The CostaRicto Campaign: Cyber-Espionage Outsourced

blogs.blackberry.com/en/2020/11/the-costaricto-campaign-cyber-espionage-outsourced

The BlackBerry Research & Intelligence Team

With the undeniable success of Ransomware-as-a-Service (RaaS), the cybercriminal market has expanded its portfolio to add dedicated phishing and espionage campaigns to the list of illicit services on offer…

During the past six months, the BlackBerry Research and Intelligence team have been monitoring a cyber-espionage campaign that is targeting disparate victims around the globe. The campaign, dubbed CostaRicto by BlackBerry, appears to be operated by "hackers-for-hire", a group of APT mercenaries who possess bespoke malware tooling and complex VPN proxy and SSH tunnelling capabilities.
Mercenary groups offering APT-style attacks are becoming more and more popular. Their tactics, techniques, and procedures (TTPs) often resemble highly sophisticated state-sponsored campaigns, but the profiles and geography of their victims are far too diverse to be aligned with a single bad actor's interests.

Although in theory the customers of a mercenary APT might include anyone who can afford it, the more sophisticated actors will naturally choose to work with patrons of the highest profile – be it large organizations, influential individuals, or even governments. Having a lot at stake, the cybercriminals must choose very carefully when selecting their commissions to avoid the risk of being exposed.

Outsourcing an espionage campaign, or part of it, to a mercenary group might be very compelling, especially to businesses and individuals who seek intelligence on their competition yet may not have the required tooling, infrastructure and experience to conduct an attack themselves. But even notorious adversaries experienced in cyber-espionage can benefit from adding a layer of indirection to their attacks. By using a mercenary as their proxy, the real attacker can better protect their identity and thwart attempts at attribution.

Key Findings:

- CostaRicto targets are scattered across different countries in Europe, Americas, Asia, Australia and Africa, but the biggest concentration appears to be in South Asia (especially India, Bangladesh and Singapore), suggesting that the threat actor could be based in that region, but working on a wide range of commissions from diverse clients.

- The command-and-control (C2) servers are managed via Tor and/or through a layer of proxies; a complex network of SSH tunnels are also established in the victim's environment. These practices reveal better-than-average operation security.

- The backdoor used as a foothold is a new strain of never-before-seen malware – a custom-built tool with a suggestive project name, well-structured code, and detailed versioning system. The earliest timestamps are from October 2019, and based on the version numbers, the project appears to be in the debug testing phase. It's not clear as of now if it's something that the threat actors developed in-house or obtained for exclusive use as part of beta testing from another entity.

- The timestamps of payload stagers go back to 2017, which might suggest the operation itself has been going on for a while, but used to deliver a different payload. It's not impossible, though, that the stagers are simply being reused without recompilation (i.e.: by changing the C2 URLs via binary editing).

- The backdoor project is called Sombra, which is a reference to an Overwatch game persona – an agent of the antagonist organization, who specializes in espionage and intelligence assessment and is characterized by stealth, infiltration and hacking skills.

- Some of the domain names hardcoded in the backdoor binaries seem to spoof legitimate domains (e.g.: the malicious domain sbibd[.]net spoofing a legitimate domain of the State Bank of India Bangladesh, sbibd.com). However, victims affected by these backdoors are unrelated, suggesting reuse of existing infrastructure which served another purpose.

- One of the IP addresses which the backdoor domains were registered to overlaps with an earlier phishing campaign attributed to APT28 (i.e.: according to RiskIQ data, the SombRAT domain akams[.]in was at the time of attack registered to the same IP address as the phishing domain mail.kub-gas[.]com). However, BlackBerry researchers believe that a direct link between CostaRicto and APT28 is highly unlikely. It might be that the IP overlap is coincidental, or – just as plausible – that the earlier phishing campaigns have been outsourced to the mercenary on behalf of the actual threat actor.

Targeting

Unlike most of the state-sponsored APT actors, the CostaRicto adversary seems to be indiscriminate when it comes to the victims' geography. Their targets are located in numerous countries across the globe with just a slight concentration in the South-Asian region:

- India
- Bangladesh
- Singapore
- China
- U.S.
- Bahamas
- Australia
- Mozambique
- France
- Netherlands

- Austria
- Portugal
- Czechia

The victims' profiles are diverse across several verticals, with a large portion being financial institutions.

Delivery

After gaining access to the victim's environment (presumably by using stolen credentials, either obtained via phishing, or bought on the dark web), the attacker sets up remote tunnelling using a SSH tool. The tool is configured to redirect traffic from a malicious domain to a proxy that is listening on a local port. The tunnel is authenticated using the attacker's private key.

In order to pull down the backdoor, a payload stager, either HTTP or reverse-DNS, is executed with the use of a scheduled task.

The backdoor comes either wrapped up in a PowerSploit reflective loader, or in the form of a custom-built dropper that uses a simple virtual machine (VM) mechanism to decode and inject the payload.

Toolset

- SombRAT: A custom backdoor (with both x86 and x64 versions)
- CostaBricks: A custom VM-based payload loader (seen only with x86 SombRAT payloads so far)
- PowerSploit's reflective PE injection module (seen with x64 SombRAT payloads)
- HTTP and reverse-DNS payload stagers
- nmap: Port scanner
- PsExec

PS1 Loader (x64)

The 64-bit backdoor is deployed in a fairly standard way. It is distributed as a set of scripts and encrypted files and utilizes a PowerShell loader based on the Invoke-ReflectivePEInjection PowerSploit module to decode and inject the final payload DLL into memory:

File Name	Function
autorun.bat	Obfuscated batch script that sets PowerShell execution policy to unrestricted and executes autorun.ps1
autorun.ps1	Obfuscated PowerShell script that decodes and executes another PowerShell loader stored in ntuser.c file
ntuser.a	XOR key used to decode the PowerShell loader and payload binary
ntuser.b	XOR encoded payload binary
ntuser.c	XOR encoded Invoke-ReflectivePEInjection module, modified to add payload decryption routine

CostaBricks Loader (x86)

The loader used with 32-bit backdoors is more technically compelling. It implements a simple custom-built virtual machine mechanism that will execute an embedded bytecode to decode and inject the payload into memory.

This attempt at obfuscation, although not new, is rather uncommon in relation to targeted attacks. Code virtualization has been most prevalent in commercial software protectors which use much more advanced solutions; simpler virtual machines are sometimes also featured in off-the-shelf malicious packers used by widespread financial crimeware. This particular implementation, however, is unique (there are just a handful of samples in the public domain) and seems to be used only with SombRAT payloads – which makes us believe it is a custom-built tool that is private to the attackers.

To further confuse anti-malware solutions, the loader contains the entire unobfuscated code of a legitimate open source application called Blink (https://github.com/crosire/blink), which never gets executed:

.rdata:00261...	00000003	C	-a
.rdata:00261...	00000022	C	Enter PID of target application:
.rdata:00261...	0000002B	C	Failed to open target application process!
.rdata:00261...	0000004F	C	Machine architecture mismatch between target application and this application!
.rdata:00261...	00000024	C	Launching in target application ...
.rdata:00261...	00000029	C	Failed to create new communication pipe!

Figure 1: Strings belonging to Blink code

There is also an unused zlib decompression routine that seems to be leftover code from an older version of the loader.

The compilation timestamps suggest that both the loader and the embedded payload are compiled at the same time (with only a few seconds difference).

One of the loaders had the following PDB path, suggesting that the internal name of the project is CostaRicto/ CostaBricks:

```
.rdata:00472B58 ; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
.rdata:00472B58 asc_472B58      db 'RSDS'               ; DATA XREF: .rdata:00471F74â†'o
.rdata:00472B58                                         ; CV signature
.rdata:00472B5C                 dd 0A94A6088h           ; Data1 ; GUID
.rdata:00472B5C                 dw 494h                 ; Data2
.rdata:00472B5C                 dw 4BE7h                ; Data3
.rdata:00472B5C                 db 86h, 9Dh, 18h, 11h, 6Bh, 22h, 4Ah, 0FCh; Data4
.rdata:00472B6C                 dd 1                    ; Age
.rdata:00472B70                 db 'C:\Wokrflow\CostaRicto\Release\CostaBricks.pdb',0 ; PdbFileName
```

Figure 2: PDB path from one of the x86 loader samples

Virtual Machine Internals

The virtual machine mechanism is implemented with the usage of C++ objects and classes. There are 20 different VM instructions, each having between zero and three operands. A pointer to the bytecode to execute is passed as a parameter to the VM initialization routine:

```
.text:00223EC3        mov     esi, esp
.text:00223EC5        mov     [esi+vm_stack.native_ebp], ebp
.text:00223ECB        lea     edi, [esi+vm_stack.native_seh_frame]
.text:00223ED1        lea     ecx, [esi+vm_stack.VMBASERUNNER]
.text:00223ED4        mov     [edi-4], esp                          ; native_esp
.text:00223ED7        mov     [edi+native_stack.retval], -1
.text:00223EDE        mov     [edi+native_stack.frame_handler], offset cxx_frame_handler
.text:00223EE5        mov     eax, large fs:0
.text:00223EEB        mov     [edi+native_stack.seh_frame], eax
.text:00223EED        mov     large fs:0, edi
.text:00223EF4        call    VMBASERUNNER_constr
.text:00223EF9        lea     ecx, [esi+vm_stack.vmbytecode_cmem]
.text:00223EFC        xor     eax, eax
.text:00223EFE        mov     [ecx+vm_stackparam.cmem_vftable], offset ??_7CMemory@@6B@ ; const CMemory::`vftable'
.text:00223F04        mov     [ecx+vm_stackparam.data], eax
.text:00223F07        mov     [ecx+vm_stackparam.len], eax
.text:00223F0A        mov     [edi+native_stack.retval], eax
.text:00223F0D        push    78356                                 ; Size
.text:00223F12        push    offset vmbytecode                     ; bytecode to be executed by the VM
.text:00223F17        call    insert_or_replace_element
.text:00223F1C        lea     ecx, [esi+vm_stack.encpayload_cmem]
.text:00223F1F        xor     eax, eax
.text:00223F21        mov     [esi+vm_stack.native_retval], 1
.text:00223F2B        mov     [ecx+vm_stackparam.cmem_vftable], offset ??_7CMemory@@6B@ ; const CMemory::`vftable'
.text:00223F31        mov     [ecx+vm_stackparam.data], eax
.text:00223F34        mov     [ecx+vm_stackparam.len], eax
.text:00223F37        push    130048                                ; Size
.text:00223F3C        push    offset enc_payload                    ; encrypted UPX-packed EXE
.text:00223F41        call    insert_or_replace_element
.text:00223F46        lea     eax, [esi+vm_stack.encpayload_cmem]
.text:00223F49        lea     ecx, [esi+vm_stack.VMBASERUNNER]
.text:00223F4C        push    eax
.text:00223F4D        lea     eax, [esi+vm_stack.vmbytecode_cmem]
.text:00223F50        push    eax
.text:00223F51        call    init_vm_decode_call_payload
```

