A Deep Dive into DoubleFeature, Equation Group's Post-Exploitation Dashboard

By itayc

Published: 2021-12-27 · Archived: 2026-04-05 20:42:38 UTC

Earlier this year, Check Point Research published the story of "Jian" — an exploit used by Chinese threat actor APT31 which was "heavily inspired by" an almost-identical exploit used by the Equation Group, made publicly known by the Shadow Brokers leak. The spicy part of the story was that Jian had been roaming in the wild and abusing Equation Group ingenuity to compromise systems before it was cool — as early as 2014, a full two years before the Shadow Brokers leaks made the original exploit public. Evidently, the authors of Jian had acquired early access to it some other way.

While this discovery had undoubtedly added an extra tinge of paranoia to an already complicated affair, we were still left with some questions of our own. Chief among those questions was, "how come that little nugget was still just *lying there* for us to find, a full 4 years after the fact?". In information security terms, 4 years is an eternity. What else would we find, if we dug deep enough? Have the Russians actually had access to these tools back in 2013? The Iranians in 2006? The Babylonians in 700 BC?

First of all, we are glad to say the answer is (probably) no. Best that we can tell, APT31's apparent early access to the leaked exploit was the exception, not the rule. This makes for a less exciting headline, but should help all of us sleep better at night. During our research, we combed over the DanderSpritz framework — a major part of the "Lost in Translation" leak — in excruciating technical detail; and we present our findings here, with a pedagogical focus on its `DoubleFeature` logging tool which provides a unique view into the rest of the framework.

What is DanderSpritz?

DanderSpritz is a full-featured post-exploitation framework used by the Equation Group. This framework was usually leveraged after exploiting a machine and deploying the PeddleCheap "implant". DanderSpritz is very modular and contains a wide variety of tools for persistence, reconnaissance, lateral movement, bypassing Antivirus engines, and other such shady activities. It was leaked by The Shadow Brokers on April 14th, 2017 as part of the "Lost in Translation" leak.

DanderSpritz Structure and Execution Flow

DanderSpritz logic can be found effectively split in two inside the directory tree of the "Lost in Translation" leak:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

leak root

----(...)

----windows

--------bin

--------exploits

--------fuzzbunch

--------implants

--------payloads

--------resources

--------specials

--------touches

----(...)

leak root ----(...) ----windows --------bin --------exploits --------fuzzbunch --------implants --------payloads --------resources --------specials --------touches ----(...)

```
leak root
----(...)
----windows
--------bin
--------exploits
--------fuzzbunch
--------implants
--------payloads
--------resources
--------specials
--------touches
----(...)
```

The core functionality of DanderSpritz is contained in the file `DszLpCore.exe`, which can be found at `windows/bin`. The framework's plugins and complex components, including DoubleFeature which we will later discuss in detail, can be found under `windows/resources`. `fuzzbunch`, `implants`, and the other directories under `windows` contain modules separate from DanderSpritz which are used for exploitation itself, taking control of victim systems, initial data gathering and so on; these are all beyond the scope of this publication.

The basic logical unit inside DanderSpritz is what we dub a “plugin”. These reside in `windows/resources`; there are about a dozen of them and they have a very specific directory structure, seen in the diagram below (though some of these subdirectories are optional).

There are also some other directories under `windows\\resources` that are not plugins (and therefore do not have this structure), and instead contain miscellaneous auxiliary scripts (such as validation scripts to verify the affiliation of victim machines).

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

plugin root

----aliases

----commands

----modules

--------descriptions

--------files-dsz

------------x86

----------------<module_name>

----------------<module_name>

----------------(...)

------------x64

----------------<module_name>

----------------<module_name>

----------------(...)

------------(...)

----payloads

--------descriptions

----pylp

----pyscripts

----scripts

----tools

----uploads

----version

plugin root ----aliases ----commands ----modules --------descriptions --------files-dsz ------------x86 ---------------- <module_name> ----------------<module_name> ----------------(...) ------------x64 ----------------<module_name> ---- ------------<module_name> ----------------(...) ------------(...) ----payloads --------descriptions ----pylp ----pyscripts ---scripts ----tools ----uploads ----version

```
plugin root
----aliases
----commands
----modules
--------descriptions
--------files-dsz
------------x86
----------------<module_name>
----------------<module_name>
----------------(...)
------------x64
----------------<module_name>
----------------<module_name>
----------------(...)
------------(...)
----payloads
--------descriptions
----pylp
----pyscripts
----scripts
----tools
----uploads
----version
```

- `Aliases` and `Commands` – These both contain XML files that declare support for “aliases” and “commands”, respectively, which serve a similar function. When a user of the DanderSpritz framework issues a shell command (in the general sense of the word – in the same way that a Bash user would run `ls`, `top` and so on), DanderSpritz will iterate over every plugin, check these XMLs and verify whether they declare support for the shell command the user typed. If the command appears under `Aliases` it will be simply mapped to an existing script; a `Command` will typically, under the hood, invoke the inner logic of the plugin in some way. This effectively means that a user of DanderSpritz can run many different shell

commands to achieve various results without being aware that behind the scenes, the same plugin handled the execution of all these requests. Under `Commands` (but not `Aliases`), additionally to the XML, there is an XSL file which specifies a format for the command's output as returned to the DanderSpritz user (XSL is a markup language for specifying presentation style for XML data — it is to XML as CSS is to HTML).

- `Modules` – Most of the plugin logic is contained in this directory. As can be inferred from the name, the logic is further divided into smaller "modules" of functionality. The `descriptions` subdirectory contains an XML file which is a sort of "manifest". It details what scripts and binaries should be run on the victim machine and on the "LP" ("Listening Post" — an attacker-controlled machine remotely monitoring the victim). It also lists the plugin's dependencies on other modules, its interface data, what computing architectures it supports, and whether it should run on the victim machine or on the LP. A few plugins also contain a `payloads` directory with a similar function.
- `PyLp` – Contains XML files for formatting incoming information exfiltrated from the victim machine. For every "message type" (kind of exfiltrated information), an XML specifies a Python script that formats the data for convenient display. This formatting script resides in the `PyScripts` directory.
- `PyScripts` – All the miscellaneous Python scripts used by the framework are in this directory.
- `Scripts` – This directory also contains miscellaneous scripts, written in some sigil-heavy scripting language that might have seemed reasonable to use before Python's rise to prominence.
- `Tools` – A grab-bag of self-contained material (PEs, DLLs, scripts, JARs, text files, …) which the authors figured they'd rather just include and invoke as-is.
- `Uploads` – stand-alone binaries which are pushed to the victim system by the plugin.
- `Version` – contains an XML file containing the plugin version.

