Babadeda Crypter targeting crypto, NFT, and DeFi communities

Posted by Hido Cohen & Arnold Osipov on November 23, 2021
Find me on:
LinkedIn

- Tweet
-

The cryptocurrency market is now worth more than $2.5 trillion. Unfortunately, this fact is not lost on threat actors. As well as using cryptocurrency themselves to extract ransoms, cybercriminals are now also tailoring malware to exploit the booming market for NFTs and crypto games. In a

discovery of critical importance to anyone familiar with this space, Morphisec Labs has encountered a new campaign of malware targeting cryptocurrency enthusiasts through Discord.

Crucially, the crypter that this campaign deploys, which we have termed Babadeda (a Russian language placeholder used by the crypter itself which translates to “Grandma-Grandpa”), is able to bypass signature-based antivirus solutions. Although some variants of this crypter have been noted by other vendors, Morphisec is the first to fully disclose how it works.

For victims, this makes infections highly likely — and dangerous. We know that this malware installer has been used in a variety of recent campaigns to deliver information stealers, RATs, and even LockBit ransomware. Fortunately, however, even as the threat level for cryptocurrency users rises, we also know that Morphisec’s Moving Target Defense technology is capable of both seeing and stopping Babadeda.

In this blog post, we will explore how Babadeda is being delivered, what an in-depth technical analysis of this malware tells us about it, and how it can be stopped.

Crypto and NFT Communities Are Prime Targets

Since May 2021, we have observed several malware distribution campaigns. However, many of the recent infections we have seen appear to be related to a sophisticated campaign that exclusively targets the Crypto, NFT, and DeFi communities. It is precisely for this reason, as well as the fact that NFTs are rising in popularity, that we have decided to take a look at this particular campaign distribution in more detail.

For those who are not familiar with NFTs (Non-fungible token): the term refers to unique tokens that provide proof of ownership on data that is stored on the blockchain technology. In recent years, NFTs have exploded in popularity, and are now starting to enter the mainstream consciousness. Naturally, this growing trend in the crypto space has opened up a new vector for threat actors to exploit.

The DElivery Chain

The vast majority of today's NFT and crypto communities are based on Discord (a group chatting platform) channels. Discord channels are publicly accessible and allow users to send private messages to one another within a channel.

In the campaign that we observed, a threat actor took advantage of these features in order to phish victims. The threat actor sent users a **private message inviting them to download a related application** that would supposedly grant the user access to new features and/or additional benefits. Because the actor created a Discord bot account on the official company discord channel, they were able to successfully impersonate the channel’s official account.

Below is an example of a phishing message that targeted users of “Mines of Dalarna”, a PC game built on the blockchain.

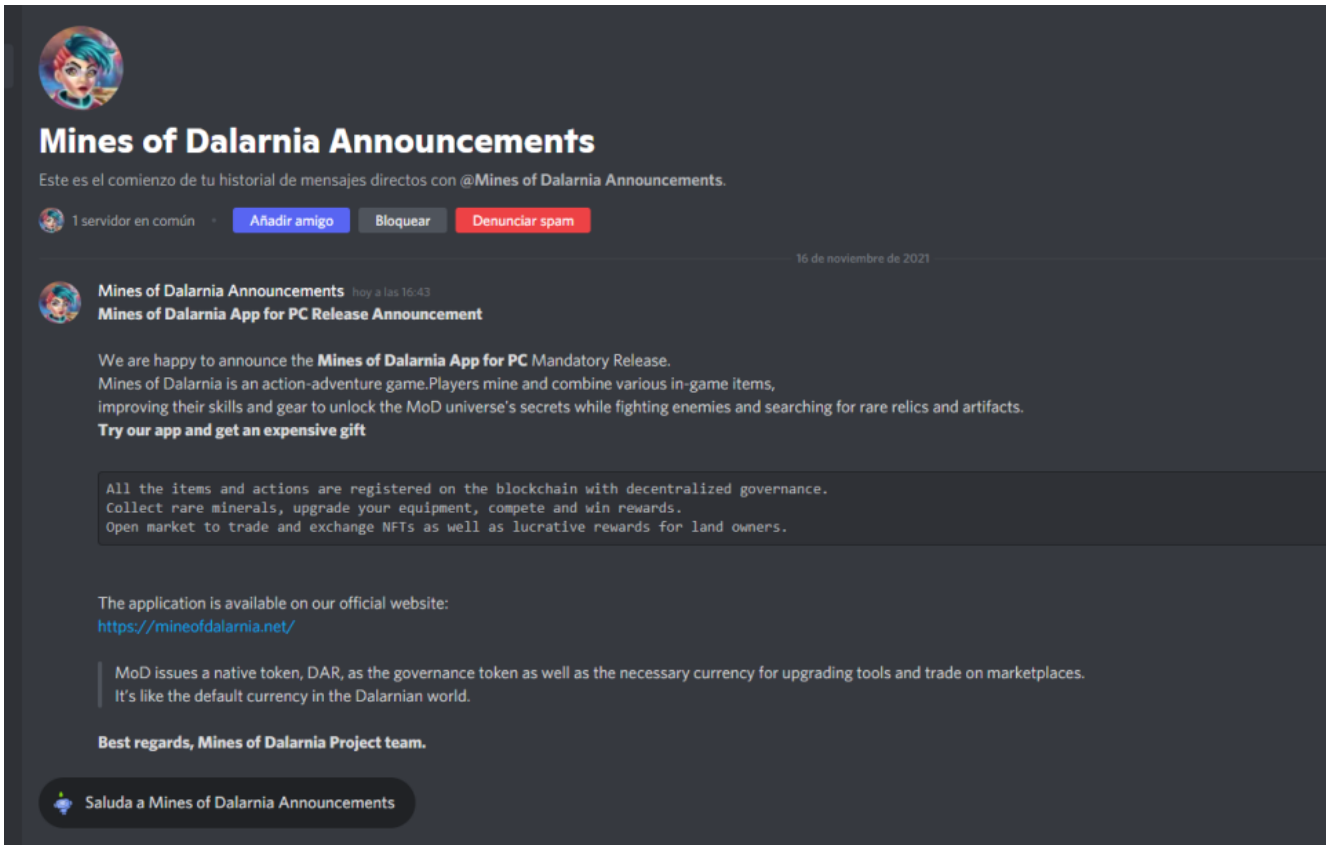


Figure 1: Fake message on the discord channel.

If a user clicks on the URL within the message, it will direct them to a decoy site. There, the user will be encouraged to download a malicious installer that embeds the Crypter with the payload.

