DUDELOCKER

MALWARE ANALYSIS AND INCIDENT FORENSICS
M.Sc. in Cyber Security

MALWARE ANALYSIS
M.Sc. in Engineering in Computer Science

A.Y. 2022/2023

SAPIENZA
UNIVERSITÀ DI ROMA

CIS SAPIENZA
CYBER INTELLIGENCE AND INFORMATION SECURITY

FLARE-ON 2016 CHALLENGE #2

- Sample: DudeLocker.exe from Flare-On Challenge #2 from 2016
- Many write-ups online. What, and spoil all the fun? Check them afterwards 😄
- Exits prematurely when executed with basic dynamic analysis tools
  - You need to tweak the environment if you want to use them…
- Reference: http://flare-on.com/files/Flare-On3_Challenges.zip (“flare”)

GETTING WARMED UP

- Extract DudeLocker.exe and BusinessPapers.doc from the archive (keep it, you may need it later…), then load the PE in a debugger
- Let's see about the prologue of subroutine 0x4019A0
  - What is the sample building with the first bunch of mov instructions?
    Hint: with IDA, double click on the local variables to see their contents in the main IDA view and use ESC to go back (with x32dbg: check the Memory Dumps view)
  - What is the output of function SHGetFolderPathW? (look up MSDN, or run it…)
  - Do you need to step into the call to lstrlenW at 0x4109F4?
  - What are the two error conditions that bring EIP to block 0x401A01?

SUB_4019A0 TO SUB_401040

- What is the purpose of the call to CreateFileW in block 0x401A08?
- Can you adjust the environment? Or patch the code for the call?
- Let us analyze the subsequent call to subroutine 0x401040
  - What is the purpose of the call to GetVolumeInformationA?
  - Where is the output written?
  - What value can EAX assume when leaving the function?
  - Immediate operand dword 74AB1D35h
    - Can you patch some instructions to reach a meaningful state later?
      (Hint: look at how EAX is used in the caller function)
    - Alternatively, can you adjust the environment?
      (Hint: VOL command-line utility)

BACK TO SUB_4019A0

- Let us continue from block 0x401A94
  - How large is the heap chunk being allocated?
  - What are the arguments for the internal call to subroutine 0x401940?
- Let us step into subroutine 0x401940
  - How many bytes are processed at a time?
  - What is being written to the heap chunk?
    (Hint: check memory for address dereferenced at 0x401983 by mov [eax], cl)
  - What encoding has been used this time?
  - Now we can return to the caller function at address 0x401AC7

ENTERING SUB_401080

- We can dive into subroutine 0x401080
  - What are the arguments passed to the function?
  - What API functions may be called by this subroutine?
  - To understand the meaning of the parameters for the first call: https://docs.microsoft.com/it-it/windows/desktop/api/wincrypt/nf-wincrypt-cryptacquirecontexta
  - To figure out the meaning of 18h, do your hex->dec math and check this: https://referencesource.microsoft.com/#mscorlib/system/security/cryptography/utils.cs,47878f4e20fb166f
  - So what provider type is being used for the CSP?
- Reach block 0x4010CE
  - What arguments are being passed to subroutine 0x401180? Step into it

A GLANCE AT SUB_401180

- What API functions are referenced in this subroutine?
- What algorithm is being used for the function called at 0x4011AC?
- What is the second argument being passed at 0x4011D4?
- Now reaching a milestone in our reversing session: CryptDeriveKey
  - Look up the MSDN documentation for the meaning of its arguments
  - What is being used as AlgId?
  - What is being used as hBaseData?
- And we are ready to return to subroutine 0x401080

LEAVING SUB_401080

- When returning from subroutine 0x401180, one more API call is on the way: CryptSetKeyParam
  - What may DudeLocker attempt to do here? *(You don't need to figure all details)*
- There is nothing much left here. Let us return to subroutine 0x4019A0
  - Shall we take the branch to block 0x40A1F0?
  - How many subroutines we can expect to execute next?
  - Why do we better focus on subroutine 0x401300 for now?

SUB_401300

- This seems a good candidate as a relevant subroutine to orchestrate the malicious actions. Let's see where its control flow brings us…
  - What kind of enumeration is performed? And what are the strings being concatenated?
  - Any idea what REP MOVSD stands for?
  - What about subroutines being called?
    - 0x401990 just applies an AND mask between EAX and 10h (16)
    - We saw 0x401000 in the initial stages
    - What about 0x401730? *A decompiler may have been handy here…* Interesting?
    - What about 0x401770? Well, this one looks interesting.

SUB_401770

- What might be the arguments for the call? Look at API calls inside
- What is the algorithm used for CryptCreateHash?
- Has the pbData string for CryptHashData changed?
- What about pbData for CryptSetKeyParam? *(Hint: check IDA's offsets)*
- Note: you don't need to know all the details of this function, unless you want to decrypt your files. But you can try that from another angle…

BACK TO SUB_401300 & MORE

- We have analyzed an important part of the ransomware: the crypto one
- There is one interesting subroutine left that gets called: 0x401500
  - What are the API functions called from there?
  - What may their presence in a loop indicate?
- Subroutine 0x401300 has been analyzed to a large extent. We can now return to subroutine 0x4019A0 and analyze what is left. Go to block 0x401AF0 right after we return from the call.
  - What happens when the `jz` jump is not taken?
  - What is the effect of executing subroutine 0x401220? Check your files 🙂

BONUS POINTS

Try to reverse the effects of the ransomware!

- BusinessPapers.doc has been encrypted by DudeLocker
- Possible approaches
  - Write a C/C++ decrypter program that calls the same crypto functions as DudeLocker to perform decryption instead of encryption
  - Do that in Python using the wincrypto library (you can use hashlib to compute the hash used in the initial stage)
  - Think outside the box 🗣 alter the code to make the ransomware carry out decryption *(hint: look at the IAT and tools like LordPE)*
- By the way, don't trust file extensions too much…

REFERENCES

Useful write-ups and other resources on DudeLocker:

- https://www.fireeye.com/content/dam/fireeye-www/global/en/blog/threat-research/flareon2016/challenge2-solution.pdf
- https://github.com/quanyang/reversing-workshop/blob/master/2/challenge2.md
- https://blog.superponible.com/2016/11/05/2016-flare-on-challenge-2/
- https://www.endgame.com/blog/technical-blog/dude-wheres-my-ransomware-flare-challenge