# Malware Code Analysis & Detection

Dipl. Eng. **Ciprian Pungilă**, M. Sc.

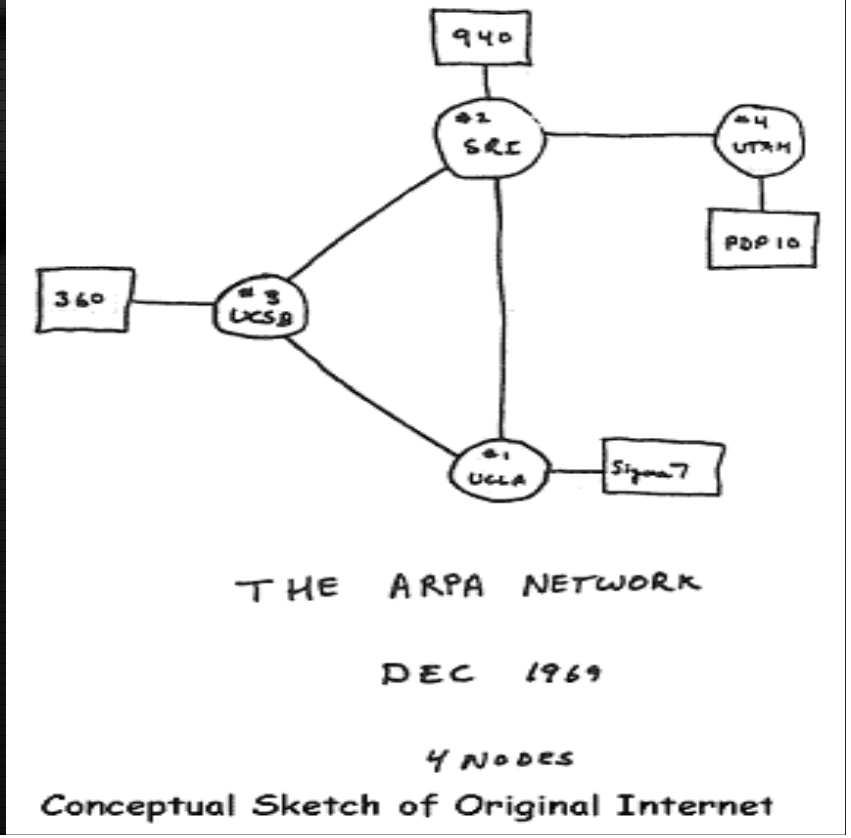
*Western University of Timisoara*

*ciprian@ciprianpungila.com*

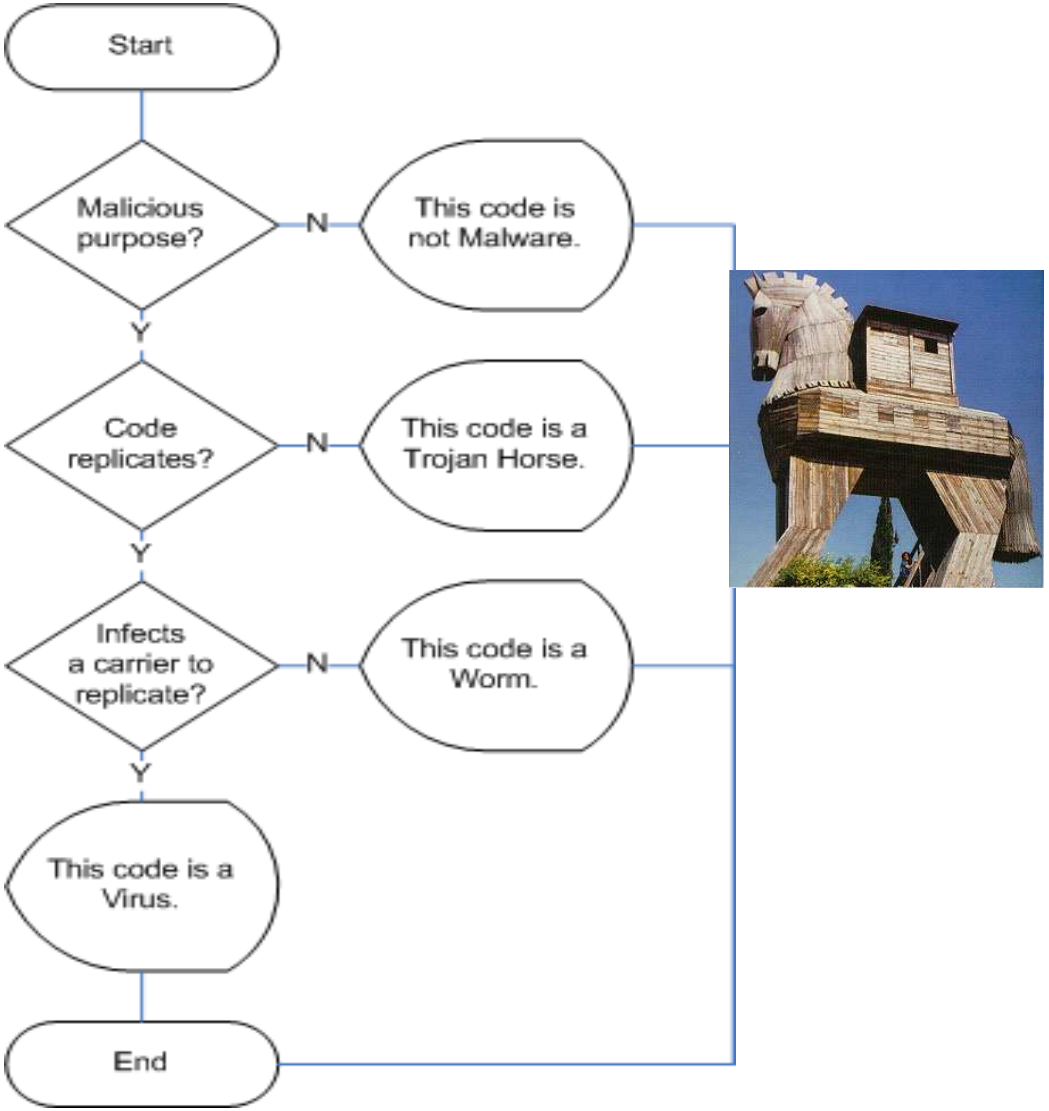
StudIT – 1-3 April, 2011

# Quick **Contents**

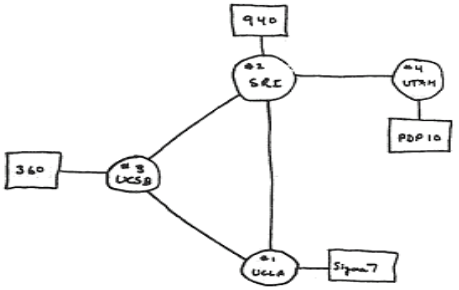
- What Is **Malware**?
- **Evolution** of Malware
- **Technical** Overview
  - **Static** Analysis
  - **Dynamic** Analysis
  - Code **Obfuscation**
- **Protection** Methods
  - **Tools**
  - **Exploit** Frameworks
- **Conclusion**