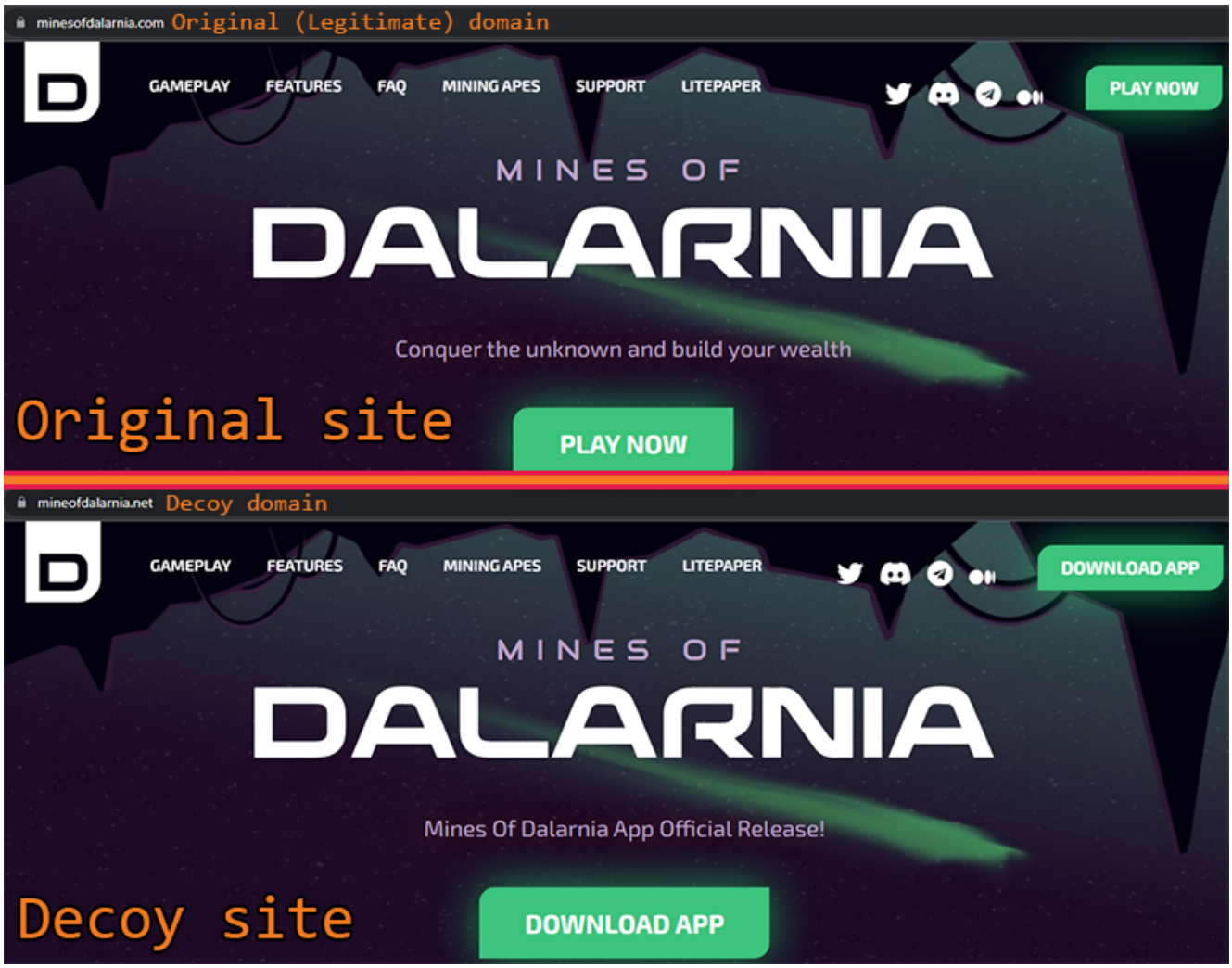


Figure 2: Original and decoy sites comparison

As you can see from the example above, the threat actor took extended measures to ensure that the delivery chain looks legitimate even to technical users. Typically:

- Cybersquatting - the domain names of the decoy sites look a lot like the domain names of the original sites Threat actors will usually remove/add a letter from/to the domain name or change the top-level domain.
- The domains are signed with a certificate (via LetsEncrypt), which enables an HTTPS connection.
- The UI of the decoy pages is very similar to the UI of the original pages.
- Upon clicking “Download APP”, the site will generally navigate to */downland.php,* which will redirect the download request to a different domain (this makes it less likely that someone will detect a decoy site).

Interestingly, on one of these decoy sites, we noticed an HTML object written in Russian. This suggests that the threat actor's origins may be in a Russian-speaking country since they most likely forgot to translate the HTML object from their native language into English

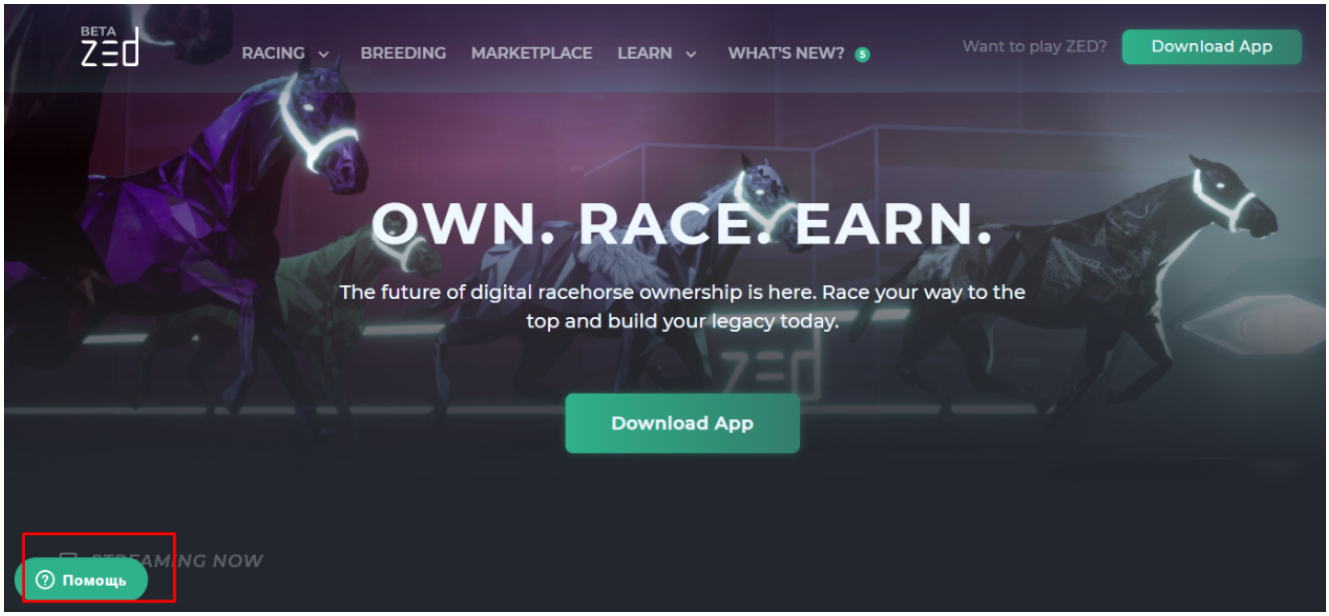


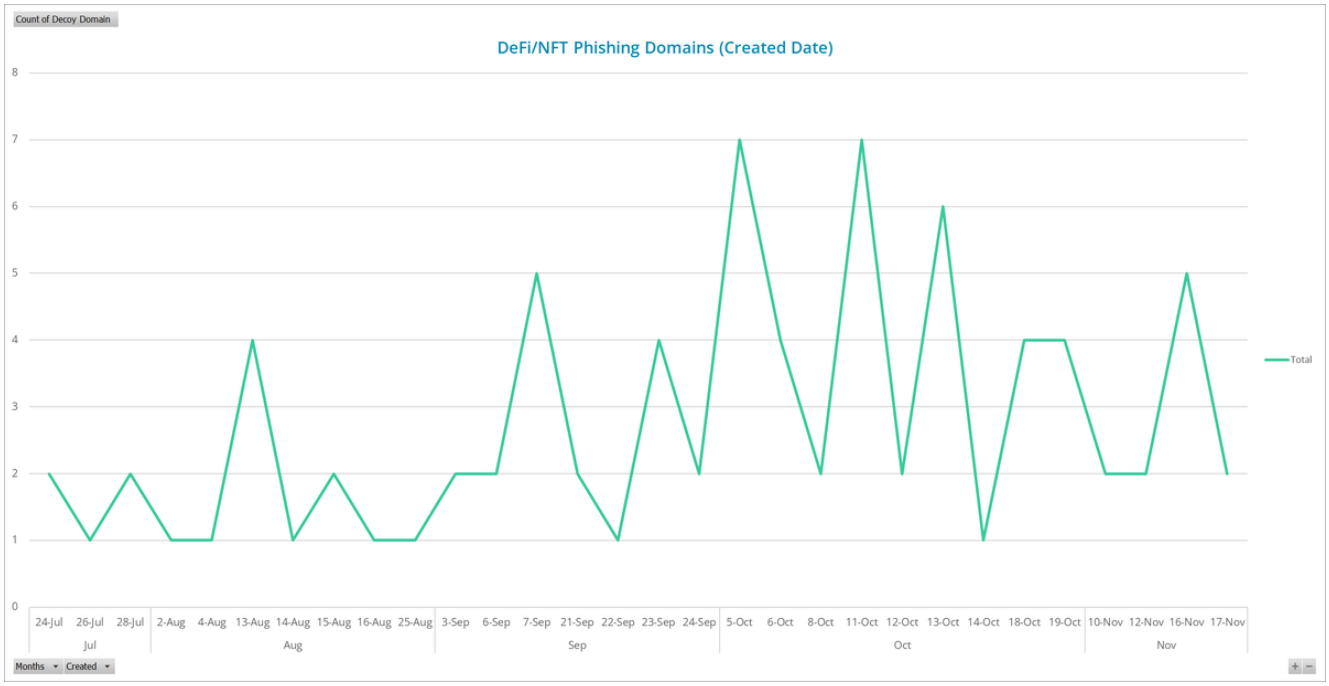
Figure 3: Lost in translation?

Decoy site examples

The following table shows a few examples of the decoy sites used in the campaigns we have observed.

Original Domain	Decoy Domain	Description	IP Resolved	Installer Name
opensea.io	openseea[.]net openseaio[.]net	The most popular NFT marketplace	185.117.2[.]82	OpenSea-App_v2.1-setup.exe
larvalabs.com	larvaslab[.]com larva-labs[.]net	The creators of CryptoPunks - The most popular PFP NFTs	185.117.2[.]81 185.117.2[.]82 45.142.182[.]160	LarvaLabs-App_v2.1.1-setup.exe
boredapeyachtclub.com	boredpeyachtclub[.]com	BAYC - one of the most popular PFP NFTs	185.117.2[.]4 185.212.130[.]64	BAYC-App-v2.1-release.exe

We have identified at least 82 domains created between July 24, 2021, and November 17, 2021, with the following registration time distribution (credit to @msuiche).



