In depth analysis of Lazarus validator

blog.malwarelab.pl/posts/lazarus_validator/

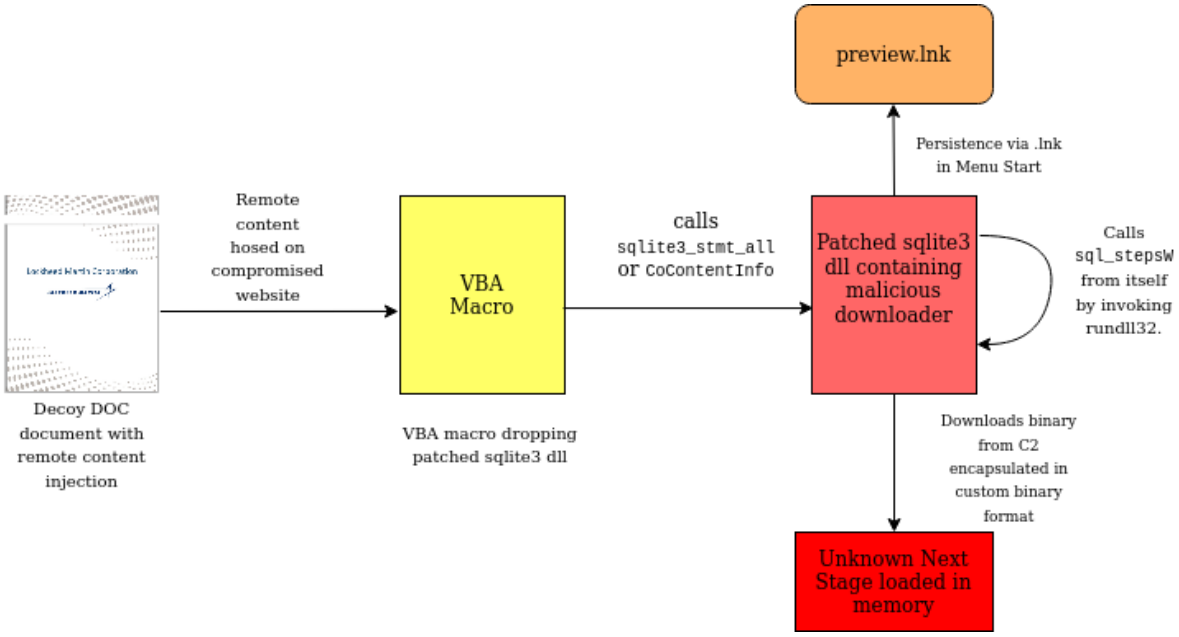
May 15, 2020

Intro

Few days ago we found interesting Word document impersonating Lockheed Martin[1]. Some time later we realized that this sample was a part of larger and older campaign conducted probably against various military contractor conducting businesses with South Korea and that this campaign was already described, however w couldn't find any in depth analysis of a validator used by Lazarus so here it is.

Infection vector

There is already a very good analysis done by StrangerealIntel, including an intelligence brief explaining potential reason for this campaign, so we wont into much details here. Whole infection vector can be summarized by flowing picture.



This script can be used to extract files embedded into DOC file as well as key and dll name. It is important to notice that beside two dll's malicious document also contains another DOC file with a rest of a lure document so target is presented with whole document not only with a first page which can put away suspicion.

During our analysis we where worked with following DOC files

MD5 Hash	Drop name	Key	Hashes of embedded files
65df11dea0c1d0f0304b376787e65ccb	wsuser.db	S-6-38-4412-76700627-315277-3247	PE x64:2b02465b65024336a9e15d7f34c1f5d9 DLL x86:f6d6f3580160cd29b285edf7d0c647ce DOC:223e954fd67c6cf75c3a6f987b94784b

MD5 Hash	Drop name	Key	Hashes of embedded files
0071b20d27a24ae1e474145b8efc9718	wsuser.db	S-6-81-3811-75432205-060098-6872	PE x64:78d42cedb0c012c62ef5be620c200d43 DLL x86:14d79cd918b4f610c1a6d43cadeeff7b DOC:59fab92d51c50467c1356080e6a5dead
1f254dd0b85edd7e11339681979e3ad6	wsdts.db	S-6-81-3811-75432205-060098-6872	PE x64:075fba0c098d86d9f22b8ea8c3033207 DLL x86:c0a8483b836efdbae190cc069129d5c3 DOC:074c02f7f5badd5c94d840c1e2ae9f72
2efbe6901fc3f479bc32aaf13ce8cf12	onenote.db	S-6-38-4412-76700627-315277-3247	PE x64:f4b55da7870e9ecd5f3f565f40490996 DLL x86:11fdc0be9d85b4ff1faf5ca33cc272ed DOC:3aa8eddf26f5944a24dfeb57c9f49a17
265f407a157ab0ed017dd18cae0352ae	thumnail.db	S-6-38-4412-76700627-315277-3247	PE x64:59cb8474930ae7ea45b626443e01b66d DLL x86:d1c652b4192857cb08907f0ba1790976 DOC:b493f37ee0fddb1d832ddacaaf417029

Validator

Binary is responsible of getting next stage, we found 2 variants, one will contact C2 server directly while other will act as intermediary loader extracting dll from itself and that extracted dll will contact C2 server. Both variants are used to download and load next stage malware, which we unfortunately didn't obtain.

