Revisiting BLISTER: New development of the BLISTER loader

By Salim Bitam, Daniel Stepanic

Published: 2023-08-24 · Archived: 2026-04-05 19:01:50 UTC

Preamble

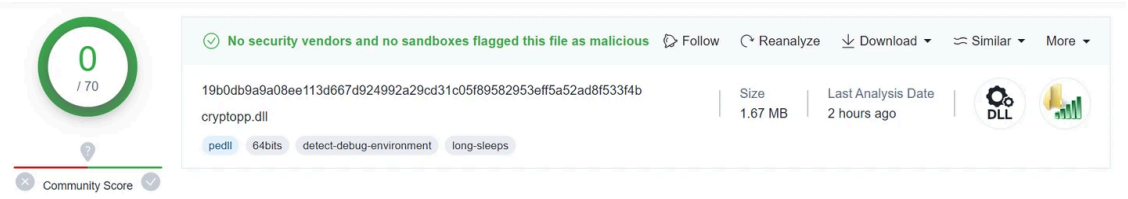
In a fast-paced and ever-changing world of cybercrime threats, the tenacity and adaptability of malicious actors is a significant concern. BLISTER, a malware loader initially discovered by Elastic Security Labs in 2021 and associated with financially-motivated intrusions, is a testament to this trend as it continues to develop additional capabilities. Two years after its initial discovery, BLISTER continues to receive updates while flying under the radar, gaining momentum as an emerging threat. Recent findings from Palo Alto's Unit 42 describe an updated SOCGHOLISH infection chain used to distribute BLISTER and deploy a payload from MYTHIC, an open-source Command and Control (C2) framework.

Key takeaways

- Elastic Security Labs has been monitoring malware loader BLISTER ramping up with new changes, and ongoing development with signs of imminent threat activity
- New BLISTER update includes keying feature that allows for precise targeting of victim networks and lowers exposure within VM/sandbox environments
- BLISTER now integrates techniques to remove any process instrumentation hook and has modified its configuration with multiple revisions, now encompassing additional fields and flags.

Overview

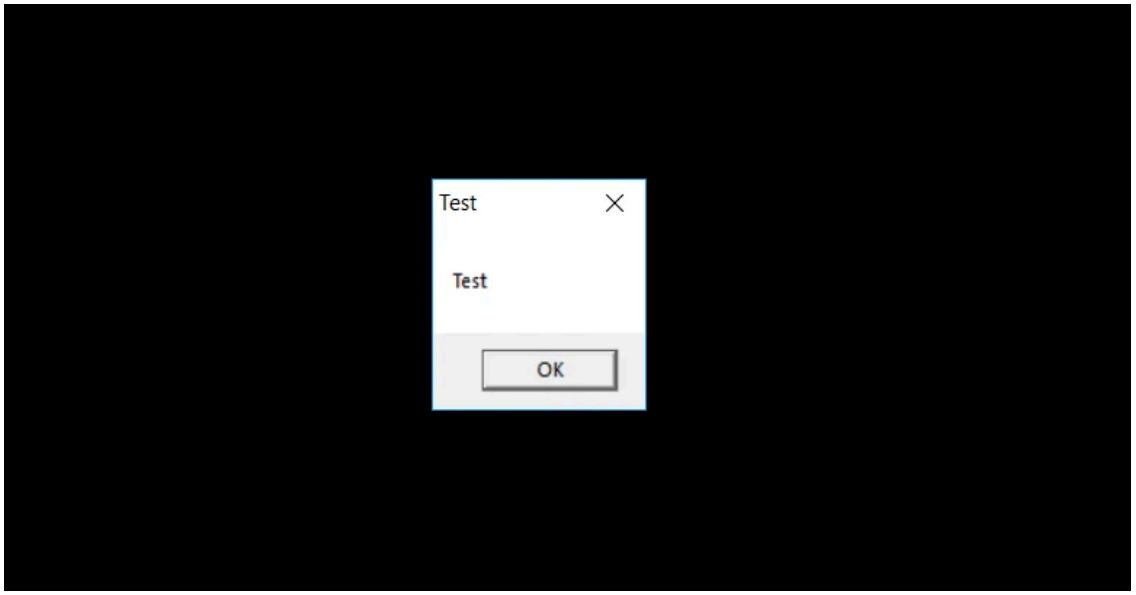
Our research uncovered new functionality that was previously absent within the BLISTER family, indicating ongoing development. However, the malware authors continue to use a distinctive technique of embedding malicious code in otherwise legitimate applications. This approach superficially appears successful, given the low rates of detection for many vendors as seen in VirusTotal. The significant amount of benign code and use of encryption to protect the malicious code are likely two factors impacting detection.



Example of BLISTER detection rates on initial upload

Recently, Elastic Security Labs has observed many new BLISTER loaders in the wild. After analyzing various samples, it's clear that the malware authors have made some changes and have been watching the antivirus

industry closely. In one sample from early June, we can infer that the authors were testing with a non-production loader that displays a Message Box displaying the strings “Test”.



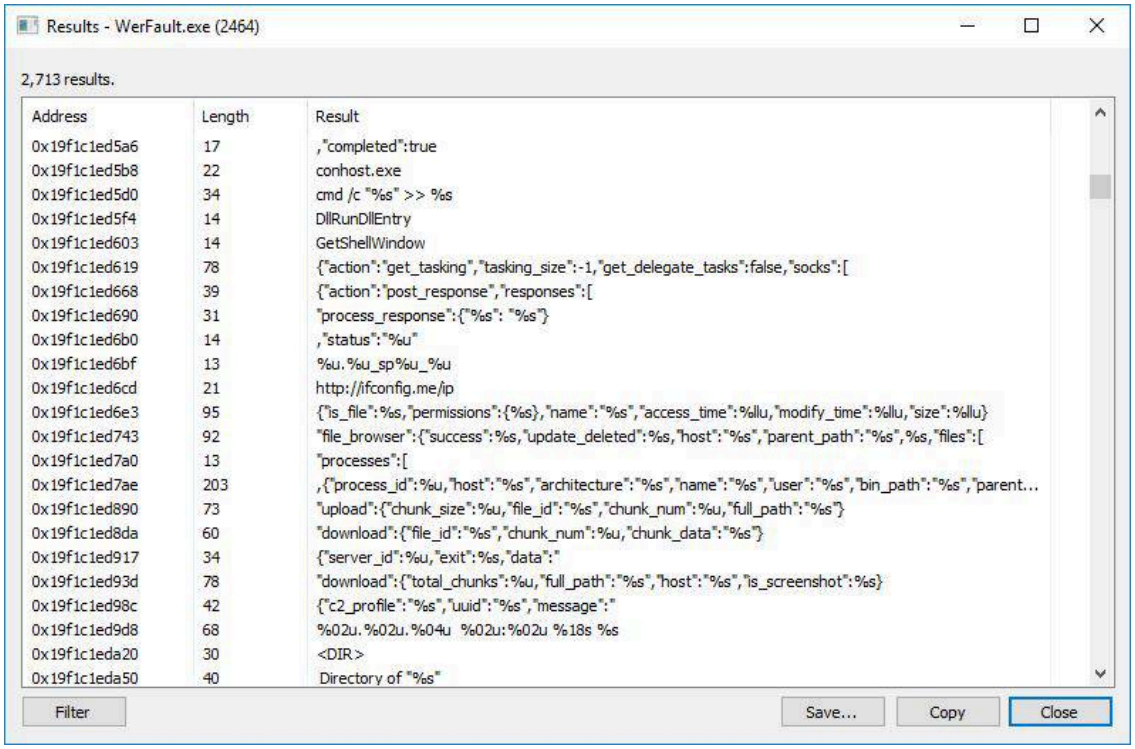
BLISTER payload with Message Box test

Readers can see a disassembled view of this functionality below.

```
                        ; int __fastcall main(int argc, const char **argv, const char **envp)
                        main proc near
EC 28                   sub     rsp, 28h
C9                      xor     r9d, r9d        ; uType
05 9A E4 02 00          lea     r8, Caption     ; "Test"
15 A3 E4 02 00          lea     rdx, Text       ; "Test"
                        xor     ecx, ecx        ; hWnd
B3 67 01 00             call    cs:MessageBoxW
                        xor     eax, eax
C4 28                   add     rsp, 28h
                        retn
                        main endp
```