The Payloads

The following table tracks the RATs used by this specific campaign's threat actor:

Dates Observed	RAT	C2
11 Nov 2021 - 22 Nov 2021	Remcos	65.21.127.164[:]4449
14 Oct 2021 - 22 Oct 2021	BitRAT	135.181.6.215[:]7777
09 Sep 2021 - 14 Oct 2021	BitRAT	135.181.140.153[:]7777
24 Aug 2021 - 07 Sep 2021	BitRAT	135.181.140.182[:]7777

Technical Analysis of the Babadeda Crypter

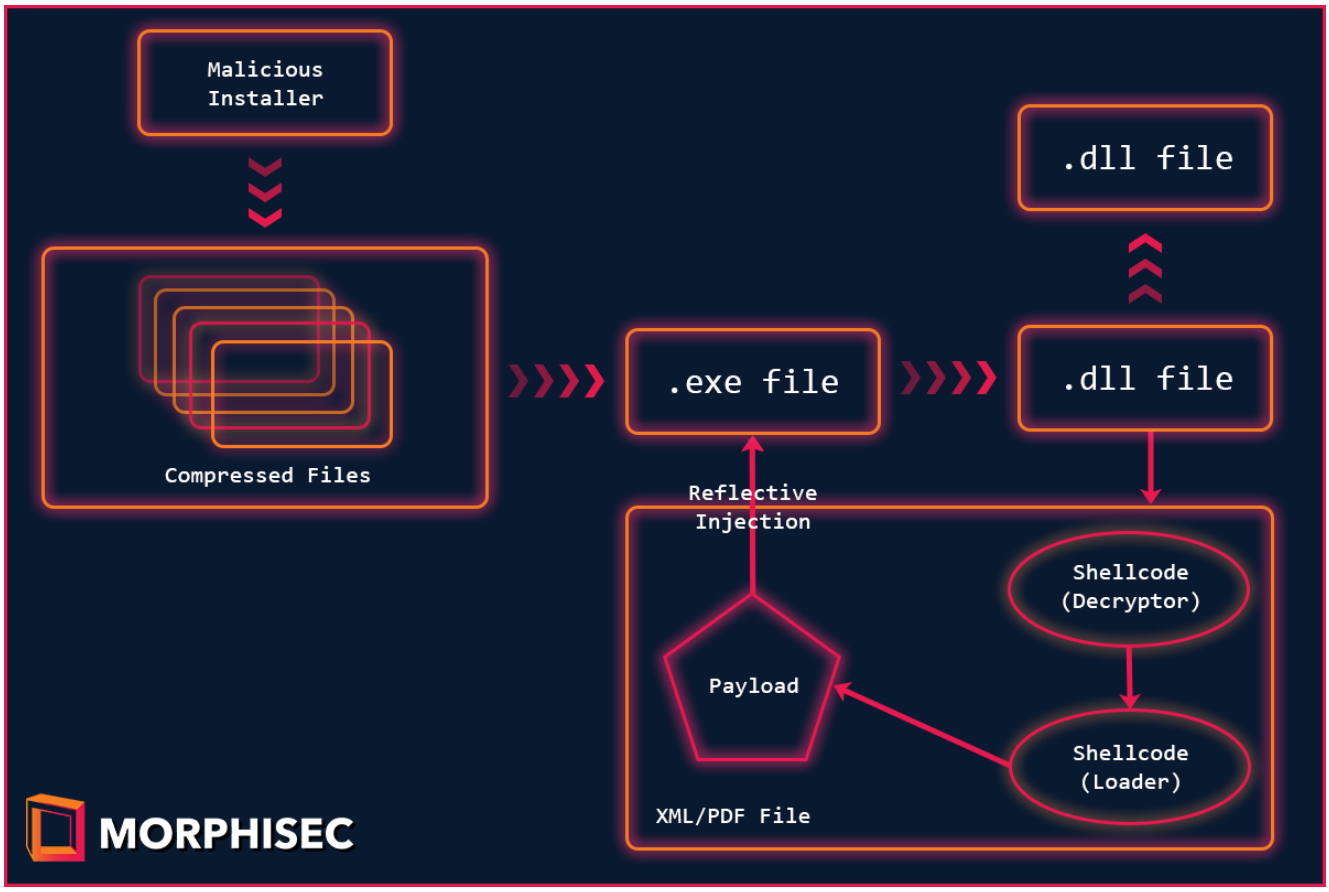


Figure 4: Execution flow diagram

During our research, we found different variants of the same Crypter — all of which contain the same main execution flow (denoted by the figure above). While investigating the Crypter, we saw how important it was for the threat actor to hide its malicious intentions inside legitimate applications in order to avoid detection. The following figure emphasizes the complexity of the evasive techniques that are implemented in the Crypter.

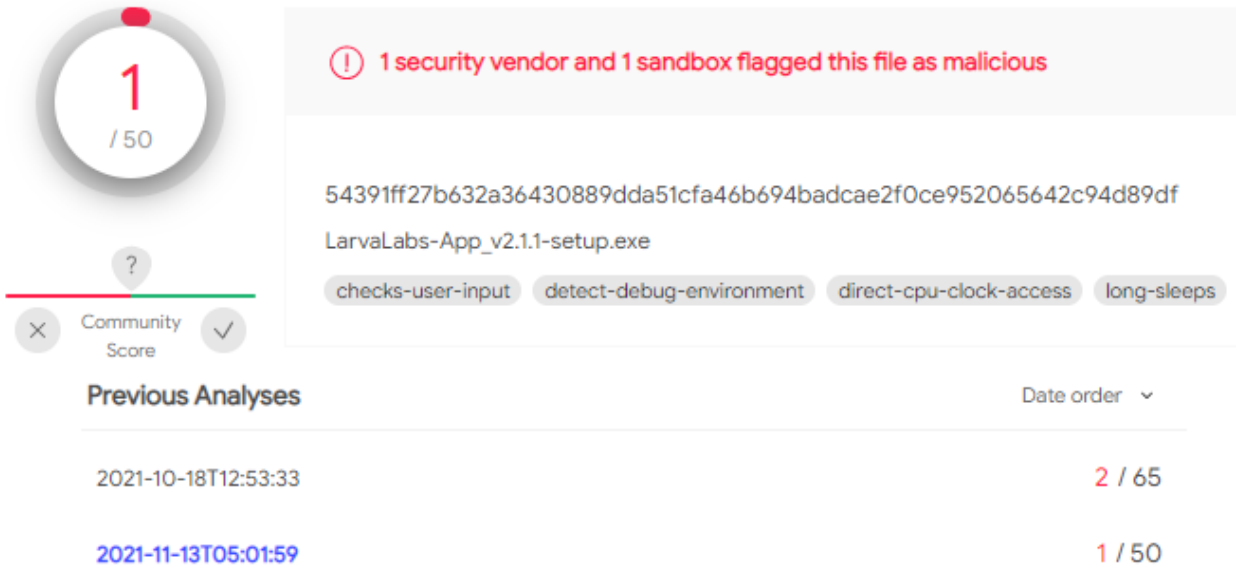


Figure 5: Low detection rate on VT

The Installer

Once downloaded and executed, the malicious installer copies its compressed files into a newly created folder with a legitimate-looking name (i.e., *IIS Application Health Monitor*) in one of the following directory paths:

```
C:\Users\<user>\AppData\Roaming\
C:\Users\<user>\AppData\Local\
```

The malicious files are copied along with many other open-source or free application-related files. At first glance, the files within the directory may seem legitimate. However, looking at these files carefully it becomes apparent that some of them are suspicious and should be inspected, as shown by the figure below.

