Project Proposal: Process Behavior Analysis with Focused Clustering

Isaac Amann

12/13/2023

Contents

1 Abstract

Windows programs have to use Windows API functions in order to interact with the operating system. These are often abstracted through the use of libraries, but programs still use them even if the programmer does not interact with them directly [3]. By counting the API calls made by processes, a data set can be created where process behavior can be analyzed. A motivating example of this would be ransomware. Ransomware programs would make many file access API calls since most files on the system would be opened in order to encrypt them. Other programs with a similar number of file access API calls may also be ransomware. In a large enough data set, it may be possible to group programs by their behavior.

2 Introduction

Windows applications interface with the Windows operating system through a series of API calls found within DLL files on the system. A process's behavior could be modeled as a data point consisting of the total of times a process made a specific API call during execution for each API call monitored by the system. The system will use DLL injection to overwrite the addresses of Windows API calls in the target process's memory. The functions overwriting the default functions will include code for counting API calls made before redirecting to the original call. The data set created by this system can then be analyzed by different data analysis techniques including clustering and focused clustering.

Initially I planned on using the C++ library Detours released by Microsoft in order to reroute API calls. The library contains functions for attaching to specific API calls and altering their behavior [2]. The library included the functionality needed, but could only attach to processes at start of execution and not into already running processes. To attach to an already running process, a remote thread should be opened on the target process where loadLibrary can be called in order to inject a DLL that will overwrite the monitored API calls in the target process's memory [1]. This technique should not disrupt a process since it is only inserting a small operation that should be executed before calling the original API call, but testing will be required to ensure it does not make programs unstable. The article showing this technique was posted in 2002, but everything used is still present in the Windows API documentation from Microsoft so it should still be viable.

The main data analysis technique used by the system will be focused clustering using the Sow And Grow algorithm currently being developed by Dr. Che. The algorithm is based on DBSCAN but instead of clustering the entire data set, certain data points are passed as seed points. The algorithm then uses this seed points to grow clusters around them. Processes of interest will be used as seed points to grow clusters in the data set that should contain data points corresponding to processes with similar behavior to that of the processes used as seed points.

The main motivation for doing this is malware detection. A known sample of a specific type of malware could be used as a seed point and processes that happen to fall into the cluster grow from this seed point may be malware of the same type. Using this method, malware in the data set that has eluded other detection methods could be revealed.

Version control will be done through Github. The following repositories will be used.
Collection Server: https://github.com/IsaacAmann/Windows-API-Monitor-Collection-Server
Collection Client: https://github.com/IsaacAmann/Windows-API-Monitor

3 Requirements

3.1 Collection Methods

Requirements were collected over multiple meetings with Dr. Che and other students working on the Sow And Grow research project. Requirements were also collected while working on an early prototype.

3.2 Functional Requirements

FR1 Server Data Collection		
Goal: Collection Server should collect Data Points from Client		
Collection Server should host an API endpoint that allows Client software to send Data Points collected while monitoring processes on the host system		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: high

FR2 Client Side Data Collection		
Goal: Client software should monitor running processes		
Client should monitor processes running on the host system and collect Data Points that include the number of Windows API calls made by the processes.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: high

FR3 Process Information		
Goal: Data points should include identifying information		
In addition to the sums for the API calls made by the process, the data point should include additional information including run time, executable name, and possibly a signature of the executable. While the clustering algorithm only needs the unlabeled data, these labels will be needed to make use of the data after being analyzed.		
Origin: Meeting with Dr. Che		
Version: 1.0	Date: 11/15/2023	Priority: medium

FR4 Data Storage		
Goal: Collected data should be stored for later analysis		
Server should store received Data Points inside of a relational database. Clusters generated should also be stored in this database for later use.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: high

FR5 Data Analysis		
Goal: Server should use Sow and Grow algorithm to analyze data		
The server should contain methods that allow the User to provide Seed Points to grow Clusters from the stored Data Set. Data Points that fall into the Clusters generated by the Seed Points should be similar to the Seed Points used.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: high

FR6 Front End		
Goal: Server should provide a front end for accessing the system		
The server should interface with a web dashboard to allow authorized users to view data, diagnostic information, and analyze data.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: high