Below we detail the typical control flow when a plugin alias or command is invoked.

1. The DanderSpritz user types a shell command in the DanderSpritz user interface which is, behind the scenes, implemented using that specific plugin.

Figure 1: The User Interface of DanderSpritz and its shell commands.

1. DanderSpritz's main logic iterates over the `resources` directory, looking at one plugin directory after the other. For each plugin directory, DanderSpritz looks at the `aliases` subdirectory and the `commands` subdirectory, and scrutinizes the XML file within, looking for a declared exported functionality matching the shell command. The match is found, and the matched XML element specifies a path inside the plugin's `pyscripts` directory.
2. DanderSpritz computes the fully qualified path of the invoked script (by appending the path specified in the matched XML element to the path of the plugin's `pyscripts` directory) and executes the file. This is where the user interface of the invoked shell command is displayed, and the plugin can be said to be properly running. (Ideally, this Python script is just UI and glue, while the core functionality that interacts with the victim machine resides in a separate remote component; but this appealing abstraction is broken somewhat in the `DoubleFeature` plugin which we will dig into later.)
3. Now the attacker gets to stare at the UI of the tool they invoked for as long as they like. Eventually, they will probably want to invoke some functionality through this UI. Depending on the functionality chosen, the Python UI constructs a Remote Procedure Call (taken from raw hardcoded data inside the Python — no XMLs here). It sends this RPC to the DanderSpritz component on the victim machine. This component on the victim side then executes the call and returns a result. In this way, RPCs are used as the API which the component on the LP accesses to perform actions on the victim machine (such as collecting screenshots or recording voice). This API is decoupled from the way these actions are actually implemented on the victim component side.
4. The RPC returns with the precious information required by the attacker (or maybe just a terse "action accomplished"). The Python UI consults the XML in the Plugin's `PyLP` directory that matches the result's

message type. This XML specifies how to display the returned information on the LP end, and the UI does so.

Figure 2: Example of XML files (both LP and Target) of a specific command.

Focus on DoubleFeature

To better understand the above structure and flow, we focused our research on a component of DanderSpritz named Doublefeature (or `Df` for short). According to its own internal documentation, this plugin "Generates a log & report about the types of tools that could be deployed on the target"; a lot of the framework tools, in their own internal documentation, make the chilling claim that DoubleFeature is the *only* way to confirm their existence on a compromised system. After some pause, we figured that at least this means DoubleFeature could be used as a sort of Rosetta Stone for better understanding DanderSpritz modules, and systems compromised by them. DoubleFeature effectively, well, doubles as a diagnostic tool for victim machines carrying DanderSpritz — It's an incident response team's pipe dream.

Figure 3: Code of strangeland.py referring to the fact that the only way to confirm is with DF.

In a perfect world, we wouldn't need to explain anything about the inner workings of DoubleFeature under the hood. After all, we just went through a whole section on how DanderSpritz plugins *in general* work under the hood; and DoubleFeature is one such plugin; therefore, everything above about RPC calls whose return values are formatted per an XSL specification should still hold — right?

Unfortunately, because of DoubleFeature's unique function as a logging module, it collects a large amount of data of various types. RPC return values and XSL markup are just not suited to transfer and display information on this scale. An unexpected corner use case emerged, an ad-hoc solution was created specifically for it, and the "pretty and elegant framework for everything" vision was quietly taken to the backyard and shot. It's a tale as old as time.

Figure 4: DoubleFeature main menu

DoubleFeature's `PyScripts` directory contains its Python UI interface (doublefeature.py) — but when the attacker chooses an option from the UI menu, behind the scenes instead of simply issuing an RPC, the script transmogrifies a "template" DLL, `DoubleFeatureDll.dll.unfinalized` , that resides in the plugin's `uploads` directory. The Python invokes the external tool `AddResource.exe` , found in the plugin's `tools` directory, to implant a resource into the already-compiled DLL and make it ready to detonate, under a new name: `DoubleFeatureDll.dll.configured` . The exact command run is:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

local run -redirect command "<g_dfconfiguretool> cmpf 6 1104 <configureddllpath> <g_dfrscfile>""

local run -redirect command "<g_dfconfiguretool> cmpf 6 1104 <configureddllpath> <g_dfrscfile>""

```
*local run -redirect  command "<g_dfconfiguretool> cmpf 6 1104 <configureddllpath> <g_dfrscfile>"*"
```

The flags used by the command are explained below.

- `c` (compressed) – Zlib compress the data
- `m` (munge) = Obfuscate the resource by XORing with pseudo-random bytes. The bytes are generated by running a PRNG (a 32-bit LCG, if you insist) and using the execution timestamp as the seed; to allow recovery, the seed is prepended to the obfuscated resource.
- `p` (place) = Place the resource into a homebrew resource directory (more detail about this later).
- `f` (finalize) = Finalize the proprietary resource directory.
- `6` = Type of resource (in this case, the enum value 6 translates to `RT_STRING` , a string-table entry)

- `1104` = Name of the resource.

After the main plugin DLL **is endowed with this new resource, the Python UI uses the DanderSpritz `dllload` shell command to load it on the victim machine:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

dllload -ordinal 1 -library <configuredDllPath>

dllload -ordinal 1 -library <configuredDllPath>

```
dllload -ordinal 1 -library <configuredDllPath>
```

Once the DLL on the victim side finishes running and writing the report to the log file on the victim machine, the Python UI exfiltrates the log file back to the attacker machine using the following DanderSpritz shell command:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

foreground get <log_file_name> -name DFReport

foreground get <log_file_name> -name DFReport

```
foreground get <log_file_name> -name DFReport
```

While (as mentioned above) most output of DanderSpritz commands is viewed according to XSL specifications, the output of DoubleFeature is too large and varied for this approach to be feasible. Instead, the attacker typically views the log file using a specialized program written for this purpose — `DoubleFeatureReader.exe`, which can be available in the plugin's `tools` directory.