# What Is **Malware**?



# Evolution Of Malware (1)

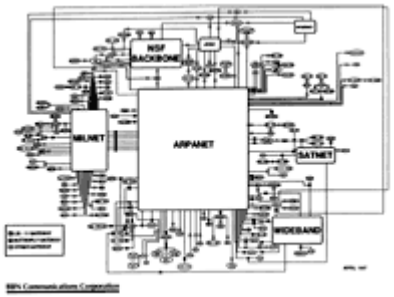


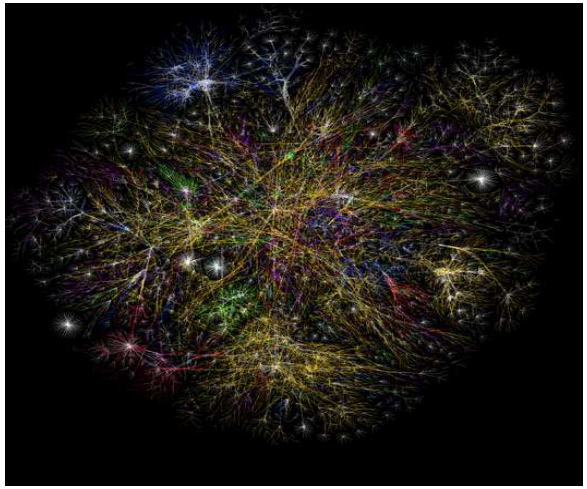

THE ARPA NETWORK

DEC 1969

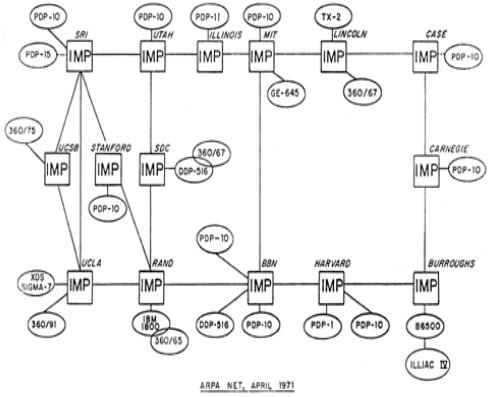
4 NODES

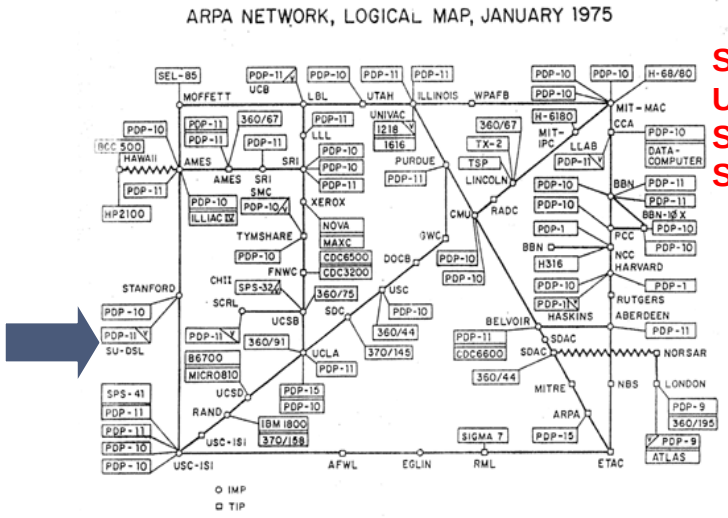
Conceptual Sketch of Orignal Internet



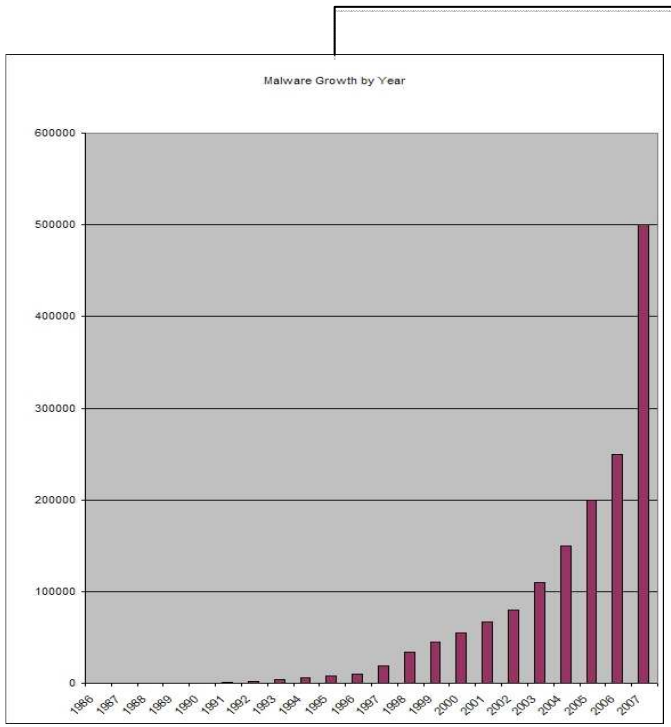
**Email virus + social engineering: Xmas Exec**
**Large scale pandemics: Morris worm**
**Infected 10% of the Internet.**

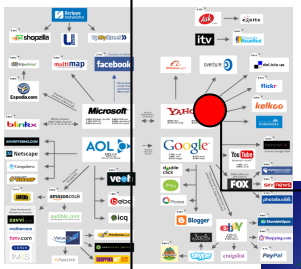


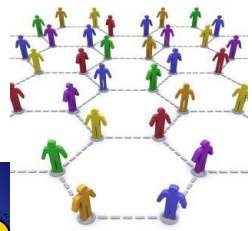



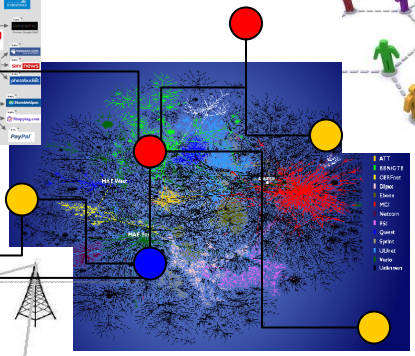

**Sophisticated engineering: Conficker**
**Use of Crypto.**
**Social Networks/cell phone worms.**
**Stuxnet,…**

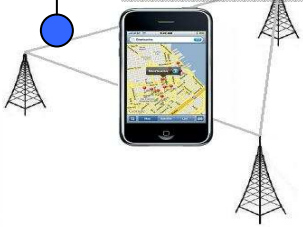
# Evolution Of Malware (2)