FR7 API Key Generation		
Goal: The server should provide a method to allow new clients to be registered		
API calls should be provided to allow new clients to register and receive a API key used to authenticate when submitting data.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: medium

3.3 Non-Functional Requirements

NF1 Backups		
Goal: Backups should be kept to prevent data loss		
The database containing the data set should be backed up regularly to prevent loss of data or to allow for rollbacks in the case bad data was submitted to the data base.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: low

NF2 Client Stability		
Goal: The client software should not make the host system unstable		
The client software will need to intercept Windows API calls made by other processes. This should be done without causing processes to crash or significantly decreasing performance		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: low

NF3 Client Authentication		
Goal: Server should authenticate clients before accepting data		
The server should use API keys to authenticate clients preventing unwanted data from being submitted to the data set by bad actors. Data points should also be labeled with identifying information indicating which client they originate from.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: medium

NF4 Data Transfer Security		
Goal: Data from clients should be sent securely		
Data sent through API endpoint should not be sent as plain text. Data may include sensitive data including system information or API keys that could be used to send requests in place of the genuine client.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: low

NF5 Rate Limits		
Goal: The server should not be vulnerable to brute force attacks		
The server should implement rate limits on API calls to prevent API keys or passwords from being leaked with a brute force attack.		
Origin:		
Version: 1.0	Date: 11/01/2023	Priority: medium

3.4 Use Cases

Use Case Name	Post Data Point
Related Requirements	FR1, FR3
Goal In Context	A Client posts a new data point to the Server through HTTP
Preconditions	Client is registered with the Server and possesses a valid API key
Successful End Condition	Data point is processed by the Server and stored on the Database. A success message is returned.
Failed End Condition	Server rejects data point and returns an error message in its response.
Primary Actors	Client, Server
Secondary Actors	Database
Trigger	Client makes a post data point request
Main Flow	1. Client makes a post data point request to the Server 2. Server authenticates the Client 3. Server creates a new data point object 4. Server pushes new data point to Database
Extensions	• None

Use Case Name	Register Client
Related Requirements	FR7
Goal In Context	A new Client registers with the Server and is provided an API key for making future requests from the server
Preconditions	Client is installed on host and Server is running
Successful End Condition	An API key is generated and returned to the Client. API key is also stored to Database along with other Client information.
Failed End Condition	New Client is rejected and an error message is returned.
Primary Actors	Client, Server
Secondary Actors	Database
Trigger	Client software makes a register request
Main Flow	1. New Client instance makes a register request to the Server 2. Server authenticates the new Client 3. Server creates a new registered client object 4. Server generates an API key for the new Client 5. Server stores new registered client object to Database 6. Server returns the API key in its response
Extensions	• None

Use Case Name	Start Process Monitor
Related Requirements	FR2, FR3
Goal In Context	Client begins monitoring Processes running on the host system
Preconditions	Client is running with elevated permissions
Successful End Condition	Client has access to a list of running processes along with their ID's
Failed End Condition	Client halts execution outputting an error
Primary Actors	Client, Host System
Secondary Actors	Process
Trigger	Client software started or refreshes its list of running processes on the host system
Main Flow	1. Client requests a list of processes from the Host System 2. Client checks for new processes
Extensions	• 2.1: Client attaches process if it is a new process • 2.2: Client detaches a process as it exits

Use Case Name	Get Process Info
Related Requirements	FR2, FR3
Goal In Context	Client retrieves info about a Process including a handle referencing it
Preconditions	Client is running and was able to successfully retrieve a list of process ID's
Successful End Condition	Client has a valid handle referencing a target Process
Failed End Condition	Client does not receive a valid handle to a target Process. Client discards the ID of the target Process
Primary Actors	Client, Process
Secondary Actors	None
Trigger	A new process is found when refreshing the running process list
Main Flow	1. Client requests process information from the Host System using process PID 2. Client receives a Handle referencing the target Process
Extensions	• None