DoubleFeature writes all its log data to a debug log file named `~yh56816.tmp`; this artifact was covered in Kaspersky's 2015 report on the remote access tool dubbed "EquationDrug" (more on that below). This log file is encrypted using the AES algorithm. Unless the user changes the key manually, the default one used is `badc0deb33ff00d` (possibly to spite vegan developers).

Main DLL of DoubleFeature

When the patched DLL (`DoubleFeatureDll.dll.configured`) is first loaded on the victim machine, it looks for a resource named “106” in a homebrew resource directory. This directory resides in the “.text” section right after the actual code, and the DLL is able to find it by searching for a distinct magic value. The homebrew resource directory has the following structure:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

Resource_Directory_struct:

word word_0

word num_of_resources

Resource_data[] resource_array

dword resource_directory_size

dword magic_hash

Resource_data:

word resource_type

word resource_num

dword offset_from_directory_start

dword resource_size

Resource_Directory_struct: word word_0 word num_of_resources Resource_data[] resource_array dword resource_directory_size dword magic_hash **Resource_data**: word resource_type word resource_num dword offset_from_directory_start dword resource_size

```
**Resource_Directory_struct**:
        word word_0
        word num_of_resources
        Resource_data[] resource_array
  dword resource_directory_size
  dword magic_hash

**Resource_data**:
        word resource_type
        word resource_num
```

```
        dword offset_from_directory_start
        dword resource_size
```

This resource (which is distinct from the resource earlier grafted onto the DLL by invoking `AddResource.exe`) is encrypted at rest, and in order to be used, it must be decrypted and decompressed. The (equivalent Python of the) logic is below.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

def decrypt_decompress_resource(buf, seed):

output = bytearray(b'')

for i in range(len(buf)):

seed = (0xDD483B8F - (0x6033A96D * seed) % (2**32)) % (2**32)

cur_xor_key = seed >> 8

output.append((cur_xor_key & 0xff) ^ (buf[i] & 0xff))

uncompressed_resource = zlib.decompress(output[4:])

return uncompressed_resource

def decrypt_decompress_resource(buf, seed): output = bytearray(b'') for i in range(len(buf)): seed = (0xDD483B8F - (0x6033A96D * seed) % (2**32)) % (2**32) cur_xor_key = seed >> 8 output.append((cur_xor_key & 0xff) ^ (buf[i] & 0xff)) uncompressed_resource = zlib.decompress(output[4:]) return uncompressed_resource

```
def decrypt_decompress_resource(buf, seed):
        output = bytearray(b'')
        for i in range(len(buf)):
                seed = (0xDD483B8F - (0x6033A96D * seed) % (2**32)) % (2**32)
                cur_xor_key = seed >> 8
                output.append((cur_xor_key & 0xff) ^ (buf[i] & 0xff))
  uncompressed_resource = zlib.decompress(output[4:])
        return uncompressed_resource
```

Resource 106, once decompressed, is a driver called **hidsvc.sys.** It is loaded into the kernel by invoking the EpMe exploit of CVE-2017-0005 (this is the very same exploit that had its logic find its way into the Jian exploit somehow). After the driver is loaded, the DLL begins communicating with it using `DeviceIoControl` s. The most

interesting IOControlCode supported by the driver is 0x85892408, which allows user-mode code to directly invoke kernel functions by simply specifying the function name and the arguments. The driver expects incoming messages with this code to be bundled with the following struct:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

ControlCode_input_buffer:

dword export_func_hash

dword num_of_arguments_bufs

dword [0x20*num] arguments_buf

ControlCode_input_buffer: dword export_func_hash dword num_of_arguments_bufs dword [0x20*num] arguments_buf

```
**ControlCode_input_buffer**:
        dword export_func_hash
        dword num_of_arguments_bufs
        dword [0x20*num] arguments_buf
```

Most arguments are self-explanatory, given the purpose of this control code. The one detail that bears explanation is the `export_func_hash` – the function name is not passed explicitly, but instead a checksum of it. Upon receiving the struct, the driver iterates over every exported function of `ntoskernl.exe` computes the resulting checksum and compares the result with the provided `export_func_hash`. Once a match is found, the driver concludes it has found the correct function. This is a standard method to obfuscate API calls, seen in many other pieces of malware.

The checksum computation logic can be seen below.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

def checksum(name, len):

val = 0

for i in range(1, len+1):

temp = (0x1A6B8613 * i) % (2**32)

val = val ^ (temp * ord(name[i-1]) % (2**32))

return val

def checksum(name, len): val = 0 for i in range(1, len+1): temp = (0x1A6B8613 * i) % (2**32) val = val ^ (temp * ord(name[i-1]) % (2**32)) return val

```
def checksum(name, len):
        val = 0
        for i in range(1, len+1):
                temp = (0x1A6B8613 * i) % (2**32)
                val = val ^ (temp * ord(name[i-1]) % (2**32))
        return val
```

Some sample checksum values:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

