Bookworm Trojan: A Model of Modular Architecture

By Robert Falcone, Mike Scott, Juan Cortes

Published: 2015-11-10 · Archived: 2026-04-05 17:09:31 UTC

Recently, while researching attacks on targets in Thailand, Unit 42 discovered a tool that initially appeared to be a variant of the well-known PlugX RAT based on similar observed behavior such as the usage of DLL side-loading and a shellcode file. After closer inspection, it appears to be a completely distinct Trojan, which we have dubbed **Bookworm** and track in Autofocus using the tag Bookworm.

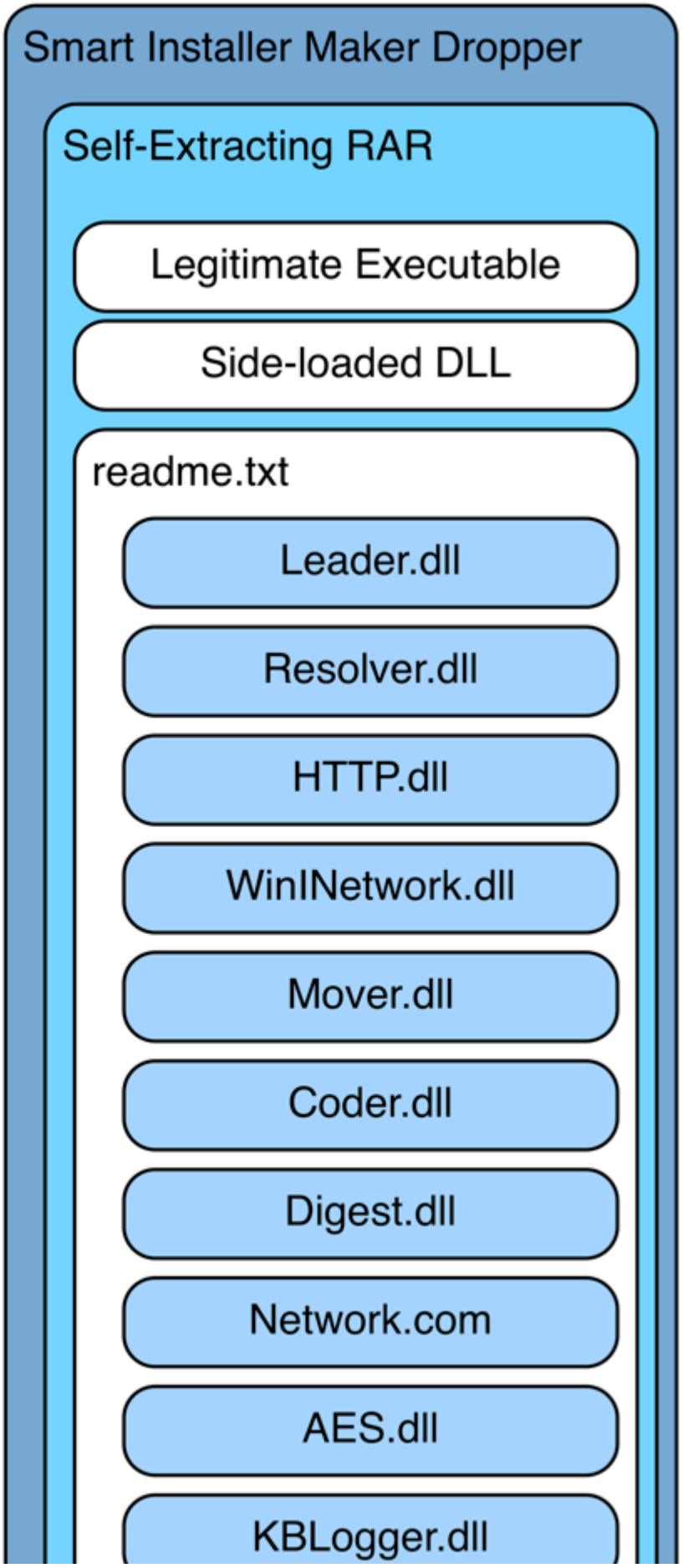
Bookworm's functional code is radically different from PlugX and has a rather unique modular architecture that warranted additional analysis by Unit 42. Bookworm has little malicious functionality built-in, with its only core ability involving stealing keystrokes and clipboard contents. However, Bookworm expands on its capabilities through its ability to load additional modules directly from its command and control (C2) server. This blog will provide an analysis of the Bookworm Trojan and known indicators of compromise. A later blog will explore the associated attack campaigns and attributions surrounding Bookworm.