Use Case Name	Attach Process
Related Requirements	FR2
Goal In Context	Client injects a DLL into the target process to reroute Windows API calls to include code for counting API calls made by the target process
Preconditions	Client has a valid handle referencing the target Process
Successful End Condition	Target Process is maintaining a list of counts for each monitored API call in its memory space and its execution is not disrupted
Failed End Condition	Client discards the reference to the target Process
Primary Actors	Client, Process
Secondary Actors	None
Trigger	A new process is found when refreshing the running process list
Main Flow	1. Client creates an object to store information about the Process 2. Client loads a DLL into the address space of the target Process
Extensions	• None

Use Case Name	Detach Process
Related Requirements	FR2
Goal In Context	Client removes previously injected DLL from target process and receives collected data from the target process
Preconditions	Process is about to close or has reached a monitoring time limit
Successful End Condition	Client receives data from the target Process
Failed End Condition	Client discards Process and logs error
Primary Actors	Client, Process
Secondary Actors	None
Trigger	Process attempts to close or reaches monitor time limit
Main Flow	1. Client receives a message from the Client containing its counted API calls 2. Client creates a new data point object 3. Client adds data point object to send queue 4. Client removes object tracking the Process 5. Process closes normally
Extensions	• None

Use Case Name	Verify Client
Related Requirements	FR7, NF3
Goal In Context	Server verifies Client sending a request before accepting data
Preconditions	Server is running and maintaining a table of API keys corresponding to individual Clients
Successful End Condition	API key is valid and the Client request is accepted
Failed End Condition	API key is invalid and the request is rejected
Primary Actors	Client, Server
Secondary Actors	Database
Trigger	Client makes a request that requires authentication
Main Flow	1. Server compares the passed API key to the corresponding key stored in the Database 2. Allows request if the keys match
Extensions	• None

Use Case Name	Process Data Point
Related Requirements	FR1, FR3, FR4
Goal In Context	Server stores a new data point after accepting a post data point request from a Client
Preconditions	Server accepted a post data point request
Successful End Condition	New data point is stored on the Database
Failed End Condition	Error is logged
Primary Actors	Client, Server
Secondary Actors	Database
Trigger	Server accepts a post data point request
Main Flow	1. Server checks that the data point is in the correct format 2. Server creates a new data point object 3. Server pushes new data point object to Database
Extensions	• None

Use Case Name	Run Data Analysis
Related Requirements	FR5
Goal In Context	Server accepts a request to start a new data analysis job
Preconditions	Request contained a valid API key and the request was valid
Successful End Condition	Data analysis job is started and a UUID corresponding to the job is returned
Failed End Condition	Error is logged and an error message is returned
Primary Actors	Server
Secondary Actors	Database
Trigger	Server receives a run data analysis request through its API
Main Flow	1. New Analysis Job object is created using the passed parameters 2. Server pushes the new Analysis Job to the Database 3. Server adds new job to the job queue 4. Server returns the UUID of the new job in its response
Extensions	• None

Use Case Name	Get Analysis Result
Related Requirements	FR5
Goal In Context	Server accepts a request for information about a analysis job
Preconditions	Request contained a valid UUID corresponding to a analysis job started on the server
Successful End Condition	Information is returned about the analysis job
Failed End Condition	Error is logged and an error message is returned in the response
Primary Actors	Server
Secondary Actors	Database
Trigger	Server receives a get analysis result request
Main Flow	1. Server looks finds Analysis Job object by the passed UUID 2. Server returns the Analysis Job object information in its response
Extensions	• None