Figure 3: Initialization of the virtual machine

A VM instance is initialized by setting its context structure, which contains the instruction pointer, zero flag, instructions list and pointer to the registers:

Offset	Field	Description
0x00	Instruction pointer	Index of the bytecode instruction to execute
0x04	Zero flag	Used for conditional jumps
0x08	Instructions_list.first	Points to the first instruction in the list
0x0C	Instructions_list.current	Points to the current instruction in the list
0x10	Instructions_list.next	Points to the next instruction in the list
0x14	Registers pointer	Points to the list of registers
0x18	Registers count	Incremented when new register is allocated

Instructions and Operands

Instructions, operands, and opcode handlers are implemented as doubly linked lists. Each VM instruction has its own index and contains information such as the opcode number, flags, operands count, and the operands:

```
__opc_04_SUB___ <0, 4, 0, 2, 1, 0, 9435C739h, 0, 0, 1, 0, 9435C73Ah, 0, 0>
```
Figure 4: An example of VM instruction format for the SUB opcode

The operands can either be immediate values or "registers". Dynamically allocated “registers” are small memory regions organized in the form of dictionary objects in doubly linked list. Each register has its own unique index that can store up to 8 bytes of data (including pointers to larger memory buffers) and can be either read or written to.

If the operand metadata specifies the index value, the operand is a "register"; otherwise the operand contains an immediate value. The value (either immediate or pointed to by a "register") is an integer: qword by default, but different lengths (byte, word or double-word) can be specified in the metadata:

Offset	Field	Notes
0x00	Instruction index	Consecutive numbers starting with 0
0x04	Opcode	0 – 0x13 (19.)
0x06	Skip bool	If set, then the instruction will be ignored
0x08	Operands count	0 – 3
0x0C	Operand type	read (0) or write (1)
0x0E	Operand flag	Specifies length: 0x10 = byte, 0x20 = word, 0x40 = dword

0x10	Operand register index	Consecutive numbers starting with 0x9435C739
0x14	Operand value	Immediate value (if operand is not a register)
0x1C	Operand 2	Optional
0x2C	Operand 3	Optional

Opcodes

Each opcode has its own handler routine, which is executed in the main VM loop:

```
.text:00226AF4
.text:00226AF4 execute_instructions_loop:                          ; CODE XREF: execute_vm_bytecode+191â†“j
.text:00226AF4                 mov     ebx, [ebp+vm_context]
.text:00226AF7                 lea     edi, [edx+1]
.text:00226AFA                 mov     [ebx+vm_context.instr_pointer], edi ; increment instruction pointer
.text:00226AFC                 mov     ebx, [eax+edx*4]            ; pointer to the structure containing current vm instruction
.text:00226AFF                 cmp     [ebx+vm_instruction.skip_bool], 0
.text:00226B04                 jnz     short next
.text:00226B06                 mov     ecx, [ebp+VMBASERUNNER]     ; VMBASERUNNER.opcodes_list
.text:00226B09                 lea     eax, [ebx+vm_instruction.opcode]
.text:00226B0C                 push    eax                         ; opcode
.text:00226B0D                 push    esi                         ; points to the last vm_instruction
.text:00226B0E                 call    set_or_get_opcode_handler
.text:00226B13                 mov     eax, [ebp+vm_instruction]   ; points to the returned opcode handler
.text:00226B16                 add     ebx, vm_instruction.operands_ptr
.text:00226B19                 push    ebx                         ; vm_instruction.operands_ptr
.text:00226B1A                 mov     ecx, [ebp+vm_context]
.text:00226B1D                 mov     ebx, ecx
.text:00226B1F                 push    ecx                         ; vm_context
.text:00226B20                 call    [eax+vm_opcode_handler.routine] ; EXECUTE INSTRUCTION
.text:00226B23                 add     esp, 8
.text:00226B26                 mov     edi, [ebx+vm_context.instr_pointer]
.text:00226B28                 mov     eax, [ebx+vm_context.instr_first]
.text:00226B2B                 mov     ecx, [ebx+vm_context.instr_current]
.text:00226B2E
.text:00226B2E next:                                               ; CODE XREF: execute_vm_bytecode+15Câ†‘j
.text:00226B2E                 mov     edx, ecx
.text:00226B30                 sub     edx, eax
.text:00226B32                 sar     edx, 2
.text:00226B35                 cmp     edi, edx
.text:00226B37                 mov     edx, edi
.text:00226B39                 jnz     short execute_instructions_loop
```

Figure 5: Loop processing VM instructions

The handler routine will check to see if the number and types of operands are valid, read operand values from VM “registers”, perform a specific action (arithmetic/byte operation, comparison, jump, API call), and save results to a destination “register”:

```
.text:00224A72                 mov     eax, [ecx+vm_operands_list.first]
.text:00224A74                 mov     ecx, [ecx+vm_operands_list.next]
.text:00224A77                 sub     ecx, eax
.text:00224A79                 cmp     ecx, 8                          ; must have two operands
.text:00224A7C                 jnz     ret_0
.text:00224A82                 mov     edi, [eax]
.text:00224A84                 cmp     [edi+vm_operand_1.is_writable], 1   ; operand_1 must be writable
.text:00224A88                 jnz     ret_0
.text:00224A8E                 mov     eax, [eax+vm_operands_list.next]
.text:00224A91                 mov     esi, [ebp+vm_context]
.text:00224A94                 movzx   ecx, [eax+vm_operand_2.writable]
.text:00224A97                 test    cx, cx
.text:00224A9A                 jz      short operand_2_immediate       ; operand_2 is an immediate value
.text:00224A9C                 cmp     cx, 1
.text:00224AA0                 jnz     ret_0
.text:00224AA6                 push    eax                             ; operand_2 (register)
.text:00224AA7                 push    esi
.text:00224AA8                 call    get_operand_value               ; return operand value in edx:eax
.text:00224AAD                 add     esp, 8
.text:00224AB0                 add     esi, vm_context.registers
.text:00224AB3                 mov     ebx, edx                        ; operand_2_value_h
.text:00224AB5                 add     edi, vm_operand_1.index
.text:00224AB8                 mov     [ebp+operand_2_value_l], eax
.text:00224ABB                 lea     eax, [ebp+var_18]
.text:00224ABE                 mov     ecx, esi
.text:00224AC0                 push    edi                             ; operand_1 (register)
.text:00224AC1                 mov     esi, eax
.text:00224AC3                 push    eax
.text:00224AC4                 call    set_or_get_register
.text:00224AC9                 mov     eax, [esi]                      ; operand_1_value
.text:00224ACB                 mov     ecx, [ebp+operand_2_value_l]
.text:00224ACE                 xor     [eax+vm_register.data_l], ecx   ; XOR value pointed by operand 1
.text:00224ACE                                                         ; with value pointed by operand_2
.text:00224AD1                 xor     [eax+vm_register.data_h], ebx
.text:00224AD4                 mov     bl, 1
.text:00224AD6                 jmp     short endp
```

Figure 6: XOR opcode handler routine

Opcode (hex)	Operands	Instruction	Description
0x00	dst, src	mov	Move from src (either immediate value or pointer/register) to register at dst. If no operands, this acts as a NOP instruction, used mostly as a label to jump to
0x01	dst, src	xor	Exclusive or dst with src, result pointed by dst
0x02	dst, src	add	Add src to dst, result pointed by dst
0x03	dst, src	and	And dst with src, result pointed by dst
0x04	dst, src	sub	Subtract src from dst, result pointed by dst
0x05	addr	call	Call address in operand 1 (can be immediate value or register)
0x06	-	ret	Return 1
0x07	mem_ptr, size	virtual_alloc	Allocate memory (call VirtualAlloc), size in operand 2, pointer returned in operand 1 (register)
0x08	mem_ptr	virtual_free	Free memory (VirtualFree), pointer in operand 1 (register)

0x09	dst, src, size	memmove	Source pointed by operand 2, destination pointed by operand 1, size in operand 3
0x0A	dst, src	cmp	Compare value at dst (register) with src (immediate or register value), set zero flag in VM context structure
0x0B	dst, src	alldiv	Dividend in operand 1 register, divisor in operand 2 (immediate or register), result in operand 1 register
0x0C	dst	jnz	If zero flag not set, jump to location specified by operand
0x0D	dst	jz	If zero flag set, jump to location specified by operand
0x0E	dst	jmp	Unconditional jump; set instruction pointer to the value of operand
0x0F	dll_handle, dll_name	load_library	Call LoadLibraryA, pointer to library name in operand 2 (register), handle to loaded library in operand 1 (register)
0x10	dll_handle, proc_name, api_address	get_proc_addr	Call GetProcAddress, pointer to DLL handle in operand 1, pointer to process name in operand 2, API address returned in operand 3 (all operands are registers)
0x11	-	exit_proc	Call ExitProcess(0)
0x12	dst, src	shr	Shift right (divide dst by src)
0x13	dst, src	shl	Shift left (multiply dst by src)

The Bytecode

All of the x86 loaders BlackBerry has seen thus far embed the exact same bytecode that is 1800 (0x708) lines long. Most of these 1800 instructions are superfluous (i.e.: have no influence on the code functionality) and were inserted there for obfuscation only.

