( true );
      }
      if (((longlong)param_1 + 0xcU & 3) == 0) {
        *(undefined4 *)((longlong)param_1 + 0xc) = 0xbb67ae85;
        if ((undefined8 *)0xfffffffffffffff7 < puVar1) goto LAB_140001419;
        if (((ulonglong)(param_1 + 2) & 7) == 0) {
          *(undefined4 *)(param_1 + 2) = 0x3c6ef372;
          if ((undefined8 *)0xfffffffffffffff3 < puVar1) goto LAB_140001419;
          if (((longlong)param_1 + 0x14U & 3) == 0) {
            *(undefined4 *)((longlong)param_1 + 0x14) = 0xa54ff53a;
            if ((undefined8 *)0xffffffffffffffef < puVar1) goto LAB_140001419;
            if (((ulonglong)(param_1 + 3) & 7) == 0) {
```

Crackme 4

Great news! We figured out it is hashing our input. Bad news, we can’t do better than brute forcing the flag with that.



But remember...

- Strings
- Sections
- Resources
- … :-)
- No, I am not just trolling you.
- Malware will commonly hide data inside of resources.
- You should always check the resources

Dynamic Analysis

Nobody asked **me** the following question:

- Why do memory addresses in Ghidra differ from the addresses displayed in debuggers like x64dbg?
- Once I find the real main function in Ghidra, how do I set a breakpoint there in x64dbg?

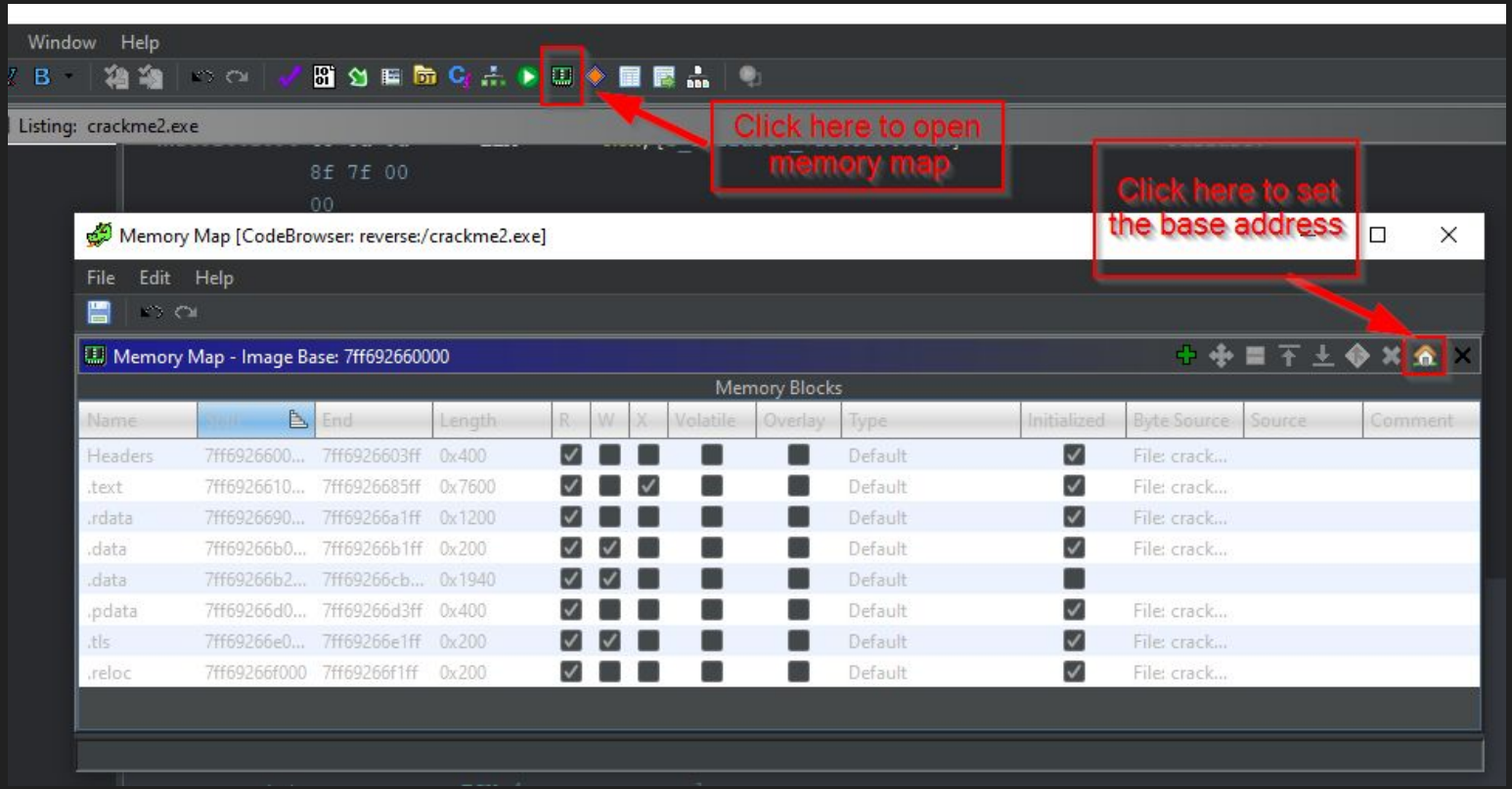
Exploit Mitigations that are a Pain for Reverse Engineering

Address Space Layout Randomization (ASLR): randomizes the base address

You should disable this on your windows VM, or if you are not using a VM, you can rebase the image in Ghidra

File	NX	Canary	ASLR	Dynamic Base	High Entropy VA	SEH
crackme2.exe	Yes	No	Yes	Yes	Yes	Yes

Rebasing a PE in Ghidra



Example: Crackme 2

- Find the main function in Ghidra
- Run the PE in x64dbg
- Rebase the image based on the base address in x64dbg
- Set a breakpoint at main
- Run the binary with command line argument with “aaaaaaa”

Maldoc

- Checks if a file exists. If not, it crashes
- The maldoc reaches out to ch0nky.chickenkiller.com
- Chickenkiller.com is a **dynamic dns provider!**
- The request uses WinHTTP from VBA to download and run (with powershell) a second stage binary
- Note the User-Agent header set by the malware is ch0nky
- If you try to visit the site and download the file as is, without setting the correct UA, it will download calc.exe
- Afterwards, it crashes word.

If you tried to reverse engineer calc.exe, remember, **you should always search for the hash of the binary you are reversing to see if someone has already done your work for you!**