**Malware incidents are rising dramatically:**

- increase of infection vectors
- increase in the complexity of botnet structures

**From Biology: Connected World Gives Viruses The Edge**

**"as human activity makes the world more connected, natural selection will favor more virulent and dangerous parasites."**

THE WORLD'S ONLY RELIABLE NEWSPAPER

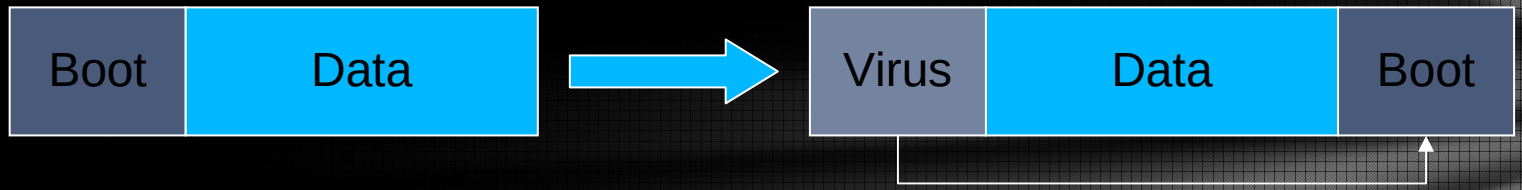
# COMPUTER VIRUS SPREADS TO HUMANS!

## BAR GLASSES HELP YOU SEE STRAIGHT WHEN YOU'RE DRUNK!

# **Technical** Overview

# **Boot Sector** Viruses

First sector of disk executed at boot




Worked well back when people traded floppies

- Could come back; “autorun.inf” on CDs

# Executables

Attach itself to executable

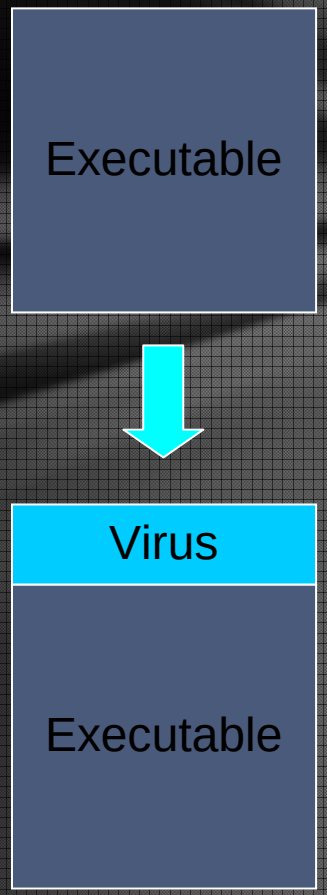
- Virus executes before normal executable is run

Can be multi-platform

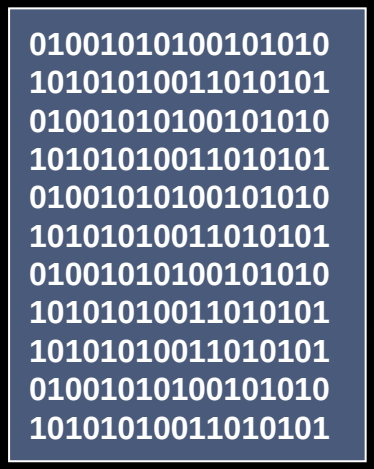
Popular method, esp. when BBS's used to trade software

- Also has infected commercial software distributions

Still in use today



# **Static** Analysis




.exe

Typically a stripped binary with no debugging information.