Bookworm: Chapter One

So far, it appears threat actors have deployed the Bookworm Trojan primarily in attacks on targets in Thailand. Bookworm has many layers (see Figure 1) that increase the complexity of its overall architecture. To make matters worse, the author used multiple algorithms not only to encrypt and decrypt files saved to the system, but also to encrypt and decrypt network communications between Bookworm and its C2 servers (known servers are listed in the Indicators of Compromise section near the end of this post).

Layered Loading Approach

The threat actors use a commercial installation tool called Smart Installer Maker to encapsulate and execute a self-extracting RAR archive and in some cases a decoy slideshow or Flash installation application. The self-extracting RAR writes a legitimate executable, an actor-created DLL called Loader.dll and a file named readme.txt to the filesystem and then executes the legitimate executable.


Smart Installer Maker Dropper
Self-Extracting RAR
Legitimate Executable
Side-loaded DLL
readme.txt
Leader.dll
Resolver.dll
HTTP.dll
WinINetwork.dll
Mover.dll
Coder.dll
Digest.dll
Network.com
AES.dll
KBLogger.dll

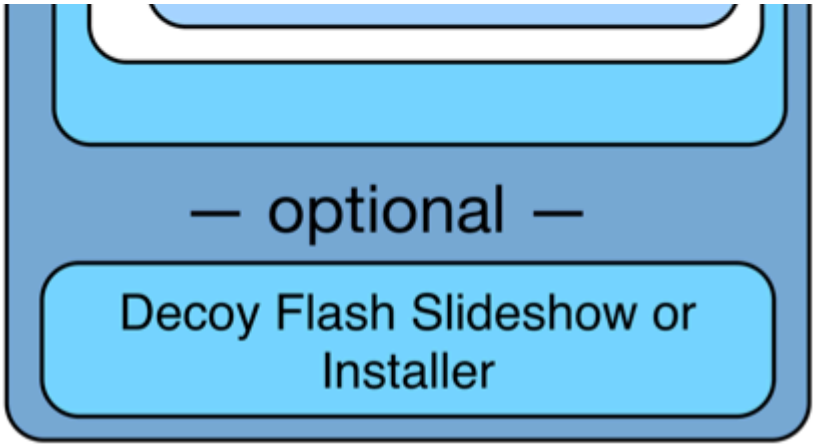


Figure 1 Architecture of Bookworm

Thus far, the actors deploying bookworm have used the legitimate executables Microsoft Malware Protection (MsMpEng.exe) and Kaspersky Anti-Virus (ushata.exe) to perform DLL side-loading and load the Loader.dll. Loader.dll decrypts the readme.txt file using a three byte XOR algorithm with 0xd07858 as a key, which results in shellcode that is responsible for decrypting the remainder of readme.txt containing the actual Bookworm Trojan. The shellcode then loads Bookworm by manually loading another DLL named “Leader.dll” embedded in the decrypted readme.txt and passes a buffer to Leader.dll containing additional DLLs. Leader.dll is the main component of Bookworm, which will we refer to as “Leader” for the remainder of this blog.

The initial execution of Leader results in the installation of Bookworm. The installation process involves moving the legitimate executable and actor-created DLL to a new location. Bookworm also creates an additional file in this new location that has the same filename as the actor-created DLL but with no file extension. Figures 2 and 3 shows the newly created files based on the legitimate application used to side-load the actor created DLL.

%AllUsersProfile%\Application Data\Microsoft\DeviceSync\MsMpEng.exe
%AllUsersProfile%\Application Data\Microsoft\DeviceSync\MpSvc.dll
%AllUsersProfile%\Application Data\Microsoft\DeviceSync\MpSvc

Figure 2 Files created if the Microsoft Malware Protection was used to Sideload the DLL

%AllUsersProfile%\Application Data\Microsoft\DeviceSync\ushata.exe
%AllUsersProfile%\Application Data\Microsoft\DeviceSync\ushata.dll
%AllUsersProfile%\Application Data\Microsoft\DeviceSync\ushata

Figure 3 Files created if the Kaspersky Antivirus application was used to Sideload the DLL

After this process is completed, Bookworm changes how it loads itself, now reading the newly created file “MpSvc” or “ushata” instead of readme.txt. The newly created file is encrypted with RC4 using the contents of the following registry value as its key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Registration\ProductID

The decrypted contents of this new file contain a path to the following file:

%AllUsersProfile%\Application Data\Microsoft\Crypto\RSA\MachineKeys\sgkey.data

The sgkey.data file contains the shellcode from readme.txt that loads the Bookworm modules, but instead of being encrypted with the three byte XOR algorithm like readme.txt, sgkey.data is encrypted with RC4 using the "ProductID" value as the key. The installation process finishes with the creation of a service named "Microsoft Windows DeviceSync Service", which results in the addition of registry keys listed in Figure 4, which will run Bookworm when the system starts.

HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\Type: 0x00000120
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\Start: 0x00000002
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\ErrorControl: 0x00000001
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\ImagePath: "C:\Documents and Settings\All Users\Application Data\Microsoft\DeviceSync\MsMpEng.exe"
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\DisplayName: "Microsoft Windows DeviceSync Service"
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\ObjectName: "LocalSystem"
HKLM\SYSTEM\CurrentControlSet\Services\DeviceSync\Description: "Allows USB devices to be hosted on this computer. If this service is stopped, any hosted USB devices will stop functioning and no additional hosted devices can be added. If this service is disabled, any services that explicitly depend on it will fail to start."

Figure 4 Registry Keys Resulting from the Creation of the Bookworm Service

Bookworm Modules

Leader is Bookworm's main module and controls all of the activities of the Trojan, but relies on the additional DLLs to provide specific functionality. The developers of Bookworm use these modules in a rather unique way, as the other embedded DLLs provide API functions for Leader to carry out its tasks. To load additional modules, Leader parses the buffer passed to it by the shellcode in readme.txt for the other DLLs, which exist in the following structure:

```
struct embedded_dll {
DWORD dll_identifier;
DWORD length_of_dll;
char[length_of_dll] embedded_dll;
};
```

Table 1 contains all of the embedded DLLs in each Bookworm sample, their ID numbers, and a description of the functionality of each DLL's API functions provided to Leader. It should be noted that Bookworm does not write any of these DLLs to the filesystem, as the Trojan operates entirely in memory.

Name	DLL ID #	Description
Leader.dll	0x0	Main module. Communicates with the C2 server and other activities by interacting with other modules in this table.

Resolver.dll	0x1	Used to resolve C2 server locations.
Mover.dll	0x2	Moves the Bookworm files from the RAR archive to a new folder and runs it from the new location. Only used on initial infection during installation.
Coder.dll	0xA	Used to carry out RC4 encryption and decryption, base64 encoding and decoding and the generation of CRC32 hashes of data.
Digest.dll	0xB	Used to generate MD5 hashes of data.
AES.dll	0xC	Used to encrypt and decrypt data using AES.
Network.dll	0xE	Sets the network interface into promiscuous mode and gathers network traffic for traffic destined to the system to receive data from C2 responses. Also provides the ability to send data to the C2 as well.
HTTP.dll	0x13	Used to create HTTP Requests to send to the C2.
WinINetwork.dll	0x17	Used to interact with the C2 server, specifically by sending HTTP GET and POST requests.
KBLogger.dll	0x5	Key logger that records keystrokes and the contents saved to the clipboard.

Table 1 Bookworm's Embedded Modules with their Corresponding Identification Number and a Brief Description

Leader loads each DLL into memory and then resolves an exported function named "ProgramStartup" within the loaded DLL. Leader then uses the "dll_identifier" (DLL ID # in Table 1) value to determine the appropriate arguments to send to the DLL when calling the ProgramStartup function. Leader then passes a pointer to a structure to each loaded DLL with each DLL receiving a different offset that it will set with addresses of its internal functions. The purpose of passing a structure to each DLL is to populate one large structure that allows Leader to call specific functions within each DLL, which is very similar conceptually to the import address table of a portable executable. Figure 5 below visualizes this concept, showing Leader calling example functions in the Bookworm modules to carry out various tasks.