Dropper

The next stage of the malware uses URLDownloadToFile and CreateProcess on the downloaded file.

In other words, it downloads a file, and executes it. This is behavior consistent with a dropper.

There isn’t too much going on with this binary other than the check for the ch0nky path, in addition to a Doge icon file

Payload

The final payload appears to be some sort of troll. It doesn't try to hide itself. In fact, it does everything it can to make itself visible

- It shakes the screen, messes with the color scheme, and plays the Soviet National anthem
- This type of behavior is consistent with trolling
- What is strange, is when we observe the resources, there are none
- In fact, this binary doesn't even declare imports. How does it even do anything?

Final Payload Bonus: It is packed!

We haven’t talked about packing yet, but you will often come across techniques used by malware authors that you haven’t seen before! By looking at the sections with Ghidra, you would see a few labeled “UPX”

A quick google search gives you a way to “unpack” the binary and observe that it is indeed only trolling

It plays the music by loading a wav file from a resource and playing it

To get full credit on this assignment, you needed to find the resource file

Lecture 10: C2 Engineering 101

For this lecture, we are going to assume the C2 channel is HTTP(s)

- Implant: malware communicating with the server
- Client: Client code used to allow the operator to interact with implants through the server
- Server: Handles connections from Clients and malware

Team Server

Consider the following:

Multiple implants connect to the C2

Multiple operators connect to the C2 to control the agents

Operators need to be appraised of the activity of their colleagues and status updates of their agents

Messages are sent from the teamserver to operators

- Example messages that operators would want to see
- A new bot has connected to the server
- A bot has pulled down a task
- A bot has sent data to the server
- An operator has issued a command to an agent

Server Components and Concepts

Listeners: an HTTP listener that handles connections from implants

Database: Where we store information about clients, implants, messages...etc

Messaging: How the server communicates with clients

Databases

We use Databases to store information. Databases can be fairly simple or complex, live on disk or in memory, on one machine, or across many.