3.5 Use Case Diagrams

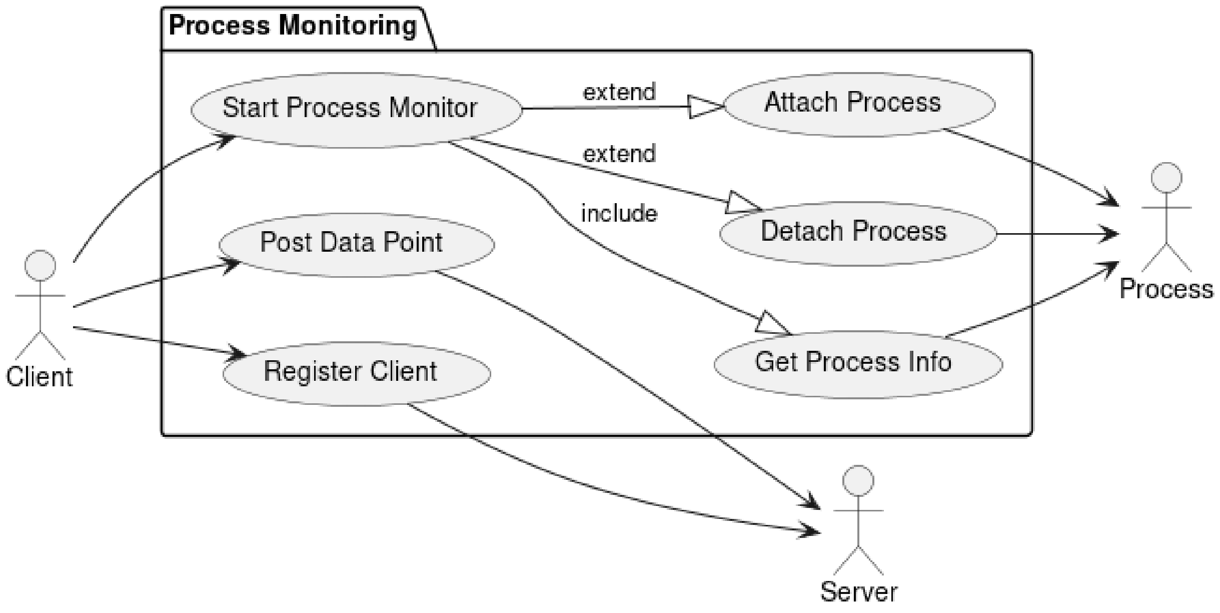

Figure 1: Use case diagram for Client

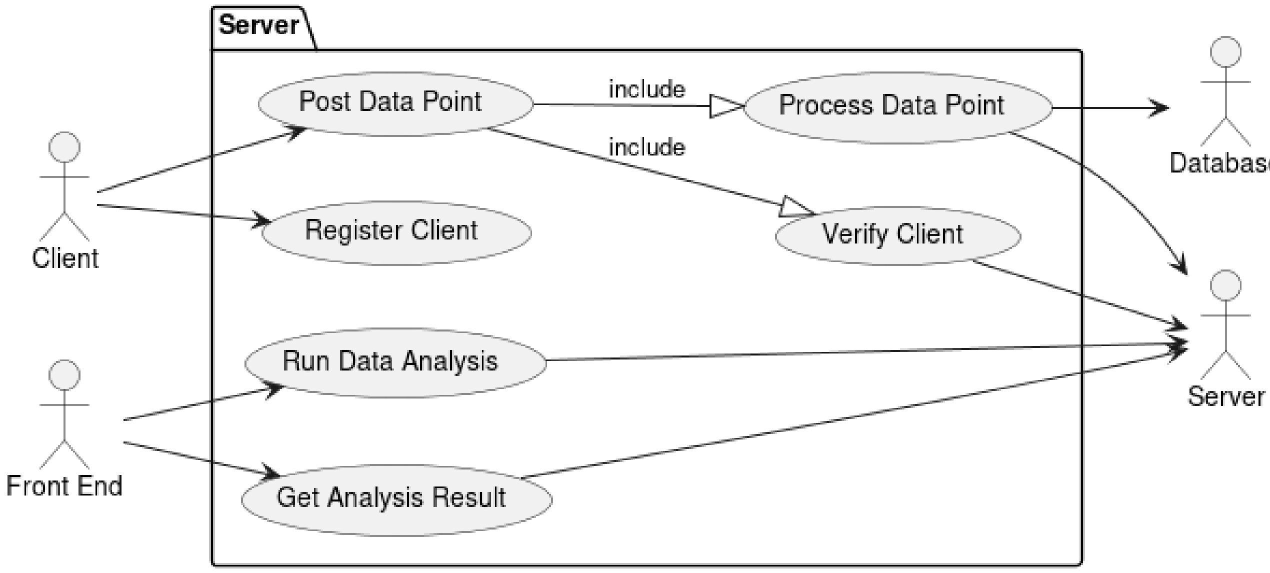

Figure 2: Use case diagram for Server

3.6 Class Diagrams

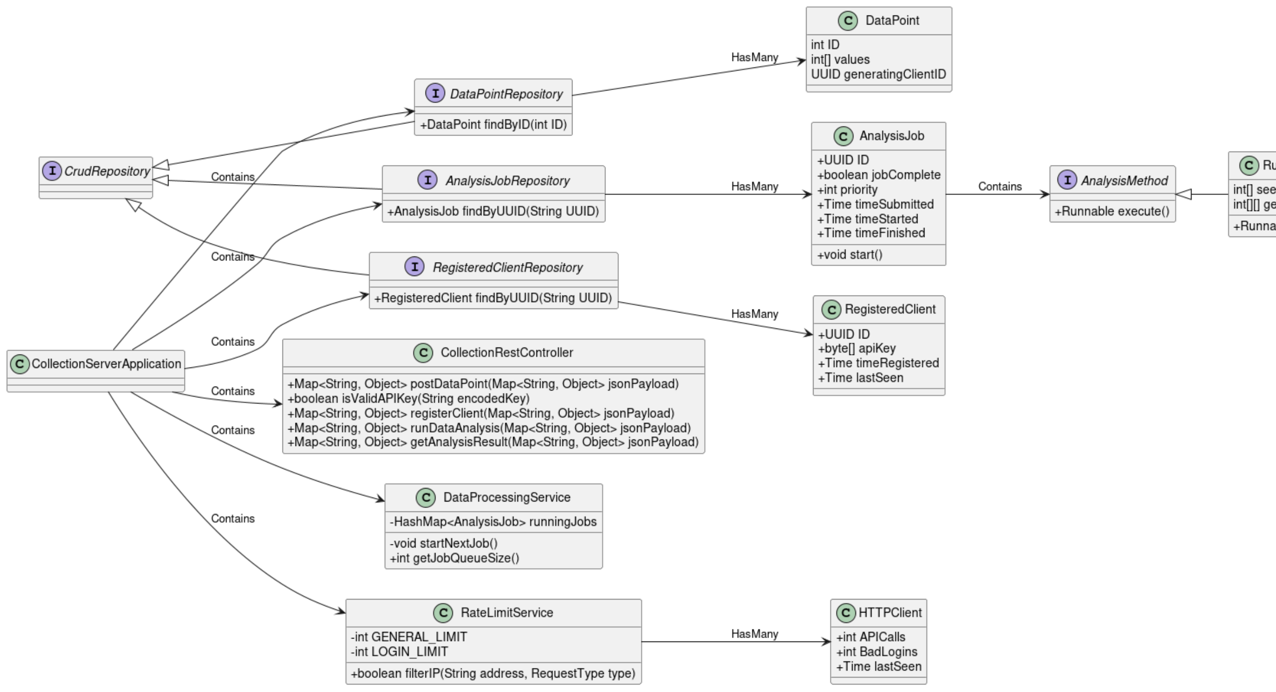
DataPoint
int ID
int[] values
UUID generatingClientID
DataPointRepository
+DataPoint findByID(int ID)
HasMany
CrudRepository
Contains
AnalysisJobRepository
+AnalysisJob findByUUID(String UUID)
AnalysisJob
+UUID ID
+boolean jobComplete
+int priority
+Time timeSubmitted
+Time timeStarted
+Time timeFinished
+void start()
AnalysisMethod
+Runnable execute()
RegisteredClientRepository
+RegisteredClient findByUUID(String UUID)
RegisteredClient
+UUID ID
+byte[] apiKey
+Time timeRegistered
+Time lastSeen
CollectionServerApplication
CollectionRestController
+Map<String, Object> postDataPoint(Map<String, Object> jsonPayload)
+boolean isValidAPIKey(String encodedKey)
+Map<String, Object> registerClient(Map<String, Object> jsonPayload)
+Map<String, Object> runDataAnalysis(Map<String, Object> jsonPayload)
+Map<String, Object> getAnalysisResult(Map<String, Object> jsonPayload)
DataProcessingService
-HashMap<AnalysisJob> runningJobs
-void startNextJob()
+int getJobQueueSize()
RateLimitService
-int GENERAL_LIMIT
-int LOGIN_LIMIT
+boolean filterIP(String address, RequestType type)
HTTPClient
+int APICalls
+int BadLogins
+Time lastSeen

