Behind the CARBANAK Backdoor | Mandiant

By Mandiant

Published: 2017-06-12 · Archived: 2026-04-05 15:50:17 UTC

Written by: James T. Bennett, Barry Vengerik

In this blog, we will take a closer look at the powerful, versatile backdoor known as CARBANAK (aka *Anunak*). Specifically, we will focus on the operational details of its use over the past few years, including its configuration, the minor variations observed from sample to sample, and its evolution. With these details, we will then draw some conclusions about the operators of CARBANAK. For some additional background on the CARBANAK backdoor, see the papers by Kaspersky and Group-IB and Fox-It.

Technical Analysis

Before we dive into the meat of this blog, a brief technical analysis of the backdoor is necessary to provide some context. CARBANAK is a full-featured backdoor with data-stealing capabilities and a plugin architecture. Some of its capabilities include key logging, desktop video capture, VNC, HTTP form grabbing, file system management, file transfer, TCP tunneling, HTTP proxy, OS destruction, POS and Outlook data theft and reverse shell. Most of these data-stealing capabilities were present in the oldest variants of CARBANAK that we have seen and some were added over time.

Monitoring Threads

The backdoor may optionally start one or more threads that perform continuous monitoring for various purposes, as described in Table 1.

Thread Name	Description
Key logger	Logs key strokes for configured processes and sends them to the command and control (C2) server
Form grabber	Monitors HTTP traffic for form data and sends it to the C2 server
POS monitor	Monitors for changes to logs stored in C:\NSB\Coalition\Logs and nsb.pos.client.log and sends parsed data to the C2 server
PST monitor	Searches recursively for newly created Outlook personal storage table (PST) files within user directories and sends them to the C2 server
HTTP proxy monitor	Monitors HTTP traffic for requests sent to HTTP proxies, saves the proxy address and credentials for future use

Table 1: Monitoring threads

Commands

In addition to its file management capabilities, this data-stealing backdoor supports 34 commands that can be received from the C2 server. After decryption, these 34 commands are plain text with parameters that are space delimited much like a command line. The command and parameter names are hashed before being compared by the binary, making it difficult to recover the original names of commands and parameters. Table 2 lists these commands.

Command Hash	Command Name	Description
0x0AA37987	loadconfig	Runs each command specified in the configuration file (see the Configuration section).
0x007AA8A5	state	Updates the state value (see the Configuration section).
0x007CFABF	video	Desktop video recording
0x06E533C4	download	Downloads executable and injects into new process
0x00684509	ammyy	Ammyy Admin tool
0x07C6A8A5	update	Updates self
0x0B22A5A7		Add/Update klgconfig (analysis incomplete)
0x0B77F949	httpproxy	Starts HTTP proxy
0x07203363	killos	Renders computer unbootable by wiping the MBR
0x078B9664	reboot	Reboots the operating system
0x07BC54BC	tunnel	Creates a network tunnel
0x07B40571	adminka	Adds new C2 server or proxy address for pseudo-HTTP protocol
0x079C9CC2	server	Adds new C2 server for custom binary protocol
0x0007C9C2	user	Creates or deletes Windows user account
0x000078B0	rdp	Enables concurrent RDP (analysis incomplete)
0x079BAC85	secure	Adds Notification Package (analysis incomplete)
0x00006ABC	del	Deletes file or service
0x0A89AF94	startcmd	Adds command to the configuration file (see the Configuration section)

0x079C53BD	runmem	Downloads executable and injects directly into new process
0x0F4C3903	logonpasswords	Send Windows accounts details to the C2 server
0x0BC205E4	screenshot	Takes a screenshot of the desktop and sends it to the C2 server
0x007A2BC0	sleep	Backdoor sleeps until specified date
0x0006BC6C	dupl	Unknown
0x04ACAFC3		Upload files to the C2 server
0x00007D43	vnc	Runs VNC plugin
0x09C4D055	runfile	Runs specified executable file
0x02032914	killbot	Uninstalls backdoor
0x08069613	listprocess	Returns list of running processes to the C2 server
0x073BE023	plugins	Change C2 protocol used by plugins
0x0B0603B4		Download and execute shellcode from specified address
0x0B079F93	killprocess	Terminates the first process found specified by name
0x00006A34	cmd	Initiates a reverse shell to the C2 server
0x09C573C7	runplug	Plugin control
0x08CB69DE	autorun	Updates backdoor

Table 2: Supported Commands

Configuration

A configuration file resides in a file under the backdoor's installation directory with the .bin extension. It contains commands in the same form as those listed in Table 2 that are automatically executed by the backdoor when it is started. These commands are also executed when the loadconfig command is issued. This file can be likened to a startup script for the backdoor. The state command sets a global variable containing a series of Boolean values represented as ASCII values '0' or '1' and also adds itself to the configuration file. Some of these values indicate which C2 protocol to use, whether the backdoor has been installed, and whether the PST monitoring thread is running or not. Other than the state command, all commands in the configuration file are identified by their hash's decimal value instead of their plain text name. Certain commands, when executed, add themselves to the configuration so they will persist across (or be part of) reboots. The loadconfig and state commands are executed during initialization, effectively creating the configuration file if it does not exist and writing the state command to it.

Figure 1 and Figure 2 illustrate some sample, decoded configuration files we have come across in our investigations.

```
state 0011
adminka force <IP address>:443
```

Figure 1: Configuration file that adds new C2 server and forces the data-stealing backdoor to use it

```
state 0011
tunnel 441 <IP address>:80
tunnel 442 <IP address>:443
tunnel 440 <IP address>:443
video online
```

Figure 2: Configuration file that adds TCP tunnels and records desktop video

Command and Control

CARBANAK communicates to its C2 servers via pseudo-HTTP or a custom binary protocol.

Pseudo-HTTP Protocol

Messages for the pseudo-HTTP protocol are delimited with the '|' character. A message starts with a host ID composed by concatenating a hash value generated from the computer's hostname and MAC address to a string likely used as a campaign code. Once the message has been formatted, it is sandwiched between an additional two fields of randomly generated strings of upper and lower case alphabet characters. An example of a command polling message and a response to the listprocess command are given in Figure 3 and Figure 4, respectively.

```
xVjMTzzJthISUXrHhrSEtaUvnMlC|myserih0cf3f75290c7e5905|lNCjyGFgGCAexwNgv
```

Figure 3: Example command polling message

```
qIPhfIFGUXDQXDMbgnlFmpZU|myserih0cf3f75290c7e5905|data=listprocess|process=svc
host.exe|idprocess=4294967295|gmGKiJcHQzYLTje
```

Figure 4: Example command response message

Messages are encrypted using Microsoft's implementation of RC2 in CBC mode with PKCS#5 padding. The encrypted message is then Base64 encoded, replacing all the '/' and '+' characters with the '.' and '-' characters, respectively. The eight-byte initialization vector (IV) is a randomly generated string consisting of upper and lower case alphabet characters. It is prepended to the encrypted and encoded message.

The encoded payload is then made to look like a URI by having a random number of '/' characters inserted at random locations within the encoded payload. The malware then appends a script extension (php, bml, or cgi) with a random number of random parameters or a file extension from the following list with no parameters: gif, jpg, png, htm, html, php.

This URI is then used in a GET or POST request. The body of the POST request may contain files contained in the cabinet format. A sample GET request is shown in Figure 5.

```
GET       /CybftIPMyw/vt/rcAic/8c7FIc/1iVpvYtU86u-XdvXTD.cgi?Vbkd8HN=sPknIWnZ-
X&33m=kgxr&8JxTUp7oupRViD97=gtU6yG706rj1eL-lYRmllCd HTTP/1.1
Host: <hostname>
User-Agent: <system user-agent>
Accept: */*
```

Figure 5: Sample pseudo-HTTP beacon

The pseudo-HTTP protocol uses any proxies discovered by the HTTP proxy monitoring thread or added by the adminka command. The backdoor also searches for proxy configurations to use in the registry at HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings and for each profile in the Mozilla Firefox configuration file at %AppData%\Mozilla\Firefox\\prefs.js.

Custom Binary Protocol

Figure 6 describes the structure of the malware's custom binary protocol. If a message is larger than 150 bytes, it is compressed with an unidentified algorithm. If a message is larger than 4096 bytes, it is broken into compressed chunks. This protocol has undergone several changes over the years, each version building upon the previous version in some way. These changes were likely introduced to render existing network signatures ineffective and to make signature creation more difficult.

```
typedef struct binaryProtocolMsg{
    uint8_t cmdId;
    uint8_t flag;
    uint32_t messageLength;
    uint16_t chunkLength;
    uint16_t chunkIndex;
    uint8_t chunkFlag; //compressed, more chunks coming
    uint32_t unknown;
    uint8_t hdrXORKey1[4]; //random, unused by some versions
    uint8_t hdrXORKey2[5]; //random, unused by some versions
    uint8_t chunkData[chunkLength];
}
```

Figure 6: Binary protocol message format

Version 1

In the earliest version of the binary protocol, we have discovered that the message bodies that are stored in the field are simply XORed with the host ID. The initial message is not encrypted and contains the host ID.

Version 2

Rather than using the host ID as the key, this version uses a random XOR key between 32 and 64 bytes in length that is generated for each session. This key is sent in the initial message.

Version 3

Version 3 adds encryption to the headers. The first 19 bytes of the message headers (up to the field) are XORed with a five-byte key that is randomly generated per message and stored in the field. If the field of the message header is greater than one, the XOR key used to encrypt message bodies is iterated in reverse when encrypting and decrypting messages.

Version 4

This version adds a bit more complexity to the header encryption scheme. The headers are XOR encrypted with and combined and reversed.

Version 5

Version 5 is the most sophisticated of the binary protocols we have seen. A 256-bit AES session key is generated and used to encrypt both message headers and bodies separately. Initially, the key is sent to the C2 server with the entire message and headers encrypted with the RSA key exchange algorithm. All subsequent messages are encrypted with AES in CBC mode. The use of public key cryptography makes decryption of the session key infeasible without the C2 server's private key.

The Roundup

We have rounded up 220 samples of the CARBANAK backdoor and compiled a table that highlights some interesting details that we were able to extract. It should be noted that in most of these cases the backdoor was embedded as a packed payload in another executable or in a weaponized document file of some kind. The MD5 hash is for the original executable file that eventually launches CARBANAK, but the details of each sample were extracted from memory during execution. This data provides us with a unique insight into the operational aspect of CARBANAK and can be downloaded here.

Protocol Evolution

As described earlier, CARBANAK's binary protocol has undergone several significant changes over the years. Figure 7 illustrates a rough timeline of this evolution based on the compile times of samples we have in our collection. This may not be entirely accurate because our visibility is not complete, but it gives us a general idea as to when the changes occurred. It has been observed that some builds of this data-stealing backdoor use outdated versions of the protocol. This may suggest multiple groups of operators compiling their own builds of this data-stealing backdoor independently.