For our purposes, we are going to work with databases that exist on one machine and store structured data (i.e., it has a schema of some kind)

We already talked about an example of a database: SQLite

Non example: Document databases like MongoDB

SQL Databases: The big 4

The query language across all 4 is basically the same, with some small changes (characters for comments, built in functions, supported data types, scalability..etc)

SQLite: Lightweight, file based database. Scapable for single users but has some growing pains. Example use case: maintain state and store data for a browser

MySQL: Scalable, single machine database that is simple to manage and setup

PostgreSQL: Scalable, single machine database that is considerably more feature rich than MySQL but also more complex to manage.

Microsoft SQL: If you want to be a red teamer, you probably have to deal with Microsoft SQL at some point, but since we only care about DBs as a means to build better C2s, we don’t use it in this class.

Basic SQL: Really, again with learning another language?

While SQL is a language, it isn’t a programming language per say. It is a way of interacting with SQL databases to perform CRUD operations (Create, Retrieve, Update, Destroy)

It is still in high demand in a professional setting, and is something I would recommend you learn for a wide variety of reasons.

Red teamers and blue teamers alike should know how databases work, as this is typically where an organization's the crown jewels live :) (Think customer databases, payment databases….etc)

SQL: Seriously oversimplified

- SQL databases are composed of tables. Tables are composed of rows. Rows are composed of data entries of a fixed type defined in the table's schema.
- In the simple case, you can think of a SQL database as an Excel Sheet, where different tabs within a sheet correspond to tables in a database.
- Each tab is filled with rows in a table, where each column has a name and type
- Data can be queried based on a standard language called SQL (Structured Query Language) to retrieve the “structured” data
- Here structured means there is a tabular schema associated with each table

Which database should you use for your C2?

Questions you need to answer first:

- How many implants do you expect to connect to your server?
- How many operators do you plan on having active at once?
- How much structured data are you collecting from implants and operators?
- What channel are you using for your C2?
- What RPC are you using for your client and implant?

You probably shouldn’t use SQLite past the POC phase

SQLite CRUD operations are blocking, and as a database can only support one “connection” at a time.

This can result in incredibly slow CRUD when there are multiple applications trying to interact with the DB

Recommendation: Stick to Postgres or MySQL

Both are production ready, incredibly powerful, and have more than enough features to handle. MySQL is probably the easier choice, but is outclassed by Postgres once the number of active connections balloons, and the schema’s complexity grows.

SQLAlchemy

- Use SQL without writing any SQL! When paired with Flask you should use Flask SQLAlchemy.
- This allows you to declare python classes instead of creating SQL schemas, and can make performing CRUD operations easy!
- Read the quickstart for a basic understanding of how to accomplish this
- https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/

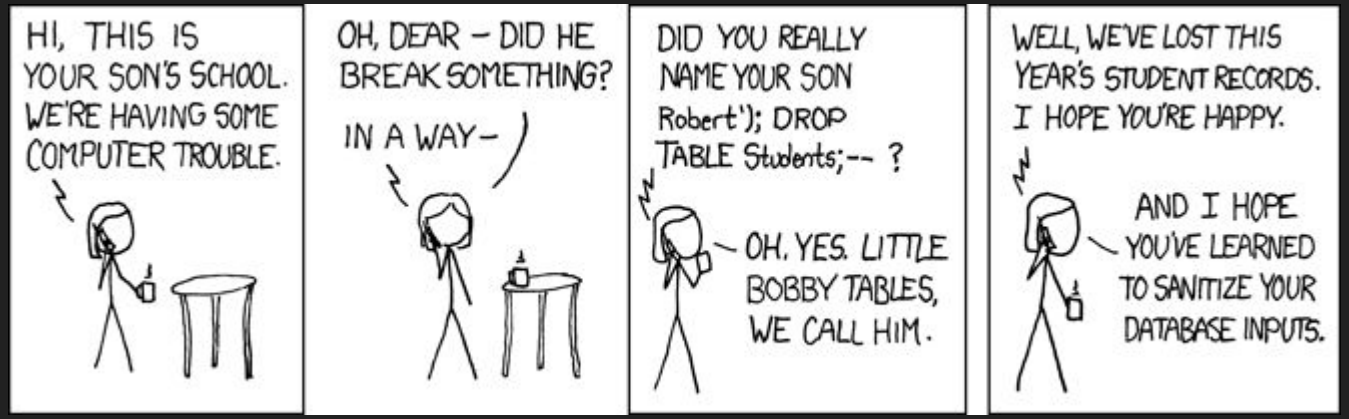
Example using Sqlite3

- All SQL DB workflows start by establishing a connection to the database
- Execute a command
- You are then given a “cursor” object that allows your to iterate through results

Remarks about Safety

SQLAlchemy or other ORMs are usually safer than writing SQL queries yourself, and is also usually easier. Many malware authors shoot themselves in the foot by interacting with SQL databases in an insecure way.