Figure 3: Class diagram for Server

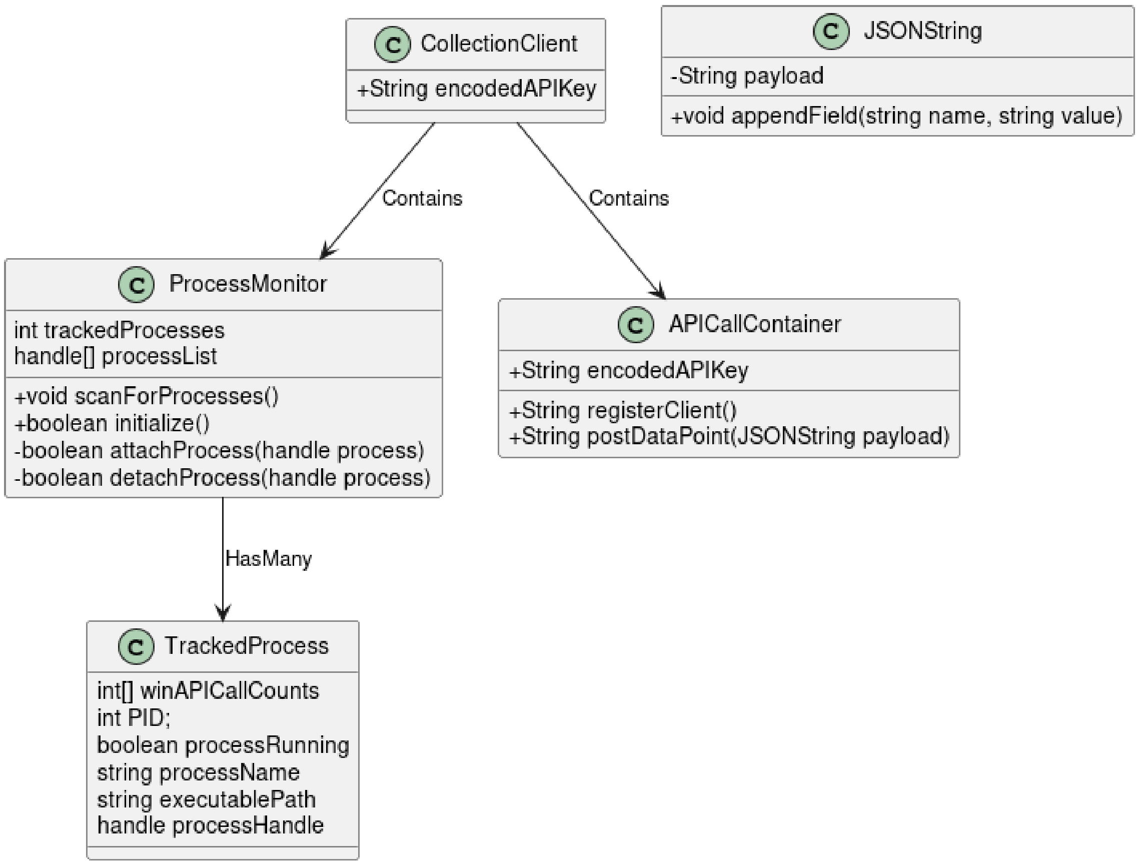

Figure 4: Class diagram for Client

3.7 Sequence Diagrams

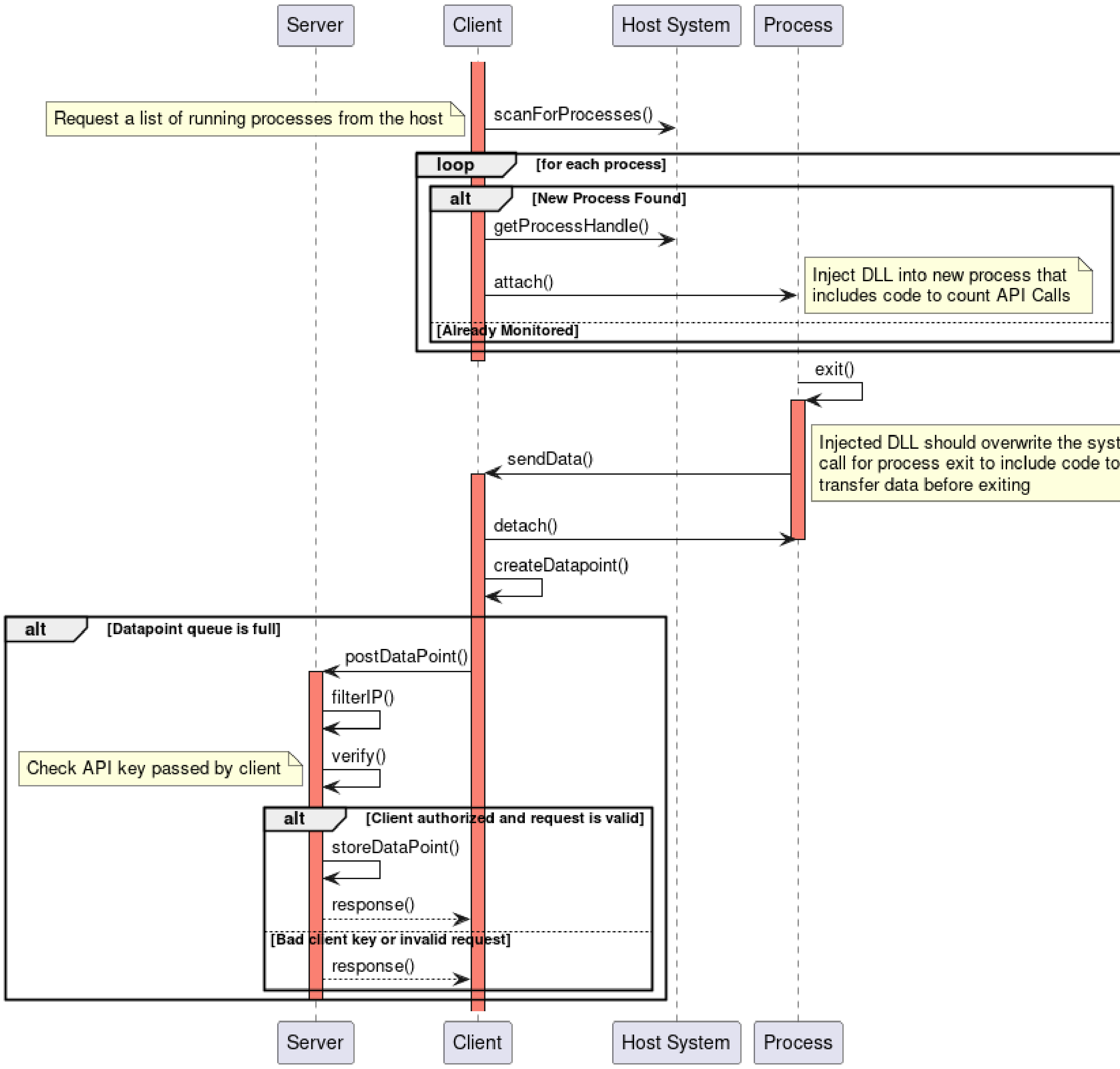
Server
Client
Host System
Process
Request a list of running processes from the host
scanForProcesses()
loop [for each process]
alt [New Process Found]
getProcessHandle()
attach()
Inject DLL into new process that includes code to count API Calls
[Already Monitored]
exit()
Injected DLL should overwrite the syst call for process exit to include code to transfer data before exiting
sendData()
detach()
createDatapoint()
alt [Datapoint queue is full]
postDataPoint()
filterIP()
Check API key passed by client
verify()
alt [Client authorized and request is valid]
storeDataPoint()
response()
[Bad client key or invalid request]
response()
Server
Client
Host System
Process

