
CS696 Fall 2007
17 September 2007

**David Evans**
University of Virginia
http://www.cs.virginia.edu/evans/

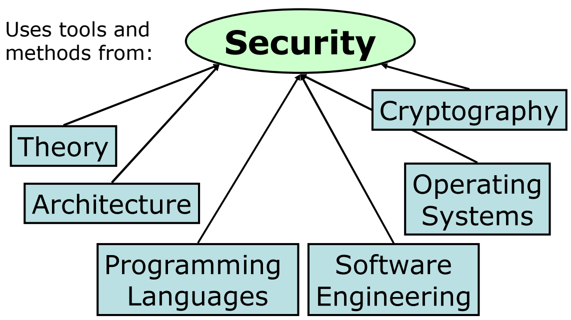
# Computer Security

Study of computing systems in the presence of *adversaries*

about what happens when people don't follow the rules

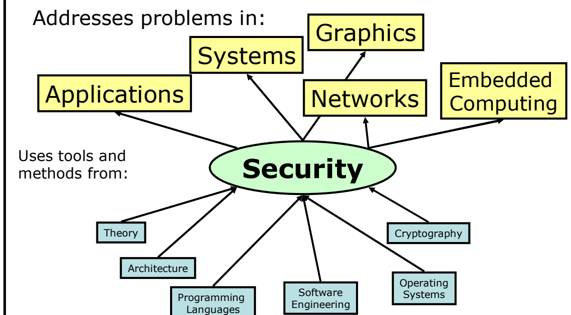


# Security in Context






# Security in Context






# Menu

(0) RFID Privacy
(Karsten Nohl)

(1) User Intent Based Policies
(Jeff Shirley)

(2) Malware Detection
(with Sudhanva Gurumurthi, Nate Paul, Adrienne Felt)

(3) Security through Diversity
(w/John Knight, Jack Davidson, ..., UC Davis, UNM, UCSB)



# RFID Privacy - Karsten Nohl

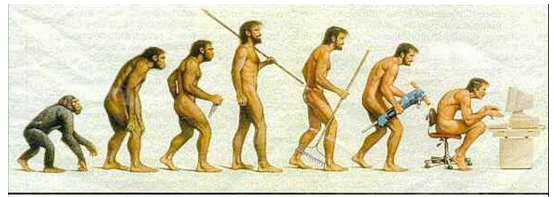

Can we provide adequate privacy and authenticity with simple, cheap primitives?

# User-Intent Based Access Control



Jeff Shirley



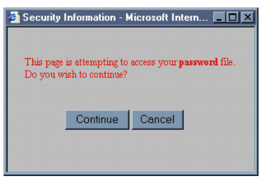


Somewhere, something went terribly wrong

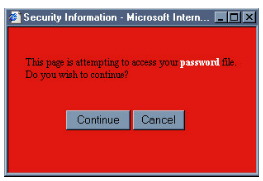
Michael Sinz's Comic



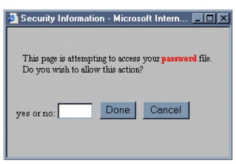
Jennifer Kahng's undergraduate thesis experiment




**37**% clicked Continue




**31**% clicked Continue




**2%** typed in "yes"



# Radical Assumption

Most users are not COMPLETE MORONS!









# User-Intent Based Access Control

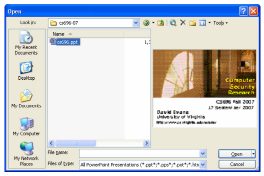
- For desktop systems: the **user** is not the enemy, the **programs** are
- How users interact with programs indicates what they trust them to do
- Polices that incorporate user intent:
  - More precise
  - (Mostly) Universal
  - Dynamic
  - Understandable

## Example: Universal File Policy

FileOpen(file $f)
⇒ read($f)

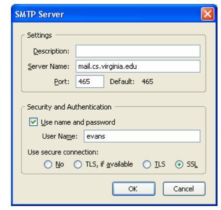
FileSave(file $f)
⇒ write($f)

InstallCreate(file $f)
⇒ read($f), write($f)





## Network Policy



EnterInSmallBox(host $h)
⇒ connect($h)

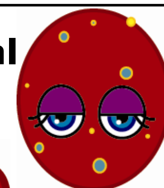


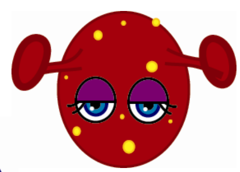
## Challenges

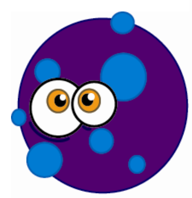
- Securely recording user actions
- Inferring intentions from actions
- Finding and evaluating interesting policies
- Automatically deriving policies



## Disk-Level Behavioral Virus Detection







work with
Nathanael Paul,
Adrienne Felt,
and Sudhanva Gurumurthi
http://www.cs.virginia.edu/malware

David Smith
"Melissa" 1999

Michael Buen Onel de Guzman
"ILoveYou" Worm, 2000

Stereotypical Malwarist, circa 2000



## "ILoveYou" Worm Code

```
rem  barok -loveletter(vbe) <i hate go to school>
rem by: spyder  /  ispyder@mail.com  /
        @GRAMMERSoft Group  / