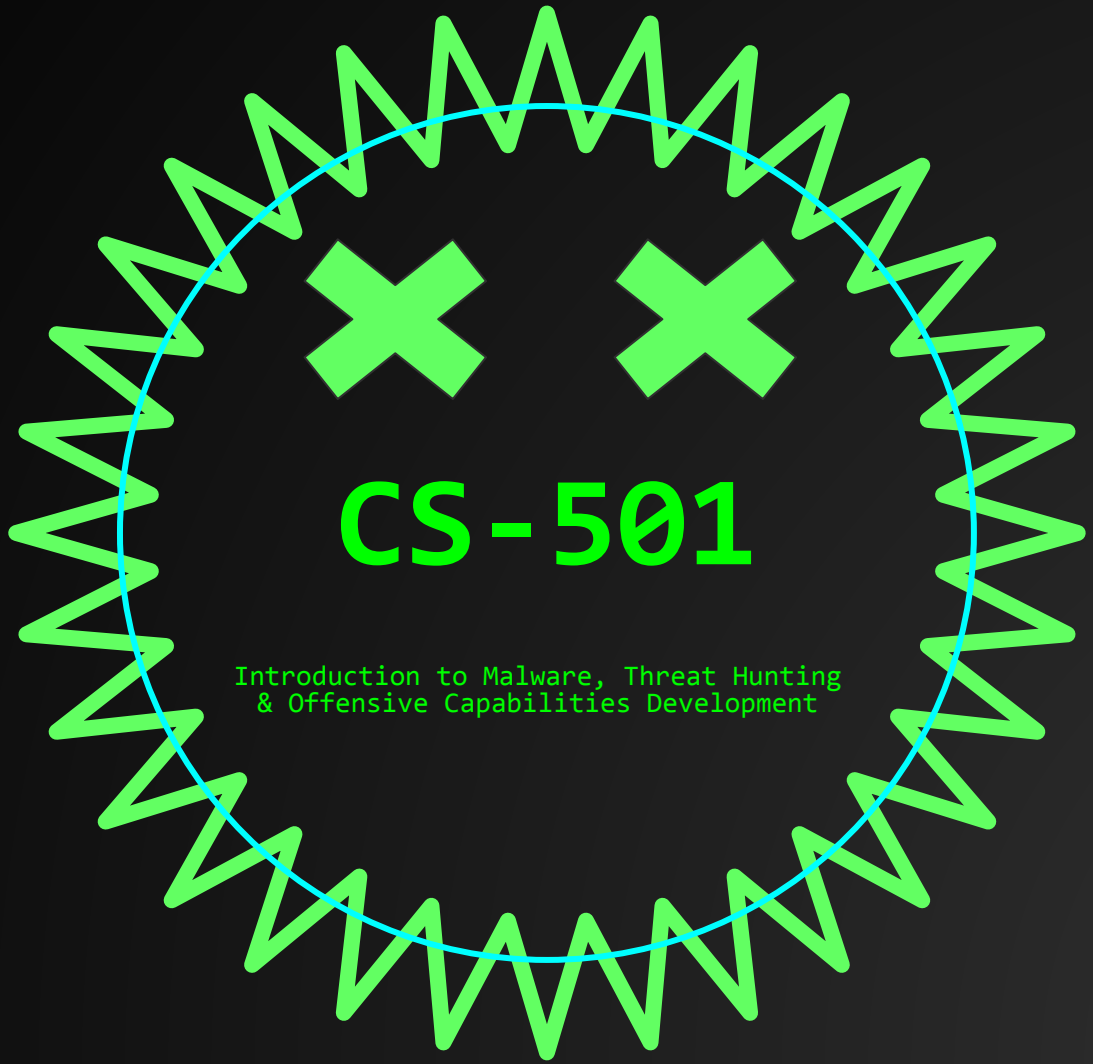

CS-501
Introduction to Malware, Threat Hunting
& Offensive Capabilities Development

Lecture 14:
C2 Engineering, Classes and Containers

Tasking and jobs

- The C2 server + DB should allow clients to task an agent to perform a job
- The job is identified with a Globally unique identifier and is associated to an implant
- When the implant checks in, it pulls down the task, and executes the job
- The Agent the responds with the Identifier for the task, in addition to the result
- The server then notifies all operators of the new information

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.