ZwQueryInformationProcess 0x62A0A841

ZwQueryObject 0xB7241E54

ZwOpenEvent 0xDE2837FA

ZwSetEvent 0x662E22E1

ZwOpenKey 0xA20F6388

ZwFsControlFile 0x407CC9F5

ZwQueryVolumeInformationFile 0x161C4B69

ZwQueryInformationFile 0xC0E4A30A

ZwSetInformationFile 0x535ACCEA

ZwReadFile 0xB8075119

ZwWriteFile 0xBAE70F4B

ZwClose 0x69023181

ZwCreateFile 0x01862336

ZwQueryDirectoryFile 0x41483801

ZwQuerySystemInformation 0x178B07C8

ZwCreateKey 0x01862336

ZwDeleteKey 0x2C9F7CA8

ZwQueryKey 0xBDD598E2

ZwEnumerateKey 0x8D3E3E7A

ZwSetValueKey 0x90A71127

ZwEnumerateValueKey 0x040F9817

ZwQueryValueKey 0xF655B34B

ZwDeleteValueKey 0x2A1BF746

ZwWaitForSingleObject 0x86324d14

ZwQueryInformationProcess 0x62A0A841 ZwQueryObject 0xB7241E54 ZwOpenEvent 0xDE2837FA ZwSetEvent 0x662E22E1 ZwOpenKey 0xA20F6388 ZwFsControlFile 0x407CC9F5 ZwQueryVolumeInformationFile 0x161C4B69 ZwQueryInformationFile 0xC0E4A30A ZwSetInformationFile 0x535ACCEA ZwReadFile 0xB8075119 ZwWriteFile 0xBAE70F4B ZwClose 0x69023181 ZwCreateFile 0x01862336 ZwQueryDirectoryFile 0x41483801 ZwQuerySystemInformation 0x178B07C8 ZwCreateKey 0x01862336 ZwDeleteKey 0x2C9F7CA8 ZwQueryKey 0xBDD598E2 ZwEnumerateKey 0x8D3E3E7A ZwSetValueKey 0x90A71127 ZwEnumerateValueKey 0x040F9817 ZwQueryValueKey 0xF655B34B ZwDeleteValueKey 0x2A1BF746 ZwWaitForSingleObject 0x86324d14

```
ZwQueryInformationProcess    0x62A0A841
ZwQueryObject                0xB7241E54
ZwOpenEvent                  0xDE2837FA
ZwSetEvent                   0x662E22E1
ZwOpenKey                    0xA20F6388
ZwFsControlFile              0x407CC9F5
ZwQueryVolumeInformationFile 0x161C4B69
ZwQueryInformationFile       0xC0E4A30A
ZwSetInformationFile         0x535ACCEA
ZwReadFile                   0xB8075119
ZwWriteFile                  0xBAE70F4B
ZwClose                      0x69023181
ZwCreateFile                 0x01862336
```

```
ZwQueryDirectoryFile        0x41483801
ZwQuerySystemInformation    0x178B07C8
ZwCreateKey                 0x01862336
ZwDeleteKey                 0x2C9F7CA8
ZwQueryKey                  0xBDD598E2
ZwEnumerateKey              0x8D3E3E7A
ZwSetValueKey               0x90A71127
ZwEnumerateValueKey         0x040F9817
ZwQueryValueKey             0xF655B34B
ZwDeleteValueKey            0x2A1BF746
ZwWaitForSingleObject       0x86324d14
```

This isn't the only aspect of DoubleFeature (and other Equation Group tools) to make life difficult for forensic analysts. The strings used in DoubleFeature are decrypted — that alone is very standard — but they are decrypted on-demand per function, which is somewhat more frustrating than usual, and they are *re-encrypted* once function execution completes, which is *much* more frustrating than usual. DoubleFeature also supports additional obfuscation methods, such as a simple substitution cipher:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

def deobfuscate_strings(enc_strings):

for enc_string in enc_strings:

replace_buffer = [0x37, 0x3B, 0x5D, 0x4B, 0x45, 0x44, 0x3C, 0x5C, 0x7B, 0x4F,

0x74, 0x41, 0x7D, 0x7E, 0x35, 0x46, 0x23, 0x2B, 0x72, 0x71,

0x40, 0x78, 0x4C, 0x55, 0x39, 0x56, 0x30, 0x5F, 0x50, 0x2C,

0x29, 0x2D, 0x79, 0x59, 0x3A, 0x57, 0x53, 0x69, 0x77, 0x63,

0x26, 0x70, 0x2A, 0x76, 0x60, 0x3D, 0x33, 0x31, 0x22, 0x47,

0x49, 0x4E, 0x75, 0x58, 0x34, 0x68, 0x6B, 0x20, 0x67, 0x32,

0x27, 0x65, 0x51, 0x28, 0x5B, 0x2E, 0x7C, 0x6F, 0x24, 0x4A,

0x3E, 0x64, 0x73, 0x6D, 0x7A, 0x3F, 0x6A, 0x54, 0x62, 0x42,

0x6C, 0x48, 0x2F, 0x25, 0x43, 0x52, 0x21, 0x66, 0x38, 0x5A,

0x61, 0x5E, 0x36, 0x4D, 0x6E, 0x00]

dec_string = ''

for i in range(len(enc_string)):

cur_place = ord(enc_string[i]) - 0x20

dec_string += chr (replace_b

uffer[cur_place])

print(dec_string)

def deobfuscate_strings(enc_strings): for enc_string in enc_strings: replace_buffer = [0x37, 0x3B, 0x5D, 0x4B, 0x45, 0x44, 0x3C, 0x5C, 0x7B, 0x4F, 0x74, 0x41, 0x7D, 0x7E, 0x35, 0x46, 0x23, 0x2B, 0x72, 0x71, 0x40, 0x78, 0x4C, 0x55, 0x39, 0x56, 0x30, 0x5F, 0x50, 0x2C, 0x29, 0x2D, 0x79, 0x59, 0x3A, 0x57, 0x53, 0x69, 0x77, 0x63, 0x26, 0x70, 0x2A, 0x76, 0x60, 0x3D, 0x33, 0x31, 0x22, 0x47, 0x49, 0x4E, 0x75, 0x58, 0x34, 0x68, 0x6B, 0x20, 0x67, 0x32, 0x27, 0x65, 0x51, 0x28, 0x5B, 0x2E, 0x7C, 0x6F, 0x24, 0x4A, 0x3E, 0x64, 0x73, 0x6D, 0x7A, 0x3F, 0x6A, 0x54, 0x62, 0x42, 0x6C, 0x48, 0x2F, 0x25, 0x43, 0x52, 0x21, 0x66, 0x38, 0x5A, 0x61, 0x5E, 0x36, 0x4D, 0x6E, 0x00] dec_string = '' for i in range(len(enc_string)): cur_place = ord(enc_string[i]) - 0x20 dec_string += chr (replace_b uffer[cur_place]) print(dec_string)

```
def deobfuscate_strings(enc_strings):
        for enc_string in enc_strings:
                replace_buffer = [ 0x37, 0x3B, 0x5D, 0x4B, 0x45, 0x44, 0x3C, 0x5C, 0x7B, 0x4F,
                  0x74, 0x41, 0x7D, 0x7E, 0x35, 0x46, 0x23, 0x2B, 0x72, 0x71,
                  0x40, 0x78, 0x4C, 0x55, 0x39, 0x56, 0x30, 0x5F, 0x50, 0x2C,
                  0x29, 0x2D, 0x79, 0x59, 0x3A, 0x57, 0x53, 0x69, 0x77, 0x63,
                  0x26, 0x70, 0x2A, 0x76, 0x60, 0x3D, 0x33, 0x31, 0x22, 0x47,
                  0x49, 0x4E, 0x75, 0x58, 0x34, 0x68, 0x6B, 0x20, 0x67, 0x32,
                  0x27, 0x65, 0x51, 0x28, 0x5B, 0x2E, 0x7C, 0x6F, 0x24, 0x4A,
                  0x3E, 0x64, 0x73, 0x6D, 0x7A, 0x3F, 0x6A, 0x54, 0x62, 0x42,
                  0x6C, 0x48, 0x2F, 0x25, 0x43, 0x52, 0x21, 0x66, 0x38, 0x5A,
                  0x61, 0x5E, 0x36, 0x4D, 0x6E, 0x00]

                dec_string = ''
                for i in range(len(enc_string)):
                        cur_place = ord(enc_string[i]) - 0x20
                        dec_string += chr (replace_b

uffer[cur_place])
                print(dec_string)
```