In the case of malicious code, it is often obfuscated and packed

Often has embedded suicide logic and anti-analysis logic

- What does the malware do
- How does it do it
- identify triggers
- What is the purpose of the malware
- is this an instance of a known threat or a new malware
- who is the author
- …

Challenges:

- lack of automation
- time-critical analysis
- labor intensive
- requires a human in the loop

# **Dynamic** Analysis

- Techniques that profile actions of binary at runtime
- More popular
  - CWSandbox, TTAnalyze, multipath exploration
  - Only provides partial ``effects-oriented profile'' of malware potential

*...while on the other hand...*

## **Static** Analysis

- Can provide complementary insights
- Potential for more comprehensive assessment

# Code **Obfuscation**

To defeat signature based detection schemes

- Polymorphism, metamorphism: started appearing in viruses of the 90's primarily to defeat AV tools

To defeat Dynamic Malware Analysis

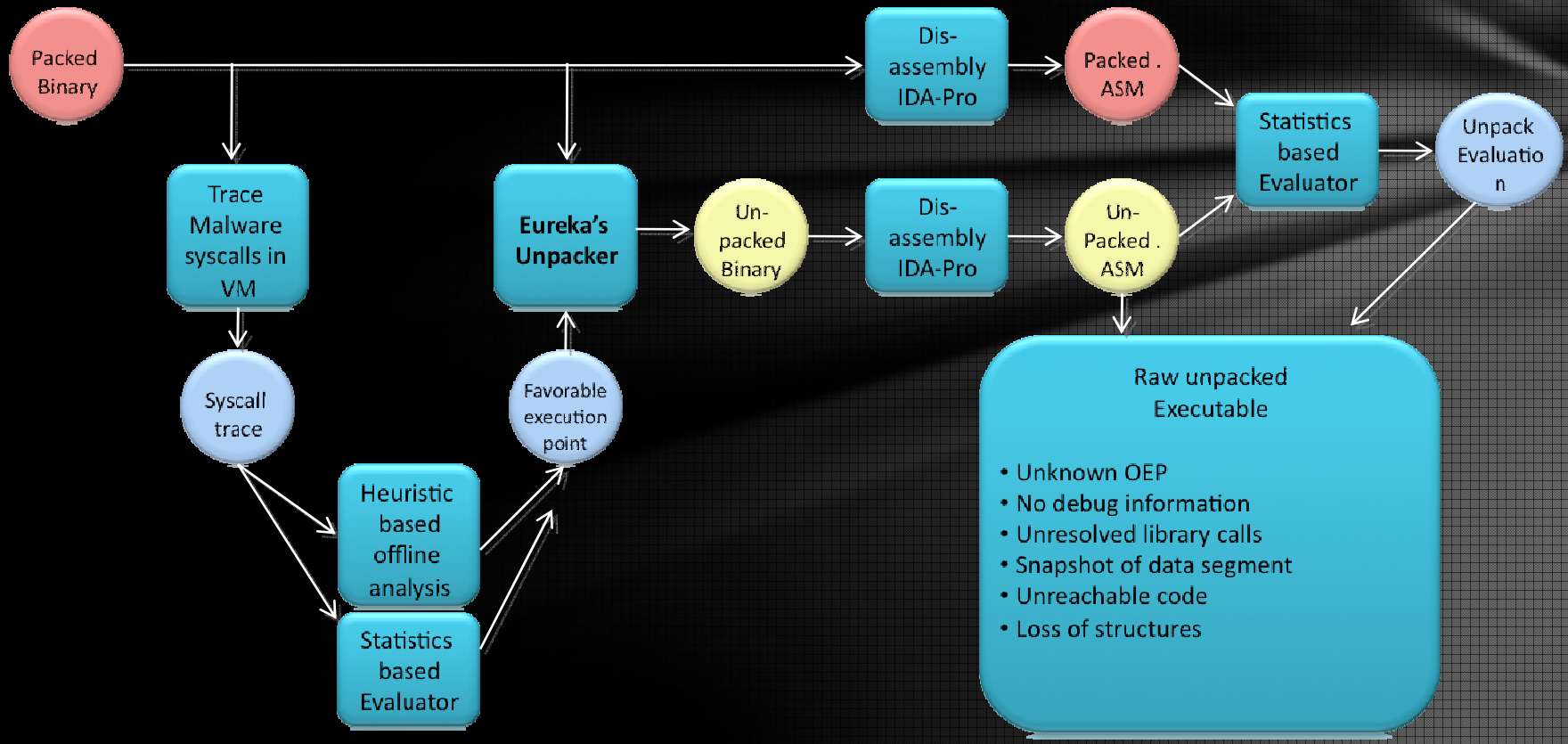
- Anti-debugging, anti-tracing, anti-memory dumping
- VMM detection, emulator detection