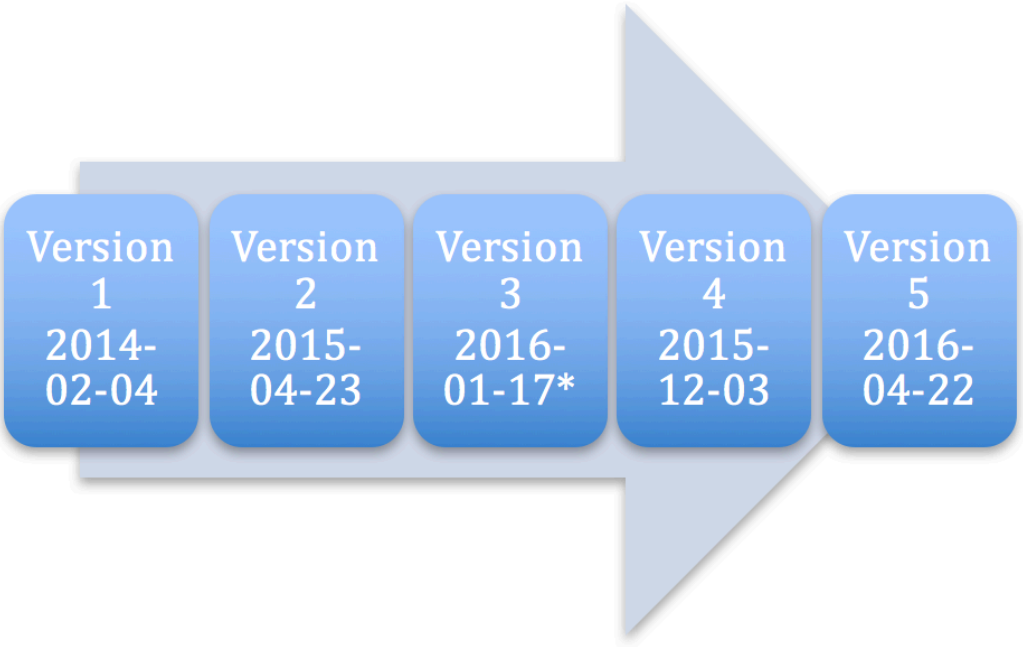

Figure 7: Timeline of binary protocol versions

*It is likely that we are missing an earlier build that utilized version 3.

Build Tool

Most of CARBANAK's strings are encrypted in order to make analysis more difficult. We have observed that the key and the cipher texts for all the encrypted strings are changed for each sample that we have encountered, even amongst samples with the same compile time. The RC2 key used for the HTTP protocol has also been observed to change among samples with the same compile time. These observations paired with the use of campaign codes that must be configured denote the likely existence of a build tool.

Rapid Builds

Despite the likelihood of a build tool, we have found 57 unique compile times in our sample set, with some of the compile times being quite close in proximity. For example, on May 20, 2014, two builds were compiled approximately four hours apart and were configured to use the same C2 servers. Again, on July 30, 2015, two builds were compiled approximately 12 hours apart.

What changes in the code can we see in such short time intervals that would not be present in a build tool? In one case, one build was programmed to execute the runmem command for a file named wi.exe while the other was not. This command downloads an executable from the C2 and directly runs it in memory. In another case, one build was programmed to check for the existence of the domain blizko.net in the trusted sites list for Internet Explorer while the other was not. Blizko is an online money transfer service. We have also seen that different monitoring threads from Table 1 are enabled from build to build. These minor changes suggest that the code is quickly modified and compiled to adapt to the needs of the operator for particular targets.

Campaign Code and Compile Time Correlation

In some cases, there is a close proximity of the compile time of a CARBANAK sample to the month specified in a particular campaign code. Figure 8 shows some of the relationships that can be observed in our data set.

Campaign Code	Compile Date
Aug	7/30/15
dec	12/8/14
julyc	7/2/16
jun	5/9/15
june	5/25/14
june	6/7/14
junevnc	6/20/14
juspam	7/13/14
juupd	7/13/14
may	5/20/14
may	5/19/15
ndjun	6/7/16
SeP	9/12/14
spamaug	8/1/14
spaug	8/1/14

Figure 8: Campaign code to compile time relationships

Recent Updates

Recently, 64 bit variants of the backdoor have been discovered. We shared details about such variants in a recent blog post. Some of these variants are programmed to sleep until a configured activation date when they will become active.

History

The “Carbanak Group”

Much of the publicly released reporting surrounding the CARBANAK malware refers to a corresponding "Carbanak Group", who appears to be behind the malicious activity associated with this data-stealing backdoor. FireEye iSIGHT Intelligence has tracked several separate overarching campaigns employing the CARBANAK tool and other associated backdoors, such as DRIFTPIN (aka *Toshliph*). With the data available at this time, it is unclear how interconnected these campaigns are – if they are all directly orchestrated by the same criminal group, or if these campaigns were perpetrated by loosely affiliated actors sharing malware and techniques.