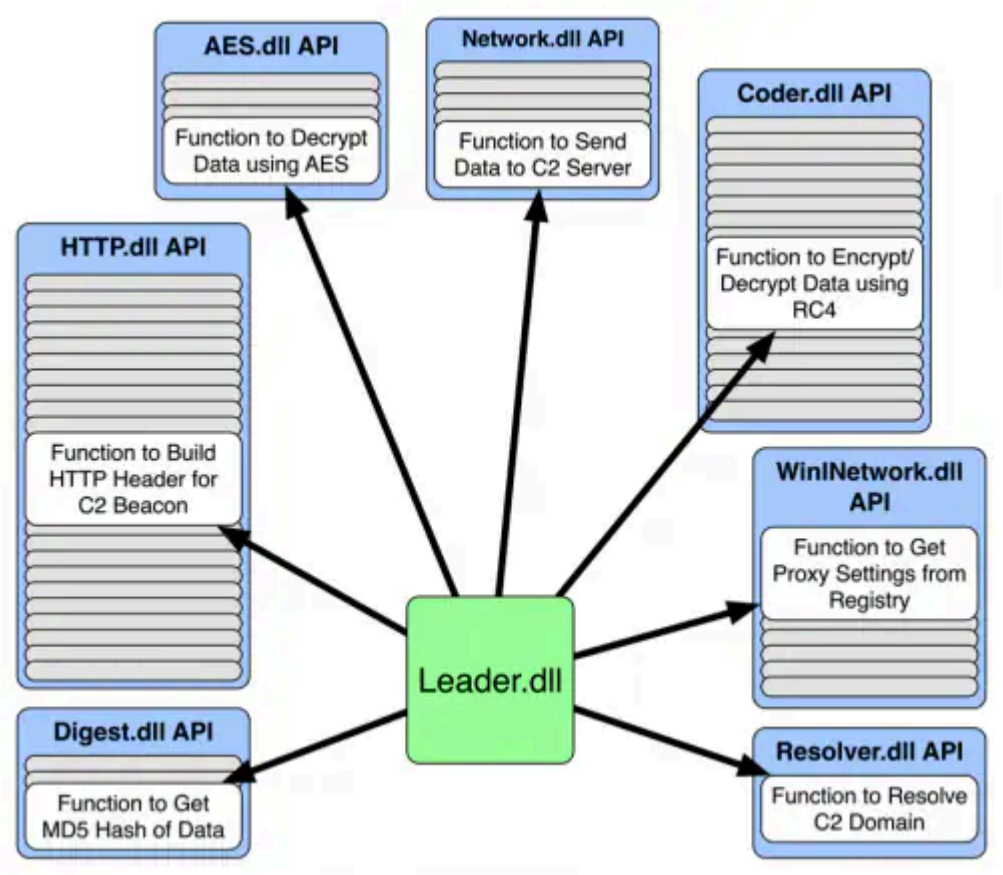


Figure 5 Visualization of Leader using other Bookworm Modules' API Functions to Carry out Tasks

By using this type of modular framework, the developers of Bookworm have made static analysis of the Trojan quite challenging. To perform static analysis of Bookworm, an analyst must recreate the structure used by Leader to store the API functions of each DLL and apply them throughout the entire Trojan. Without performing this task, an analyst would be unable to determine which API function Leader calls within the supporting DLLs. For example, Figure 6 below shows a code block within Leader that is responsible for encrypting a buffer using functions within the AES module; however, the red, blue, and green boxes show calls to functions based on an offset in a structure.

Figure 6 Bookworm Calling API Functions using an Offset to its Structure

At first glance, an analyst would be unable to determine the purpose of the code block displayed in Figure 6, as the functions called are not readily apparent. By creating a structure and populating it with the correct API functions however, an analyst can determine the API functions called in this code block. In Figure 7 below, the red, blue and green boxes show calls to three functions within the AES module that allow Leader to encrypt data using the AES algorithm.