To defeat Static Malware analysis

- Encryption (packing)
- API and control-flow obfuscations
- Anti-disassembly

The main purpose of obfuscation is to slow down the security community

# **Eureka** Framework Workflow



# Eureka's Model


**Coarse-grained execution tracing**

**NtTerminateProcess**
**NtCreateProcess**

**Statistical bigram analysis**

**bigram.**

# **Static Analysis** of Executable Code

- Find patterns of malicious code inside the executable
- Various approaches possible, most of them inefficient
  - Simple-pattern searching: Karp-Rabin, Knuth-Morris-Pratt, Boyer-Moore, etc.
  - Repeated parsing of the input data: low performance
- Best approach: multiple-pattern searching
  - Data structures + Formal languages + Graph theory
  - Known approaches: Aho-Corasick, Commentz-Walter, RegEx extensions, etc.
  - Performance vs. memory-usage
    - e.g. 100,000 patterns in a lookup tree => 4GB of RAM used; an optimized version uses 128 MB
    - Multi-core development

# **Dynamic Analysis** of Executable Code

- System-call analysis
  - Analysis of disassembled output
    - Control flow graphs (CFG): http://en.wikipedia.org/wiki/Control_flow_graph
    - Control dependence graphs (CDG): http://www.grammatech.com/research/papers/staticAnalysis/imgSlides/sld021.html
    - Flow dependence graphs (FDG): http://www.grammatech.com/research/papers/staticAnalysis/imgSlides/sld022.html
    - Program dependence graphs (PDG): http://www.grammatech.com/research/papers/staticAnalysis/imgSlides/sld023.html
    - System dependence graphs (SDG): http://www.grammatech.com/research/papers/slicing/slicingWhitepaper.html
  - Intraprocedural & interprocedural slicing algorithms (e.g. Weiser's backward slicing)
  - System-call sequence analysis
    - Several approaches available (e.g. Markov chains, statistical analysis, neural networks, weight-analysis, etc.)
    - Classify program based on detected behavior

# **How** Do We **Protect** Ourselves?

- Avoid creating bugs
  - Write correct code
- Change environments for detecting errors
  - Use tools to exploit effectiveness
  - Find our own bugs (ethical hacking?)
- Use appropriate tools
  - Languages that are type-safe and ensure bound-checks (e.g. Java, Smalltalk, ML, Perl)
  - Subsections of languages and/or code standards (e.g. C++ with smart pointers, std::strings, STL containers)
  - Performance vs. correctness (e.g. bounds checking in Pascal vs. C)

# Tools

LibSafe

- http://www.research.avayalabs.com/project/libsafe/
- Intercept calls to functions with known problems and perform extra checks
- Source is not necessary

StackGuard and SSP/ProPolice

- Place "canary" values at key places on stack
  - http://en.wikipedia.org/wiki/Stack-smashing_protection
- Terminator (fixed) or random values
- ProPolice patch to gcc

# **Run-Time & Compile-Time** Analysis

## BoundsChecker and related tools

- http://www.compuware.com/products/devpartner/
- Augments code with bounds checking code
- Coverage Analysis

