C**木马程序**

本项目实现以下功能：

* 主控端连接服务器

只要服务器在运行，主控端就能够通过指定服务器 ip 地址来连接服务器。一旦建立连接，就能发送指令与之交互。

* 服务器控制受控端

服务器运行过程中，会接受受控端的请求并与之建立连接。主控端发送attack攻击指令，服务器就会发送控制指令给受控端，受控端接收到后会发送确认指令，并执行攻击任务。

* 受控端发动攻击

受控端一旦在宿主机上运行，就会设置自动化任务重启自身（每分钟）。一旦服务器启动，最多一分钟就能与服务器建立连接，接受服务器的控制，发动攻击。

# **组成**

## 3个**子**项目

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 路径 |
| controller | 主控端 | ./project/controller |
| controllee | 受控端 | ./project/controllee |
| server | 服务器 | ./project/server |

## 5个第三方库

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 路径 |
| argparse | 用于转换命令行参数 | ./library/argparse |
| chalk.c | 用于在输出中带有色彩 | ./library/chalk.c |
| libhv | 用于方便地创建网络主机 | ./library/libhv |
| SHA256 | 用于计算密码的SHA256数值 | ./library/SHA256 |
| libevent | 事件循环库，处理事件 | 通过 apt 安装 |

## **3个自开发库**

|  |  |  |
| --- | --- | --- |
| 名称 | 说明 | 路径 |
| option | Rust Option类型的C近似实现 | ./library/lib/option.h |
| result | Rust Result 类型的 C 近似实现 | ./library/lib/result.h |
| util | 其他好用的工具函数 | ./library/lib/util.[ch] |

## 1个**应用层协议定义（./library/protocol）**

定义了本项目的应用层协议相关宏和工具函数。

# **核心内容**

## controller

|  |  |  |  |
| --- | --- | --- | --- |
| 文件 | 名称 | 种类 | 说明 |
| main.c | on\_sigint() | 函数 | 接收到SIGINT信号的回调函数。执行关闭已有连接的动作。 |
| main() | 函数 | 主函数，先执行 init 初始化，再添加 on\_sigint 回调函数，最后执行主事件循环来与服务器建立连接、交换信息。 |
| account/account.c | user\_name | 字符数组 | 用户输入的用户名存放在这里 |
| user\_password | 字符数组 | 用户输入的密码存放在这里 |
| account\_init() | 函数 | 从控制台获取用户名和密码，并存放到user\_name 和 user\_password 字符数组里面。 |
| config/config.c | config | Config对象 | 存放与程序配置相关的信息，包括服务器ip地址和服务器端口。 |
| config\_init() | 函数 | 初始化 config 对象 |
| config\_report() | 函数 | 输出config对象的信息 |
| hios/hios.c | sockio | hio\_t对象 | 套接字的libhv事件循环对象，存放与服务器连接的套接字的文件描述符 |
| stdinio | hio\_t 对象 | 标准输入流的libhv事件循环对象，存放标准输入流的文件描述符 |
| hloop | 主事件循环对象 | sockio和stdinio的事件回调都在这个事件循环中触发和调用 |
| on\_connect() | 函数 | 与服务器建立连接的回调函数。先用 hio\_read开启读流，然后向服务器连续地发送用户名与用户密码。 |
| on\_close() | 函数 | 与服务器的连接关闭的回调函数。输出一条信息并关闭标准输入流。 |
| on\_recv() | 函数 | 接收服务器消息的回调函数。先原封不动输出服务器消息，若服务器消息以协议规定的分隔符结尾，则输出"clr"提示用户输入下一条命令。 |
| on\_stdin() | 函数 | 从标准输入流读取到数据的回调函数。将输入原封不动转发给服务器。 |
| hios\_init() | 函数 | 初始化hloop、sockio和stdinio对象，为sockio设置on\_connect、on\_close、on\_recv回调，为stdinio设置on\_stdin回调 |
| init/init.c | sip\_str | 字符串 | 指向用户从命令行参数输入的服务器ip字符串 |
| sport\_str | 字符串 | 指向用户从命令行参数输入的服务器端口字符串 |
| options | argparse\_option对象数组 | 描述clr可执行文件的详细用法 |
| usages | 字符串数组 | 描述clr可执行文件的大致用法 |
| argparse | argparse对象 | 用于执行命令行参数的转换 |
| init\_account() | 函数 | 调用 account\_init函数，并检查是否成功 |
| init\_config() | 函数 | 转换命令行参数，调用 config\_init函数，并检查结果 |
| init\_hio() | 函数 | 调用hios\_init函数，并检查结果 |
| init() | 函数 | 用 argparse 转换命令行参数，调用 init\_config、init\_account、init\_hio，并检查结果 |