And a stream cipher based on a simple homebrew linear PRNG:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

def decrypt(seed, buffer, mask, first_hex_seed, second_hex_seed):

outut = ''

for i in range(len(buffer)):

seed = (((first_hex_seed * seed) % (2 ** 32)) + second_hex_seed) % (2 ** 32)

cur_xor_key = ((seed >> 16)) | mask

output += chr ((cur_xor_key & 0xff) ^ buffer[i])

return output

def decrypt(seed, buffer, mask, first_hex_seed, second_hex_seed): outut = '' for i in range(len(buffer)): seed = (((first_hex_seed * seed) % (2 ** 32)) + second_hex_seed) % (2 ** 32) cur_xor_key = ((seed >> 16)) | mask output += chr ((cur_xor_key & 0xff) ^ buffer[i]) return output

```
def decrypt(seed, buffer, mask, first_hex_seed, second_hex_seed):
        outut = ''
        for i in range(len(buffer)):
                seed = (((first_hex_seed * seed) % (2 ** 32)) + second_hex_seed) % (2 ** 32)
                cur_xor_key = ((seed >> 16)) | mask
                output += chr ((cur_xor_key & 0xff) ^ buffer[i])
        return output
```

As mentioned above, by virtue of its function, DoubleFeature is a unique source of knowledge pertaining to Equation Group tools — after all, the entire logging module depends on an ability to query these tools on a victim system and verify which are present. Below we list some of the tools probed by the logging module, some of which were unknown.

Apart from resources 106 and 1104, which are actively used in the DLL's execution flow, the main DLL's homebrew resource directory also contains the following resources:

- Resource 1004 – UnitedRake Restart DLL.
- Resource 1005 – UnitedRake Shutdown DLL.
- Resource 1006 – StraitBiZarre Restart DLL.
- Resource 200 – Hashes of known boot managers that are being compared to BCD partition data.
- Resource 1007 – Upgrade KillSuit module DLL — references to it can be found in the code, but it can no longer be physically found in the directory. Possibly it existed in earlier versions of the DLL and was removed later.

Plugins Monitored by DoubleFeature

UnitedRake

UnitedRake (UR) is a remote access tool that can be used to target Windows machines. It is an extensible and modular framework that is provided with a large number of plugins that perform different information collection functions. This is the tool that Kaspersky dubbed "EquationDrug" in their original report, published before the Shadow Brokers leak. The leak also included the UnitedRake manual, which contained the configuration, commands, and modules of this tool. DoubleFeature supports many management functions related to UnitedRake such as Shutdown, TipOff, KickStart and Enabling/Disabling logging.

We came across the following indicators of UnitedRake:

- MSNDSRV.sys – Kernel mode stage 0 and rootkit. Implements an NDIS driver for filtering the network traffic. Until UR version 4.0.
- ATMDKDRV.sys – Network-sniffer/patcher. Since UR version 4.1.
- "*Software\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2109}\TypeLib*" or "*\Registry\Machine\SOFTWARE\Classes\CLSID\{091FD378-422D-A36E-8487-83B57ADD2209}\TypeLib*" – contains the GUID of UR, the special key registry key.
- "*\Registry\Machine\System\CurrentControlSet\Control\Session Manager\MemSubSys\{95FFB832-8B00-6E10-444B-DC67CAE0118A-F6D58114}*" – KillSuit Logging data related registry key.
- "*Global\64322D88-0CEA-4ce0-8562-67345B70C655*" – File Mapping created in TipOff command.
- "*Global*6F27089A-3482-4109-8F5B-CB3143A1AB9A" and "*Global*667FBF02-F406-4C0A-BA65-893747A0D372" – Events created in UR Shutdown.
- {A0CCDC61-7623-A425-7002-DB81F353945F-5A8ECFAD} – UnitedRake 3/4 Config Data and Transport Info CLSID
- {30F3976F-90F0-B438-D324-07E031C7507E-981BE0DD} – UnitedRake Plugins Info CLSID
- {95FFB832-8B00-6E10-444B-DC67CAE0118A-F6D58114} – UnitedRake Logging data CLSID
- {01C482BA-BD31-4874-A08B-A93EA5BCE511} – UnitedRake's mutex name.

StraitBizarre

StraitBizarre (SBZ) is an implant used for stealthy data exfiltration which is performed over **FriezeRamp** – a custom network protocol that is similar to IPSEC. It's a cross-platform project, and different versions exist supporting Windows, Linux and mobile platforms (e.g. **DROPOUTJEEP** for iPhone, and there's even **TOTEGHOSTLY** for Windows Mobile).

Figure 5: StraitBizzare information. Source: Der Spiegel

We came across the following indicators of StraitBizarre inside DoubleFeature:

- {1B8C5912-8BE4-11D1-B8D3-F5B42019CAED} – SBZ CLSID for GUID, version and Special Status Keys.

KillSuit

KillSuit (KiSu) (“GrayFish” in the original Kaspersky report) is an unusual plugin in that once deployed on the victim machine, its entire mission is running other plugins, providing a framework for persistence and evasion. Some (not all) DanderSpritz plugins can be either run individually, or be invoked through KillSuit. Its design is such that every instance of KillSuit running on the victim side can host a single tool (such as MistyVeal, below); and so, it can easily happen that a victim machine will have several instances of KillSuit installed on it, each hosting a different post-exploitation tool. The data for each KillSuit instance, including all its modules, is kept encrypted in registry entries. This is something unique to KillSuit and is not a feature of DanderSpritz plugins in general.

