Unpacking, a Hybrid Approach

Mario Alberto López (mariol@f-prot.com)
Frisk Software International, Reykjavík, Iceland.

2nd International CARO Workshop
1st & 2nd May 2008, Hoofddorp, The Netherlands.

Introduction

Emulation based unpacking? Too slow.

Static unpacking? Too specific.

Why not both?

Here we present an overview of the current F-PROT AV engine's Win32 PE file format packers/protectors handling capabilities. It uses both approaches to deal with the problem.
In combination with a flexible database-driven AV engine architecture, it has become a very important element in F-PROT's technology to deal with the always increasing malware production volume.
Let's take a look.

Why?

- Need to implement a reasonable solution as soon as possible.
- Very limited time and human resources.
- Limitations given by F-PROT's engine implementation requirements (platform independence).
- Oportunity to re-use already deployed and widely tested F-PROT engine code.

Architecture

- Emulator:
  - IA-32 instructions
  - flat memory
  - Win32 APIs
  - Win32 system (PE loader, TIB, TEB, PEB, PML, DLLs images, SEH, timing, threads, IDT)

Architecture (cont.)

- Static unpackers, "mini-emulators"(MEs):
  - 3 NRV unpackers (UPX)
  - ApLib 0.26b? unpacker (old ASPack, JDPack, NSPack, Pex, TeLock, PE-Pack)
  - ApLib 0.34? unpacker (FSG, AHPack, DBPE, Joiner, MEW, Obsidium, Packman)
  - several PECompact unpackers
  - UPX defilters
  - some API function names hashing algorithms
  - some CRCing algorithms
  - more...

Architecture (cont.)

- Database "Known code chunks"(KCCs) detection entries:
  - one detection entry for each KCC
  - any particular ME can be triggered by many different KCCs detection entries (different implementations of exactly the same algorithm)
- Database "Known packers/protectors"(KPPs) detection entries:
  - one detection entry for each specific KPP implementation
  - can configure the engine behaviour on the fly

Components interaction

```
// an oversimplified version in "pseudo C code" of the current F-PROT engine's Win32 main emulation loop

emulate_win32_pe_file()
{
    do  // ask the database if anything at this emulated EIP is something it knows:
    {   // malware, packers, "known code chunks"
        found_what = check_current_EIP_for_known_stuff();

        if ( found_what == SOME_KNOWN_MALWARE )
        {
            do_report_malware();
            break;
        }
        else if ( found_what == SOME_KNOWN_CODE_CHUNK_TO_MINI_EMULATE )
        { // the database told us which mini-emulator to use, and how
            do_mini_emulate_known_code_chunk(); // ultra fast! :)
        }
        else
        {   // the database told us it is a known packer, and what to do next (increase emulation limit?)
            if ( found_what == SOME_KNOWN_PACKER )
            {
                do_report_known_packer();
            }
            do_emulate_single_ia32_instruction(); // very slow :(
        }
    }
    while ( should_keep_emulating );
}
```

Example, some UPX sample

; An arbitrarily chosen UPX-compressed Win32 PE file
; entry point partial disassembly
; split in functional code blocks

- **; Entry point, Decompressor initialization, 6 instructions (generically emulated)**

```
0040DA30 pushad
0040DA31 mov esi,0040B015h ; esi = compr.stream VA
0040DA36 lea edi,[esi-0000A015] ; edi = decompr.stream
VA
```

(... some code removed ...)

- **; Decompressor, 82 instructions ("mini-emulated")**

```
0040DA48 mov al,[esi]
0040DA4A inc esi
```

(... some code removed ...)

Example, some UPX sample (cont.)

- **; Defilter initialization, 3 instructions (generically emulated)**

```
0040DB02 pop esi ; esi = decompressed stream VA
0040DB03 mov edi,esi
0040DB05 mov ecx,016Bh
```

- **; Defilter, 19 instructions ("mini-emulated")**

```
0040DB0A mov al,[edi]
0040DB0C inc edi
0040DB0D sub al,E8
```

(... some code removed ...)

