Embracing the new threat: towards automatically, self-diversifying malware

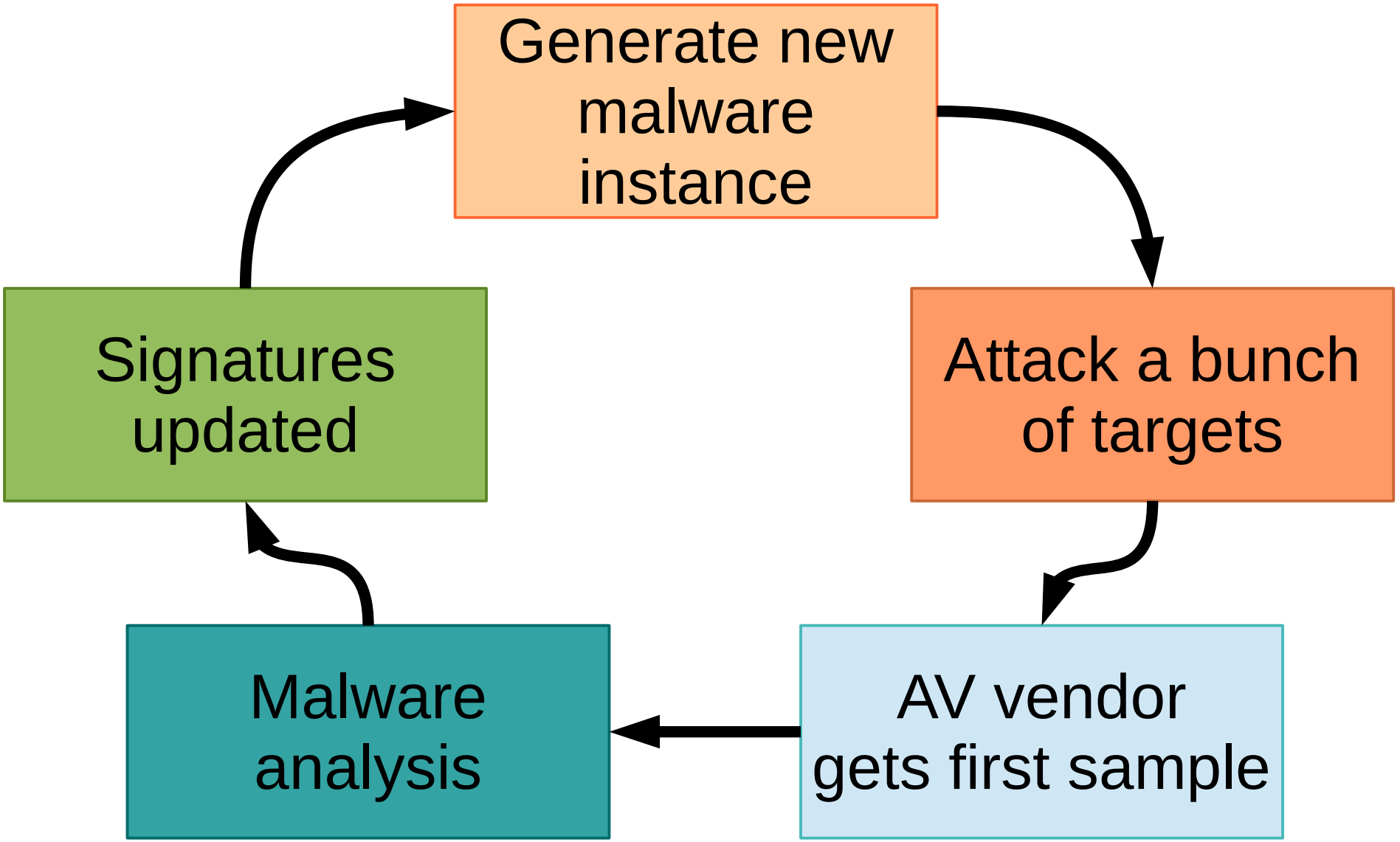
Mathias Payer <mathias.payer@nebelwelt.net>