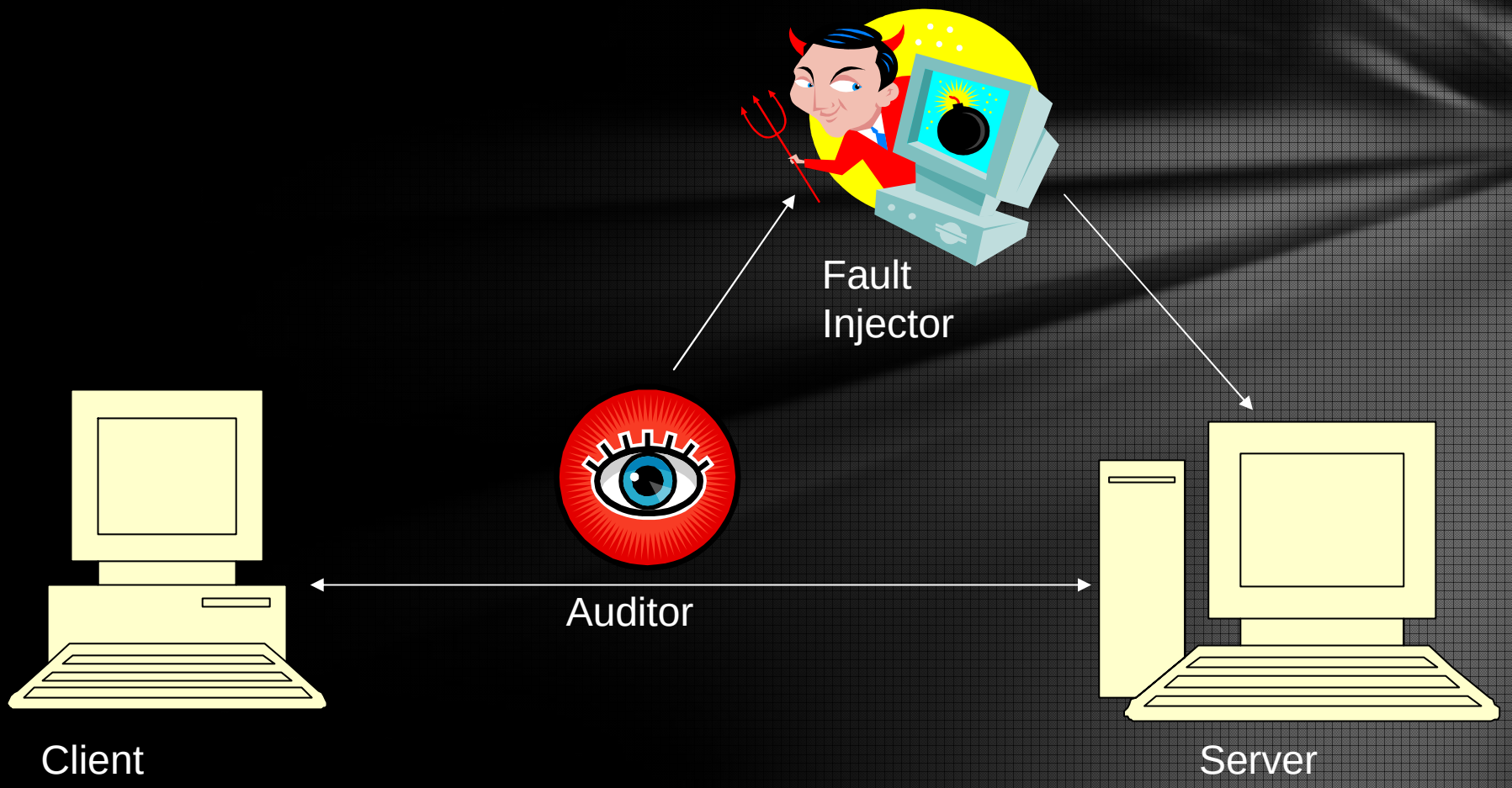
## Rational Purify

- http://www-306.ibm.com/software/awdtools/purify/

# Software Fault Injection

- Hardware fault injection well used and understood
  - Software fault injection still emerging
  - Active research area at CSL
- Identify input areas
  - Generally network, but could also be files, environment variables, command line
- Inject bad inputs and see what happens

# Software Fault Injection – **Model**



# Other **Techniques**

Fuzzing

- A variant of the fault injection model
  - Create "fuzzed" input to cause errors
- ShareFuzz
  - Intercept all getenv() calls to return very, very long strings

SPIKE

- An input language for creating variant network packets
- From ethereal output, make it easy to express new packets
  - a_binary("00 01 02 03")
    Data: <00 01 02 03>
  - a_block_size_big-endian_word("Blockname");
    Data: <00 01 02 03 00 00 00 00>
  - a_block_start("Blockname")
    a_binary("05 06 07 08")
    Data: <00 01 02 03 00 00 00 00 05 06 07 08>
  - a_block_end("Blockname");
    Data: <00 01 02 03 00 00 00 04 05 06 07 08>