# controllee

|  |  |  |  |
| --- | --- | --- | --- |
| 文件 | 名称 | 种类 | 说明 |
| main.c | on\_sigint() | 函数 | 接收到 SIGINT的回调函数 |
|  | main() | 函数 | 先调用init函数，再开启主事件循环 |
| hios/hios.c | sockio | hio\_t 对象 | 与服务器连接的 libhv hio\_t对象。处理相关事件回调 |
| hloop | hloop\_t 对象 | 全局主事件循环对象 |
| timer\_close() | 函数 | 定时关闭的回调函数，关闭与服务器的连接 |
| on\_connect() | 函数 | 与服务器建立连接的回调函数，设置关闭回调并开启读流 |
| on\_recv() | 函数 | 从服务器接受消息的回调函数，当指令为 \x01时发送确认消息，并执行攻击动作。 |
| hios\_init() | 函数 | 初始化hloop和sockio，设置相关回调函数。 |
| init/init.c | init\_crontab() | 函数 | 设置自动化任务，每分钟重启自身。 |
|  | init\_hio() | 函数 | 调用 hios\_init函数，并检查结果 |
|  | init() | 函数 | 调用 init\_hio和init\_crontab，并检查结果 |

## server

|  |  |  |  |
| --- | --- | --- | --- |
| 文件 | 名称 | 种类 | 说明 |
| mian.c | main() | 函数 | 调用 init函数并检查结果，然后调用 run 函数开始主事件循环 |
| config/config.c | config | Config对象 | 保存程序配置相关信息，包括主控服务器监听端口、受控服务器监听端口、用户名、用户密码的sha256数据 |
|  | config\_init() | 函数 | 初始化config对象 |
|  | config\_report() | 函数 | 输出config数据 |
| core/cmd.c | cmds | Cmd对象数组 | 保存服务器可接受的所有来自主控端的命令。 |
| on\_cmd\_help() | 函数 | help命令的回调函数 |
| on\_cmd\_quit() | 函数 | quit命令的回调函数 |
| on\_cmd\_list() | 函数 | list命令的回调函数 |
| on\_cmd\_attack() | 函数 | attack命令的回调函数 |
| on\_cmd\_undefined() | 函数 | 其他未知命令的回调函数 |
| attack\_id | int | 要攻击攻击的id |
| cmd\_init() | 函数 | 初始化cmds 对象，添加回调函数 |
| cmd\_exe() | 函数 | 根据 name、argc、argv执行cmds中的命令回调函数 |
| core/server.c | server | Server对象 | 保存两个SingleServer数据，一个SingleServer包含port（端口）、hloop（事件循环对象）、io（监听端口的套接字）和pthread（线程） |
| server\_m\_on\_accept() | 函数 | 当主控端请求建立时的回调函数 |
| server\_e\_on\_accept() | 函数 | 当受控端请求建立时的回调函数 |
| server\_init() | 函数 | 初始化server的两个SingleServer，即创建套接字，创建ssl上下文 |
| server\_mpthread\_routine() | 函数 | 主控端服务器线程运行的函数，执行主控端hloop的事件循环 |
| server\_epthread\_routine() | 函数 | 受控端服务器线程运行的函数，执行受控端hloop的事件循环 |
| server\_start() | 函数 | 启动以上两个线程例程 |
| server\_quit() | 函数 | 停止服务器线程例程，释放hloop内存 |
| core/session.c | session | Session对象 | 保存当前主控端会话id、会话建立状态、互斥锁和套接字 |
| session\_on\_close() | 函数 | 会话套接字关闭的回调函数 |
| session\_on\_read() | 函数 | 处理主控端请求的回调函数 |
| session\_on\_login() | 函数 | 当主控端成功登录的回调函数 |
| session\_id\_alloc() | 函数 | 为当前会话分配一个id |
| session\_occupied() | 函数 | 当前是否已经建立或正在建立一个会话 |
| session\_reset() | 函数 | 用来重设一些session的字段 |