Entry point

Entrypoint function takes 3 parameters,

- file path of document responsible for infection - this file will be deleted!
- 32-byte key
- campaign id

usage is clearly visible in macro embedded in documents used for infection.

```
a = sqlite3_stmt_all(orgDocPath, "S-6-38-4412-76700627-315277-3247", "43")
```

While this function can have different names in all samples looks roughly the same

```
int __cdecl sqlite3_stmt_all(char *file_path, LPCSTR lpString, char *campain_id)
{
  _BYTE *v4; // esi
  int v5; // ecx
  char cmd_line[512]; // [esp+10h] [ebp-508h]
  char Buffer[512]; // [esp+210h] [ebp-308h]
  CHAR Filename[260]; // [esp+410h] [ebp-108h]

  memset(&Filename, 0, 0x104u);
  memset(&Buffer, 0, 0x200);
  memset(&cmd_line, 0, 0x200);
  if ( lstrlenA(lpString) != 32 )
    return 0;
  v4 = LocalAlloc(0x40u, 0x104u);
  strcpy(v4, (int)file_path);
  CreateThread(0, 0, remove_file, v4, 0, 0);
  GetModuleFileNameA((HMODULE)0x10000000, Filename, 0x104u);
  sprintf(
    Buffer,
    512,
    "C:\\Windows\\System32\\rundll32.exe \"%s\", sqlite3_steps %s 0 0 %s 1",
    Filename,
    lpString,
    campain_id);
  run_cmd(Buffer);
  sprintf(cmd_line, 512, "\"%s\", sqlite3_steps %s 0 0 %s 1", Filename, lpString, campain_id);
  drop_link(v5, cmd_line);
  return 1;
}
```

This function boil downs to:

- check length of second argument which later will be used as the key to decrypt next stage payload or configuration details - depending on variant.
- remove file, in new thread, pointed by path passed as first parameter, usually remote template fetch by by document send to victim
- run function from itself, using `rundll32.exe`, responsible for further actions
- persists itself on computer by dropping a `LNK` file into autostart mimicking Microsoft utilities such as `preview` or `onedrive`

Intermediary loader

In some cases we found samples that will decode and load intermediary dll before call to C2. Inner dll is encoded by two rounds of xor, first one looks like well known `visual_decrypt` from zeus with a little twist of first byte being xored with the last one.

```
// g_Data - 10096348h
// g_Key - 10096347h
g_Data[0] ^= g_Key[g_Size];
for ( i = g_Size - 1; i >= 1; --i )
    g_Data[i] ^= g_Key[i];
```

Second round of xoring is done against key passed in parameter. This double xor gives us potential to extract payload key due to amount of null bytes in PE file

```
p70<98-.\x08\x0f\x02S}7-$0-38\x01)\x1d\x03\x05\x06\x04\x03-\x072\x078\x0e4\x1a&\x0f#\x05K\x1a*\x1a\x18\x06\t\x04\x10\
\x02\x07\x07\x0e\x1a\x0f\x05\x1a\x1a\x06\x04\x06\x1a\x02\x04\x05\x02\x07\x07\x0e\x1a\x0f\x05\x1a\
\x0bv\nx\ra\x08f\x08d\x01`\x15\x00\n\x15\x15\t1\n\x10\x08\x1d\x14\x07\x0c\x00\n\n\x0b\x0c\x0cm\tr\t\x03\x17\x02\
\x0f\x08\r\r\x048-6X62_-6-X
\x07-3811-75432205-060098-6872S-6-81-3811-75432205-060098-6872S-6-81-3811-75432205-060098-6872S-6-81-3811-75432205-06009
Python>visDecry(d)[:100]
"&&-;1-3<11-43205-060p98-6872S-6-81-3811-75432205-060098-872]2#80$\x190a\x14`[[A\x10E__QBQT\x18NWVY]'\rT"
Python
```

Heart of validator

Upon reaching this state malware tries to decrypt it config, this is pretty straightforward and can be done with following script. Config is encrypted with AES and key is derived by applying md5 hash to a value provided in command line. Config contains following fields