# Exploit **Frameworks**

Metasploit

- http://www.metasploit.com/index.html

Canvas

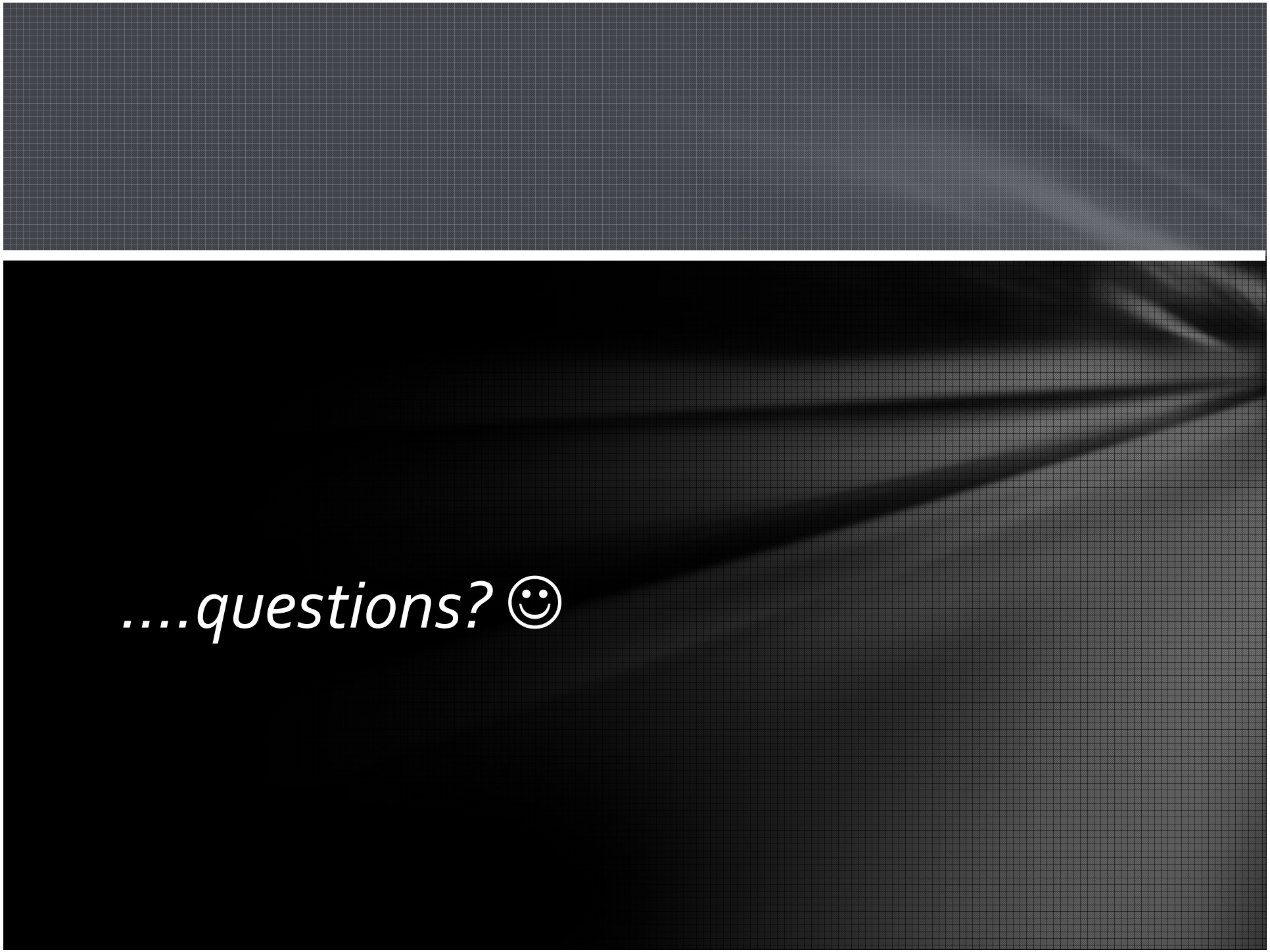
- http://www.immunitysec.com

Core Impact

- http://www.coresecurity.com/products/coreimpact/index.php

# Conclusion

- No 100% accurate methods for analysis
  - Godel's incompleteness theorem
  - Turing's halting-problem
- Exponentially-increasing databases cause problems in static analysis
- Perpetually evolving polymorphic and metamorphic techniques disrupt heuristic/dynamic analysis easily
- New proactive methods of defense emerge embedded in kernels of OSs (e.g. PatchGuard in Vista, etc.)


....questions? ☺

# Bibliography


- *Malware and Protections*, CyberSecurity Lab, 2006
- *Reverse Engineering Malware*, SRI International, Hassen Saidi, Computer Science Laboratory
- *Hybrid Analysis and Control of Malware*, Kevian A. Roudy, Barton P. Miller
- *Reverse Engineering Malware*, Lenny Zeltser, Spring 2010
- *Eureka: A Framework for Enabling Static Malware Analysis*, Wang Zhi, The 13th European Symposium on Research in Computer Security (ESORICS) conference 2008
- *Malware Analysis,* Jaimin Shah, Krunal Patel, Vishal Patel, Shreyas Patel, Georgia Institute of Technology, School of Electrical and Computer Engineering
- *Malware Analysis and Playpen Recuritment Talk*, Alan S.H. Lam
- *COEN 252 Computer Forensics - Investigating Hacker Tools*
- *Malware and Exploit Enabling Code*, CS498IA, Spring 2007