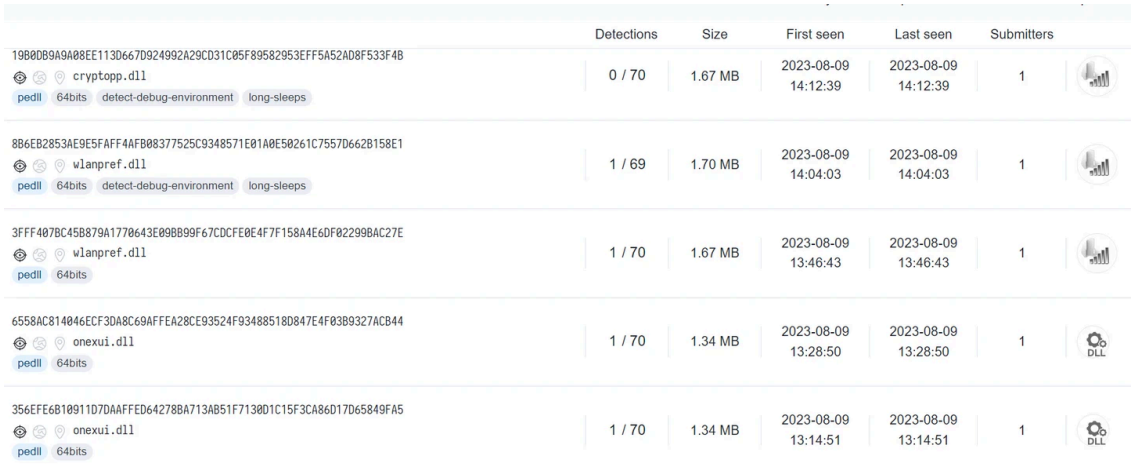
BLISTER testing payloads with Message Box

By the end of July, we observed campaigns involving a new BLISTER loader that targeted victim organizations to deploy the MYTHIC implant.



MYTHIC running inside injected WerFault process

At the time of this writing, Elastic Security Labs is seeing a stream of BLISTER samples which deploy MYTHIC and have very low rates of detection.



Wave of BLISTER samples in August 2023

Comparative analyses

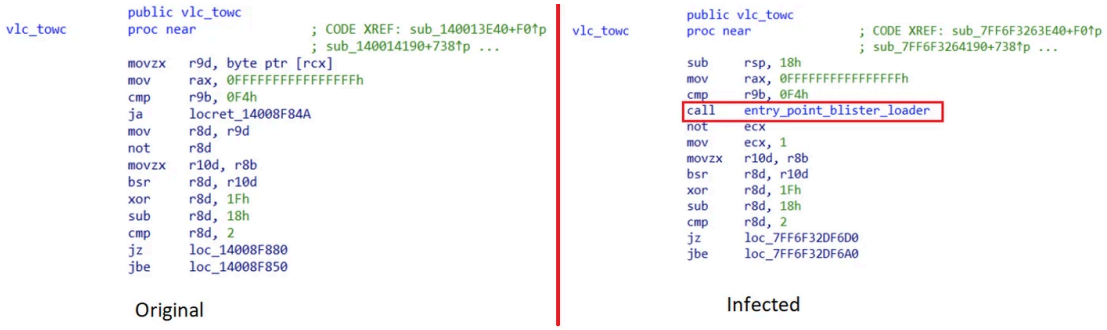
Smuggling malicious code

The authors behind BLISTER employ a consistent strategy of embedding BLISTER's malicious code within a legitimate library. The most recent variants of this loader have targeted the VLC Media Player library to smuggle their malware into victim environments. This blend of benign and malicious code seems effective at defeating some kinds of machine-learning models.

property	value
md5	7E11229465A8FDF879912C31B17D30A9
sha1	CD1FABB1C38AE1ED5689456AF46D96F2706E1E2D
sha256	46ED2D6C9C183B09F051D253DA1F5C5F711680F9460C6109B2106E07D2630D6C
language	English-US
code-page	ANSI Latin 1
CompanyName	VideoLAN
ProductName	VLC media player
ProductVersion	3,0,16,0
FileVersion	3.0.16
FileDescription	VLC media player
LegalCopyright	Copyright © 1996-2021 VideoLAN and VLC Authors
LegalTrademarks	VLC media player, VideoLAN and x264 are registered trademarks from VideoLAN

Meta data of BLISTER sample

The following is a comparison between a legitimate VLC DLL and one that is infected with BLISTER's code. In the infected sample, the entry point that references malicious code has been indicated in red. This methodology is similar to prior BLISTER variants.



