



# T-110.6220: Antivirus Engine Basics

Antti Tikkanen, F-Secure Corporation

# Detecting Malware

---

- Blacklisting
  - Detecting badness
  - Typically fairly reactive but heuristic and behavior blocking approaches are also widely used
  - Not perfect but good enough
- Whitelisting
  - Allow only known good and block rest
  - Solid theory but practical implementation is challenging
  - Usually current technology combines both approaches
- This presentation focuses on blacklisting technologies

# File Infectors vs. Standalone Malware

---

- See “Introduction to Malware” lecture by Gergely Erdelyi
  - Virus: Self-replicating (most often refers to parasitic infectors)
- A lot of the literature on the subject focuses on detecting viruses (i.e. file infectors)
- Nowadays most malware is Trojans, Backdoors, and other malware that does not replicate
  - New file infector viruses do still appear frequently, though
- While detection methods are pretty much the same, this lecture tries to cover malware detection in general, not just detecting file infectors

# Detection Outcomes

---

		Object is malicious?	
		Yes	No
Malware detected?	Yes	✓	<b>False Positive</b>
	No	<b>False Negative</b>	✓

From [Aycok06]

- “Ghost Positive”: A misdisinfected file that is actually no longer infected with the virus

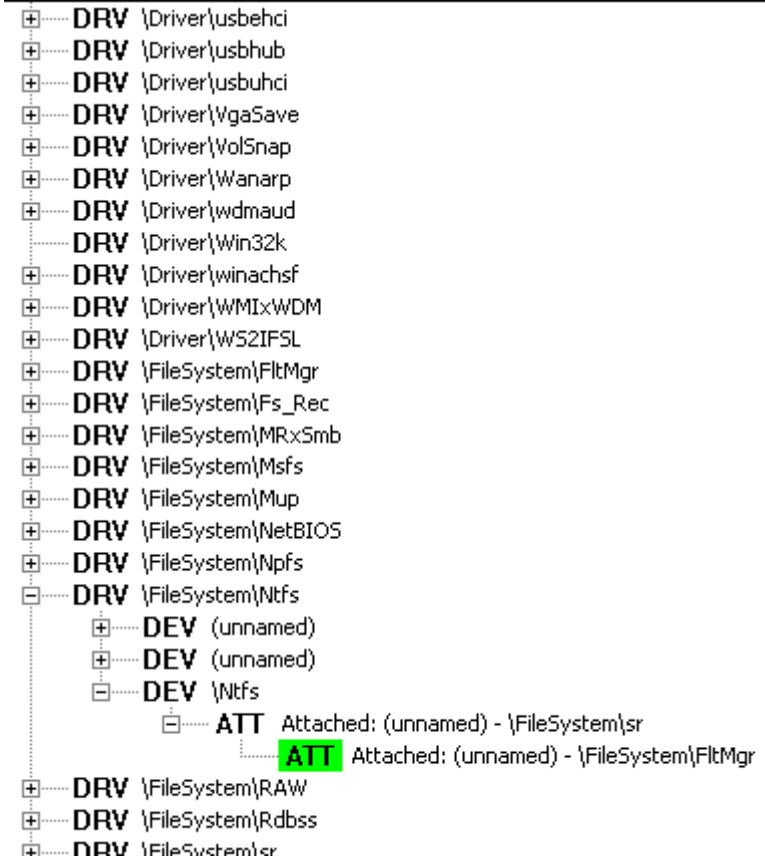