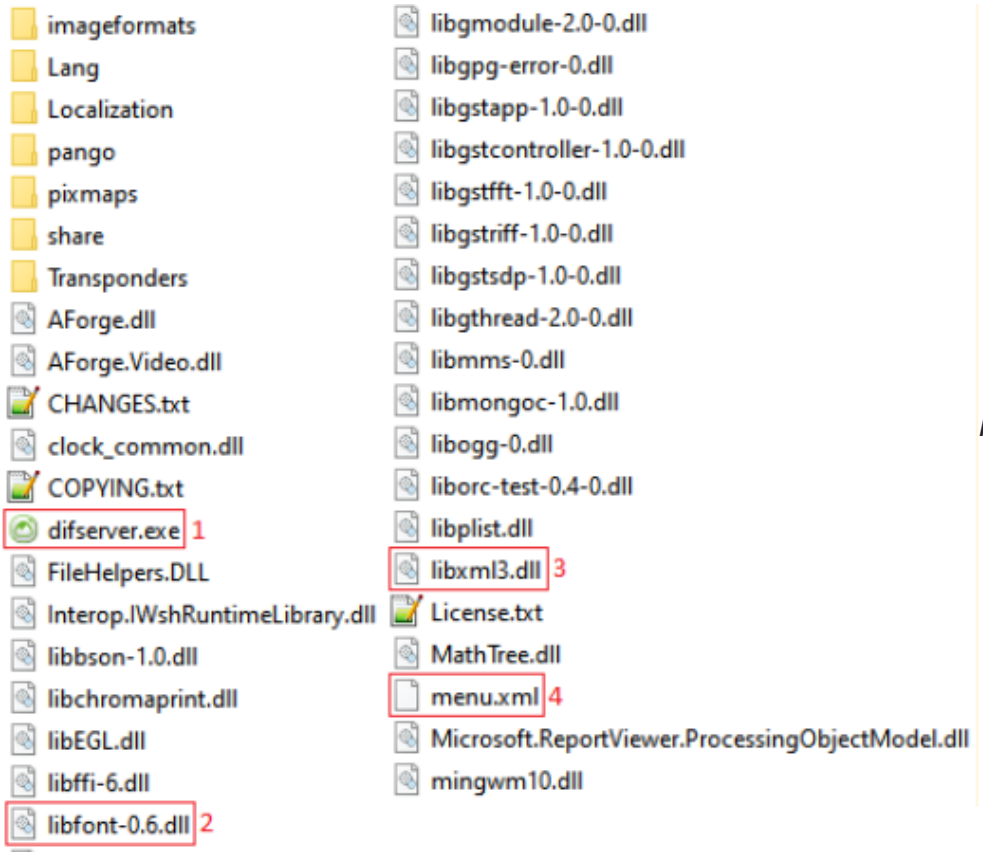


Figure 6: The compressed files (malware files selected with stage numbers)

Crypter Execution

After dropping the mentioned files, the Installer starts execution via the main executable (number 1 in the figure above).

We have noticed that at this point, some variants display a fake error message that stops the execution until the user interacts with the message. This fake message might be used as a security solutions evasion technique. Alternatively, its role may be to deceive the user into thinking that the application has failed to execute, even as it silently continues the malicious execution in the background.

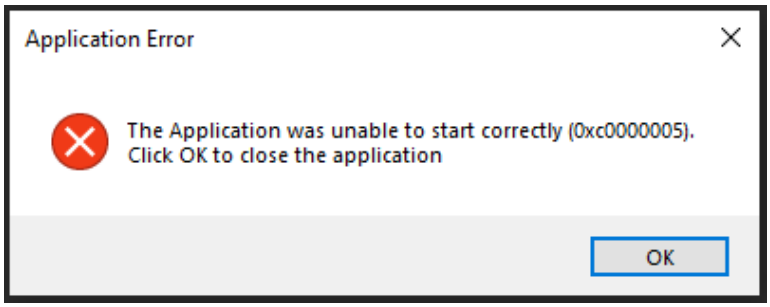


Figure 7: Fake error message

By analyzing the two different variants, we can see the implementation of this message box:

```
MessageBoxA(
  0,
  "The Application was unable to start correctly (0xc0000005). Click OK to close the application",
  "Application Error",
  0x10u);
hModule = LoadLibraryA(lpLibFileName);        // JdbcOdbc.dll
TrueTypeValidate = (void (*)(void))GetProcAddress(hModule, lpProcName);// TrueTypeValidate
if ( TrueTypeValidate )
  TrueTypeValidate();
```

Variant (1) with the fake message box code

```
hModule = LoadLibraryA(lpLibFileName);        // libfont-0.6.dll
gzprintf = GetProcAddress(hModule, lpProcName);// gzprintf
if ( gzprintf )
  gzprintf();
```

Variant (2) without the fake message box

Figure 8: Comparison between variants

As we can see in the figure below, the function's code is much longer compared to the actual DLL loading code. That's because the actor has implanted its actions within a legitimate application code in order to confuse analysts, obfuscate its real intentions, and make it harder for antivirus solutions to detect.

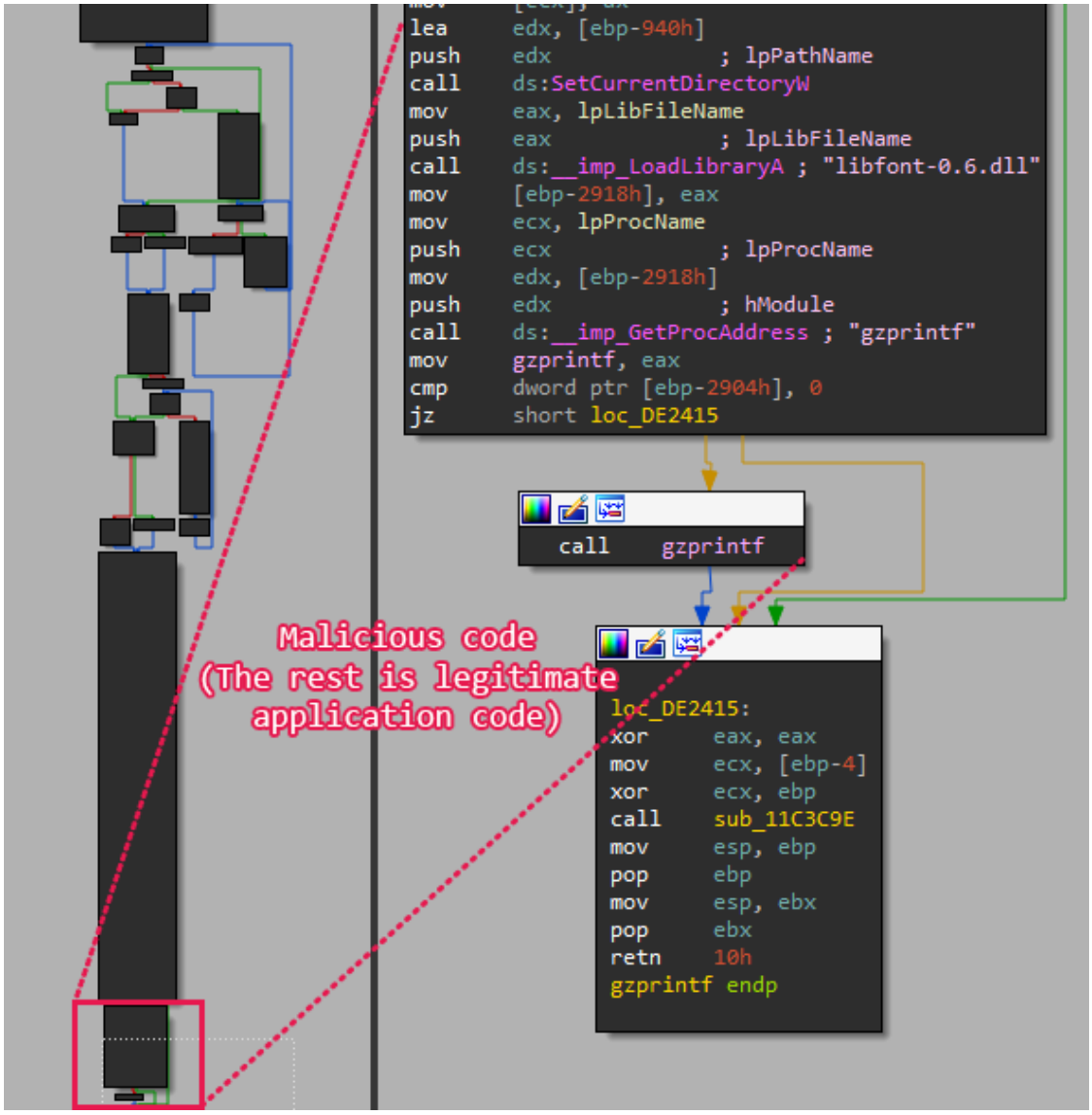


Figure 9: Left - the full function. Right - the DLL loading code

The Shellcode's Loader DLL

The threat actor generally embeds the next stages of the execution inside an additional file, usually an XML or a PDF file. Nonetheless, we have also observed additional file types such as JavaScript, Text, and PNG.

Here, just like before, the actor embeds the malicious code inside different legitimate codes. We have extracted the relevant sections to clearly demonstrate the malware's activity:

```
h_xml_file = CreateFileA("menu.xml", 0xC0000000, 1u, 0, OPEN_EXISTING, 0x80u, 0);
sleep_param = 35i64;
mw_thread_sleep(&sleep_param);
if ( !h_xml_file )
  goto LABEL_32;
file_size = GetFileSize(h_xml_file, 0);
file_buffer = (char *)mw_wrap_operator_new(4 * file_size);
ReadFile(h_xml_file, file_buffer, file_size, &bytes_read, 0);
CloseHandle(h_xml_file);
*(_DWORD *)(file_buffer + 0x893F3) = file_buffer;
exe_base_address = GetModuleHandleA(0);
```

```
// Copy Shellcode
shellcode_address = (unsigned int)(exe_base_address + 0x580);
shellcode_address_1 = (void (*)(void))(exe_base_address + 0x580);
qmemcpy(exe_base_address + 0x580, file_buffer + 0x88D8C, 0x6C0u);
```

```
if ( *(_BYTE *)(file_buffer + 0x893FB) )
  shellcode_address_1();
```

Figure 10: Exported function logic

The malicious logic starts by reading the additional file (in this case an XML file) and calling kernel32!Sleep for 35 seconds (the duration changes between variants). Next, it loads this entire file to memory and starts its parsing task.