Field	Value	Description
c2_domain	`https://www.astedams.it/include/inc-elenco-offerter.asp`	Address of C2
comm_key_1	7f39a52fc6a51b5dd3830064c63f9d490000000000000000000000000000000000	AES key used to encrypt data send to C2
comm_key_2	7b5089c2bea3ec4aa98c5cdf69dac2150000000000000000000000000000000000	AES key used to encrypt data received from C2
sample_id	9370	ID of sample, used when doing request to C2
param_name_cid	ned	Name of HTTP Param containing campaign id
param_name_size	gl	Name of HTTP Param containing size of data send to C2
param_name_zero	hl	Name of HTTP Param always containing zero

Fingerprinting machine

Malware collects following informations

- Computer name
- User name
- List of attached drives with their type and capacity and free space
- List of running processes

C2 Communication

After collecting all the information, data is packed into a binary blob described below and encoded with base64 and send to C2 using a wrapper class calls `CWininet_Protocol`. Malware tries to obtain value of default `User-Agent`, if it fails constant string `Mozilla` is used. Communication happens in loop using one socket, until any response is received. If response is received and it has a specific hardcoded length, decoding begins. Decoding is mirror process to encoding C2 request, response is encoded with both base64 and packed into binary format. After successful decoding received PE file is being loaded into memory and run finishing work of a validator.

Binary Format

All binary blobs have a following header

```
struct blob_hdr {

        DWORD random;
        DWORD is_compressed;
        DWORD size;
        DWORD size2; // size == size2
        DWORD padded_size;
        DWORD decoded_hash[4] // md5 hash of fully decoded blob, without header
        DWORD unk_1[4];
        DWORD data_hash[4] // md5 hash of encoded data, without header
        DWORD unk_2[4]
}
```

While data in configuration is only encrypted, all data exchanged between C2 server is additionally compressed, algorithm used here is lz4

PE Loading

Next stage binary is loaded into memory and run, code responsible for it is borrowed MemoryLoadLibraryEx

Attribution

Attribution to Lazarus is done based on a blog by Telsy. They mention the same scheme of infections as well the same dll however in older version[2]

Yara

```
rule apt_NK_Lazarus_DllImplat_cmd_line {

        meta:
                reference = "https://blog.malwarelab.pl/posts/lazarus_validator/"
                hash = "141931bf718c5c4d3931f64b04e2112b65a6bcd46c0092300ff6824b573f8b36"
                hash = "b76b6bbda8703fa801898f843692ec1968e4b0c90dfae9764404c1a54abf650b"
                hash = "37a3c01bb5eaf7ecbcfbfde1aab848956d782bb84445384c961edebe8d0e9969"
                hash = "bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114"
                hash = "a8647a04563b746b1d8d4cdd67616cb646a3f6766d9c2d447541b9dc26452d8b"
                hash = "bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114"
                hash = "48b8486979973656a15ca902b7bb973ee5cde9a59e2f3da53c86102d48d7dad8"
                hash = "21515fd6e6eb994defb589b4d0d9d956f7b4cb07823aaec501134ab063d883e9"
                hash = "26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa"
                author = "Maciej Kotowicz, mak@malwarelab.pl"
                copyright = "MalwareLab.pl"
                date = "2020-05-18"

        strings:
                $s0 = " %s 0 0 %s 1"
                $s1 = / %s 0 0 [0-9]+ 1/

        condition:
                any of them and filesize <2MB
}

import "pe"

rule apt_NK_Lazarus_DllImplat_cfg_decoder {

        meta:
                reference = "https://blog.malwarelab.pl/posts/lazarus_validator/"
                hash = "bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114"
                hash = "26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa"
                author = "Maciej Kotowicz, mak@malwarelab.pl"
                copyright = "MalwareLab.pl"
                date = "2020-05-18"

        strings:
                $c0 = { 6A 15 59 8B F2 8D 7C 24 ?? F3 A5 8B ?? 24 ?? 83 E? 0F 6A 10 58 2B C? F7 D? 1B ??
23 ?? 03 }
                $c1 = { B9 15 00 00 00 8B F2 8D 7C 24 ?? F3 A5 8B ?? 24 ?? [2] 83 E? 0F }

        condition:
                uint16(0) == 0x5a4d and any of them
}
```