# Detection Strategies

---

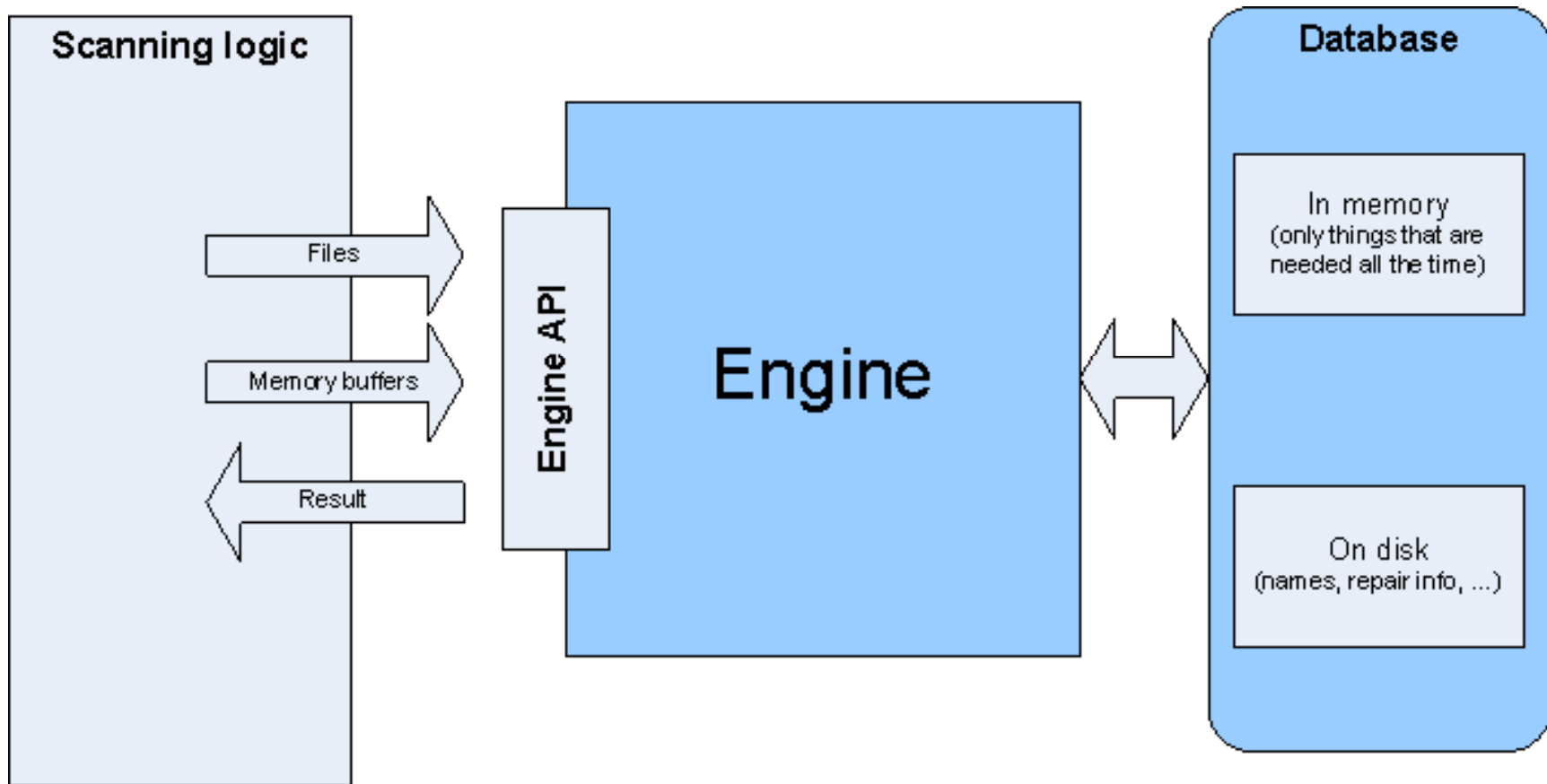
- Static
  - Analysis of file structure and contents
- Dynamic
  - Target of analysis is executing
    - Behavior monitoring on real system
    - Emulation

# Scanning Methods

- Scan types
  - On-demand (ODS)
  - On-access (OAS)
- ODS enumerates files and scans them one by one
- OAS scans files as they are being accessed
  - Typically implemented using Windows filter drivers



# An Antivirus Engine



# Design Principles

---

- Scanning of clean files needs to be as fast as possible
  - “Front end fast” [Kuo99] since most things are not viruses
- Scanning or disinfecting malware can take a while longer
- Memory vs. scanning speed tradeoff
  - Not always possible
- Platform issues
  - Engines for many uses: Servers, gateways, desktop, mobile phones, ...
  - Supported languages (e.g. C++ is not always that portable)
  - Available memory and CPU
  - Endianness



# Design Principles Continued

---

- Preprocess as much as you can when building the database
  - Long engine initialization will result in slow boot up time
- Read as little as you can
  - With larger files, file IO becomes very costly if your approach always reads the full file

# Subcomponents of an Antivirus Engine

---

I/O abstraction layer

Fingerprint scanning (string scan, hashing, ...)