The first piece that is parsed from the file is a shellcode located in a pre-calculated offset (in this case, 0x88D8C and overwrites the executable at offset 0x1600).

menu.xml

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00088CB0	63	65	73	2E	56	61	72	69	61	6E	74	2E	54	6F	4F	62	ces.Variant.ToOb
00088CC0	6A	65	63	74	22	3E	0D	0A	20	20	20	20	20	20	20	20	ject">..
00088CD0	20	20	20	20	3C	73	75	6D	6D	61	72	79	3E	0D	0A	20	<summary>..
00088CE0	20	20	20	20	20	20	20	20	20	20	20	47	65	74	20	74	Get t
00088CF0	68	65	20	6D	61	6E	61	67	65	64	20	6F	62	6A	65	63	he managed objec
00088D00	74	20	72	65	70	72	65	73	65	6E	74	69	6E	67	20	74	t representing t
00088D10	68	65	20	56	61	72	69	61	6E	74	2E	0D	0A	20	20	20	he Variant...
00088D20	20	20	20	20	20	20	20	20	20	3C	2F	73	75	6D	6D	61	</summa
00088D30	72	79	3E	0D	0A	20	20	20	20	20	20	20	20	20	20	20	ry>..
00088D40	20	3C	72	65	74	75	72	6E	73	3E	3C	2F	72	65	74	75	<returns></retu
00088D50	72	6E	73	3E	0D	0A	20	20	20	20	20	20	20	20	3C	2F	rns>.. </
00088D60	6D	65	6D	62	65	72	3E	0D	0A	20	20	20	20	20	20	20	member>..
00088D70	20	3C	6D	65	6D	62	65	72	20	6E	61	6D	65	3D	22	4D	<member name="M
00088D80	3A	53	79	73	74	65	6D	2E	4D	61	6E	61	55	8B	EC	83	:System.ManaU‹ìƒ
00088D90	EC	58	53	E8	F8	03	00	00	89	45	FC	8B	45	FC	83	C0	ìXSèø...‰Eü‹EüƒÀ
00088DA0	11	89	45	CC	8B	45	FC	8B	40	09	8B	4D	CC	8D	04	C1	.‰EÌ‹Eü‹@.‹MÌ..Á
00088DB0	89	45	C4	83	65	F0	00	83	65	C8	00	83	65	EC	00	EB	‰EÄƒeð.ƒeÈ.ƒeì.ë
00088DC0	07	8B	45	EC	40	89	45	EC	8B	45	FC	8B	4D	EC	3B	48	.‹Eì@‰Eì‹Eü‹Mì;H
00088DD0	09	73	1A	8B	45	EC	8B	4D	CC	8D	04	C1	89	45	B4	8B	.s.‹Eì‹MÌ..Á‰E´‹
00088DE0	45	B4	8B	4D	F0	03	48	04	89	4D	F0	EB	D4	FF	75	F0	E´‹Mð.H.‰MðëÔÿuð
00088DF0	FF	75	FC	E8	8A	02	00	00	59	59	89	45	D0	83	65	E8	ÿuüèŠ...YY‰EÐƒeè
00088E00	00	EB	07	8B	45	E8	40	89	45	E8	8B	45	FC	8B	4D	E8	.ë.‹Eè@‰Eè‹Eü‹Mè
00088E10	3B	48	0D	73	1A	8B	45	E8	8B	4D	C4	8D	04	C1	89	45	;H.s.‹Eè‹MÄ..Á‰E
00088E20	B0	8B	45	B0	8B	4D	C8	03	48	04	89	4D	C8	EB	D4	8B	°‹E°‹MÈ.H.‰MÈëÔ‹
00088E30	45	FC	8B	00	8B	4D	FC	03	41	04	8B	4D	FC	8B	55	FC	Eü‹.‹Mü.A.‹Mü‹Uü
00088E40	8B	49	0D	0F	AF	4A	09	03	C1	89	45	BC	8B	45	BC	89	‹I..¯J..Á‰E¼‹E¼‰

Figure 11: The shellcode bytes inside the XML file

The executable .text section's characteristics are configured to RWE (Read-Write-Execute) -- that way the actor doesn't need to use VirtualAlloc or VirtualProtect in order to copy the shellcode and transfer the execution. This helps with evasion since those functions are highly monitored by security solutions. Once the shellcode is copied to the executable, the DLL calls to the shellcode's entry point (shellcode_address).

Persistency Implementation

If the crypter is configured to install persistence, the loader DLL will execute a new thread that loads another DLL (from the compressed files) that will handle this task.

```
// Set persistence
hThread = _beginthreadex(
            0,
            0,
            (unsigned int (__stdcall *)(void *))mw_load_dll_and_call_function,
            context,
            0,
            (unsigned int *)&hThread_1 + 1);
```

Figure 12: New thread creation for calling the persistent mechanism

The newly loaded DLL will either use one of the following logics or both of them to implement the persistency:

Write a.lnk file in the startup folder that executes the Crypter's main executable.

```
qmemcpy(ValueName, L"Virtual Diffractor Server", sizeof(ValueName));
memset(v26, 0, sizeof(v26));
if ( SHGetFolderPathW(0, CSIDL_STARTUP, 0, 0, startup_folder_path) >= 0 )
  lstrcatW(startup_folder_path, L"\\virtual difserver.lnk");
```

```
if ( CoCreateInstance(&rclsid, 0, 1u, &riid, (LPVOID *)&IShellLinkW) >= 0 )
{
  IShellLinkW->lpVtbl->SetPath(IShellLinkW, Filename);
  IShellLinkW->lpVtbl->SetDescription(IShellLinkW, ValueName);
  if ( IShellLinkW->lpVtbl->QueryInterface(IShellLinkW, &stru_100F0530, (void **)&IPersistFile) >= 0 )
  {
    if ( GetFileAttributesW(startup_folder_path) == -1 )
      IPersistFile->lpVtbl->Save(IPersistFile, startup_folder_path, 1);
    IPersistFile->lpVtbl->Release(IPersistFile);
  }
  IShellLinkW->lpVtbl->Release(IShellLinkW);
}
```

Figure 13: .lnk file persistence implementation

Write a registry Run key that executes the Crypter's main executable.

```
memset(Filename, 0, 0x208u);
GetModuleFileNameW(0, Filename, 0x104u);
if ( !RegOpenKeyExW(
        HKEY_CURRENT_USER,
        L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
        0,
        0x20006u,
        &phkResult) )
{
  v8 = phkResult;
  if ( !phkResult )
  {
    sub_100140D0(62001);
    v8 = phkResult;
  }
  RegSetValueExW(v8, ValueName, 0, 1u, (const BYTE *)Filename, 2 * wcslen(Filename));
  RegCloseKey(phkResult);
}
```

Figure 14: registry run key persistence implementation

The Decryption Shellcode

The XML file (or any other file type used by the crypter) contains the following components:

1. The first shellcode (referred to in this section).
2. An encrypted additional shellcode (referred to in the next section, the Loader shellcode)
3. An encrypted payload.

The Decryption shellcode has three main tasks: first, it extracts the Loader shellcode and the payload, then it decrypts them, and finally, it transfers the execution to the decrypted Loader shellcode.

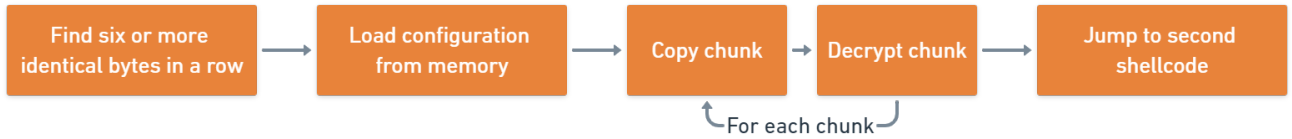


Figure 15: Decryption shellcode execution flow

The Decryption shellcode begins with dynamically locating the configuration structure by searching for a sequence of six or more identical bytes. This configuration holds pointers to the loader shellcode and the final payload; these are encrypted and split inside the XML.

```
struct config
{
  _DWORD file_content_ptr;
  _DWORD file_length;
  _BYTE decryption_key;
  _DWORD number_of_chunks_loader;
  _DWORD number_of_chunks_payload;
  file_conf* chunk_config;
};

struct chunk_config
{
  _DWORD offset;
  _DWORD size;
};
```