Figure 5: Sequence diagram for data collection

4 Architecture

4.1 Architecture Diagram

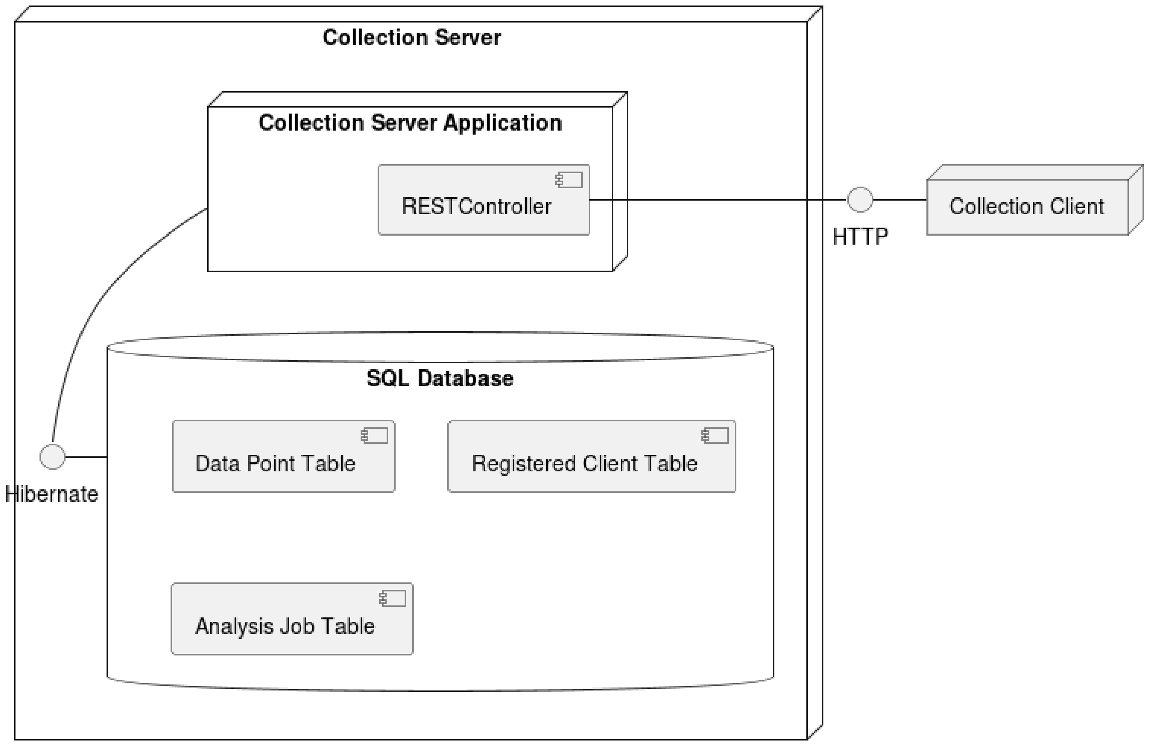

Figure 6: Drawing of system architecture

4.2 Architecture Description

The system uses a Client-Server architecture. This architecture fit best since the system needs to service an arbitrary number of clients running the client software. The server will also implement a REST API to handle requests from the client software and the front end web page. The database will also not be exposed to the client or web front end. Any operations on the database will have to be done through API calls to the server.

5 Conclusion

I have previous experience in developing back end servers and deploying them. The main technical challenge with this project will likely be the client software. This is my first time doing Windows development. I have made some progress with an early prototype of the client software but there are several topics I will need to read into including Windows threads, DLL files, and Windows process message passing. The main motivation of this project is to create an interesting high-dimensional data set to test the Sow and Grow algorithm with, but if the system is able to consistently identify malware it could have more practical applications.

6 References

[1] Ivo Ivanov. *API hooking revealed.* https://www.codeproject.com/Articles/2082/API-hooking-revealed. 2002.

[2] Microsoft. *Microsoft Research Detours Package Overview.* https://github.com/microsoft/Detours/wiki. 2020.

[3] Andrew Steane. *Quick Introduction to Windows API.* https://users.physics.ox.ac.uk/ Steane/cpp$_h$*elp/wind*
2009.