Comparison between original and patched VLC library

Different hashing algorithm

One of the changes implemented since our last write-up is the adoption of a different hashing algorithm used in the core and in the loader part of BLISTER. While the previous version used simple logic to shift bytes, this new version includes a hard-coded seed with XOR and multiplication operations. Researchers speculate that changing the hashing approach helps to evade antimalware products that rely on YARA signatures.

```
__int16 _CX; // cx
unsigned int v3; // [rsp+0h] [rbp-18h]

v3 = 0x78F1E5BF;
while ( *a1 )
{
  _CX = *a1 ^ v3;
  __asm { rcl     cl, 56h }
  v3 = ((0x5BD1E995 * (*a1 ^ v3)) >> 15) ^ (0x5BD1E995 * (*a1 ^ v3));
  ++a1;
}
return v3;
}
```

Disassembled hashing algorithm

Configuration retrieval

Following the decryption of malicious code by the BLISTER'd loader, it employs an identical memory scanning method to identify the configuration data blob. This is accomplished by searching for a predetermined, hardcoded memory pattern. A notable contrast from the earlier iteration of BLISTER lies in the fact that the configuration is now decrypted in conjunction with the core code, rather than being treated as a separate entity.

Environmental keying

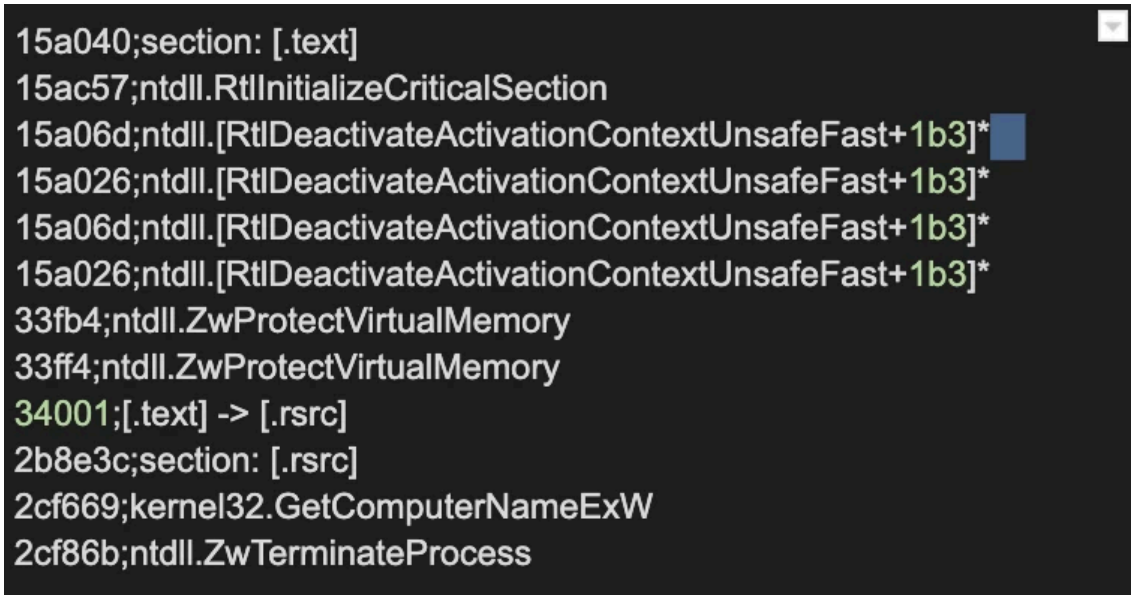
A recent addition to BLISTER is the capability to exclusively execute on designated machines. This behavior is activated by configuring the appropriate flag within the malware's configuration. Subsequently, the malware proceeds to extract the machine's domain name using the `GetComputerNameExW` Windows API. Following this, the domain name is hashed using the previously mentioned algorithm, and the resulting hash is then compared to a hash present in the configuration. This functionality is presumably deployed for the purpose of targeted attacks or for testing scenarios, ensuring that the malware refrains from infecting unintended systems such as those employed by malware researchers.

```
_BOOL8 __fastcall fxh::check_domain_name(struct_000 *a1, int domain_hash_config)
{
  const CHAR *kernel32_module; // rax
  FARPROC kernel32_dll_GetComputerNameExW; // rax
  int nSize; // [rsp+0h] [rbp-228h] BYREF
  int status_; // [rsp+4h] [rbp-224h]
  __int16 dns_domain[268]; // [rsp+10h] [rbp-218h] BYREF

  nSize = 0x20A;
  memset_(dns_domain, 0, 0x20Aui64);
  kernel32_module = fxh::utils::GetModuleHandle(a1, kernel32_dll_kernel32_dll);
  kernel32_dll_GetComputerNameExW = fxh::utils::GetProcAddress(a1, kernel32_module, kernel32_dll_GetComputerNameExW);
  status_ = (kernel32_dll_GetComputerNameExW)(ComputerNameDnsDomain, dns_domain, &nSize);
  return status_ != 1 || strlen_(dns_domain) <= 0 || hashing_algo(dns_domain) != domain_hash_config;
}
```