DoubleFeature logs a great amount of data pertaining to KillSuit. In fact, there is also some dead code inside DoubleFeature that allows deleting, upgrading and pushing module updates into running KillSuit instances (we agree with the decision to deprecate this code; after all, DoubleFeature is supposed to be used for logging, and we’ll soon see this functionality in the KillSuit Python UI, where it belongs). While “KillSuit” is the name used inside DoubleFeature and in the outer-layer DanderSpritz CLI that the attacker will actually invoke, actually the

Plugin folder name used internally is DecibalMinute (DeMi for short). The Python UI logic can mainly be found inside 3 scripts that, unsurprisingly, reside in the plugin's `pyscripts` directory.

- "**Mcl_Cmd_DiBa_Tasking.py**" – handles KiSu installation, uninstallation and upgrades. As a parameter, this script accepts the type of persistence mechanism to use; there are 4 types of persistence, helpfully named "Default", "Launcher", "SoTi" and "JuVi". We elaborate on their internal workings a bit further below. Under the hood, the Python UI implements this via an RPC call (RPC_INFO_INSTALL).
- "**Mcl_Cmd_KisuComms_Tasking.py**" – used to establish a connection with a running instance of KillSuit on the victim end, and provides functionality for dynamically loading and unloading modules/drivers.
- "**_KiSu_BH_enable.py**" – One of KillSuit's internal drivers is called "BroughtHotShot", or BH for short. This script does *not* enable it, but checks whether it is enabled (via DanderSpritz commands `available -command kisu_install -isloaded` and `available -command kisu_install -load`). If you want to enable the driver, you need to do `KiSu_BH_enable.py on`, and disabling it is `KiSu_BH_enable.py off`.
- "**Mcl_Cmd_KiSuFullList_Tasking.py**" – Produces a list of current KiSu installations on the target machine. Behind the scenes, this is done by invoking the `kisu_list` DanderSpritz command, and then for every returned installation, retrieving its configuration via the DanderSpritz command `kisu_config -instance id -checksum`. This configuration contains various technical details such as the KillSuit version, the installation's registry key and value, the loaders for the kernel and user modules, the directory of the encrypted virtual filesystem used to keep the hosted plugin's modules, the legitimate driver that'd been victimized by injecting the hosted plugin into it, and the flags used internally when launching KillSuit on the victim.

Every KillSuit instance has an internal record of an "ID" of the tool hosted inside the instance, which is universally the same per tool. We found referenced inside DoubleFeature to the following possible instances:

- PC (PeddleCheap) – 0x7A43E1FA – provides an interactive shell and some feature for long-term persistence. Also serves as a post-exploitation tool in itself, and can install other KillSuit instances on a compromised host.
- UR (UnitedRake) – 0x91FD378 – see above
- STLA (StrangeLand) / GROK – 0x1A0F5582 – these are both keyloggers. Their encrypted logs are stored inside files with names of the form `tm154*.da`.
- SNUN (SnuffleUnicorn) – 0x23A4732A
- WRWA (WraithWrath) – 0x502BB710
- SLSH (SleepySheriff) – 0x32A7032D
- WORA (WoozyRamble) – 0x68A40E49
- TTSU (TiltTsunami) – 0x8F1D6511
- SOKN (SoberKnave) – 0x8F1D6510 – This tool has functionality for data exfiltration through unused/disabled WiFi cards. It is used in Air-Gapped targets.
- MAGR (MagicGrain) – 0x437E52E8
- DODA (DoubleDare) – 0x1C9D4A8A
- SAAN (SavageAngel) – 0x9D801C63
- MOAN (MorbidAngel) – 0x9D801C62

- DEWH (DementiaWheel) – 0xAE37690B – Hacking tool also known as "Fanny".
- CHMU (ChinMusic) – 0x39B2DA17
- MAMO (MagicMonkey) – 0x2D473AB3
- MABE (MagicBean) – 0x8675309 – used for WiFi Man In The Middle

DiveBar

DiveBar (DiBa) is DoubleFeature's name for the part of KillSuit responsible for persistence methods (such as "KSLA" (KillSuit loader), "SolarTime", "JustVisiting" and "DoctorOcopus".

The different persistence methods, which we mentioned above by name, are:

- KSLA (Launcher) – Simply installs a new driver on the victim system and uses it for persistence. This worked up until Microsoft introduced Driver Signing Enforcement (DSE), which does not allow unsigned drivers to run. This method is not supported in Windows Vista and later.
- JustVisiting (JuVi) – In order to bypass DSE, this persistence mechanism abuses a known vulnerability in the signed driver ElbyCDIO.sys, which is a part of the software "CloneCD" by RedFox. The vulnerable driver is made to be loaded, and exploited, on system startup. The elevated privileges obtained in this way are then used to add DiveBar's persistence driver to LSAExtensionConfig/interfaces. This method is only compatible with Windows 8.
- SolarTime (SoTi) – An advanced persistence mechanism that works by modifying one of the victim system's VBRs. More details about this method can be found in this report by F-Secure. Only compatible with NTFS filesystems with FVEBOOT and a certain boot sector format. SoTi compares the hash of the boot sector to a list of "known good" hashes, which are listed below.

