Module 3

Understanding and countering malware's evasion and self-defence

https://github.com/hasherezade/malware_training_vol1

Introduction
Malware: Evasion and self-defense

Malware: Evasion and self-defense

- In order to carry on its mission, malware must remain undetected
- Malware needs to defend itself from:
  - Antimalware products (on the victim machine)
  - Analysis tools and sandboxes (on a researcher's machine)

Malware: Evasion and self-defense

- Approaches:
  - Passive:
    - obfuscation (at the level of: code, control flow, strings, used APIs)
  - Active:
    - environment fingerprinting, detection of the analysis tools and:
      - interference in them (i.e. uninstalling AV products, unhooking hooks)
      - altering own behavior (deploying a decoy, or terminating execution)

The passive approach: obfuscation

- Related with the way code is designed: i.e. using exception handlers to switch between various code blocks, using dynamically loaded functions, string obfuscation, polymorphic code, etc
- Added at the compilation level: i.e. adding junk instructions, complicating control flow (example: movfuscator)
- Added at linking level: atypical PE header, atypical sections alignment
- Post-compilation: using protectors
- Depending on the degree with the obfuscation, may be difficult to defeat

Deobfuscation

- Approaches:
  - Dynamic:
    - Code intrumentation, tracing: allows to quickly find out what the code does, without reconstructing all details of the implementation – quick and generic, but we may miss the parts that haven't been executed during the test runs
  - Static:
    - analysis of the code and cleaning/resolving the obfuscated parts, reconstruction of the control flow – may be more accurate, but laborious, and requires different approach depending on a particular case

The active approach: fingerprinting

- Mostly related with the way code is designed: additional functions doing enviromnent fingerpriting to find artefacts indicating analysis
- Post-compilation: using protectors with added antidebug/anti-VM layer, underground crypters specialized in AV/sandbox evasion
- Most of the used methods are well-known, and the fact of using them can be relatively easily detected

Anti-evasion

- Approaches:
  - Sample-oriented:
    - Patching: finding the checks and removing them
  - Enviromnent oriented:
    - VM hardening: changing default settings, strings, that are commonly checked to identify VM
    - Using plugins for debuggers, specialized in hiding its presence (i.e. by overwriting values in PEB), changing default windows names
    - Using tools that are less often targeted by the checks: i.e. Intel Pin