Environmental keying feature

One of the few malware analysis tools capable of quickly exposing this behavior is the awesome Tiny Tracer utility by hasherezade. We've included an excerpt from Tiny_Tracer below which captures the BLISTER process immediately terminating after the `GetComputerNameExW` validation is performed in a sandboxed analysis VM.



TinyTracer logs

Time-based anti-debugging feature

Similar to its predecessors, the malware incorporates a time-based anti-debugging functionality. However, unlike the previous versions in which the timer was hardcoded, the updated version introduces a new field in the configuration. This field enables the customization of the sleep timer, with a default value of 10 minutes. This default interval remains unchanged from prior iterations of BLISTER.

```
   || (flag & fxh::config_setup::flag_1000) != 0
   && ((flag & fxh::config_setup::flag_2000) == 0 ? (sleep_timer = 600000) : (sleep_timer = config->sleep_timer),
       fxh::utils_sleep(struct_000, sleep_timer)) )
{
```

Time-Based Anti-Debug Feature

Unhook process instrumentation to detect syscalls

In this latest version, BLISTER introduces noteworthy functionality: it unhooks any ongoing process instrumentation, a tactic designed to circumvent userland syscall detection mechanisms upon which certain EDR solutions are based.

```
__int64 __fastcall fxh::anti_detection::unhook_syscall_detection(struct_000 *a1)
{
  FARPROC ntdll_dll_NtSetInformationProcess; // rax
  const CHAR *ntdll_module; // [rsp+20h] [rbp-28h]
  PROCESS_INSTRUMENTATION_CALLBACK_INFORMATION process_instrumentation_callback; // [rsp+28h] [rbp-20h] BYREF

  process_instrumentation_callback.Callback = 0i64;
  process_instrumentation_callback.Reserved = 0;
  process_instrumentation_callback.Version = 0;
  if ( a1->mapped_ntdll )
    ntdll_module = a1->mapped_ntdll;
  else
    ntdll_module = a1->ntdll_module;
  ntdll_dll_NtSetInformationProcess = fxh::utils::GetProcAddress(a1, ntdll_module, ntdll_dll_NtSetInformationProcess);
  return (ntdll_dll_NtSetInformationProcess)(
           -1i64,
           PROCESS_INFO_CLASS_INSTRUMENTATION,
           &process_instrumentation_callback,
           0x10i64);
```

Unhooking process instrumentation

BLISTER's configuration

The BLISTER configuration structure has also been changed with the latest variants. Two new fields have been added and the flag field at offset 0 has been changed from a WORD to a DWORD value. The new fields pertain to the hash of the domain for environmental keying and the configurable sleep time; these field values are at offset 4 and 12 respectively. The following is the updated structure of the configuration:

```
BLISTER configuration structure
struct Config {
    uint32_t flag;
    uint32_t domain_hash;
    uint32_t payload_export_hash;
    uint32_t sleep_timer;
    wchar_t w_payload_filename_and_cmdline[783];
    size_t compressed_data_size;
    size_t uncompressed_data_size;
    uint8_t pe_deciphering_key[16];
    uint8_t pe_deciphering_iv[8];
};
```