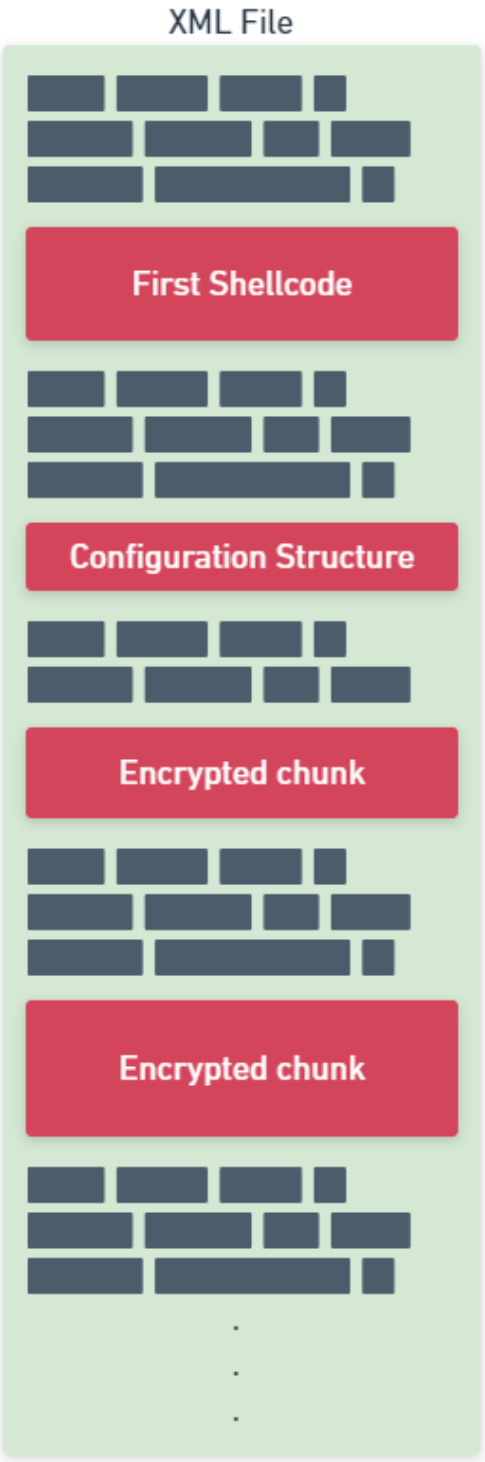


Figure 16: Configuration and XML file's structures

Based on this, we can identify the configuration inside the XML file:

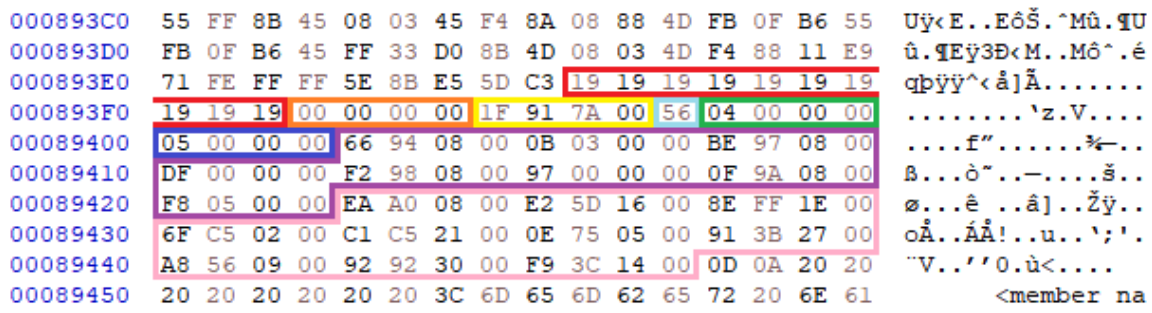


Figure 17: Configuration structure

Using this configuration the malware iterates over each chunk copies it, and decrypts it using the denoted decryption key (the configuration changes between samples).

Then, the shellcode searches for two DWORD placeholders, **0xBABADEDA** and **0xDEADBEAF**. It replaces the first placeholder with the address of the decrypted payload and the latter with the size of the payload. This data is used in the next stage, the Loader shellcode.

The Loader Shellcode

The purpose of the Loader shellcode is to inject the decrypted payload within the currently running process (itself).

We can divide the loading mechanism into three stages: initialization, injection, and correction.

Initialization

This stage is responsible for setting the relevant data that will be used during the injection and correction stages.

```
new_pe_address = (IMAGE_DOS_HEADER *)0xBABADEDA;// placeholder changes in runtime
new_pe_size = 0xDEADBEAF;                       // placeholder changes at runtime
new_pe_optional_header_1 = mw_get_image_optional_header((IMAGE_DOS_HEADER *)0xBABADEDA);
mw_get_nt_header((IMAGE_DOS_HEADER *)0xBABADEDA);
mw_get_image_section_header((IMAGE_DOS_HEADER *)0xBABADEDA);
new_pe_size_of_image = new_pe_optional_header_1->SizeOfImage;
current_exe_peb = mw_get_ntcurrpeb();
mw_get_ntcurrteb();
current_exe_address = (IMAGE_DOS_HEADER *)current_exe_peb->ImageBaseAddress;
current_exe_ldr_entry = (LDR_DATA_TABLE_ENTRY *)mw_get_ldr_list_entry(current_exe_peb);
new_pe_target_entry_point = (char *)current_exe_address + new_pe_optional_header->AddressOfEntryPoint;
VirtualProtect = (void (__stdcall *)(IMAGE_DOS_HEADER *, DWORD, MACRO_PAGE, int *))mw_get_VirtualProtect_by_hash();
```

Figure 18: Extracting the data for injection and correction stages

To start initializing, the Loader first saves the decrypted payload address and payload size according to the placeholder's addresses. Next, it parses the PE headers of the payload to extract the image size and the entry point according to the current executable's base address. The Loader parses the _PEB structure in order to find the base address of the current executable and the LDR_DATA_TABLE_ENTRY which will be used later. Finally, it dynamically loads the VirtualProtect function using a pre-calculated hash value (0xF1C25B45 in our case).

Injection

This stage is pretty straightforward. Within it, the Loader overwrites the current PE with the final payload's PE. It does so by copying the PE headers and each section according to the current executable's base address.

```
original_protection = 0;
VirtualProtect(
  current_exe_address,
  new_pe_optional_header->SizeOfHeaders,
  PAGE_READWRITE,
  (int *)&original_protection);
memset_w((int)current_exe_address, 0, new_pe_size_of_image);
mw_overwrite_mapped_pe_with_new_pe(new_pe_address, (int)current_exe_address, 0);
```

Figure 19: Change headers protection and clear memory bytes for the new PE

Once previous bytes have been cleared, the Loader copies the new PE headers to the base address and each section to the relevant location according to the IMAGE_SECTION_HEADER.

Correction

The final stage is responsible for fixing the import address table and relocation table of the newly injected PE.

```
current_exe_ldr_entry->EntryPoint = new_pe_target_entry_point;
current_exe_ldr_entry->SizeOfImage = new_pe_optional_header->SizeOfImage;
mw_construct_IAT(current_exe_address);
mw_construct_RELOC(current_exe_address);
oldProtection = 0;
VirtualProtect(current_exe_address, new_pe_optional_header->SizeOfHeaders, original_protection, &oldProtection);
memset_w((int)new_pe_address, 0, 0x1000u);    // delete header
```

Figure 20: Fix tables and remove altering evidence

- mw_construct_IAT
  - Load GetModuleHandleA, LoadLibraryA and GetProcAddress functions by hash (0x9FE4FCE1, 0x85557334 and 0xF23B576D respectively).
  - Iterate over the IAT of the new PE.
  - Load each function and update its address.
- mw_construct_RELOC
  - Calculate the delta between the previous image base and the current one.
  - Iterate over each entry in the relocation table.
  - Add the delta to the entry value.

In addition to fixing the import address and relocation tables, the Loader removes evidence of injection by using the following methods:

1. Update the LDR data table entry to match the injected PE.
2. Remove the injected PE headers from memory.

These steps attempt to evade memory scanners that seek mismatching LDR data and in-memory PEs.

Finally, the malware jumps to the entry point of the newly injected PE with the original command-line arguments.

Conclusion

As demonstrated above, Babadeda is a highly dangerous crypter. Targeting cryptocurrency users through trusted attack vectors gives its distributors a fast-growing selection of potential victims. Once on a victim's machine, masquerading as a known application with a complex obfuscation also means that anyone relying on signature-based malware effectively has no way of knowing Babadeda is on their machine — or of stopping it from executing.

Mitigating the threat posed by Babadeda requires securing the device memory it targets. Morphisec does this through Moving Target Defence, a technology that morphs process memory trapping crypters like Babadeda before they are able to deploy.