File type (doc, exe, html, vbs, ...) recognizer and parsers

Archive uncompressor (zip, arj, mime, ...)

Unpacker for runtime packers (UPX, ASPack, ...)

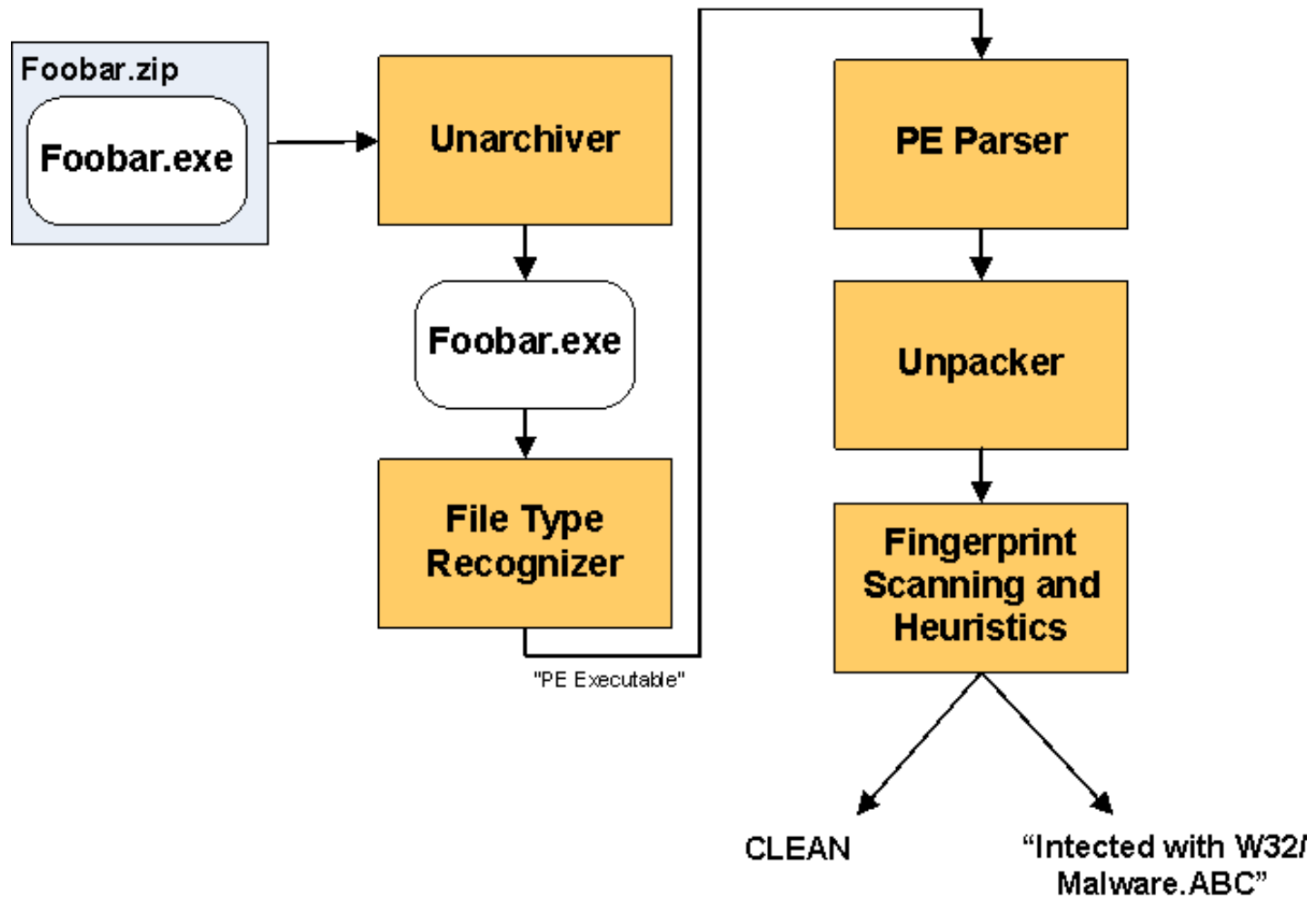
Emulator

File disinfection and system clean-up

Database

Database update logic

# Example Flow



# Basic Scanning Techniques

---

- String scanning
  - Byte strings, not necessarily text strings
- Hash scanning
  - Using checksums as signatures
- Virus-specific detection algorithms
  - Custom detection code
- Heuristics
  - Several types
- Emulation
  - Used in conjunction with the other techniques

# String Scanning

---

- Search for pattern  $P$  (length  $n$ ) in in text  $T$  (length  $m$ )
- Naïve string search
  - Poor performance, not usable in practice
- Boyer-Moore
  - Not optimal for AV engines, where multiple patterns are searched in each file
- Aho-Corasick
  - Finds a set of patterns in a given text
- Regular expressions
  - Typically requires a state machine, performance issues
- Wildcarding
  - “04 A3 56 ?? ?? ?? ?? 67 AA F0”
  - See example of implementation of Aho-Corasick with wildcards in [Kumar92]

# Issues in String Scanning

---

- Search Range
  - Full file search (slow)
  - Limited search (start, length): How to select starting point?
- Number of Signatures
  - One signature can lead to false positives
  - It can be a good idea to use a second signature to verify infection [Fernandez01]
- False positives
  - There have been instances where antivirus application detects the database of another antivirus app [Fernandez01]

# Hash Scanning

---

- Instead of looking for a byte string, look for a hash of a byte string
- Matching short strings can be prone to false positives
  - Hash Scanning: “Nearly Exact identification” [Ször05]
- Reduced memory consumption
  - A proper search string can be much longer than 20 bytes
  - A CRC16 checksum is 2 bytes and a MD5 hash is 16 bytes
- Selection of hash algorithm is important
  - Short hash: More prone to false positives
  - Long hash: Consumes more memory
  - Collision resistance: A hash function (that produces a fixed size output) will always have collisions
- Speed: All algorithms are very fast compared to disk IO