Analysis Artifacts

Hashes

```
cf44576adcfc51a062457398797f99e85b504a70208938180639c2e0cde7ca95  074c02f7f5badd5c94d840c1e2ae9f72
805183c19f4bffca871fb344247bd5d31357480c536c052c8b7c4109f507f3ef  075fba0c098d86d9f22b8ea8c3033207
37a3c01bb5eaf7ecbcfbfde1aab848956d782bb84445384c961edebe8d0e9969  11fdc0be9d85b4ff1faf5ca33cc272ed
23d73fc8f10588944d8dc2073ce6af6d159943f11ac0c140c9b2e67fb0ad8b89  14d79cd918b4f610c1a6d43cadeeff7b
2627b7c827404ee49271bfc6bb152e52cf28e35c4bef1ad256018c5c08daea21  1bd0ca304cdecfa3bd4342b261285a72
f3e4947e32c6b1d0303b342a74426d4de0ad2de8d2bfb83f0003a1ebc650b7a7  223e954fd67c6cf75c3a6f987b94784b
26a2fa7b45a455c311fd57875d8231c853ea4399be7b9344f2136030b2edc4aa  25b37c971fd7e9e50e45691aa86e5f0a
48b8486979973656a15ca902b7bb973ee5cde9a59e2f3da53c86102d48d7dad8  2b02465b65024336a9e15d7f34c1f5d9
d79bfa19e4d32692030d15c2767beb8cd88dc552d51245c88f42edd5129326f0  3aa8eddf26f5944a24dfeb57c9f49a17
7fdfc719935d938651f45aafef3cd2ecc0020e9b77ac0780edb3ba585c16c9e2  59cb8474930ae7ea45b626443e01b66d
69e50a20ea6be94e4336ba8cea3c438e8ce7c4aab66ce1baf26bc3a77200fb27  59fab92d51c50467c1356080e6a5dead
805183c19f4bffca871fb344247bd5d31357480c536c052c8b7c4109f507f3ef  7228705813d5bc6c6a62fc53ac019344
454734dca530d54c4e8f543bdd33b5eb4b50f3039a953b54281dc67a09af4ca6  78d42cedb0c012c62ef5be620c200d43
a8647a04563b746b1d8d4cdd67616cb646a3f6766d9c2d447541b9dc26452d8b  8fc7b0764541225e5505fa93a7376df4
21515fd6e6eb994defb589b4d0d9d956f7b4cb07823aaec501134ab063d883e9  92657b98c2b4ee4e8fa1b83921003c74
a769b39d0c80d1a035dd51efa28b09221cbb6c912def95a57c8910455e333756  b493f37ee0fddb1d832ddacaaf417029
3c5c1a7e7efe4eee3b7650167c664f730f40923a38c3e6640cbb2a4bfe9f64c6  c0a8483b836efdbae190cc069129d5c3
6590f66d6afe155b1109e81e2c36ece73236223ab17ae1a1c77a027be9f7d400  cd5357d1045948ba62710ad8128ae282
ec254c40abff00b104a949f07b7b64235fc395ecb9311eb4020c1c4da0e6b5c4  d1c652b4192857cb08907f0ba1790976
b76b6bbda8703fa801898f843692ec1968e4b0c90dfae9764404c1a54abf650b  f4b55da7870e9ecd5f3f565f40490996
bff4d04caeaf8472283906765df34421d657bd631f5562c902e82a3a0177d114  f6d6f3580160cd29b285edf7d0c647ce
1b0c82e71a53300c969da61b085c8ce623202722cf3fa2d79160dac16642303f  65df11dea0c1d0f0304b376787e65ccb
d7ef8935437d61c975feb2bd826d018373df099047c33ad7305585774a272625  0071b20d27a24ae1e474145b8efc9718
66e5371c3da7dc9a80fb4c0fabfa23a30d82650c434eec86a95b6e239eccab88  1f254dd0b85edd7e11339681979e3ad6
1076b25d5fa5cccdddcaf3f788789ae3c4ea9b034066693b6a0560af129ceda6  2efbe6901fc3f479bc32aaf13ce8cf12
```

URLs

```
https://www.curiofirenze.com/newsl/include/inc-map.asp
http://www.elite4print.com/admin/order/batchPdfs.asp
http://www.astedams.it/include/inc-elenco-offerter.asp
https://www.astedams.it/uploads/frame/61.dotm
https://www.astedams.it/uploads/template/17.dotm
https://www.sanlorenzoyacht.com/newsl/uploads/docs/43.dotm
https://www.sanlorenzoyacht.com/newsl/uploads/docs/1.dotm
```

File Names

```
wsuser.db
wsdts.db
onenote.db
thumnail.db
```

Communication keys:

```
0f0b242fb5d73d08b856bc43432b350000000000000000000000000000000000
cb38822697af45210d2759889c2eb2bf0000000000000000000000000000000000
7f39a52fc6a51b5dd3830064c63f9d490000000000000000000000000000000000
7b5089c2bea3ec4aa98c5cdf69dac2150000000000000000000000000000000000
```

Paths

```
\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\OneDrive.lnk
\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\onenote.lnk
\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\preview.lnk
\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\thumbnail.lnk
```

---

1. `9724ee7a6baf421ac5a2a3c93d32e796e2a33d7d75bbfc02239fc9f4e3a41e0` ↩
2. Different size of config and hardcoded campaign id ↩