FIN7

In all Mandiant investigations to date where the CARBANAK backdoor has been discovered, the activity has been attributed to the FIN7 threat group. FIN7 has been extremely active against the U.S. restaurant and hospitality industries since mid-2015.

FIN7 uses CARBANAK as a post-exploitation tool in later phases of an intrusion to cement their foothold in a network and maintain access, frequently using the video command to monitor users and learn about the victim network, as well as the tunnel command to proxy connections into isolated portions of the victim environment. FIN7 has consistently utilized legally purchased code signing certificates to sign their CARBANAK payloads. Finally, FIN7 has leveraged several new techniques that we have not observed in other CARBANAK related activity.

We have covered recent FIN7 activity in previous public blog posts:

- FIN7 Spear Phishing Campaign Targets Personnel Involved in SEC Filings
- FIN7 Evolution and the Phishing LNK
- To SDB, Or Not To SDB: FIN7 Leveraging Shim Databases for Persistence

The FireEye iSIGHT Intelligence MySIGHT Portal contains additional information on our investigations and observations into FIN7 activity.

Widespread Bank Targeting Throughout the U.S., Middle East and Asia

Proofpoint initially reported on a widespread campaign targeting banks and financial organizations throughout the U.S. and Middle East in early 2016. We identified several additional organizations in these regions, as well as in Southeast Asia and Southwest Asia being targeted by the same attackers.

This cluster of activity persisted from late 2014 into early 2016. Most notably, the infrastructure utilized in this campaign overlapped with LAZIOK, NETWIRE and other malware targeting similar financial entities in these regions.

DRIFTPIN

DRIFTPIN (aka *Spy.Agent.ORM*, and *Toshliph*) has been previously associated with CARBANAK in various campaigns. We have seen it deployed in initial spear phishing by FIN7 in the first half of 2016. Also, in late 2015, ESET reported on CARBANAK associated attacks, detailing a spear phishing campaign targeting Russian and Eastern European banks using DRIFTPIN as the malicious payload. Cyphort Labs also revealed that variants of

DRIFTPIN associated with this cluster of activity had been deployed via the RIG exploit kit placed on two compromised Ukrainian banks' websites.

FireEye iSIGHT Intelligence observed this wave of spear phishing aimed at a large array of targets, including U.S. financial institutions and companies associated with Bitcoin trading and mining activities. This cluster of activity continues to be active now to this day, targeting similar entities. Additional details on this latest activity are available on the FireEye iSIGHT Intelligence MySIGHT Portal.

Earlier CARBANAK Activity

In December 2014, Group-IB and Fox-IT released a report about an organized criminal group using malware called "Anunak" that has targeted Eastern European banks, U.S. and European point-of-sale systems and other entities. Kaspersky released a similar report about the same group under the name "Carbanak" in February 2015. The name "Carbanak" was coined by Kaspersky in this report – the malware authors refer to the backdoor as Anunak.

This activity was further linked to the 2014 exploitation of ATMs in Ukraine. Additionally, some of this early activity shares a similarity with current FIN7 operations – the use of Power Admin PAExec for lateral movement.

Conclusion

The details that can be extracted from CARBANAK provide us with a unique insight into the operational details behind this data-stealing malware. Several inferences can be made when looking at such data in bulk as we discussed above and are summarized as follows:

1. Based upon the information we have observed, we believe that at least some of the operators of CARBANAK either have access to the source code directly with knowledge on how to modify it or have a close relationship to the developer(s).
2. Some of the operators may be compiling their own builds of the backdoor independently.
3. A build tool is likely being used by these attackers that allows the operator to configure details such as C2 addresses, C2 encryption keys, and a campaign code. This build tool encrypts the binary's strings with a fresh key for each build.
4. Varying campaign codes indicate that independent or loosely affiliated criminal actors are employing CARBANAK in a wide-range of intrusions that target a variety of industries but are especially directed at financial institutions across the globe, as well as the restaurant and hospitality sectors within the U.S.

Posted in

- Threat Intelligence
- Security & Identity