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

C454045E1299C5AD5E2932A7B0016D7A

C1544A2220F5DD61A62C697D9A2C5B77

05422319E7821018401F477B3621F8E2

4C85F9D2D0B02E0B3BDFC34D0F63B414

0023DE8F74BF9F932AFC9E288082E660

58B9130DEEFF83F1185C372595CD4607

B4A78F824A7F0FA688DF729F2AEF7F7F

DCE6AAAD1574BC72A25DC4551D52A2C1

C454045E1299C5AD5E2932A7B0016D7A C1544A2220F5DD61A62C697D9A2C5B77
05422319E7821018401F477B3621F8E2 4C85F9D2D0B02E0B3BDFC34D0F63B414
0023DE8F74BF9F932AFC9E288082E660 58B9130DEEFF83F1185C372595CD4607
B4A78F824A7F0FA688DF729F2AEF7F7F DCE6AAAD1574BC72A25DC4551D52A2C1

```
C454045E1299C5AD5E2932A7B0016D7A
C1544A2220F5DD61A62C697D9A2C5B77
05422319E7821018401F477B3621F8E2
4C85F9D2D0B02E0B3BDFC34D0F63B414
0023DE8F74BF9F932AFC9E288082E660
58B9130DEEFF83F1185C372595CD4607
B4A78F824A7F0FA688DF729F2AEF7F7F
DCE6AAAD1574BC72A25DC4551D52A2C1
```

As mentioned above, KillSuit keeps inside the victim registry something called a “module store”. Traditionally the registry has been used in malware to store simple configuration data, as per the registry’s legitimate purpose; but as years have passed, more and more malware has gotten bold in using the registry to store arbitrary data. Here the registry is made to swallow a whole Virtual File System containing the module store, which is generated by concatenating two words chosen pseudo-randomly from two hard-coded dictionaries (the creation time of the victim’s root directory is used as the seed). The list of possible values for the first word is reproduced below:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

Account

Acct

Adapter

App

Audit

Boot

Class

Correction

Debug

Dir

Directory

Domain

Driver

Event

Font

Hardware

Hiber

Host

Language

Legacy

Locale

Logon

Manufacturer

Media

Net

Network

OEM

Power

Prefetch

Privilege

Process

Remote

Scheduler

Security

Server

Shared

Shutdown

Startup

Task

Trust

Uninstall

User

Win16

Win32

Account Acct Adapter App Audit Boot Class Correction Debug Dir Directory Domain Driver Event Font Hardware Hiber Host Language Legacy Locale Logon Manufacturer Media Net Network OEM Power Prefetch Privilege Process Remote Scheduler Security Server Shared Shutdown Startup Task Trust Uninstall User Win16 Win32

```
Account
Acct
Adapter
App
Audit
Boot
Class
Correction
Debug
Dir
Directory
Domain
Driver
Event
Font
Hardware
Hiber
Host
Language
Legacy
Locale
Logon
Manufacturer
Media
Net
Network
OEM
```

```
Power
Prefetch
Privilege
Process
Remote
Scheduler
Security
Server
Shared
Shutdown
Startup
Task
Trust
Uninstall
User
Win16
Win32
```

And the possible values for the second word:

Plain text

Copy to clipboard

Open code in new window

EnlighterJS 3 Syntax Highlighter

Cache

Cfg

Config

Data

Database

Db

Exts

Flags

Hierarchy

History

Info

Libs

List

Logs

Mappings

Maps

Mgmt

Mon

Monitor

Performance

Plugins

Policy

Profile

Records

Registry

Settings

Setup

Support

Usage

Cache Cfg Config Data Database Db Exts Flags Hierarchy History Info Libs List Logs Mappings Maps Mgmt Mon Monitor Performance Plugins Policy Profile Records Registry Settings Setup Support Usage

```
Cache
Cfg
Config
Data
Database
Db
Exts
Flags
Hierarchy
History
Info
```

```
Libs
List
Logs
Mappings
Maps
Mgmt
Mon
Monitor
Performance
Plugins
Policy
Profile
Records
Registry
Settings
Setup
Support
Usage
```

Looking at the architecture of “GrayFish” as reported by Kaspersky, we are left with the impression that it is one and the same as KillSuit:

Figure 6: Architecture of GrayFish. Source: Kaspersky

The resources in the diagram are in a one-to-one correspondence with the DiveBar resources:

102 – fvexpy.sys – F7F382A0C610177431B27B93C4C87AC1

103 – mpdkg32.dll – 0182DBF3E594581A87992F80C762C099

104 – BroughtHotShot driver – drmkflt.sys – 9C6D1ED1F5E22BF609BCF5CA6E587DEC / D3DF8781249F2C404C4935CA9FFB1155

107 – New VBR (for SolarTime)

110 – mpdkg64.dll – F01525C9EF763C49E28CEC6C2F6F6C60

114 – Elby loader – fhsvcapi.dll – 6156E50571571B233019C4EBB472899D

115 – Elby driver – AAA8999A169E39FB8B48AE49CD6AC30A

DiveBar does not limit itself to abusing ElbyCDIO.sys; it also searches for vulnerable benign drivers already present on the victim's machine to be used as a "launcher" for the hosted plugin's code. Internally, such a benign driver that is chosen by DiveBar to launch KillSuit instance is called a "thunk" (this might be the place to mention that according to the Merriam-Webster dictionary, the only meaning of the word "thunk" is as a present-tense verb meaning "to produce a flat hollow sound"; the author of the original PE file format caused enough suffering by using this mysterious word as a description for an opaque digital object, and we are sad to see this practice repeated here). For every KillSuit instance, DoubleFeature reports the ~~thunk~~ exploited dll used to load its kernel-mode module, which is called the KML (Kernel Module Launcher) for short. A similar report is made for the User Mode Launcher (UML).

FlewAvenue

FlewAvenue(FlAv) is an IPv4 driver that provides covert network access for other tools. It provides different networking faculties such as DNS queries and ICMP echo ("ping").

We came across the following indicators of **FlewAvenue**:

- "ntevt.sys " – The name of this tool's driver.

DuneMessiah

DoubleFeature diagnostics only provide very minimal information regarding this tool. For this tool, DoubleFeature reports a pseudorandomly-generated "Event Name" that the instance on the victim machine uses internally, as well a number of "registered KillSuit instances".

CritterFrenzy

DoubleFeature reports only the bare minimum of information about this plugin as well. From the information we can see DoubleFeature collects regarding this tool, it seems to be another instance of KillSuit that probably has been used in the past and its ID was 333.

We came across the following indicators of CritterFrenzy:

- “MPDKH32” – The name of this tool.

MistyVeal

MistyVeal (MV) is a “validator” implant, meaning that it is used to verify that the targeted system is indeed an authentic victim and not a research environment. It is implemented as an Internet Explorer Browser Helper Object (these are typically used for extending IE functionality; for example, Adobe’s Acrobat plugin for IE is a Browser Helper Object). MistyVeal was also part of the Equation Group’s original “Double Fantasy” implant, a precursor of UnitedRake. You can read more about it and the connection to Regin in a report by EpicTurla.

- “nethdlr.sys” – The name of this tool’s driver.