Image (c) http://ucrtoday.ucr.edu/9768/assassin-bugs

Malware landscape is changing

Image (c) Wikimedia

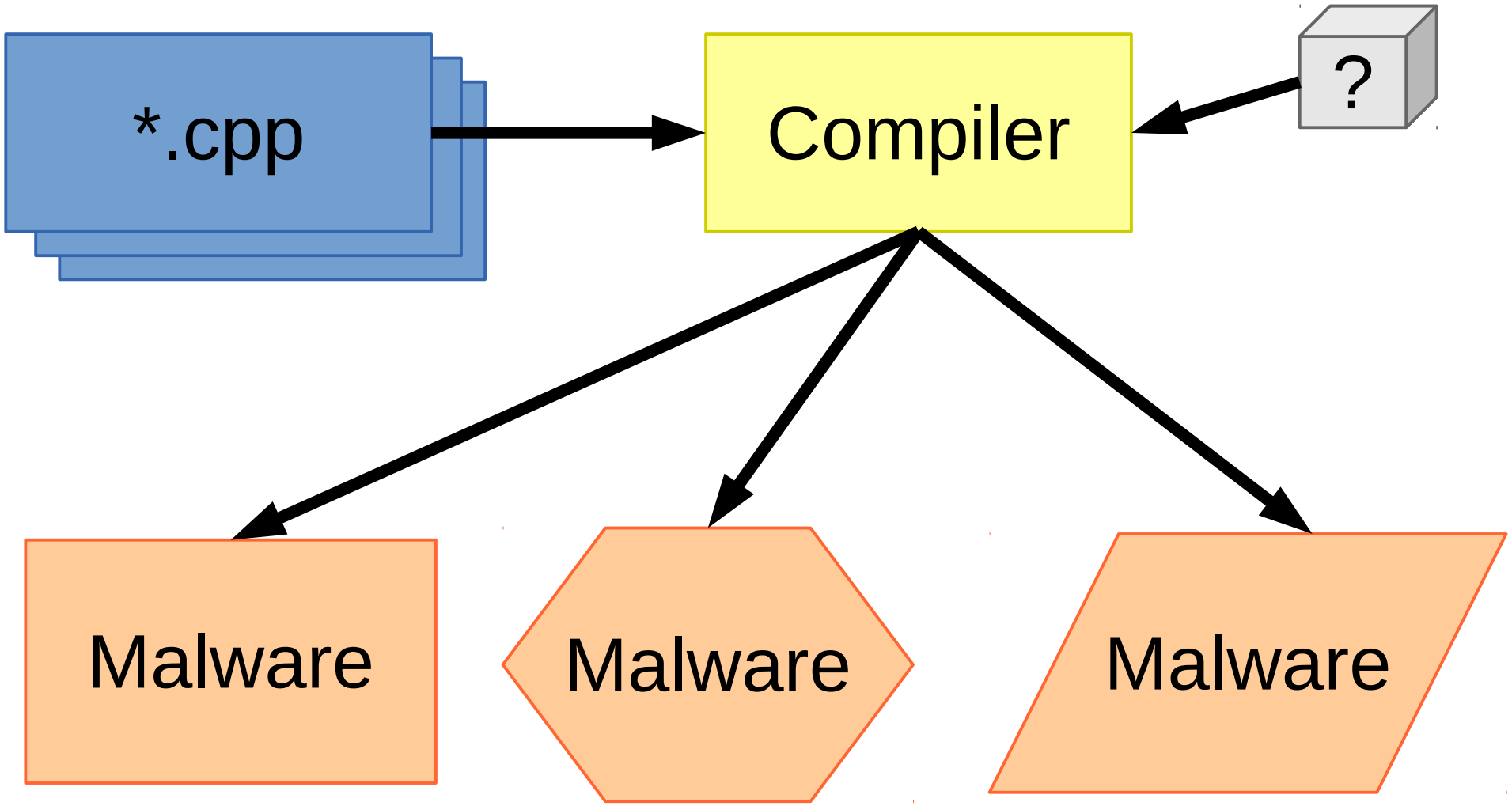
The ongoing malware arms race



Defense limitations

- Newly diversified samples are not detected
  - Basically a “new” attack
- New malware spreads fast
  - Time lag for analysis to update signatures
- What if we can make each executable unique?