Task Queues

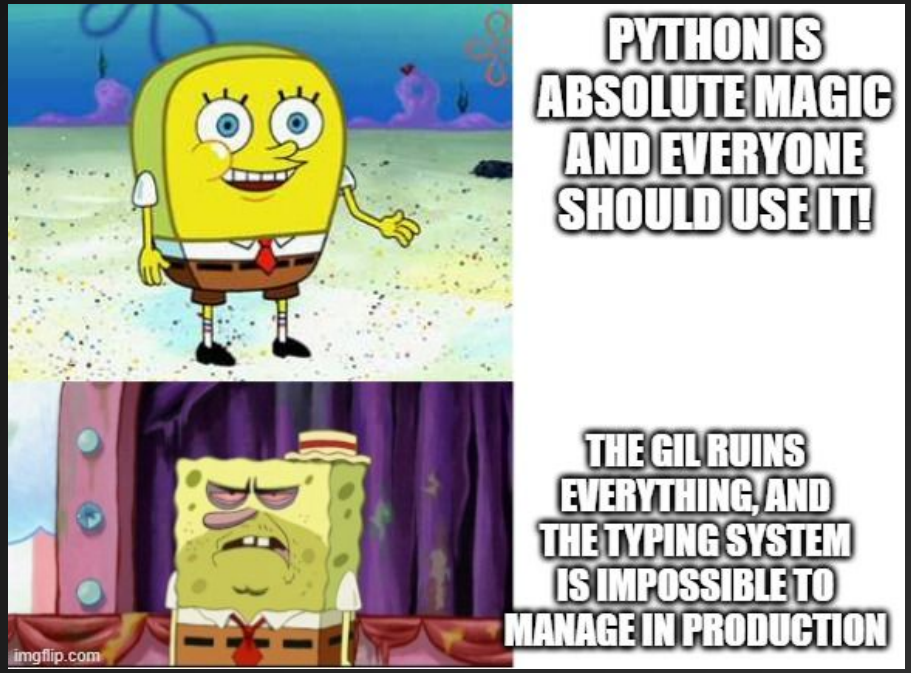
- Keyed FIFOs
- Think of this as every Implant gets their own queue of work
- Each time the implant checks into the server, they pop n tasks from the front of the queue
- Operators can queue up more work for agents

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 an 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
- Especially because of python’s GIL

Consideration: Python GIL

- GLobal Interpreter Lock
- In python, only 1 thread can actually execute code in the interpreter at a time
- This bottlenecks performance
- To achieve true parallelism, you can use Multiprocessing to send and receive pickled objects
- This is also not ideal since processes are more expensive to spawn than threads!



Alternatives

Redis: Yes

Zeromq: not recommended but will work

Custom: You could, but doing so in python is going to be a pain. The GIL only allows for process based parallelism and this can get very unwieldy

SQL DB: this can technically be implemented using a database. It is a bit slow in practice

Kafka: I mean I guess?

Neo4j: ???