IOCs

The sample used in the blog post:

File	SHA256
Installer	99e6b46a1eba6fd60b9568622a2a27b4ae1ac02e55ab8b13709f38455345aaff
difserver.exe	358211210e0bb34dd77073bb0de64bb80723f3434594caf1a95d0ed164ee87a1
libfont-0.6.dll	ce3758d494132e7bef7ea87bb8379bb9f4b0c82768d65881139e1ec1838f236c
libxml3.dll	0ceead2afcdee2a35dfa14e2054806231325dd291f9aa714af44a0495b677efc
menu.xml	080340cb4ced8a16cad2131dc2ac89e1516d0ebe5507d91b3e8fb341bfcfe7d8

YARA Rule

```
rule BABADEDA_Crypter

{

    meta:

        description = "Detects BABADEDA Crypter"
```

```
        author = "Morphisec labs"

        reference = "https://blog.morphisec.com/the-babadeda-crypter-targeting-crypto-nft--defi-communities"

    strings:

        $entry_shellcode = {55 8B EC 83 EC 58 53 E8 F8 03 00 00 89 45 FC 8B 45 FC 83 C0 11 89 45 CC 8B 45 FC 8B 40 09 8B 4D CC 8D 04}

      $placeholder_1 = {8138DADEBABA}

      $placeholder_2 = {8138AFBEADDE}

    condition:

    $entry_shellcode and all of ($placeholder_*)

}
```

Decoy Domains

aave-v3[.]com
abracodabra[.]net
alchemixfi[.]com
apeswaps[.]net
app.sushi-v3[.]com
arbitrums[.]com
artblocks[.]us
astar-network[.]com
avalanche-network[.]com
avax-bridge[.]com
avax-bridge[.]net
avax-network[.]net
avax.wallet-bridge[.]net
avax.wallet-network[.]net
avax.wallet-network[.]org
babydogescoin[.]com
boredpeyachtclub[.]com
bridge-avax[.]com
bridge-avax[.]net
bridge-avax[.]us
c-nft[.]net
casper-network[.]com
compoundfinance[.]net
cryptoblade[.]net
decentralands[.]net
diviprojects[.]com
dydxexchange[.]net
galagamesapp[.]com
hedera[.]run
illuviums[.]com
keep-network[.]net
klimadao[.]net
larva-labs[.]net
larvaslab[.]com
looprings[.]net
luckybuddhaluckyclub[.]com
mangomarkets[.]net
mineofdalarnia[.]net
monstasinfinite[.]net
moonebeam[.]com
near-protocol[.]com
network-avax[.]net
network-avax[.]org
nft-opensee[.]com
olympusdao[.]fund
openseaio[.]net
openseea[.]net
optinism[.]net
polkadot-network[.]com
projectseeds[.]net
projectsserum[.]net
rareble[.]net
rocketspool[.]net
secretswaps[.]net
sia-tech[.]net
solanarts[.]com
solsoulnft[.]com
sushi-app[.]com

```
sushi-v3[.]com
sushi-v3app[.]com
terra-money[.]net
thetatokenfund[.]com
wallet-avalanche[.]com
wallet-avalanche[.]net
wallet-avalanche[.]org
wallet-avax[.]com
wallet-avax[.]info
wallet-avax[.]net
wallet-bridge[.]net
wallet-network[.]net
wallet-network[.]org
wa let.bridge-avax[.]us
wallet.network-avax[.]org
wonderlaned[.]com
zed-run[.]net
```

Hashes

```
0098b2c38a69132bfde02d329d6c1c6e2b529d32d7b775a2ac78a369c0d10853
0115ba0f26a7b7ca3748699f782538fa761f7be4845a9dc56a679acea7b76cd3
062f019515bff366fcbf49cca3f776c21e2beb81c043a45eea81044a9391fd97
112282b873bdbeb5614fc8658934a99d666ba06c4e2840a21cd4458b426a4cad
120213353ac7bd835086e081fb85dfa4959f11d20466fd05789ded3bff30bb11
1252c9103805e02324d2aecb5219e6a071c77b72477eba961621cb09a2138972
140d9a4a2ec5507edf7db37dcc58f2176a0e704e8f91c28a60a7f3773e85e1aa
14da3566bc9f211528c1824330c46789396447c83c3c830bb91490d873025df8
18c01e1f6e0185752dbf8c9352d74ade56ac40d25ae701d4a5954b74d0c7aeea
196ec622eb7d9420b1c04b3856467abeb3ca565d841f34c3c9a628afc10775c8
214d6681f5d82d4fa43e7a8676935ef01ddab8d0847eb3018530aedffe7ebb55
2e5455e268cf12ebc0213aa5dacb2239358c316dda3ec0f99d0f36074f41fb09
2fc8dedf82997894bb31a0eca96ae3c589863ec9bf4d1e2af0a84f2e9c3ef301
3270599801099d3b5399eb898f79d7b7ec0d728c71d5177244b8110757365ade
39b4dc69dd29011135732a881152f99dc19310cb906b7255a3e9ef367258094d
3c844e66f0dafdced0861a8e2ff54fd762ba170bf5082fb2c38cdbbac5a7fecb
3e52c251dc8683e0f374bcbea27b4b700c05dc39db13336859acbbd32590fe7c
3e6a29c04270a4b62375946fdb4c392a1c9b3f64ef391f85bdd67cb78426889f
44e00bef4b6d3f03a845208b925c129a5fe1b9ef6ed8cd27144c5e94176aaa6e
462f7543326630d209b6433936f0c54f8920d6b5505e88d802ee060320ea8106
4e6eed44594054ea42f9860c1e53744649a319788e2cb7f1f624e435cbdec43d
54391ff27b632a36430889dda51cfa46b694badcae2f0ce952065642c94d89df
6342d9c9e087945651b11cec4903f083a20d31182e0be5b2b6030df0a980ff68
65363debbbb9a691838e823c34807a9770db30c2af616c5574231af2b16d6aef
6e4d56a438062210ba8ca68dee690c1692960ff36936c96586f74ee194e1c821
6f247a74aa62fea0577da869fda841170ce6f1fe0e1b9f3b0d8172d336bb7dc6
71d0c5b5916cc5f91370f42fbfd249795e7c40526ae204becdd20fe453b53e8d
72df0397893e1ac981063fbcc0ad048543ba7143ba824f2bb0aa5dfb61538ce6
7c8242812137aad072fe1cb78d49d01187b869d43ebcfcd87eb590c1bc9f1246
7e827e1981d2ccaec16a5b646976b0d492d555a20b9ba5dd4ba0d605dfcab2f7
86b1cf4e6952db195842809ffd7e88e5fdaca8b2b2c0005e995d34cbe9d157ad
8b9120fc400510de52fb5c6689f403e5c0aaba3ff58e2ee114286c2cf09615b5
8ce8c448b5958da3c59874594de428b783116d8c1cf440ab804633799d88af8e
90faf9b85d96a09cb689be3a52669a58df2e9ea53b150a97d05de641e624f634
95d226710f37a870a338344afac6350b48c5d70c7ac8518c42f694eb0f6aa7c5
9b132e1d883c4f513d4ac3a5735a28a1917cfde837ee4a4b632a66cce5aa8be2
a2545370b390e52376d12776152aff9285b9b3fe6610d2f8dd24b11ccb14c5b3
a2e090192bf0b3b00f5bbef0b81858bc17861fedd82e93f0ab6d60777ca6820e
af0c213a2cfb62e6a9ce788c3860c627e035401b75df7f60eb64d4f4bc196aa2
b5fe6db30b741f515df94238c8d1a3c51a84fe72f218751c86a254801c3233ee
b6dc8341fd38dacb7a2a38a14a21afbab8e7e3f31f2fd29f0bcd7d4eb83e203c
bcaaab0cd2178acdf025c7f23f10ab01906a99aca5d07e3a7e261928f8f91695
c21e2be7324afb67f1e5cf9fbc95dc346db2ec62d9d8db7b0da9377a00346f41
c97893d936b5e1203fb926e7ab612ffd488578e9791f07be4a6eabc83645fb5b
ca70f7b046f5909f0134a1c465fda3794344f45055ba2dfa802623bd326fe5b6
d360daf106314561e9ec57075dd4f544ad52680678a644e186758650a405b765
d548c2e3479c6c7a20ffa8a8402aa00c45aaef24102daf5c94c54a8a6013f370
d76e7a14ab20d3f28de1ecef803d8b1629ed077495db5ec7b7f5828ed33c684e
de644e637da7cd117517b1bb96ee0f58131515013a322366d680f613afa31bc4
e5f55a5ecd7315c9e028738ced66d42852569dd061e15610a054c2121c9ed4d9
e99d32952bda84f32425681229ec544849156e479b7247e3e480f3a23a39c915
f24492ceab91f70c3dd3c5040184dae3bc38804c872ae948ed1ee6906a890b16
fb04bc486bf7b6574b5b7caf1ed4f1a21e9e7463adf312219f767a58e8fb2be1
fde7bd78e2085f364e0eb145c77b57b8bfa5bacf6a3e6eaed4b9e3a97c065a80
fde8ca7c729a25e723a3738a1b5520f29ef2100ba2d9a2739aa30176b039f511
```