The purpose of the bytecode is to decrypt the embedded payload, load it into memory reflectively and execute it:

```
.text:001FE310 virtalloc_16b   __opc_07_VALLOC <4D9h, 7, 0, 2, 1, 0, 9435CDCCh, 0, 0, 1, 0, 0, 10h, 0>        ; allocate mem buffer for decryption key
.text:001FE33C                 __opc_02_ADD___ <4DAh, 2, 0, 2, 1, 0, 9435CDCDh, 0, 0, 1, 0, 0, 1E72E0DBh, 0>
.text:001FE368                 __opc_04_SUB___ <4DBh, 4, 0, 2, 1, 0, 9435CDCEh, 0, 0, 1, 0, 0, 45E5FF18h, 0>
.text:001FE394                 __opc_03_AND___ <4DCh, 3, 0, 2, 1, 0, 9435CDCFh, 0, 0, 1, 0, 9435CDD0h, 0, 0>
.text:001FE3C0                 __opc_04_SUB___ <4DDh, 4, 0, 2, 1, 0, 9435CDD1h, 0, 0, 1, 0, 0, 2E007CCBh, 0>
.text:001FE3EC                 __opc_04_SUB___ <4DEh, 4, 0, 2, 1, 0, 9435CDD2h, 0, 0, 1, 0, 9435CDD3h, 0, 0>
.text:001FE418                 __opc_02_ADD___ <4DFh, 2, 0, 2, 1, 0, 9435CDD4h, 0, 0, 1, 0, 0, 1BB4DC1h, 0>
.text:001FE444                 __opc_03_AND___ <4E0h, 3, 0, 2, 1, 0, 9435CDD5h, 0, 0, 1, 0, 0, 64982E79h, 0>
.text:001FE470                 __opc_03_AND___ <4E1h, 3, 0, 2, 1, 0, 9435CDD6h, 0, 0, 1, 0, 0, 79FE8364h, 0>
.text:001FE49C                 __opc_00_MOV___ <4E2h, 0, 0, 2, 1, 0, 9435CDD7h, 0, 0, 1, 0, 9435CDCCh, 0, 0>    ; save key_ptr at 0x9435CDD7
.text:001FE4C8                 __opc_02_ADD___ <4E3h, 2, 0, 2, 1, 0, 9435CDD8h, 0, 0, 1, 0, 0, 246B4F71h, 0>
.text:001FE4F4                 __opc_03_AND___ <4E4h, 3, 0, 2, 1, 0, 9435CDD9h, 0, 0, 1, 0, 9435CDDAh, 0, 0>
.text:001FE520                 __opc_00_MOV___ <4E5h, 0, 0, 2, 0, 20h, 9435CDCCh, 0, 0, 1, 0, 0, 14820285h, 0>  ; key_1 = 0x14820285
.text:001FE54C                 __opc_02_ADD___ <4E6h, 2, 0, 2, 1, 0, 9435CDDBh, 0, 0, 1, 0, 9435CDDCh, 0, 0>
.text:001FE578                 __opc_02_ADD___ <4E7h, 2, 0, 2, 1, 0, 9435CDCCh, 0, 0, 1, 0, 0, 4, 0>            ; key_ptr += 4
.text:001FE5A4                 __opc_02_ADD___ <4E8h, 2, 0, 2, 1, 0, 9435CDDDh, 0, 0, 1, 0, 0, 0C4FF8A06h, 0>
.text:001FE5D0                 __opc_02_ADD___ <4E9h, 2, 0, 2, 1, 0, 9435CDDEh, 0, 0, 1, 0, 9435CDDFh, 0, 0>
.text:001FE5FC                 __opc_00_MOV___ <4EAh, 0, 0, 2, 1, 0, 9435CDE0h, 0, 0, 1, 0, 0, 94EA8FDh, 0>
.text:001FE628                 __opc_03_AND___ <4EBh, 3, 0, 2, 1, 0, 9435CDE1h, 0, 0, 1, 0, 0, 0E37B3CCFh, 0>
.text:001FE654                 __opc_03_AND___ <4ECh, 3, 0, 2, 1, 0, 9435CDE2h, 0, 0, 1, 0, 9435CDE3h, 0, 0>
.text:001FE680                 __opc_00_MOV___ <4EDh, 0, 0, 2, 1, 0, 9435CDE4h, 0, 0, 1, 0, 0, 264C8A75h, 0>
.text:001FE6AC                 __opc_02_ADD___ <4EEh, 2, 0, 2, 1, 0, 9435CDE5h, 0, 0, 1, 0, 0, 7FF58AFDh, 0>
.text:001FE6D8                 __opc_00_MOV___ <4EFh, 0, 0, 2, 1, 0, 9435CDE6h, 0, 0, 1, 0, 0, 0B9837DFAh, 0>
.text:001FE704                 __opc_03_AND___ <4F0h, 3, 0, 2, 1, 0, 9435CDE7h, 0, 0, 1, 0, 9435CDE8h, 0, 0>
.text:001FE730                 __opc_00_MOV___ <4F1h, 0, 0, 2, 0, 20h, 9435CDCCh, 0, 0, 1, 0, 0, 26820323h, 0>  ; key_2 = 0x26820323
.text:001FE75C                 __opc_04_SUB___ <4F2h, 4, 0, 2, 1, 0, 9435CDE9h, 0, 0, 1, 0, 0, 0C145E87Ch, 0>
.text:001FE788                 __opc_02_ADD___ <4F3h, 2, 0, 2, 1, 0, 9435CDEAh, 0, 0, 1, 0, 0, 190C02AAh, 0>
.text:001FE7B4                 __opc_03_AND___ <4F4h, 3, 0, 2, 1, 0, 9435CDEBh, 0, 0, 1, 0, 0, 709B117Bh, 0>
.text:001FE7E0                 __opc_04_SUB___ <4F5h, 4, 0, 2, 1, 0, 9435CDECh, 0, 0, 1, 0, 9435CDEDh, 0, 0>
.text:001FE80C                 __opc_04_SUB___ <4F6h, 4, 0, 2, 1, 0, 9435CDEEh, 0, 0, 1, 0, 0, 0B232670h, 0>
.text:001FE838                 __opc_00_MOV___ <4F7h, 0, 0, 2, 1, 0, 9435CDEFh, 0, 0, 1, 0, 0, 0A2C6D200h, 0>
.text:001FE864                 __opc_00_MOV___ <4F8h, 0, 0, 2, 1, 0, 9435CDF0h, 0, 0, 1, 0, 9435CDF1h, 0, 0>
.text:001FE890                 __opc_02_ADD___ <4F9h, 2, 0, 2, 1, 0, 9435CDCCh, 0, 0, 1, 0, 0, 4, 0>            ; key_ptr += 4
.text:001FE8BC                 __opc_02_ADD___ <4FAh, 2, 0, 2, 1, 0, 9435CDF2h, 0, 0, 1, 0, 9435CDF3h, 0, 0>
.text:001FE8E8                 __opc_03_AND___ <4FBh, 3, 0, 2, 1, 0, 9435CDF4h, 0, 0, 1, 0, 0, 0B29A1AC3h, 0>
.text:001FE914                 __opc_02_ADD___ <4FCh, 2, 0, 2, 1, 0, 9435CDF5h, 0, 0, 1, 0, 9435CDF6h, 0, 0>
.text:001FE940                 __opc_00_MOV___ <4FDh, 0, 0, 2, 0, 20h, 9435CDCCh, 0, 0, 1, 0, 0, 35223562h, 0>  ; key_3 = 0x35223562
.text:001FE96C                 __opc_04_SUB___ <4FEh, 4, 0, 2, 1, 0, 9435CDF7h, 0, 0, 1, 0, 0, 0D4C83F52h, 0>
.text:001FE998                 __opc_04_SUB___ <4FFh, 4, 0, 2, 1, 0, 9435CDF8h, 0, 0, 1, 0, 0, 576E1F49h, 0>
.text:001FE9C4                 __opc_04_SUB___ <500h, 4, 0, 2, 1, 0, 9435CDF9h, 0, 0, 1, 0, 0, 3FDFD28h, 0>
.text:001FE9F0                 __opc_00_MOV___ <501h, 0, 0, 2, 1, 0, 9435CDFAh, 0, 0, 1, 0, 9435CDFBh, 0, 0>
.text:001FEA1C                 __opc_02_ADD___ <502h, 2, 0, 2, 1, 0, 9435CDCCh, 0, 0, 1, 0, 0, 4, 0>            ; key_ptr += 4
.text:001FEA48                 __opc_02_ADD___ <503h, 2, 0, 2, 1, 0, 9435CDFCh, 0, 0, 1, 0, 0, 0AFEF542Ah, 0>
.text:001FEA74                 __opc_03_AND___ <504h, 3, 0, 2, 1, 0, 9435CDFDh, 0, 0, 1, 0, 0, 5A18B61Ah, 0>
.text:001FEAA0                 __opc_00_MOV___ <505h, 0, 0, 2, 0, 20h, 9435CDCCh, 0, 0, 1, 0, 0, 41256421h, 0>  ; key_4 = 0x41256421
```

Figure 7: A fragment of VM bytecode - setting the decryption key

The payload decryption routine uses a custom symmetric algorithm based on arithmetic and byte-shift instructions – a combination of SHL/SHR/SUB/ADD/XOR – with hardcoded keys.