# Hash Scanning: Starting Position

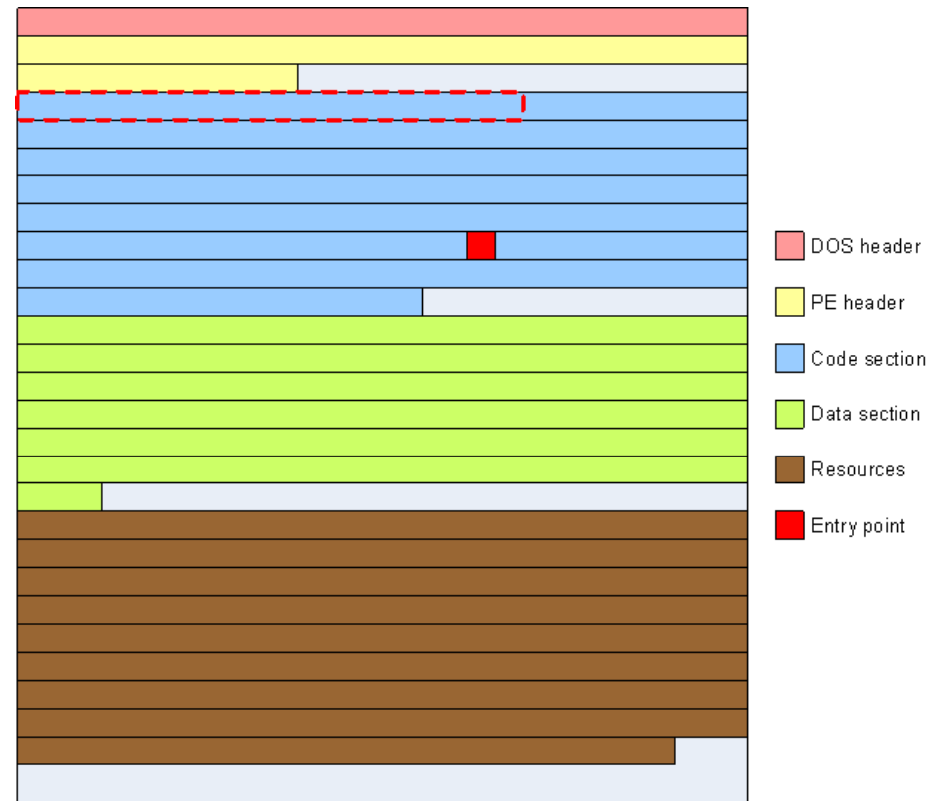
---

- In the most basic case we can calculate a hash of the full file
  - Does not work for file infectors, of course
  - Slow
  - Even a single byte of junk appended to the file will break detection
- A better solution is to use (start, length) pair for calculating hash
- Length needs to be chosen carefully:
  - Short: Risk of false positives
  - Long: A lot of disk access – slow, non-generic detection



# Hash Scanning: Start Position

- Start can be arbitrary or a fixed point, such as:
  - Beginning of the file
  - PE Entry Point
  - Beginning of the code section
  - Beginning of an exported function
  - Auto\* (AutoNew(), AutoOpen(), ...) macros in Word documents [Szappanos01]
  - ...
- The more unique (start, length) pairs, the slower the scan



# Other Speed Optimizations

---

- Top and Tail scanning
  - File infectors typically add themselves to the beginning or end of executable
- Scan only certain file types
- Cache scanning results
  - File size and modification date (easy to fool)
  - Store file checksum (in mem, on disk, or tagged onto files) – is calculating this checksum much faster than scanning?
  - What to do when antivirus database is updated?
- Code optimizations and algorithms
  - “Do not optimize, choose a better algorithm”

# Virus-specific Detections

---

- “Algorithmic Methods” [Ször05]
- Writing custom code for a single malware family or variant
- Cons:
  - Detecting a single family or variant can take thousands of lines of code
  - Writing such code takes time, quality assurance takes longer
  - Can be slow (requires proper **filtering**)
- Pros:
  - Virtually no limitation of what can be done

# Virus-specific Detections: An Example

---

```
pe = parse_pe(input_file)
section_names = pe->parse_section_names()
for name in section_names do
    if name == "ATTACH"
        return "W32/Mebrook.A"
    end
end
return "CLEAN"
```

# Disadvantages of Fingerprint Scanning

---

- Can only detect malware that matches a known signature
  - Basically only variants (or families) that have been analyzed before
- Growth in number of malware is exponential
  - Database size grows too rapidly
- A need for more generic and proactive methods

# Heuristic Scanning

---

- Malware can (at least theoretically) be detected based on its properties
- Very often malware files look different from normal files
  - E.g. strange values in fields of PE header
  - Some harmless files have weird characteristics as well
- Malware always has one or more of the following properties [Shipp01]:
  1. It replicates
  2. It has a payload that does something bad
  3. It tries to hide from anti-malware programs (e.g. obfuscation, rootkit cloaking, ...)

# Heuristic Methods

---

- Methods that are based on
  - Static analysis of the file properties
  - Dynamic analysis of the behaviour when executed
- Simple techniques
  - Rule-based heuristics
  - Weight-based heuristics
- Advanced methods include neural networks, expert systems, and data mining techniques

# Weight vs. rule-based heuristics

---

- Weights (threshold 70 pts)

Stopper Boosters	• “Connects to a web server”	20 pts
	• “Uses undocumented API calls”	10 pts
	• “Kills antivirus processes”	60 pts
	• “Contains decryption loop”	20 pts
	• “Contains string ‘virus’”	10 pts
	• “Shows a pop-up dialog to user”	-10 pts

- Rules:

- “Connects to a web server” && “Contains decryption loop”
- “Kills antivirus processes” && “Contains string ‘virus’”