| session\_init() | 函数 | 初始化 session对象，程序启动时调用一次 |
| session\_establish() | 函数 | 根据传入的套接字 hio\_t对象尝试建立一个会话 |
| core/victim.c | victims | VictimTable对象 | 保存所有受害者。VictimTable是一个前驱指针链表，一个节点包含前一个节点的 next 指针的地址，方便删除和插入操作。 |
| victims\_lock | pthread\_mutex\_t 对象 | victims的互斥锁 |
| victim\_new() | 函数 | 通过受控端连接来新建一个Victim链表节点。 |
| victim\_on\_close() | 函数 | 受控端连接关闭的回调函数，从victims链表中删除相关节点。 |
| victim\_on\_read() | 函数 | 受控端发送消息的回调函数。通常是受控端接收指令的确认消息 |
| victim\_free() | 函数 | 释放victim链表的内存 |
| victims\_init() | 函数 | 初始化victims指针以及初始化victims\_lock |
| victims\_add() | 函数 | 根据受控端连接来新建一个Victim节点，并插入到victims链表中 |
| victim\_get\_by\_io\_ptr() | 函数 | 根据hio\_t \*io的值来找到Victim节点 |
| victim\_get\_by\_id() | 函数 | 根据受控端id的值来找到Victim节点 |
| victims\_quit() | 函数 | 释放victims的内存，并摧毁victims\_lock |
| event/event.c | global\_events | GlobalEvents对象 | 保存全局事件循环base，以及所有的全局事件event\_t对象 |
| init\_event\_cb() | 函数 | 当init\_event触发时的回调函数 |
| sigint\_event\_cb() | 函数 | 当接受到 SIGINT 信号的回调函数 |
| global\_event\_loop() | 函数 | 启动全局事件循环 |
| global\_events\_init() | 函数 | 初始化global\_events对象，初始化相关event\_base和event\_t |
| global\_events\_trigger\_init\_event() | 函数 | 触发init\_event |
| global\_events\_quit() | 函数 | 释放global\_events相关内存资源 |
| init/init.c | uname | 字符串 | 用户名 |
| psw | 字符串 | 用户密码 |
| eport\_str | 字符串 | 受控端服务器端口字符串 |
| mport\_str | 字符串 | 主控端服务器端口字符串 |
| options | argparse\_option 数组 | 保存应用程序的详细用法 |
| usages | 字符串数组 | 保存应用程序的大概用法 |
| argparse | argparse对象 | 用于执行命令行参数转换 |
| init\_victims() | 函数 | 调用victims\_init函数并检查返回值 |
| init\_cmds() | 函数 | 调用cmd\_init函数并检查返回值 |
| init\_session() | 函数 | 调用session\_init并检查返回值 |
| init\_server() | 函数 | 调用server\_init并检查返回值 |
| init\_events() | 函数 | 调用global\_events\_init并检查返回值 |
| init\_config() | 函数 | 将uname等配置数据转换为合适的数据并传给config\_init函数，并检查返回值 |
| init() | 函数 | 调用init\_events、init\_config、init\_server、init\_session、init\_cmds、init\_victims并检查返回值 |
| quit/quit.c | quit() | 函数 | 调用session\_quit、global\_events\_quit、server\_quit、victims\_quit，并输出正确退出的信息 |
| run/run.c | run() | 函数 | 调用global\_event\_loop，执行主事件循环 |
| util/getip.c | getip() | 函数 | 获取子网ip |
| util/printlog.c | event2\_print\_log() | 函数 | 根据严重等级打印log信息 |
| util/string.c | split() | 函数 | 将字符串根据空白分段 |
| myatoi() | 函数 | 将字符串转换为整数，并返回转换结果 |