Crypter hashes

amadey stealer

```
4d02224a7dadfc2d8a1343fdc51e4634a98bd073f867bfd091e667efd112108a
384292cad1c05552ccbd691de48865ce75375f7e601db66b3f5cad0f8f294d6c
5dd0e9ef811c199a06758d65b66d051d3b0057971b021df0928ede727fe17371
e312af68203fd80a2dd86a69460941ce29709424310abffd66fd7323a2b8ef6e
```

Cryptbot

```
83aa33a24f0751cae83420450716387399813 04b37fc036da342f15ccebaf482
c69fd2882bcda2ae6b24235babcc570f31774a45698edbaaea70e1b9d9fd315e
58fba0f609b5363b1fbb792e0b2def924b770b2f57329f383cf691ef5988055e
1a31f5a7bc1c5782ab9e7a401a2a474ee75e571adfa1f7685c13258653e8af7d
2dc6785721bc9369090ce77d47b6b85eb4c9fba88d4c29675b5c98195c653f3c
89d3acf2cfd33516f0aaaf901226c1c8936c33ded480115cbb56b4d11fc0d405
3f3f0c883ada23e33685a015dbd59a08668fa80bf3248b4fbc3b00dac1fb4305
66b5d71b2ae6f7569b050130ccb548785925d4ff14ccfd5fa9738e8b444cbd97
d2fc2159debd0a2222673cdc028c5f88ca5cc6c72f5665d60c5d27806757cff2
8e113203dd97f0f33562db9086b0eadeb5ea1242738abd80ae872ac3552a2599
58b3a4ec25d09191c9f5cb064a4ac4ea35a51cf1dd5e26e5d5bc63662c49c2ce
3fdd54336ae1400d16fd36013844953d8cbfa2982516f3d40ed2a18f58f82609
2b7425ae37127535adf331bdba2e4b126dc7a67890f2974fa95624b06b3ff248
a49d63a099d6499875e6b46268054b63d582303c7eac93a65ff00537ab22f487
d69e8d0678be5a8da741058f0ae2a6f99ffb8e3326ac50fda54336b23a546fc6
e51597f0749cbb7b8b53795383f891158ab7a5af350d803f8bba787ba1d3af87
240e6edb33f1d5578084bd8422792770d3bafea1581b58e45eb6f89a889f41bd
dd5f6a8a3f255be6e5b8c7402be7059298bcfea15931752e10ea0be59ad08063
3b0cf91645b6ac772fc518bd5d145db4e7750af4e8239cc46734350ddf4595bd
85c2e909efb713bdb2fb402dd380ada3bcc5ff92776ab95cafaeda7e47ea6dc4
94f8745d09bc73fa393b77c944bb7230fc68235fd5049c32c31612eb31747224
fe828cf68e77f09d903d17e4318e585ca5753b5cc1a8e7fdb081244ee6e29464
36cba5140248916ece6706fee52c892e7284b8c1dde007b273cf8adf1e2565ae
979a0dd895e91925b1664d6e475f5046c1b0243a9999b6d15c96480437ddf931
65a1fe7f19a41fbf7ac6196d5e54900558c948603a86de7e4920129556293723
7c34b54d3dd6d4b36587667cb52201ca8412ec23e0a6d062cceddb703104d0c9
ca00f9b232a297b1896a96c01b4835cafa0b050d62b6b891b74d6c799e6e6d26
3f6c89a650f439f01b2435425946f5b5eab475da42ee04088fd552bc59644613
fd274f7faace98d5660ea1a13dd74cade60626bf10cc5e4a66c0c76d8e018ad8
496e1715c87a07b92d9214810bcd3fc6880b88b93246e94ad9421f3434076660
6ff84a220c1f0d6c078de2bc9961dcda11ea21eabebc86576798f5d1a0548e11
3d59fa24db23fb548796b2632a3c94ea6be2c2a64236b470bcbcb5bfc6e1d915
8a217e632ef9f099bae955699c9eb497c6227a642486f64c903a336fdd0f3ac5
03b9c509e7ff704be0431c541a3571360b52edac361b3d9ce627b4e93c53be17
bbe8ee94ed612d25d1378980dbe529ad018f1a2ed0521c0621f81ae54bc2d516
c885a22bdd7d046c4a616e639cc91dc94cabb972108bdc2d9540fcbe393241d1
1ef9df7881ad13c6865aa6161390df6580eb648c3c05a35db706c7b5d7a238f5
6009bf01b6ede3fd35ef88aee476c1cb77ed32c54fd467b2d6173b59af8510c2
d90f581c543cba58332c5c67e2a464387142e72bb9d6960bcc9dd52ef2a948f4
db9b014740b96a6b7e277cf456a19260533dadf8b36652d05e374b098c93f63c
df7f07f9b0c6ff27b0011f3a6daa5ca4b73f554b6a1ed319dce05919c3c4e18c
dc5ffac866a06926359e00872ce7cc7b85d2ddf09abdc3371ac101be4e7ed46e
98110cbc2802dc27b9d9fe5ba5ceeece06cf3ed93974dfeb1ce26f2b5c43e23c
75837a43d3df5f8ec3117279edcfc255c69be9aaf2eed9d0d3cc98bf3b06ae01
ae0cd5b88a754affa47410a0fa9d9b38582c21b8e06c32273206fa15551efdb3
beb4c0c6486545826c2ec5fa5ba44d02abeb20558e55f47c51366523cacdde27
50f424ee3a86842df558da44cc247fd7bd4d1d7bf5439b8732883aa840a9fceb
0bf886695f19c711bd63d145518301270e247830259eb29c83bd0ee135f53ee7
aa2e234a48e1f19d8dfd1885aa7ea0c73b1d22faee0f3b208dc65762e6ba374d
49caac5d027dadb4db266ac999842ed7ea10b245750f8b31af738b4bfdcc5ade
464f8fc360e64cdf07c837d5911f93b60cb99ee0ff531ffb0422652c7d6124d6
0ef94ff905ef764a4aab4cc90d657dd681b434e13df35c01c6473ee3813dd34e
64b451280c906afb57198e787eaa18780abd3932bc7cf3742a5e58ccb1ccf204
```

lockbit

```
778eb09cac51aa75b6e3c32e78adfe0e9292af40d0f800fb3ae569198945a9ef
5b9e6d9275e9523aa3945be891745442a07b936ee5236e23934250ba3844f65f
17c6f4e45d44bd4c06212139f521976b87ed5a6ddcd0e4e5e978e64dabb3883f
237bc833db8c72cedf0a09bd642567aa31cc74dd6bcfe5b67871f375d617ec85
446736e381fa8942f8d32cb4f2ae8fb6a9245fa0e70b7f7298ee7a5cb6fe9f32
668434940877f747a5d3adc745548bcfdcc881418f02e705204df2ad54a311cb
74b4d14d2d1af6642d5867eb89c277aa02f5e4ac667d87b5aca380f40eabe1bf
c920b2de025019e9a406e9b2f0ac2cbbfc18d65eac15f59ca8921c5fb4bfa240
d25116f1fe5c9a22fcf73c4c7358f93f1ad445bb9a602d18ff69f8fa29d0be0f
e80579baf175626787070bf61f75b4b810eb9d9bdb653972ad40797ee5ff82cc
1ee311c3f24397de3f6671b67a263206e78f8040f5ac2fc0182d0ee171c53228
4b3a396f8230fc87b7fc47aa1d7ed19c78867f3dd43fd570ea93748390be58d5
```

ursnif

```
04595c3111276f02b6dc2ece0778cb5829c086484aeafa24e0aac3d8479deb4b
e2c83783d6ab57ac91d99bfb9d607d0b5537e305661406bbf2347c3af92d3464
676a540a91b9ffb4a18af0f4355561f3579ee4cbbf0740a80e482af92e8cdc07
716ce7fe411f352686b4071074aa96e1456ab7a67445b3cf1c475e18a4e5ac25
ceba6a7f9a2c25a35090470c6209aefed808786c47194a18415a7898390c20cb
e203345d8120bd6d29e667bbceb92083ebb55e36b21cd22d669aa2f91830a656
```

smokeloader

```
79ae89733257378139cf3bdce3a30802818ca1a12bb2343e0b9d0f51f8af1f10
1ae5c809ea8fabce9c699c87416d73ba5ab619accef6deeb26c2c38f39323181
ee8f0ff6b0ee6072a30d45c135228108d4c032807810006ec77f2bf72856e04a
```

fickerstealer

```
bd8d1264a88d5cdd701a4ee909b70beaec39d216c988b33bfb30f25aee3540ee
1f53d6f4fb02c8663b9d377570953d07c56df297674b7c3847d1697f0e5f8165
cf88923b7d0287884870af999a8d64f90c7deeb4c4d09feed406472ff259b30d
```

Metasploit Reverse HTTP

```
b8990f204ca595e23562aa8063fd163651771626ba4acf45890f25315616fc1e
```

quasarrat

```
e8a8581cd3594a3937762f90d20ab889e7868bb88e9249f96222bd48643d7dea
```

Contact SalesInquire via Azure