These constant values are used in all x86 SombRAT droppers we've seen so far:

```
key_1 = 0x14820285
key_2 = 0x26820323
key_3 = 0x35223562
key_4 = 0x41256421
cst_1 = 0x61C88647
cst_2 = 0x9E3779B9
```

```
tmp_1a = encdw_1 << 4 & 0xffffffff
tmp_1b = encdw_1 >> 5 & 0xffffffff
tmp_1c = encdw_1 - cst_1 & 0xffffffff

tmp_2a = tmp_1a + key_3 & 0xffffffff
tmp_2b = tmp_1b + key_4 & 0xffffffff
tmp_3 = tmp_2a ^ tmp_1c
keydw_2 = tmp_3 ^ tmp_2b

decdw_2 = encdw_2 - keydw_2

magic_1 = decdw_2 << 4 & 0xffffffff
magic_2 = decdw_2 >> 5 & 0xffffffff

key_1a = key_1 + magic_1 & 0xffffffff
key_2a = key_2 + magic_2 & 0xffffffff
cst_2a = cst_2 + decdw_2 & 0xffffffff

tmp_5 = key_1a ^ cst_2a
keydw_1 = tmp_5 ^ key_2a

decdw_1 = encdw_1 - keydw_1 & 0xffffffff
```

Figure 8: Payload decoding algorithm

SombRAT Backdoor

The backdoor delivered by the above-mentioned loaders is a C++ compiled executable developed with heavy usage of objects, classes, and interfaces. It has a plugin architecture and basic functionality of a foothold RAT that is mainly used to download and execute other malicious payloads – either as its own plugins or standalone binaries. It can also perform other simple actions, like collecting system information, listing and killing processes, and uploading files to the C2.

Features:

- Communication over DNS tunnel with a hardcoded domain name and DGA-generated subdomain
- C2 traffic encrypted with RSA-2048
- Custom AES-encrypted storage format used to store configuration, plugins, and harvested data

- Unique version number for each sample

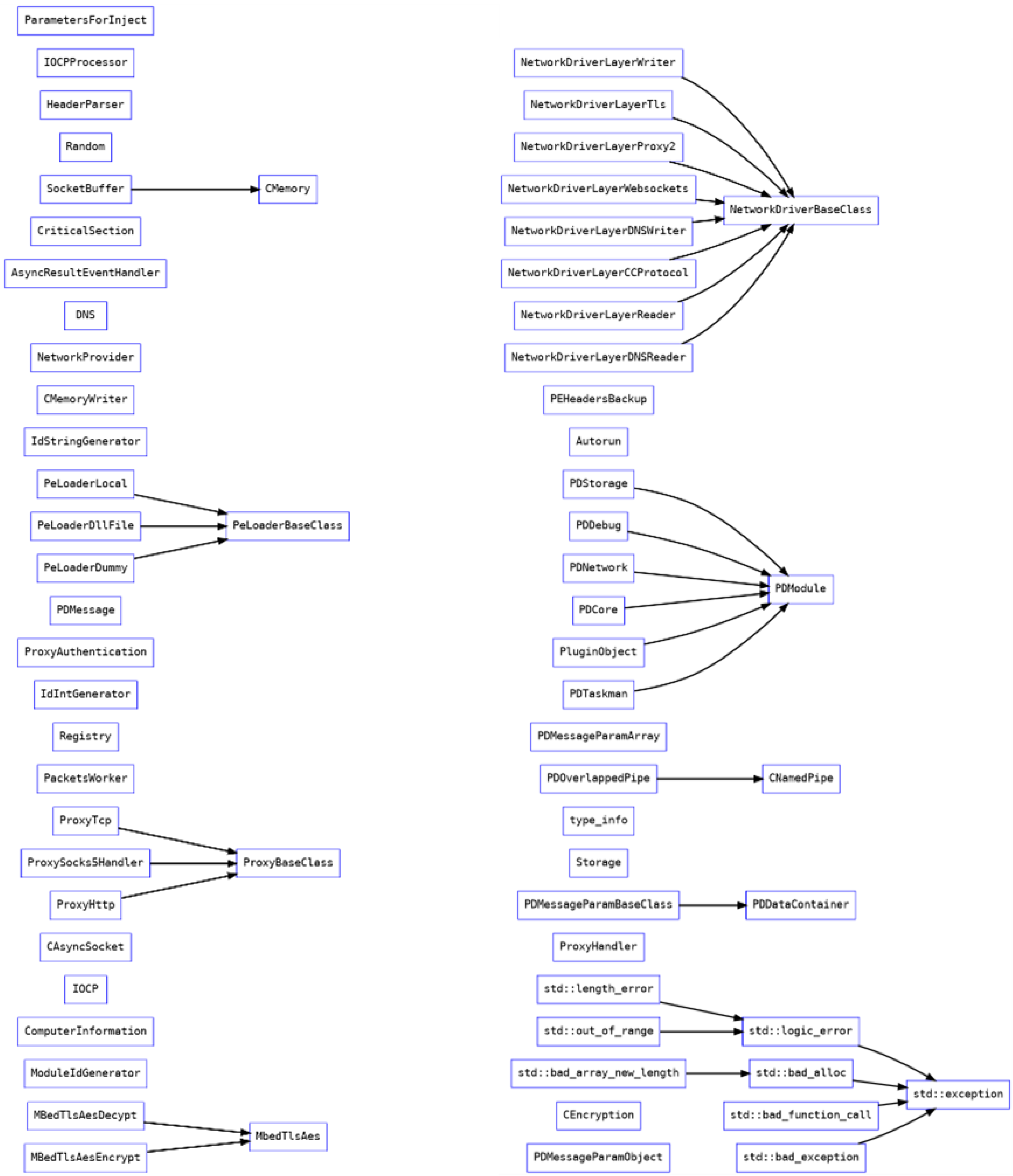

Figure 9: Backdoor classes hierarchy

According to a PDB path found in the 64-bit specimens, the project was originally called Sombra – possibly in reference to the Overwatch game character:

```
.rdata:0000000140093CE4 ; Debug information (IMAGE_DEBUG_TYPE_CODEVIEW)
.rdata:0000000140093CE4 asc_140093CE4   db 'RSDS'                ; DATA XREF: .rdata:0000000140090AB4â†'o
.rdata:0000000140093CE4                                          ; CV signature
.rdata:0000000140093CE8                 dd 833C4360h             ; Data1 ; GUID
.rdata:0000000140093CE8                 dw 0D9E3h                ; Data2
.rdata:0000000140093CE8                 dw 4B6Ch                 ; Data3
.rdata:0000000140093CE8                 db 0AAh, 0D6h, 52h, 0ACh, 9Ah, 82h, 4Eh, 2; Data4
.rdata:0000000140093CF8                 dd 1                     ; Age
.rdata:0000000140093CFC                 db 'C:\Projects\Sombra\_Bin\x64\Release\Sombra.pdb',0 ; PdbFileName
```

Figure 10: PDB path from 64-bit backdoor with project name ‘Sombra’

In the Overwatch game world, Sombra is an agent of an antagonist organization called Talon. She is skilled in computer hacking and cryptography and specializes in espionage and intelligence assessment:

“One of the world's most notorious hackers, Sombra uses information to manipulate those in power.

Sombra's skills include computer hacking and cryptography; these are activities she greatly enjoys, to the point where the desire to get past locks and solving mysteries is ingrained in her personality. She is a known associate of Reaper, specializing in espionage and intelligence assessment.

Stealth and debilitating attacks make Sombra a powerful infiltrator. Her hacking can disrupt her enemies, ensuring they're easier to take out, while her EMP provides the upper hand against multiple foes at once. Sombra’s ability to Translocate and camouflage herself makes her a hard target to pin down.”[1]

Embedded in each sample is a hardcoded version number, with the following versions observed thus far:

Version	Compilation timestamp	Architecture
0.0.1.114499	31-10-2019 21:22:39 UTC	x86
0.0.1.14630 (T)	09-11-2019 21:53:44 UTC	x86
0.1.60 (DT)	11-11-2019 14:55:45 UTC	x86
0.1.208 (DT)	17-11-2019 20:58:25 UTC	x86
0.1.724 (DT)	24-12-2019 10:33:41 UTC	x64
0.2.404 (DT)	20-08-2020 01:36:50 UTC	x64

One of the backdoor samples (0.1.60 (DT)) was found to be hosted on http[://]159.65.31[.]84/svolcdst.exe.

Behaviour

Before entering the command processing loop, the backdoor will check to see if it’s running as a service, and will create a run-once mutex consisting of %HOSTNAME% with a postfix of “S”, “U”, or “SU”, depending on which privileges it was executed with.