Obligatory: https://xkcd.com/327/



Safe File Upload

Spot the error

```python
@app.request("/upload_json", methods=["POST"])
def handle_upload():
    json_data = request.json
    data = json_data.get("data")
    raw_bytes = decode_data(data)
    filename = json_data.get("filename")
    loot_path = "static/" + filename
    with open(loot_path, 'wb+') as f:
        f.write(raw_bytes)
        print("I am 1337 and got wrote my loot to ", loot_path)
    return jsonify({"status": "ok"})
```

Safe File Upload

Do we actually control “filename”?

```
@app.request("/upload_json", methods=["POST"])
def handle_upload():
    json_data = request.json
    data = json_data.get("data")
    raw_bytes = decode_data(data)
    filename = json_data.get("filename")
    loot_path = "static/" + filename #This is super unsafe!
    with open(loot_path, 'wb+') as f:
        f.write(raw_bytes)
        print("I am 1337 and got wrote my loot to " , loot_path)
    return jsonify({"status": "ok"})
```

Brokered Message Queues

An application that can be used to “broker” messages

You connect to a message broker, and it routes messages from their senders to their intended recipient in a centralized way.

BrokerLess: ZMQ

- Relies on sockets to send and receive messages
- Fewer features than rabbitmq, but is lighter and easier to manage since it is more or less a (incredibly useful!) wrapper around sockets
- One problem with ZMQ: only one thread can publish from a socket at a time. This can get annoying when you run your application in production and need to connect multiple publishers

Alternatives

Custom: Don’t do this in python. The GIL only allows for process based parallelism and this can get very unwieldy

Redis: Yes

Publish Subscribe Pattern