Fully automatic diversity



Outline

State of the art:
Malware detection

A new threat:
Malware diversification

Possible mitigation:
Better security practices

State of the art: Malware detection

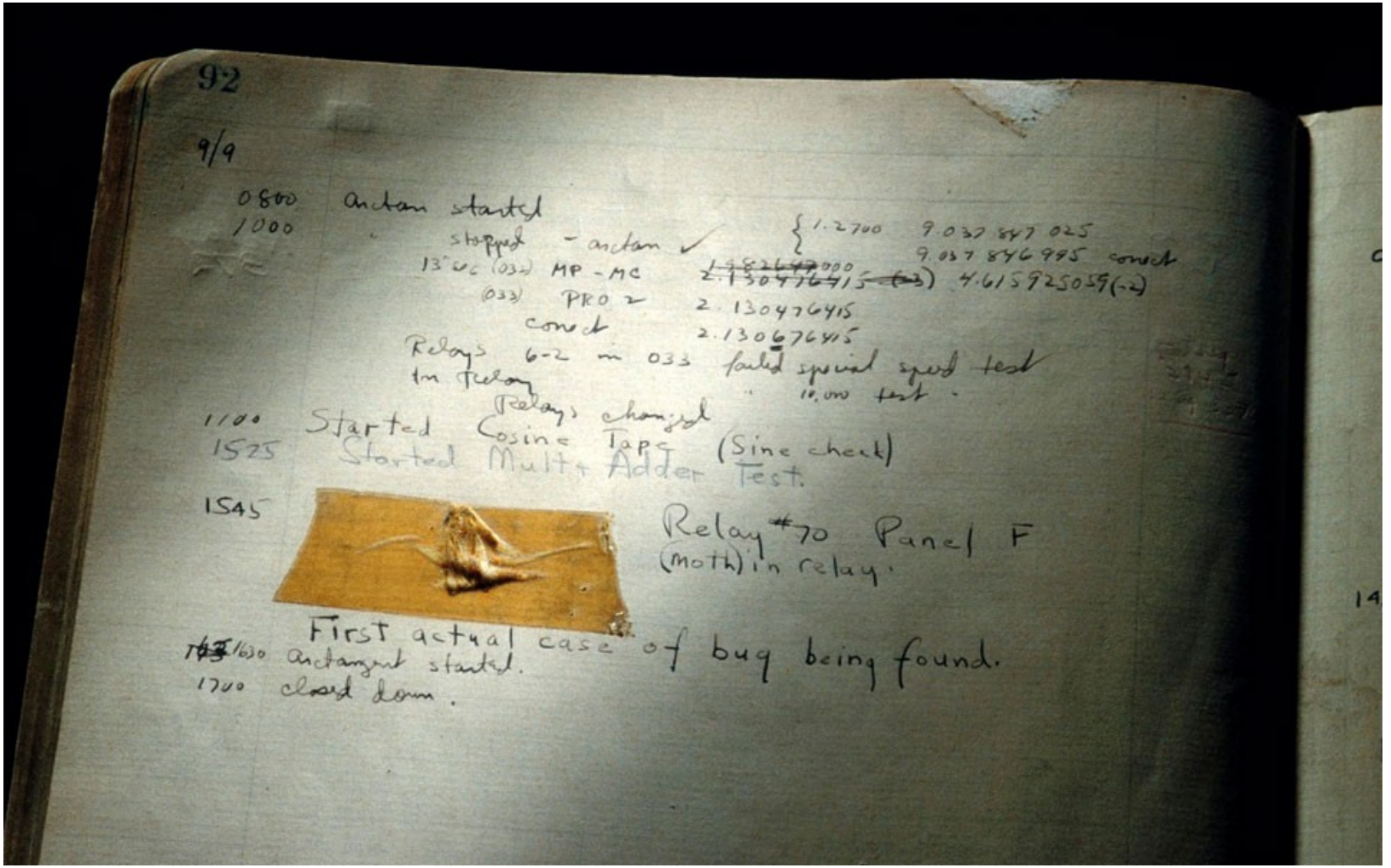


Image (c) Wikimedia

Malware detection is limited

- Performance
  - Don't slow down a user's machine (too much)
- Precision
  - Behavioral, generic matching
- Latency
  - Time lag between spread and protection

Detection mechanisms

Image (c) Wikimedia

Signature-based detection

- Compare against database of known-bad
  - Extract pattern
  - Match sequence of bytes or regular expression
- Advantages
  - Fast
  - Low false positive rate
- Disadvantages
  - Precision limited to known-bad samples

Static analysis-based detection

- Search potentially bad patterns
  - API calls
  - System calls
- Advantages
  - Low overhead
- Disadvantages
  - False positives
  - Based on well-known heuristics

Behavioral-based detection

- Execute “file” in a virtual machine
  - Detect modifications
- Advantages
  - Most precise
- Disadvantages
  - High overhead
  - Precision limited due to emulation detection