The C2 domain name for the DNS communication is hardcoded and obfuscated using XOR. The backdoor will generate a subdomain using a custom domain generation algorithm (DGA) and try to send an initial beacon to the C2 via DNS tunneling:

```
.text:00413478                 mov     cl, 68h ; 'h'
.text:0041347A                 mov     dword ptr [ebp+c2_domain], 10A1B68h
.text:00413481                 xor     eax, eax
.text:00413483                 mov     [ebp+var_1C], 6460C0Ah
.text:0041348A                 mov     [ebp+var_18], 1C0Dh
.text:00413490                 mov     [ebp+var_16], 0
.text:00413494
.text:00413494 decode_c2_domain:                       ; CODE XREF: do_stuff+C0â†“j
.text:00413494                 xor     [ebp+eax+var_1F], cl
.text:00413498                 inc     eax
.text:00413499                 cmp     eax, 9
.text:0041349C                 jnb     short loc_4134A3
.text:0041349E                 mov     cl, [ebp+c2_domain]
.text:004134A1                 jmp     short decode_c2_domain ; sbibd.net
```

Figure 11: Decoding the C2 domain name

The configuration, along with downloaded plugins and all harvested data are stored in a custom database format inside a single file under the %TEMP% directory. The file name is hardcoded and obfuscated with XOR. The storage file is encrypted with AES-256 using a hardcoded key and is decrypted each time the malware needs to read or write it and re-encrypted after new data is added:

```
51 65 54 68 57 6d 5a 71 34 74 37 77 39 7a 24 43 26 46 29 4a 40 4e 63 52 66 55 6a 58 6e 32 72 35
// ASCII: "QeThWmZq4t7w9z$C&F)J@NcRfUjXn2r5"
```

Figure 12. Hardcoded AES key for storage encryption

Strings used as backdoor commands and in debugging messages sent to the C2 are encoded with a simple alphabet substitution. These are not decrypted by the backdoor on the victim's side, and the key for decryption is not present in the binary. Most probably the backdoor client decrypts them locally:

.data:00000...	0000000C	C	vys{rdtxbyc
.data:00000...	00000020	C	~y~c~v{~mrvys{xvsg{bp~yunby~f~s
.data:00000...	00000007	C	gextrdd
.data:00000...	0000000C	C	vys{rdtxbyc
.data:00000...	00000004	C	are
.data:00000...	0000000C	C	gextrddyvzr
.data:00000...	00000005	C	qerr
.data:00000...	00000014	C	~yqxezvc~xyvttrgcrs
.data:00000...	0000000F	C	{xvsqexzzrzxen

Figure 13: Substitution-encoded strings

Command and Control (C2)

The C2 communication can either be performed via DNS tunnelling or TCP sockets. Traffic is SSL-encrypted and can bypass HTTP/SOCKS5 proxies. The C2 domain name is hardcoded in the binary and obfuscated with a single-byte XOR key which differs between samples. In order to establish communication, the malware first uses a DGA (Domain Generation Algorithm) to generate the subdomain to connect to. Depending on an internal boolean setting, one of the following URL formats is used:

- images**%x.%s**
- images**%x**.elmako.**%s**

where **%s** is the hardcoded domain name and **%x** contains 8 hexadecimal characters generated based on the result of the GetTickCount API:

```
(GetTickCount * 0x8088405 + 1) % 0xFFFFFFFF
```

If the connection is unsuccessful, the backdoor will try to generate and connect to several other URLs in the same domain, using the same algorithm but without the "images" prefix.

It seems that in most cases, the malware sends out data using DNS_TYPE_TEXT requests, while the attackers issue commands separately over the TCP channel with the IP address associated with the DGA-generated subdomain.

All the communication is compressed with zlib and encrypted with AES. Additionally, an embedded RSA public key is used to secure the AES key exchange:

```
_PUBLICKEYSTRUC <6, 2, 0, 0A400h>
RSAPUBKEY <31415352h, 800h, 10001h>
EF C9 77 B9 A3 8E 48 92 77 C8 E1 E1 0C 46 35 2B CD 5C DB 7B 66 26 85 D2 2A 22 46 0F
5E CE 7D BD 34 40 3D C1 F8 31 5F 5B 76 7F 76 7B 46 0D 58 C3 FD A4 D9 12 16 0D 40 BA
B5 2D 11 88 10 AB FF A6 84 E2 F0 E9 C8 47 32 D0 5D E0 4F 10 4A CB 85 EF 90 D6 94 79
76 64 17 7C 37 73 04 BD 87 28 E9 ED 7C FE 56 54 B0 5F 2B 5A E6 8E 1F C8 CF 6E 5D 25
A4 2C BA E2 2D A0 51 8B 32 E2 DD 59 95 DB DC 43 11 2A C2 6F 08 5E 4F 89 4E F5 0C 42
A0 27 E1 CE AC CC 03 C9 85 36 10 7A 38 A1 D5 67 88 26 BA D9 47 47 88 B6 4C 37 4E C2
A2 68 D5 A0 A4 10 8D FA 45 3F 24 42 48 17 EC C9 25 7D B3 A2 1A 87 9E C8 32 36 E7 96
A9 D6 2B 4D 05 D1 8C 1F B0 E9 06 DC FD DC 31 72 0E E6 CA B8 77 E2 66 FE F3 C4 64 40
1C F2 06 5D 81 73 39 6F ED 33 0E 6D E1 30 D3 94 83 A0 78 92 8F 6F 17 E6 26 A8 23 B7
03 D2 F8 A2
```

Figure 14: RSA key used for C2 traffic encryption

Backdoor Commands

Both the x86 and x64 versions of the backdoor feature approximately 50 different commands organized into six groups, each group served by a different interface:

- Core
- Taskman
- Config
- Storage
- Debug
- Network

Command	Type	Description
networkdisconnected broadcast	Core	Broadcast “networkdisconnected” message
informationaccepted broadcast	Core	Save provided session ID to memory struct, broadcast “informationaccepted” message
networkconnected broadcast	Core	Broadcast session ID and system info
ping	Core	Send a "ping" to the C2 server
loadasdll	Core	Load additional DLL into memory
loadfromstorage	Core	Inject DLL into memory (from storage)
loadfromfile	Core	Inject DLL into memory (from disk)
loadfrommem	Core	Inject DLL into memory (from memory)

loadplugincomplete	Core	Execute a plugin that is already loaded
initializeandloadpluginby-uniqid	Core	Load and execute a plugin; plugins are stored as zlib-compressed and AES encrypted PE files inside the storage and referred to by unique identifier
getinfo	Core	Obtain environment strings, computer name, username, OS version information, system time, etc.
restart	Core	Respawn using ShellExecuteW
shutdown	Core	Exit process
uninstall	Core	(unimplemented)
updatemyself	Core	Create backup of itself (with .old extension) and spawn new instance via CreateProcessW
pluginunload	Core	Unload and remove plugin from storage
getprocesslist	Taskman	Obtain a list of running processes
killprocessbypid	Taskman	Terminate a process by PID
killprocessbyname	Taskman	Terminate a process by name
get	Config	Read specified values from .config file in storage and send to the C2
set	Config	Set specific config fields and save to .config file in storage
del	Config	Delete specified config fields from .config file in storage
initdefaults	Config	Initialize config fields with default values and save to .config file in storage
clear	Config	Zero-out config fields and save to .config file in storage
save	Config	Save provided config values to .config file in storage
enum	Config	Read values from .config file in storage to memory
write	Storage	Encrypt and write data to storage file
create	Storage	Create new encrypted storage file
close	Storage	Encrypt and flush data to storage file
drop	Storage	Write supplied file content to storage file
delete	Storage	Delete file with specified ID from storage; enumerate files in storage

enum	Storage	Enumerate files in storage (name, written, size)
upload	Storage	Decrypt and upload file with specified ID from storage file
clearall	Storage	Remove all files from storage
archivebypath	Storage	Read file(s) from specified path and save to storage file, then enumerate storage
restorestorage	Storage	Delete storage file and open a new storage
cancel[1] /closeanddeletestorage	Storage	Remove all files from storage and delete storage temp file
closestorage	Storage	Close storage temp file
openstorage	Storage	Open storage temp file
getcontent	Storage	(unimplemented)
awaitcreate	Storage	Create new encrypted storage file
await&putcontent	Storage	Read from C2 and save to the storage file
await&getcontent	Storage	Read from content from storage and send via C2
debuglog	Debug	Enable debug logging
broadcast	Network	Set networkconnected or networkdisconnected bool in memory
touchconnect	Network	Send the networkconnected bool setting to the C2
stats	Network	Send details of sent/received bytes
reconnect	Network	Close socket and reconnect
disconnect	Network	Close socket
switchtotcp	Network	Switch C2 communication to TCP/IP
switchdns	Network	Switch C2 communication to DNS
setproxy[2]	Network	Set proxy type, host, port, domain, user, password
checkproxy[2]	Network	(unimplemented)
getproxy[3]	Network	Send current proxy configuration to C2

resetproxy[3]	Network	(unimplemented)

1 – before v0.1.60t
2 – since at least v0.1.208
3 – since at least v0.1.724

Network

Infosportals[.]com

First active during October 2019, infosportals[.]com was utilized by early SombRATs as the primary C2 domain. Since then, the domain shifted IP address multiple times, was then taken offline between February and May, before being reactivated briefly between late May and mid-June as part of another offensive:

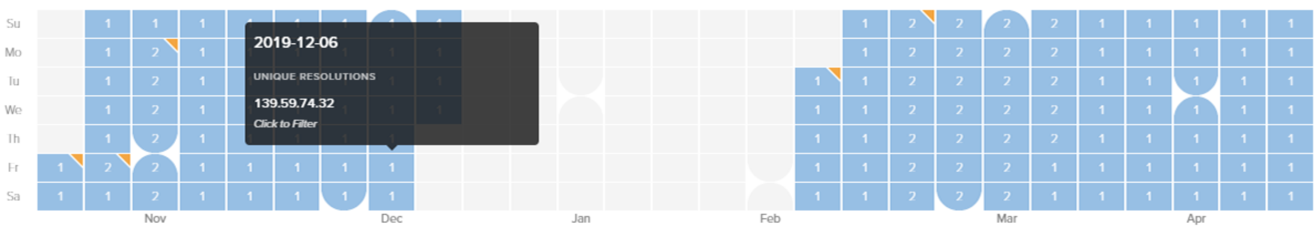

Figure 15: Timeline of IP resolutions for infosportals[.]com

	Resolve	Location	Network	ASN	First	Last
☐	212.114.52.98	DE	212.114.52.0/24	30823	2020-05-23	2020-06-16
☐	0.0.0.0		Unknown		2020-02-04	2020-05-11
☐	144.217.53.146	CA	144.217.0.0/16	16276	2020-02-16	2020-03-12
☐	139.59.74.32	IN	139.59.64.0/20	14061	2019-10-28	2019-12-11
☐	185.189.112.223	DE	185.189.112.0/24	9009	2019-10-18	2019-11-02
☐	185.189.112.223	DE	185.189.112.0/24	9009	2019-10-25	2019-10-25