Example, some UPX sample (cont.)

- **; original imported APIs entry points resolver, 27 instructions (generically emulated)**
  `0040DB36 lea edi,[esi+0000B000]`
  `0040DB3C mov eax,[edi]`
  (... some code removed ...)
- **; jump to OEP (generically emulated)**
  `0040DB7E popad`
  `0040DB7F jmp 00401000h`

Need for speed, optimizing the generic emulator

- Caching memory accesses
- independent CPU flags emulation
- avoid CPU flags setting when possible

Conclusion

- Disadvantages
  - The whole project:
    - Not as universal as Dynamic Translation.
  - MEs:
    - Need to code new MEs and add new KCC detection entries when needed.
    - Do not handle heavily polimorphic code (handled by generic emulation instead, slow).

Conclusion (cont.)

- The whole project advantages:
  - Very good efforts/results relation in the real world.
  - Universal (always, by generic emulation) and fast (MEs, not always, depending on the code being emulated) at the same time.
  - Easily expandable, can be implemented and deployed incrementally (many small specific projects instead of one huge universal project).
  - A lot of already deployed F-PROT engine code was re-used (detection, emulator).
  - Interactive (database-driven) emulation ("smart"database entries affect the emulator behaviour).

Conclusion (Advantages cont.)

- Emulator advantages:
  - Many improvements indirectly benefitted other engine modules (exact malware detection engines, heuristic engines, advanced anti-emulation parasitic virus handling).

Conclusion (Advantages cont.)

- MEs advantages:
  - Many newly implemented MEs were re-used (different KCCs detection entries trigger the same ME)
  - Relatively simple implementation (each one handles one single algorithm).
  - Transparent, simple and optimized interface to the generic emulator.
  - Much faster than generic emulation.
  - Faster than Dynamic Translation? (common sense, not proven, no need for code analysis, code generation, code optimization)

Conclusion (Advantages cont.)

- Detection entries advantages:
  - Flexible (database based).
  - In practice, the scanning performance of the engine was not and is not affected by the addition of the new features.
  - KCCs detection entries advantages:
    - Many KCC detection entries were unintentionally re-used (same KCC occurrence in different packers/protectors).
  - KPPs detection entries advantages:
    - Possibility of changing the engine behaviour at will.

Contributors

Thanks to the people that helped this technology to become a reality, by contributing with requirements, ideas, algorithms, coding practices, reasearch, code, samples, testing, profiling, bug reports, code fixes, lots of good work and more. In alphabetical order:

Ana (Анна Подольская -Anna Podolskaia-),
Bjartmar (Bjartmar Kristjánsson),
Ceco (Цветан Шалявски -Tzvetan Chaliavski-)
Einar (Einar Jón Gunnarsson),
Erlendur (Erlendur S. Þorsteinsson),
Ernir (Ernir Erlingsson),
Fjalar (Sigurður Fjalar Sigurðarson),
Frisk (Friðrik Skúlason),
Gretar (Gretar Þór Gretarsson),
Halldor (Halldór Ólafsson),
Henry (Henrý Þór Baldursson),
Krissi (Kristmundur Jón Hjaltason),
Marcin (Antoni Marcin Nawrocki),
Martein (Marteinn Þór Harðarson),
Oliver (Oliver Schneider),
Petar (Петър Костадинов Шомов -Petar Kostadinov Shomov-),
Raggi (Ragnar Gísli Ólafsson),
Robert (Robert Sandilands),
Siggi (Sigurður Arnar Stefnisson),
Sindri (Sindri Bjarnason),
Svavar (Svavar Ingi Hermannsson),
Thro (Þröstur Snær Eiðsson),
Vess (Веселин Владимиров Бончев -Vesselin Vladimirov Bontchev-).