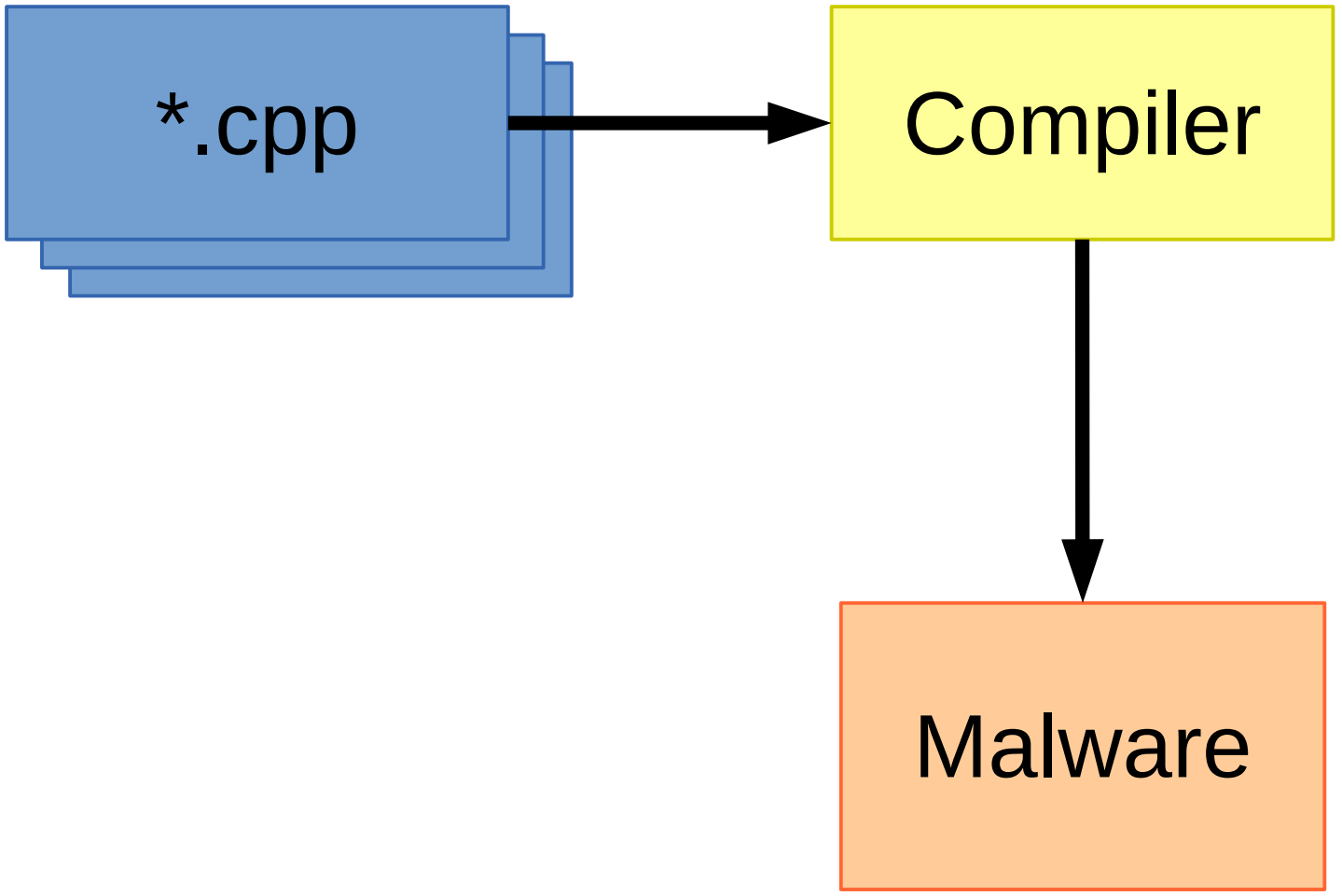
Summary: Malware protection

- Arms race
- Only partial protection
- Only limited resources available

New threat: Malware diversification

Image (c) Wikimedia

Software diversification



C/C++ liberties

- Data layout changes
  - Data structure layout on stack
  - Layout for heap objects (limited for structs)
- Code changes
  - Register allocation (shuffle or starve)
  - Instruction selection
  - Basic block splitting, merging, shuffling