Figure 16: Table of IP resolutions for infosportals[.]com

Sbibd[.]net

A phishing domain mimicking the legitimate sbibd.com (registered to the State Bank of India, Bangladesh), sbibd[.]net was first active for a short spell from early November to December 2019, then reactivated again between February and March 2020 and was used as the primary C2 with several SombRAT variants:

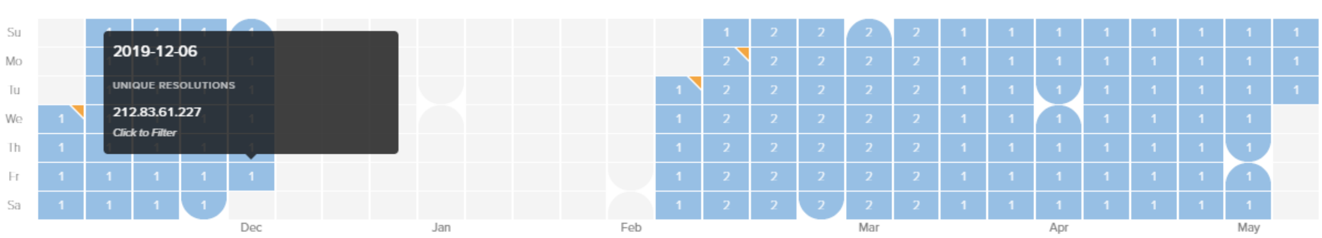

Figure 17: Timeline of IP resolutions for sbibd[.]net

	Resolve	Location	Network	ASN	First	Last
☐	0.0.0.0		Unknown		2020-02-04	2020-05-05
☐	144.217.53.146	CA	144.217.0.0/16	16276	2020-02-10	2020-03-14
☐	212.83.61.227	DE	212.83.32.0/19	47447	2019-11-06	2019-12-06

Figure 18: Table of IP resolutions for sbibd[.]net

Akams[.]in

First active for a few weeks from late December 2019 to mid-January 2020, akams[.]in was also used by multiple SombRAT samples for C2 communications. One of the prior resolutions, for IP 45.89.175.206, is particularly interesting, as it overlaps with another domain called mail[.]kub-gas[.]com, which was implicated as being associated with an APT-28/Fancy Bear/Sofacy phishing campaigns in a report by Area 1 Security. However, after much scrutiny, it would appear highly likely that there is no direct connection between the SombRAT campaign and APT-28 activity.

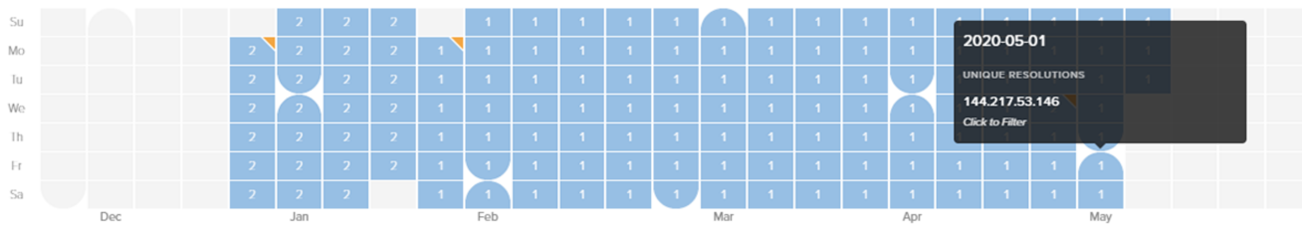

Figure 19: Timeline of IP resolutions for akams[.]in

	Resolve	Location	Network	ASN	First	Last
☐	144.217.53.146	CA	144.217.0.0/16	16276	2020-01-20	2020-05-05
☐	144.217.53.146	CA	144.217.0.0/16	16276	2020-04-22	2020-04-22
☐	45.89.175.206	RO	45.89.175.0/24	9009	2019-12-23	2020-01-17
☐	45.89.175.206	RO	45.89.175.0/24	9009	2019-12-23	2020-01-17
☐	184.168.221.67	US	184.168.220.0/22	26496	2018-03-07	2018-03-07

Figure 20: Table of IP resolutions for akams[.]in

newspointview[.]com

Registered and active during late June 2020, newspointview[.]com has been used with more recent SombRAT variants as the primary C2 domain:

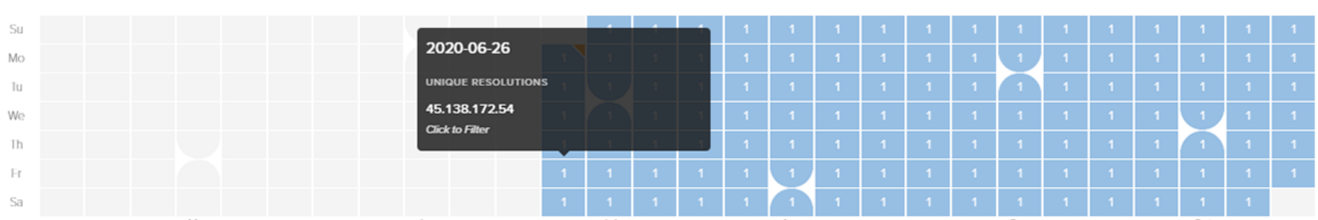

Figure 21: Timeline of IP resolutions for newspointview[.]com

	Resolve	Location	Network	ASN	First	Last
☐	45.138.172.54	DE	45.138.172.0/22	30823	2020-06-22	2020-10-16

Figure 22: Table of IP resolutions for newspointview[.]com

Timeline

The following timeline shows key domain/IP resolutions and known SombRAT releases:

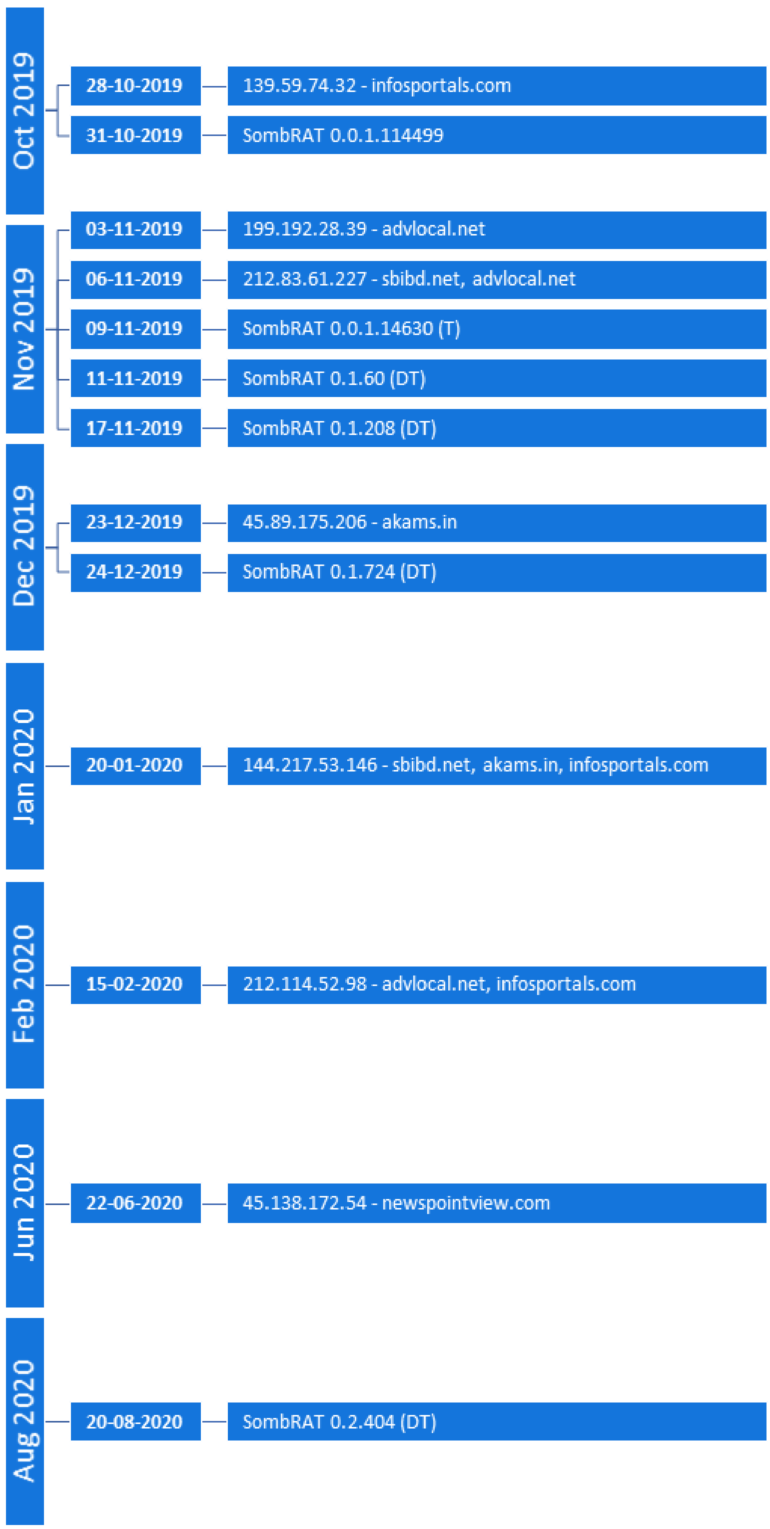

Figure 23: Timeline of IP resolutions and SombRAT versions

Conclusions

There are several factors that lead us to the assumption that the threat actor behind CostaRicto is a mercenary group:

- The toolset used in CostaRicto campaign consists of bespoke malware that appeared around October 2019 and has been rarely seen in the wild since. It therefore appears to be private to this particular adversary.