Configuration structure

Changes have also been made to the configuration flags, allowing the operator to activate different functions within the malware. Researchers have provided an updated list of functions built upon our prior research into BLISTER.

```
BLISTER configuration files enum Config::Flags {
    kDoPersistance = 0x1,
    kOwnProcessReflectiveInjectionMethod = 0x2,
    kOwnProcessHollowingMethod = 0x8,
    kRemoteProcessHollowingMethod = 0x10,
    kExecutePayloadExport = 0x20,
    kExecuteShellcodeMethod = 0x40,
    kInjectWithCmdLine = 0x80,
    kUseSysCalls = 0x100,
    kUseFreshMappedNtdll = 0x200,
    kEnableSleepBeforeInjection = 0x1000,
    kCustomSleepTimerSet = 0x2000,
    kEnableProcessInstrumentationUnhook = 0x80000,
    kEnableKeying = 0x100000,
};
```

Configuration flags enumeration

Payload extractor update

In our previous research publication, we introduced an efficient payload extractor tailored to dissect and extract the configuration and payload of the loader. To dissect the most recent BLISTER variants and capture these new details, we enhanced our extractor which is available here.

```
{
    "Configuration": {
        "flag": "0x181541",
        "domain_keying_hash": "0x9042e452",
        "payload_export_hash": "0xc7dbe4f9",
        "sleep_time": 600000,
        "rabbit_key": "0d2c198f2d1d5f4a74edd91281a58e73",
        "rabbit_iv": "22c4d67871364c52",
        "compressed_data_size": "0x2b75f",
        "uncompressed_data_size": "0x43a03",
        "injection_method": "Execute shellcode",
        "is_pe": false,
        "enabled_features": [
            "kDoPersistance",
            "kExecuteShellcodeMethod",
            "kUseSysCalls",
            "kEnableSleepBeforeInjection",
            "kEnableProcessInstrumentationUnhook",
            "kEnableKeying"
        ]
    }
}
```

Configuration extractor

Conclusion

BLISTER is one small part of the global cybercriminal ecosystem, providing financially-motivated threats to gain access to victim environments and avoid detection by security sensors. The community should consider these new developments and assess the efficacy of BLISTER detections, Elastic Security Labs will continue to monitor this threat and share actionable guidance.

Detection logic

Prevention

- Windows.Trojan.Blister

Detection

- Windows Error Manager/Reporting Masquerading
- Potential Operation via Direct Syscall
- Potential Masquerading as Windows Error Manager
- Unusual Startup Shell Folder Modification
- Potential Masquerading as VLC DLL

YARA

Elastic Security has created YARA rules to identify this activity. Below is the latest rule that captures the new update to BLISTER.

```
rule Windows_Trojan_Blister {

    meta:

        author = "Elastic Security"

        creation_date = "2023-08-02"

        last_modified = "2023-08-08"

        os = "Windows"

        arch = "x86"

        category_type = "Trojan"

        family = "Blister"

        threat_name = "Windows.Trojan.Blister"

        license = "Elastic License v2"

    strings:
```

```
        $b_loader_xor = { 48 8B C3 49 03 DC 83 E0 03 8A 44 05 48 [2-3] ?? 03 ?? 4D 2B ?? 75 }

        $b_loader_virtual_protect = { 48 8D 45 50 41 ?? ?? ?? ?? 00 4C 8D ?? 04 4C 89 ?? ?? 41 B9 04 00 00 00 4

    condition:

        all of them

}
```

Observed adversary tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that advanced persistent threats use against enterprise networks.

Tactics

Tactics represent the why of a technique or sub-technique. It is the adversary's tactical goal: the reason for performing an action.

- Execution
- Defense Evasion
- Persistence

Techniques / Sub techniques

Techniques and Sub techniques represent how an adversary achieves a tactical goal by performing an action.

- System Binary Proxy Execution: Rundll32
- Execution Guardrails: Environmental Keying
- Registry Run Keys / Startup Folder
- Masquerading
- Process Injection: Process Hollowing

References

The following were referenced throughout the above research:

- Palo Alto Unit42
- Trendmicro
- Malpedia

Observables

All observables are also available for download in both ECS and STIX format in a combined zip bundle.

The following observables were discussed in this research.

Indicator	Type	Reference
5fc79a4499bafa3a881778ef51ce29ef015ee58a587e3614702e69da304395db	sha256	BLISTER loader DLL

Source: https://security-labs.elastic.co/security-labs/revisiting-blister-new-developments-of-the-blister-loader