# **运行过程**

### controller

调用init初始化 -> 初始化成功 -> 输出配置信息 -> 尝试连接服务器 -> 连接成功 -> 尝试建立会话 -> 建立成功 -> 将用户输入的命令转发给服务器 -> 服务器关闭连接 -> 退出

-> 初始化失败 -> 退出

-> 建立失败 -> 退出

-> 连接失败 -> 退出

### **controllee**

调用init初始化（添加自动化任务重启） -> 初始化成功 -> 尝试连接服务器 -> 连接成功 -> 等待服务器发送攻击指令 -> 发动攻击 -> 目标机器关机

-> 连接失败 -> 退出

-> 初始化失败 -> 退出

### server

调用init初始化 -> 初始化成功 -> 开启主事件循环 -> 启动主控端服务器和受控端服务器线程 -> 主控端服务器监听主控端请求，并提供服务

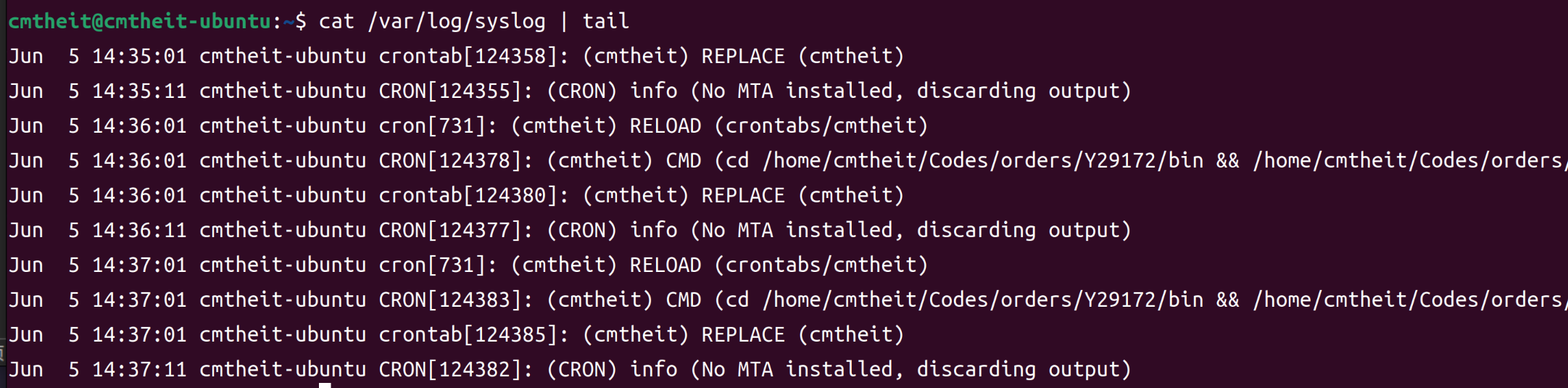
-> 受控端服务器监听受控端请求，并发起指令

-> 初始化失败 -> 打印错误信息并退出

# **受控端痕迹擦除原理**

受控端 cle 程序在运行时会首先设置一个自动化任务，具体是通过linux crontab 命令。输入 crontab -l 会输出当前用户设置的自动化任务。

然而自动化任务的设置和替换会导致在 /var/log/syslog 中留下日志。输入 cat /var/log/syslog | tail命令，可以看到具体如下：



上图中的 "REPLACE" 表示自动化任务被替换，这就可能使得宿主发现我们这一入侵行为。因此我们在自动化任务中添加一条删除日志的命令 *sudo /bin/sed -i /cron/Id /var/log/syslog*，这样就看不到crontab留下的日志了。（注意，要用 sudo ./bin/cle，否则删除日志的命令无法生效）。