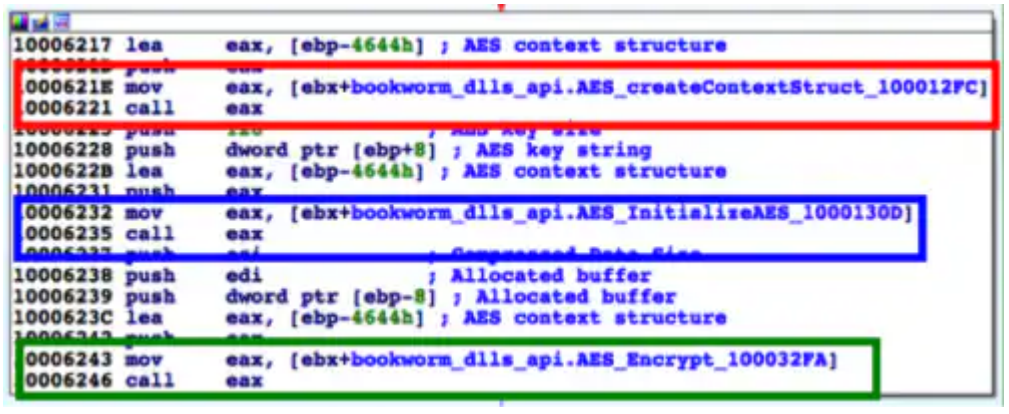


Figure 7 Applying Bookworm's API Structure Exposes the API Functions Called

Not only does this modular approach require an analyst to create a structure, but it also takes away an analyst's ability to use cross references (XREFs) on the API functions within the supporting modules. Using XREFs during static analysis is a common technique to quickly find where functions of interest are called. An analyst cannot use this method to find where Leader is calling specific Bookworm APIs because the functions are not called directly;

rather they are called based on the structure offset. We are unsure if the developers of Bookworm created this as an analysis hurdle, but it certainly contributes to anti-analysis tactics.

Bookworm's Capabilities

Although the developers of Bookworm have included only keylogging functionality in Bookworm as a core ability, as suggested in Table 1, several of the embedded DLLs provide Leader with cryptographic and hashing functions, while others support Leader's ability to communicate with its C2 server. The developers designed Bookworm to be a modular Trojan not limited to just the initial architecture of the Trojan, as Bookworm can also load additional modules provided by the C2 server. The ability to load additional modules from the C2 extends the capabilities of the Trojan to accommodate any activities the threat actors need to carry out on the compromised system.

Key Logging Functionality

The KBLogger.dll module, which we will refer to as KBLogger, provides key logging and clipboard grabbing functionality and is the only Bookworm module that does not provide Leader with API functions. Instead, Leader creates a new process "C:\WINDOWS\System32\dllhost.exe –user" that it injects itself into and uses to execute the KBLogger functionality.

KBLogger runs on its own by creating a new window called "DolefulClass<username><PID>", which is hidden so it is invisible to the user. The new window executes code that will create the following folder to store files that contain logged keystrokes and stolen clipboard contents:

%AllUsersProfile%\Application Data\Microsoft\Crypto\RSA\MachineKeys\<crc32 hash>bk

KBLogger captures keystrokes typed by the user and saves them to a file in the folder above. KBLogger also specifically monitors for the keystroke combinations "Control + C", "Control + V" and "Control + X" that it uses as triggers to copy the contents of the clipboard to a file. The keystrokes and clipboard contents are encrypted before KBlogger saves them to the file system using the RC4 algorithm using a key derived from the value at the following registry key:

HKLM\SOFTWARE\Microsoft\Internet Explorer\Registration\ProductID

KBLogger will generate the key for the RC4 algorithm by using XOR and an eight-byte key (specifically 0x6E, 0x30, 0xD0, 0x30, 0xB9, 0x30, 0xB1, 0x30) on the value of the above registry key. KBLogger creates files with the naming format "<username XOR with 2 byte key 0x5878>_<seconds since EPOCH>" to store the captured keystrokes and clipboard. For example, on November 3, 2015, we saw the following file created during by KBLogger on an analysis system with a username of "administrator":

191c351136112b0c2a192c172a_56391E90

C2 Communications and Additional Modules

Bookworm uses a state machine to keep track of and carry out communications between the compromised system and the C2 server. Also, Bookworm uses a combination of encryption and compression algorithms to obfuscate

the traffic between the system and C2 server. We have seen the following encryption and compression methods used at various stages and in differing combinations in the C2 communications:

- RC4
- AES
- XOR with 0x5a
- LZO

Bookworm first creates an HTTP request that acts as a network beacon to notify the C2 of the compromised system. The initial network beacon is either an HTTP GET or POST request, which varies between Bookworm samples. Unit 42 analyzed the contents of a beacon seen in Figure 8, which was sent to a URL that follows a structure of "http://<c2 server>:<port>/0<crc32 hash of tick count><tick count><encrypted data>". The encrypted data in the URL is a 32 character string (16 hexadecimal bytes) created by RC4 and AES encrypting an empty data buffer using the tick count and crc32 hash of the tick count as a key.