# Example: Rules for Detecting EPO Viruses

Rule 1	Rule 2	Rule 3	Rule 4
SizeOfImage is incorrect	Entry point is in a writable code section	SizeOfCode is incorrect	SizeOfImage is incorrect
Entrypoint contains a jump	SizeOfCode is incorrect	Entry point is in a writable code section	Entrypoint is in the last section
Uses suspicious API calls (FindFirstFile, SetFilePointer, ...)	TimeStamp looks like it has an infection marker or Header Checksum is incorrect	TimeStamp or DOS header looks like it has an infection marker	DOS header looks like it has an infection marker
			SizeOfCode is incorrect

# Detecting Encryption & Packers

---

- Fingerprint-based detection
  - See PEiD for an example
- Shannon's entropy
  - Packed code has higher entropy (info content) than normal code
- Analysis of a disassembly or code (script malware). The following sequence [Schmall03] could belong to an encryption loop:
  1. Pointer initialized with a valid memory address
  2. A counter is initialized
  3. A memory read from the pointer
  4. Operation on the result of 3.
  5. Memory write of the result of 4.
  6. Manipulation of the counter
  7. Branch instruction depending on the counter

# Emulation

---

- Different classes of emulation
  - CPU emulation to fight polymorphism
  - OS emulation to do behavioral analysis (“sandboxing”)
- Design problems:
  - When to stop emulating?
  - How perfectly do we want to emulate the environment (OS)
  - Speed vs. safety: A truly isolated emulator is slow
- Dynamic code translation
- Hypervisors
- Emulation was covered in Jarkko Turkulainen’s lecture

# When to Stop Emulation?

---

- A program scanned can be
  - Clean
  - Infected with a virus
  - A malicious program
- Emulated memory can be scanned for signatures periodically
- Sometimes it might take even millions of clock cycles until the malicious nature of the program is revealed
  - This could take dozens of seconds depending on the emulator design and implementation

# Sandboxing

---

- Heuristics detects programs that look like malware
- Sandboxing (emulation) detects programs that act like malware
- *“My sandbox is a virtual world where everything is simulated. It is powered by an emulator, and together they let possible virus-infected binary executables ‘run’ as they would on a real system. When execution stops, the sandbox is analysed for changes.” [Natvik01]*



Image copyright F-Secure corporation

# Further Reading & References

---

- **[Aycock06]** Aycock, John: “Computer Viruses and Malware (Advances in Information Security)”, ISBN 978-0387302362, Springer, 2006
- **[Fernandez01]** Fernandez, Francisco: “Heuristic Engines”, In proceedings of VirusBulletin Conference 2001.
- **[Kumar92]** Kumar, Sandeep & Spafford, Eugene: “A Generic Virus Scanner in C++”  
<http://ftp.cerias.purdue.edu/pub/papers/sandeep-kumar/kumar-spaf-scanner.pdf>
- **[Kuo99]** Kuo, Jimmy: “Deep Inside an AntiVirus Engine”, 16 Mar 1999. <http://crypto.stanford.edu/seclab/sem-98-99/kuo.html>
- **[Natvik01]** Natvik, Kurt: “Sandbox Technology Inside AV Scanners”. In proceedings of VirusBulletin Conference 2001.
- **[Schmall02]** Schmall, Markus: “Building an Anti-Virus Engine“, March 4, 2002. <http://www.securityfocus.com/infocus/1552>
- **[Schmall03]** Schmall, Markus: “Classification and identification of malicious code based on heuristic techniques utilizing Meta languages”, 2003.
- **[Shipp01]** Shipp, Alex: “Heuristic Detection of Viruses Within Email”, In proceedings of VirusBulletin Conference 2001.
- **[Suominen07]** Suominen, Mikko: “Win32-tiedostovirukset ja niiden havaitseminen“, University of Turku, 2007.
- **[Szappanos01]** Szappanos, Gabor: “VBA Emulator Engine Design”, In proceedings of VirusBulletin Conference 2001.
- **[Ször05]** Ször, Peter: “The Art of Computer Virus Research and Defense”, ISBN 978-0321304544, Addison-Wesley Professional, 2005

Protecting  
the  
irreplaceable