We came across the following indicators of MistyVeal:

- {B812789D-6FDF-97AB-834B-9F4376B2C8E1} – MV CLSID for GUID and version.

DiceDealer

DiceDealer (DD), mentioned in the leaked UnitedRake manual, is a parsing tool for the logging data produced by all installations and uninstallations performed by DiveBar (this is relevant to UnitedRake because DiveBar is typically used to install it). If you are looking to manually parse DiceDealer log files, the easiest method is to copy the log file into the same directory where the DiceDealerReader tool is located. The reader is dependent on several of the files within that directory and will fail to parse the log if they are not present.

PeddleCheap

PeddleCheap ****(PC) is among the first tools to be run on a victim machine, and can be used to bootstrap a complete DanderSpritz installation. PeddleCheap has minimal functionality allowing attackers to connect to the victim machine and remotely install and configure implants that allow further post-exploitation functionality, including a full install of the DanderSpritz framework. PeddleCheap is usually injected into lsass.exe by several methods, including the DOUBLEPULSAR backdoor.

Figure 7: PeddleCheap User Interface.

We came across the following indicators of PeddleCheap:

- {A682FEC0-333F-B16A-4EE6-24CC2BAF1185} – PC CLSID for GUID and version.

Control flow of DoubleFeature’s Rootkit

The rootkit used by DoubleFeature (`hidsvc.sys`) performs the following actions when it is loaded:

- It creates an unnamed device object but registers IRP dispatch functions.
- It dispatches IOCTL requests.
- It specializes in run-time patching of Windows kernel code.
- It runs kernel APIs for the user-mode module.

The rootkit is patched by the user-mode DLL before being loaded into memory — this is done to insert the PID of the user-mode process so that the rootkit knows which process to hide. The rootkit then attaches to this accomplice user-mode process via `KeAttachProcess` .

The rootkit finds the dynamic addresses of API functions using HalAllocateCommonBuffer or MmIsAddressValid (the addresses for *these* functions are earlier obtained by invoking MmGetSystemRoutineAddress). It uses encrypted stack strings which are decrypted on a need-to-use basis and encrypted again immediately after they are used, similarly to the method used in the user-mode component of DoubleFeature that we described earlier.

In order to avoid detection, the rootkit also takes care to create its own driver objects as stealthily as possible. First, instead of creating the object directly, the rootkit creates a handle to `Device\\NULL` , then hijacks its FileHandle by inserting its own device object with the name `driver\\msvss` . Then, it uses this FileObject to send a `IRP_MJ_Create` request in order to obtain a handle to the newly created driver object. Second, the rootkit calls `ObMakeTemporaryObject` and removes the name of the object from its parent object manager directory, effectively unlinking it from the structs that the OS uses internally to keep track of loaded drivers. Because of the way Windows OS handles drivers, this has the effect of keeping the driver running in the background while diagnostic tools and researchers will fail to find the driver.

The IRP_MJ_DEVICE_CONTROL handler function of the new device contains the different IoControl codes that can be sent from the user-mode DLL (such as 0x8589240c for truncating a file, and 0x85892408 for executing an API call in kernel mode and sending the output back to the user-mode process).

Conclusion

Sometimes, the world of high-tier APT tools and the world of ordinary malware can seem like two parallel universes. Cybercriminals periodically produce the umpteenth Cryptolocker clone or, at most, another modular jack-of-all-trades Emotet wannabe; in the meanwhile, nation-state actors tend to clandestine, gigantic codebases, sporting a huge gamut of features that have been cultivated over *decades* due to practical need. For those of us with our heads deep enough up the cybercrime industry's nether regions, many of the features described above — rootkits, dedicated components to thoroughly vet victims, whole systems dedicated to logging the stages of post-exploitation — are, largely, abstract theory. The cybercrime industry's DoubleFeature is typically an HTTP `GET` request containing `&OS=win10` , encrypted by some homebrew variant of RC4. The gap can really not be overstated.

It's not often that we get such a candid glimpse into tools of this degree of sophistication, as the Shadow Brokers leak allowed us. The DanderSpritz-tier projects of the world are naturally covered by a shroud of secrecy — even, as we've seen, from fellow APT actors, who can maybe at best get their hands on a rival tool once in a blue moon, as happened with EpRom which led to the creation of Jian. As an industry, it turns out we too are still slowly

chewing on the 4-year-old leak that revealed DanderSpritz to us, and gaining new insights. On the defenders' side, we have the duty to study these marvels of infosec engineering carefully and apply the lessons learned — before lower-tier, run-of-the-mill attackers do the same.

Appendix 1: Table of Command-Line Argument Supported by DoubleFeature Main DLL

Option	Arguments	Relevant command in the menu	Description
-n/-m	Registry key name		Check if a registry key exists or not.
-o	–		Returns DuneMessiah information
-p	filename		Changes the log file name.
-q	Hash, ID/Name		Deletes KillSuit module.
-s	'u'/'s'		Shutdowns UnitedRake or StraitBizarre
-t	IP, port		TipOffs UnitedRake
-u	Hash, ID/Name		Upgrades KillSuit module
-v	Hash, ID/Name		Downloads KillSuit module
-x	filename		Truncates file on victim's computer
-g	IP:port		Checks FlewAvenue feature compatibility
-h	'u'/'s'		Stops FA in UR or SBZ
-i	k = KillSuit m = Manual Instances p = Processes info q = Modules Data s = StraitBiZarre u = UnitedRake c= CritterFrenzy d = DiveBar e = Loaded Drivers f = FlewAvenue g = Special Implant i = Implant Independet		Gives information about the tool given as an argument

-j	0/1		Get DiceDealer logs – DiveBar information on the victim's computer.
-k	'u'/'s'		KickStarts UR or SBZ
-l	–	DFQuery	Query information as in 'i' command but on several tools on one command.
-f	–		Toggles FlewAvenue mode – as a network sniffer or as a memory patcher. It does this by changing the "config2" SubKey.
-a	new AES key		Replaces the AES key for encrypting the logs.
-b/-e	registry key		Deletes DiveBar registry key
-c	–		–
-d	on/off		Enable/Disable UR logging

Source: https://research.checkpoint.com/2021/a-deep-dive-into-doublefeature-equation-groups-post-exploitation-dashboard/