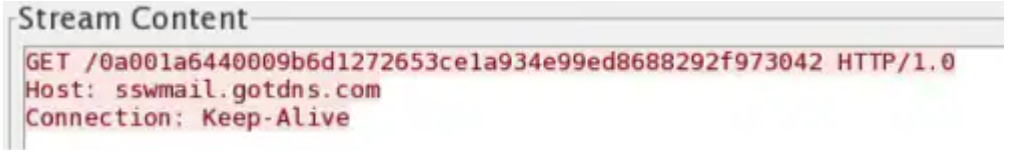


Figure 8 Initial C2 Beacon from Bookworm sample 8ae2468d3f208d07fb47ebb1e0e297d7

Subsequent HTTP POST requests from Bookworm to the C2 server include campaign identifier and system uptime. The data in the HTTP POST has a structure of “\x03\x04<campaign code>\x00<system uptime>”. Leader then compresses this string using the LZO compression algorithm and compares the compressed length to the original string length. Leader uses the shorter of the two strings and appends it to a DWORD that is the size of the data and encrypts the combined string using AES with a key of "0123456789" and XOR with a key of 0x5a. We believe the threat actors use the data in these POST requests to map the compromised system to the appropriate campaign and to filter out analysis systems. In our follow-up blog we will further discuss the campaign codes identified in our analysis.

The threat actors also deliver additional modules to Bookworm via C2 communications. To load additional modules into Bookworm, Leader parses C2 responses for data that have the following structure:

<MD5 of cleartext of module><encrypted module>

Leader skips the first 16 bytes and RC4 decrypts the remaining data with the key "0123456789". It then computes the MD5 hash of the resulting cleartext and checks this hash with the MD5 in the first 16 bytes of the C2 response data to see if it is the same. If the MD5 hashes match, then the code will carry out further checks on the decrypted data to determine if the data is a new DLL for Leader to load as an additional module. At this time, Unit 42 has not seen a Bookworm C2 server provide additional modules via network communications. By performing static analysis on Leader.dll, we know that Leader will load the additional modules and attempt to call "ProgramStartup" and "QueryBuffer" functions exported by the DLLs.

Conclusion

While we did not discuss the surrounding attacks using Bookworm in detail, we have observed threat actors deploying Bookworm primarily in attacks on targets in Thailand. The developers of Bookworm have gone to great lengths to create a modular framework that is very flexible through its ability to run additional modules directly from its C2 server. Not only is this tool highly capable, but it also requires a very high level of effort to analyze due to its modular architecture and its use of API functions within the additional modules. We believe that it is likely threat actors will continue development Bookworm, and will continue to use it for the foreseeable future.

Indicators of Compromise

Known Bookworm C2 Servers

- bkmail.blogdns[.]com
- debain.servehttp[.]com
- linuxdns.sytes[.]net
- news.nhknews[.]hk
- sswmail.gotdns[.]com
- sswwmail.gotdns[.]com
- sysnc.sytes[.]net
- systeminfothai[.]gotdns.ch
- thailandbbs.ddns[.]net
- ubuntudns.sytes[.]net
- web12.nhknews[.]hk

Bookworm Smart Install Maker Samples

- 0f41c853a2d522e326f2c30b4b951b04
- 8ae2468d3f208d07fb47ebb1e0e297d7
- 35755a6839f3c54e602d777cd11ef557
- 87d71401e2b8978c2084eb9a1d59c172
- 599b6e05a38329081b80a461b57cec37
- ba1aea40182861e1d1de8c0c2ae78cb7
- de1595a7585219967a87a909f38acaa2
- f8c8c6683d6ca880293f7c1a78d7f8ce
- 0b4ad1bd093e0a2eb8968e308e900180
- cba74e507e9741740d251b1fb34a1874
- fcd68032c39cca3385c539ea38914735
- 3e69c34298a8fd5169259a2fef506d63

Bookworm Self-Extracting RAR Samples

- 04d63e2a3da0a171e5c15d8e904387b9
- 0d57d2bef1296be62a3e791bfad33bcd
- 4389fc820d0edd96bac26fa0b7448aee

- 74c293acdda0d2c3b5087763dae27ec6
- b030c619bb24804cbcc05065530fcf2e
- 29df124f370752a87b3426dcad539ec6
- 9df45e8d8619e234d0449daf2f617ba3
- 40f1b160b88ff98934017f3f1e7879a5
- 210816c8bde338bf206f13bb923327a1
- 187cdb58fbc30046a35793818229c573
- 0b4ad1bd093e0a2eb8968e308e900180
- 499ccc8d6d7c08e135a91928ccc2fd7a
- 5e4852c8e5ef3cbceb69a9bc3d554d6c
- 5282b503b061eaa843c0bcda1c74b14f

Updated April 4, 2024, at 9:15 a.m. PT to correct Table 1.