Malware diversification

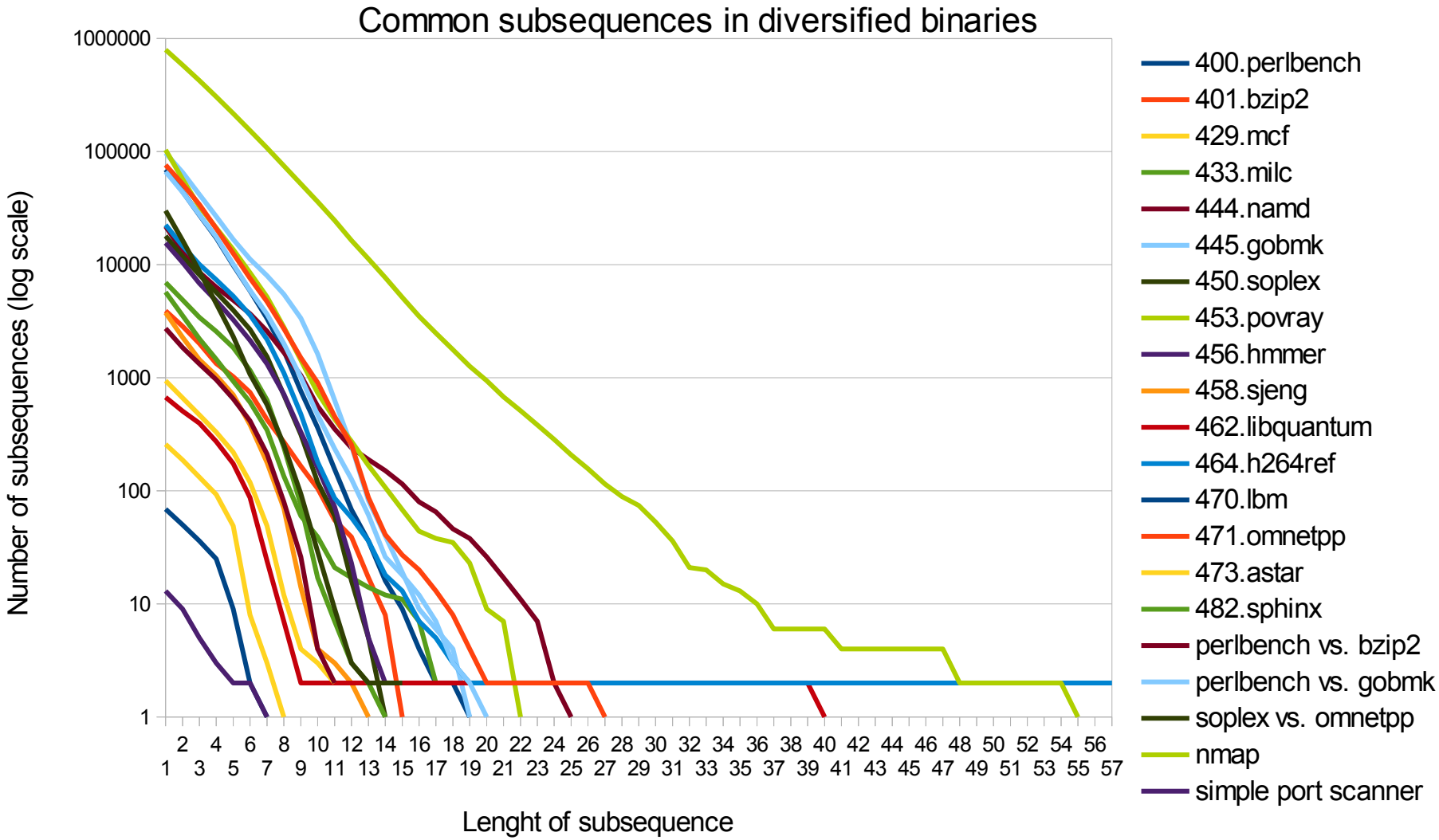
- Generate unique binaries
  - Minimize common substrings (code or data)
  - Performance overhead not an issue
- Diversify code and data layout
- Diversify static data as well

Implementation

- Prototype built on LLVM 3.4
  - Small changes in code generator, code layouter, register allocator, stack frame layouter, some data obfuscation passes
- Input: LLVM bitcode
- Output: diversified binary

- Source: http://github.com/gannimo/MalDiv

Performance



Usage scenarios

- Malware generator
  - Pumps out unique binaries
- Distributed malware generation
  - Use distributed compile bots

Possible mitigation: Better security practices

Image (c) Wikimedia

Mitigation

- Recover high-level semantics from code
  - Hard
- Full behavioral analysis
  - Harder
- Prohibit initial intrusion
  - Fix broken software & educate users
  - Hardest

Conclusion

Image (c) Wikimedia

Conclusion

- Diversity evades malware detection
  - Fully automatic, built into compiler
  - No need for packers anymore
- Adopts to new similarity metrics
- New arms race between defenders and compiler writers
  - Weapons for the masses

Questions?

Mathias Payer <mathias.payer@nebelwelt.net>
https://nebelwelt.net
https://github.com/gannimo/MalDiv