- Moreover, the constant development, detailed versioning system and well-structured code that allows for easy functionality expansion – all suggest that the toolset is part of a long-term project, rather than a one-off campaign.

- The apparent sharing of network infrastructure with a previous, seemingly unrelated phishing campaign attributed to APT28, as well as the reuse of phishing domain names as C2 servers in attacks against unrelated victims, indicates that the same entity is likely behind a diverse range of attacks.

- Finally, the diversity and geography of the victims doesn't fit a picture of a campaign sponsored by a particular state; rather, it's a mix of targets that could be explained by different assignments commissioned by disparate entities.

With the undeniable success of Ransomware-as-a-Service (RaaS), it's not surprising that the cybercriminal market has expanded its portfolio to add dedicated phishing and espionage campaigns to the list of services on offer. Outsourcing attacks or certain parts of the attack chain to unaffiliated mercenary groups has several advantages for the adversary – it saves their time and resources and simplifies the procedures, but most importantly it provides an additional layer of indirection, which helps to protect the real identity of the threat actor.

Researchers and investigators tend to group adversaries based on similar tactics, techniques and procedures, code reuse, and physical infrastructure overlap. The attribution is often derived by analyzing the nature and geography of the campaign targets in relation to geopolitical situation. However, in the case of mercenary APTs, the selection of victims might appear random and will rarely reveal a bigger picture about the motives behind the campaigns.

When dealing with threat actors that outsource their campaigns, only the entity that performed the attack can be tracked, while the actual perpetrator becomes more elusive than ever.

Indicators of Compromise (IoCs):

Indicator	Type	Description
130fa726df5a58e9334cc28dc62e3ebaa0b7c0d637fce1a66daff66ee05a9437	SHA256	SombRAT x86 loader
8062e1582525534b9c52c5d9a38d6b012746484a2714a14febe2d07af02c32d5	SHA256	SombRAT x86 loader
d69764b22d1b68aa9462f1f5f0bf18caebbcff4d592083f80dbce39c64890295	SHA256	SombRAT x86 loader
f6ecdae3ae4769aaafc8a0faab30cb66dab8c9d3fff27764ff208be7a455125c	SHA256	SombRAT x86 loader
561bf3f3db67996ce81d98f1df91bfa28fb5fc8472ed64606ef8427a97fd8cdd	SHA256	SombRAT x86 payload (memory dump)
8323094c43fcd2da44f60b46f043f7ca4ad6a2106b6561598e94008ece46168b	SHA256	SombRAT x86 payload
ee0f4afee2940bbe895c1f1f60b8967291a2662ac9dca9f07d9edf400d34b58a ee0f4afee2940bbe895c1f1f60b8967291a2662ac9dca9f07d9edf400d34b58a	SHA256	SombRAT x86 payload (UPX)

70d63029c65c21c4681779e1968b88dc6923f92408fe5c7e9ca6cb86d7ba713a	SHA256	SombRAT encoded payload (x64)
79009ee869cec789a3d2735e0a81a546b33e320ee6ae950ba236a9f417ebf763	SHA256	SombRAT decoded payload (x64)
d8189ebdec637fc83276654635343fb422672fc5e3e2818df211fb7c878a3155	SHA256	Payload stager
fa74f70baa15561c28c793b189102149d3fb4f24147adc5efbd8656221c0960b	SHA256	GO-socks5 proxy
c0db3dadf2e270240bb5cad8a652e5e11e3afe41b8ee106d67d47b06f5163261	SHA256	Pcheck proxy
6df8271ae0380737734b2dd6d46d0db3a30ba35d7379710a9fb05d1510495b49	SHA256	Pcheck proxy
7424d6daab8407e85285709dd27b8cce7c633d3d4a39050883ad9d82b85198bf	SHA256	Pscan port scanner
svolcdst.exe	Filename	SombRAT loader
tunnusvcen.exe	Filename	SombRAT loader
C:\Projects\Sombra_Bin\x64\Release\Sombra.pdb	PDB path	SombRAT x64
C:\Wokrflow\CostaRicto\Release\CostaBricks.pdb	PDB path	SombRAT loader
%HOSTNAME%UI724	Mutex	Run-once mutex
%HOSTNAME%SUI724	Mutex	Run-once mutex
sbibd[.]net	Domain	SombRAT C2
infosportals[.]com	Domain	SombRAT C2
akams[.]in	Domain	SombRAT C2
newspointview[.]com	Domain	SombRAT C2
159.65.31.84	IP	SombRAT hosting place
212.83.61.227	IP	sbibd[.]net
144.217.53.146	IP	sbibd[.]net, akams[.]in, infosportals[.]com
45.89.175.206	IP	akams[.]in
45.138.172.54	IP	newspointview[.]com

212.114.52.98	IP	infosportals[.]com

MITRE ATT&CK:

Tactic	ID	Name	Description
Initial Access	T1078	Valid Accounts	Suspected initial compromise using stolen credentials
Execution	T1106	Execution through API	SombRAT – C2 command
T1053/005	Scheduled Task/Job: Scheduled Task	Used to download SombRAT loader	
T1059/001	Command and Scripting Interpreter: PowerShell	Used to load x64 SombRAT	
Defence Evasion	T1055	Process Injection	Invoke-ReflectivePEInjection PowerSploit module
T1140	Deobfuscate/Decode Files or Information	SombRAT – Decode strings and custom storage data	
Discovery	T1057	Process Discovery	SombRAT – C2 command
T1082	System Information Discovery	SombRAT – C2 command	
T1124	System Time Discovery	SombRAT – C2 command	
T1046	Network Service Scanning	pscan, nmap	
Collection	T1560/003	Archive Collected Data: Archive via Custom Method	SombRAT – Custom storage file
Command and Control	T1572	Protocol Tunneling	SombRAT - DNS tunnelling for C2
T1071/001	Application Layer Protocol: Web Protocols	SombRAT – HTTP for C2	
T1573/002	Encrypted Channel: Asymmetric Cryptography	SombRAT – RSA for C2 encryption	
T1090/002	Proxy: External Proxy	pcheck HTTP/S proxy, GO SOCKS5 proxy, PuTTY	
Exfiltration	T1041	Exfiltration Over C2 Channel	SombRAT