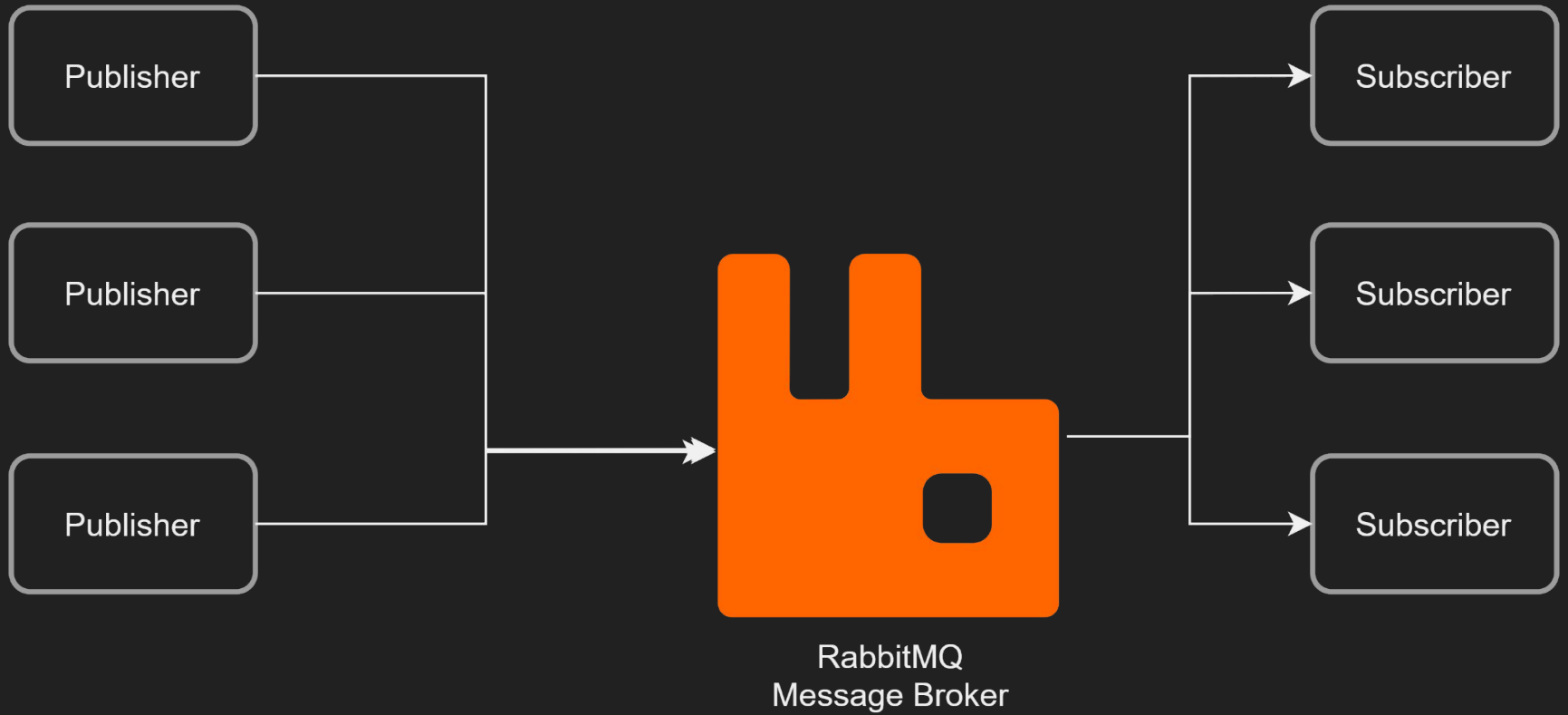
Publish Subscribe (Pub/Sub) is a messaging pattern

One or many Publishers produce messages without any knowledge where the messages will be routed. Each message has an associated topic

One or many Subscribers subscribe to a particular topic. This means they consume messages produced by the publisher

Without a message broker, you are usually limited to 1 publisher per socket.

Overview of Pub/Sub Pattern



RabbitMQ Example

```
user@LAPTOP-4RN9EIPI:~/CS-501-code-snip/pub_sub$ python3 pub.py
 [x] Sent 'Agent 0 has checked in!'
 [x] Sent 'Agent 1 has checked in!'
 [x] Sent 'Agent 2 has checked in!'
 [x] Sent 'Agent 3 has checked in!'
 [x] Sent 'Agent 4 has checked in!'
user@LAPTOP-4RN9EIPI:~/CS-501-code-snip/pub_sub$
```

```
user@LAPTOP-4RN9EIPI:~/CS-501-code-snip/pub_sub$ ipython3 sub.py
 [*] Waiting for logs. To exit press CTRL+C
 [x] b'Agent 0 has checked in!'
 [x] b'Agent 1 has checked in!'
 [x] b'Agent 2 has checked in!'
 [x] b'Agent 3 has checked in!'
 [x] b'Agent 4 has checked in!'
```

```
user@LAPTOP-4RN9EIPI:~/CS-501-code-snip/pub_sub$ ipython3 sub.py
 [*] Waiting for logs. To exit press CTRL+C
 [x] b'Agent 0 has checked in!'
 [x] b'Agent 1 has checked in!'
 [x] b'Agent 2 has checked in!'
 [x] b'Agent 3 has checked in!'
 [x] b'Agent 4 has checked in!'
```

Messaging

Messages between the operators and the server need to also be standardized.

Personally, I prefer to use JSON as it is easy to parse and serialize.

Messages are grouped into types of messages called events, which are published to operators by the server

Example events: New_implant_event, Implant_checkin_event, Implant_response_event...etc

Operators can choose which types of messages they wish to subscribe to based on a topic