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

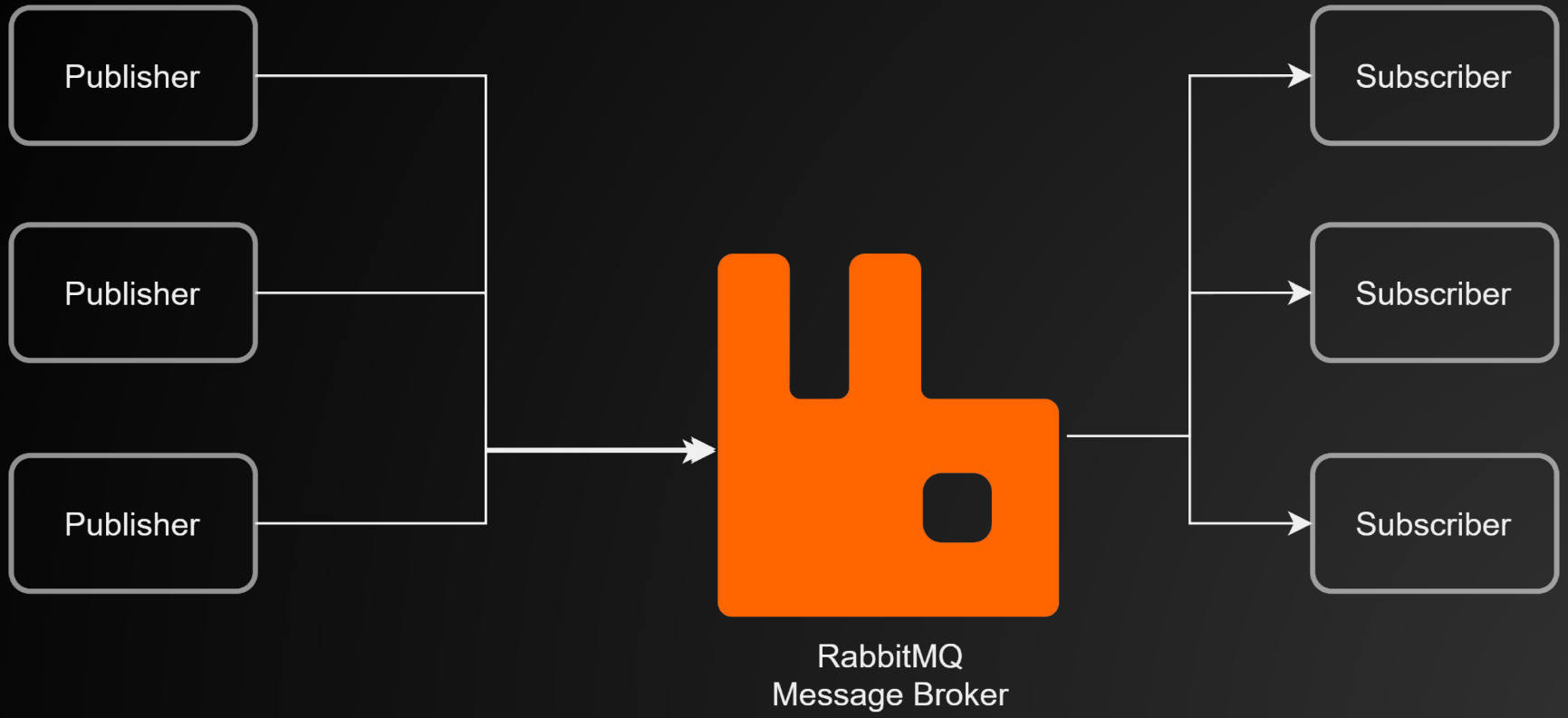
Publish Subscribe Pattern

One or many Subscribers subscribe to a particular topic(s).

Subscribers consume messages associated to their list of topics

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

Using C++: Containers

- The C++ Standard Library (STL) provides many tools to make coding easier
- Containers allow for automatic managing of dynamic data

Templates

- Special functions that allow *generic* arguments
- Compile time feature
- Allows for code reuse parameterized by a type T

Template Example

C++ std::vector

- *Sequence container* template
- Lifecycle of a std::vector is automatically managed
  - This includes automatic heap allocations/deallocations!
- Vectors are especially useful, as the data contained is guaranteed to be contiguous in memory!
- Think vectors of it as being similar to python’s list type
  - Except you cannot mix types with vectors
  - You can make a vector of pointers, and type information though!

Example: std::vector<BYTE>

LPVOID from std::vector<BYTE>

Classes

- Similar to C-Structs
- Allow for declaring data as private/public (compiler feature)
- Constructor: function invoked on creation of class object
- Destructor: function invoked when an object is destroyed

Pure Virtual Functions

- Similar to Golang’s interfaces
- Declare a base class
- Set a function to virtual
- Set its implementation to 0→ forcing any class derived from the base to implement that function

Demo: Factory Pattern