Yara Hunting Rules:

```
import "pe"
import "hash"
```

```
rule costaricto_vm_dropper
{
   meta:
      description = "Rule to detect SombRAT loader by code similarity"
      author = "BlackBerry Threat Hunting and Intelligence Team"

   strings:
      // vm class name
      $classname = "VMBASERUNNER" ascii wide nocase

      // start of vm bytecode
      $vmbytecode = {37C7359438C73594}

      // start of encrypted payload
      $encpayload_1 = {77D2C7AC59B2EB0DF37028AC950971FB}

     // binary string from enc payload (some payloads differ only in the header)
      $encpayload_2 = {06359D29C83125C321C201CF9AE7D1626B8F4281C33617EECE86BD106C628FE593936F00C2C
68E28843BE5374F876840FCD1BFD014D5DEFF4BA8EB6A5FFFB24F932138B04C1BE6D5BD8BB572B8116799AE1C8F0
D5DB774ABA4884B9E706981FC3740B4CD891F8A0EA6900D41B675CFC98A}

      // vm execution loop
      $vmcode_1 = {8B ?? 08 8B ?? 0C 89 ?? 29 ?? C1 ?? 02 39 ?? 74 4E 83 ?? ?? 08 8D ?? ?? 8B ?? ?? 8D ?? 01 89 ??
8B ?? ?? 66 83 ?? 08 00 75 28 8B ?? ?? 8D ?? 04 5? 5? E8 ?? ?? FF FF 8B ?? ?? 83 ?? 0C 5? 8B ?? 0C 89 ?? 5? FF ?? 14
83 C4 08 8B ?? 8B ?? 08 8B ?? 0C 89 ?? 29 ?? C1 ?? 02 39 ?? 89 ?? 75 B9}

      // vm execution loop (sample from Nov 2019)
      $vmcode_2 = {8B ?? 4? 89 ?? 8B ?? 08 8B ?? 88 33 ?? 66 39 ?? 08 75 19 8D ?? 04 5? 8D ?? 08 E8 ?? ?? 00 00 8B ??
8D ?? 0C 5? 5? FF ?? 5? 5? 8B ?? 8B ?? 0C 2B ?? 08 C1 ?? 02 3B ?? 75 C7}

   condition:
      uint16(0) == 0x5a4d and filesize < 5MB and filesize > 20KB and any of them
}

rule costaricto_vm_dropper_pdb_path
{
   meta:
      description = "Rule to detect samples with CostaRicto PDB path"
      author = "BlackBerry Threat Hunting and Intelligence Team"
      pdb_string = "C:\\Wokrflow\\CostaRicto\\Release\\CostaBricks.pdb"

   strings:
      $a = "CostaRicto" ascii wide nocase
      $b = "CostaBricks.pdb" ascii wide nocase
      $c1 = "C:\\Wokrflow\\" ascii wide nocase
      $c2 = "Release" ascii wide nocase
      $c3 = ".pdb" ascii wide nocase

   condition:
      uint16(0) == 0x5a4d and filesize < 5MB and filesize > 20KB and ($a or $b or all of ($c*))
}

rule costaricto_sobmrat_pdb_path
{
   meta:
      description = "Rule to detect samples with SombRAT PDB path"
      author = "BlackBerry Threat Hunting and Intelligence Team"
      pdb_string = "C:\\Projects\\Sombra\\_Bin\\x64\\Release\\Sombra.pdb"
      pdb_string_2 = "c:\\projects\\sombra\\libraries"

   strings:
      $a = "\\Projects\\Sombra\\" ascii wide nocase
      $b = "Sombra.pdb" ascii wide nocase

    condition:
      uint16(0) == 0x5a4d and filesize < 5MB and filesize > 20KB and ($a or $b)
}
```

```
rule costaricto_backdoored_blink
{
    meta:
        description = "Rule to detect backdoored Blink application"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    strings:
        $a1 = "Failed to open target application process!"
        $a2 = "Machine architecture mismatch between target application and this application!"
        $a3 = "Failed to create new communication pipe!"
        $b = "Plauger, licensed by Dinkumware, Ltd."

   condition:
     uint16(0) == 0x5a4d and filesize < 5MB and filesize > 50KB and ($b and 1 of ($a*))
}

rule costaricto_rich_header
{
    meta:
        description = "Rule to detect Rich header associated with CostaRicto campaign"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    condition:
        pe.rich_signature.toolid(0xf1, 40116) and
        pe.rich_signature.toolid(0xf3, 40116) and
        pe.rich_signature.toolid(0xf2, 40116) and
        pe.rich_signature.toolid(0x105, 26706) and
        pe.rich_signature.toolid(0x104, 26706) and
        pe.rich_signature.toolid(0x103, 26706) and
        pe.rich_signature.toolid(0x93, 30729) and
        pe.rich_signature.toolid(0x109, 27023) and
        pe.rich_signature.toolid(0xff, 27023) and
        pe.rich_signature.toolid(0x97, 0) and
        pe.rich_signature.toolid(0x102, 27023)
}

rule costaricto_rich_header_august
{
    meta:
        description = "Rule to detect Rich header associated with CostaRicto campaign"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    condition:
        pe.rich_signature.toolid(0xf1, 40116) and
        pe.rich_signature.toolid(0xf2, 40116) and
        pe.rich_signature.toolid(0xf3, 40116) and
        pe.rich_signature.toolid(0x102, 26428) and
        pe.rich_signature.toolid(0x103, 26131) and
        pe.rich_signature.toolid(0x104, 26131) and
        pe.rich_signature.toolid(0x105, 26131) and
        pe.rich_signature.toolid(0x103, 26433) and
        pe.rich_signature.toolid(0x104, 26433) and
        pe.rich_signature.toolid(0x109, 26428) and
        pe.rich_signature.toolid(0x93, 30729) and
        pe.rich_signature.toolid(0xff, 26428)
}

rule costaricto_rich_xor_key
{
    meta:
        description = "Rule to detect Rich header associated with CostaRicto campaign"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    condition:
        // x86 droppers
        pe.rich_signature.key == 0x2e8d923f or
        pe.rich_signature.key == 0x97d94c45 or

        // x86 payload
        pe.rich_signature.key == 0xef257087 or
```

```
        pe.rich_signature.key == 0x4f257087 or
        pe.rich_signature.key == 0x1e816e7e or
        // x64 payload
        pe.rich_signature.key == 0xd1e5ae6c or
        pe.rich_signature.key == 0x5df9c60b
}

rule costaricto_sombrat_unpacked
{
    meta:
        description = "Rule to detect unpacked SombRAT backdoor"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    strings:
        // class names
        $a1 = "PEHeadersBackup"
        $a2 = "PeLoaderDummy"
        $a3 = "PeLoaderLocal"
        $a4 = "PeLoaderBaseClass"
        $a5 = "PDTaskman"
        $a6 = "PDMessageParamArray"
        $a7 = "NetworkDriverLayerWebsockets"
        $a8 = "NetworkDriverLayerDNSReader"
        $a9 = "WaitForPluginIOCPFullyClosed"

        // substitution-encrypted strings
        $b1 = "~ydcv{{rs{~|r"          // installedlike
        $b2 = "~yg{vcqxez"             // winplatform
        $b3 = "~yqxezvc~xyvttrgcrs"    // informationaccepted
        $b4 = "xvsqexzdcxevpr"         // loadfromstorage
        $b5 = "xvsqexzzrzxen"          // loadfrommemory
        $b7 = "xgrydcxevpr"            // openstorage
        $b8 = "g{bp~y{xvstxzg{rcr"     // pluginloadcomplete
        $b9 = "g{bp~yby{xvs"           // pluginunload

        // AES-encrypted strings
        $c1 = {44 5B 7F 52 0C 13 52 1A 16 45 4C 75 65 72 60 53}

        // RSA public key
        $d1 = {EF C9 77 B9 A3 8E 48 92 77 C8 E1 E1 0C 46 35 2B}

    condition:
        uint16(0) == 0x5a4d and filesize < 5MB and filesize > 20KB and any of them
}

rule costaricto_pcheck_proxy
{
    meta:
        description = "Rule to detect a custom proxy tool related to the CostaRicto campaign"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    strings:
        $a = "exe.exe host host_port proxy_host proxy_port"
        $b = "Tool jobs done"

    condition:
        uint16(0) == 0x5a4d and filesize < 500KB and filesize > 10KB and ($a or $b)
}

rule costaricto_pscan_port_scanner
{
    meta:
        description = "Rule to detect a custom proxy tool related to the CostaRicto campaign"
        author = "BlackBerry Threat Hunting and Intelligence Team"

    strings:
        $a1 = "Invalid arguments count (ver "
        $a2 = "Example: ./pscan"
```

```
        $a3 = "127-130.0.0.1"
        $b1 = "[output.txt]"
        $b2 = "Invalid ip address range"

    condition:
        uint16(0) == 0x5a4d and filesize < 500KB and filesize > 10KB and any of ($a*) or all of ($b*)
}
```

IDAPython Scripts:

```
#!/usr/bin/python

import sys, os, struct, array

fin = sys.argv[1]
fout = "%s_decoded" %(fin)
f = open(fin, "r+w+b")
f2 = open(fout, "w+b")
encsize = os.path.getsize(fin) / 4

key_1 = 0x14820285
key_2 = 0x26820323
key_3 = 0x35223562
key_4 = 0x41256421
cst_1 = 0x61C88647
cst_2 = 0x9E3779B9

enc = array.array('I')
enc.read(f, encsize)

i = 0

while i < encsize:
    encdw_1 = enc[i]
    encdw_2 = enc[i+1]

    tmp_1a = encdw_1 << 4 & 0xffffffff
    tmp_1b = encdw_1 >> 5 & 0xffffffff
    tmp_1c = encdw_1 - cst_1 & 0xffffffff
    tmp_2a = tmp_1a + key_3 & 0xffffffff
    tmp_2b = tmp_1b + key_4 & 0xffffffff
    tmp_3 = tmp_2a  ^ tmp_1c

    keydw_2 = tmp_3 ^ tmp_2b
    decdw_2 = encdw_2 - keydw_2 & 0xffffffff

    magic_1 = decdw_2 << 4 & 0xffffffff
    magic_2 = decdw_2 >> 5 & 0xffffffff
    key_1a = key_1 + magic_1 & 0xffffffff
    key_2a = key_2 + magic_2 & 0xffffffff
    cst_2a = cst_2 + decdw_2 & 0xffffffff
    tmp_5 = key_1a ^ cst_2a

    keydw_1 = tmp_5 ^ key_2a
    decdw_1 = encdw_1 - keydw_1 & 0xffffffff

    data1 = struct.pack('I', decdw_1)
   data2 = struct.pack('I', decdw_2)

    f2.seek(i*4)
    f2.write(data1)
    f2.seek(i*4+4)
    f2.write(data2)

    i = i + 2
```

Figure 24: SombRAT payload decryption script

```
import idc, idaapi, idautils
import idautils
import string, array, struct, binascii

def isprintable(s, codec='ascii'):
    try: s.decode(codec)
    except UnicodeDecodeError: return False
    else: return True

def get_int(addr):
    return struct.unpack('I', get_bytes(addr, 4))[0]

def add_comment(offset, comment):
    idc.MakeComm(offset, comment)
    target = idc.DfirstB(offset)
    while target != BADADDR:
        idc.MakeComm(target, comment)
        target = idc.DnextB(offset, target)

def substitution(start, size, patch):
    dec = ""
    enclen = size
    plain = "`abcdefghijklmnopqrstuvwxyz{|}~H&\x7F"
    key =   "wvutsrqp\x7F~}|{zyxgfedcba`onmlkji&Hh"
    if len(key) != len(plain):
        warning("Lenght differs!")
    i = 0
    for i in range(enclen):
        c = Byte(start + i)
        idx = key.find(str(chr(c)))
        if idx != -1:
            c = plain[idx]
        else:
            c = str(chr(c))
        dec = dec + c
        if patch == True:
            patch_byte(start + i, c)
        i += 1
    return dec

# iterate over all segments
for s in idautils.Segments():
    if ".data" in idc.SegName(s):
        start = idc.GetSegmentAttr(s, idc.SEGATTR_START)
        end = idc.GetSegmentAttr(s, idc.SEGATTR_END)
        num = 0
        while start < end - 4:
            if get_int(start) == 0:
                enclen = get_int(start+4)
                encstrcheck = get_int(start+8)
                if enclen > 1 and enclen < 100 and encstrcheck > 0x2020:
                    encstr = idc.get_bytes(start+8, enclen)
                    if isprintable(encstr) == True:
                        num += 1
                        startaddr = start+8

                        print("#%i") %num
                        print("address = 0x{:08x}".format(start))
                        print("len = %i") %enclen
                        print("encstr = %s") %encstr
                        decstr = ""
                        decstr = substitution(startaddr, enclen, 0)
                        print("decstr =%s") % decstr
                        print("----------------------------------------")
                        idc.MakeComm(start, "{}".format(decstr))
                        decname = "s_" +
                          (''.join(e for e in decstr if e.isalnum()))[:20]
```

```
        decname = decname.strip()
        res = MakeNameEx(start, decname,
                SN_NOCHECK | SN_NOWARN | 0x800)

    start += 4
```

Figure 25: SombRAT string decoding IDA Python script (for x86 payloads)

About The BlackBerry Research & Intelligence Team

The BlackBerry Research & Intelligence team examines emerging and persistent threats, providing intelligence analysis for the benefit of defenders and the organizations they serve.

Back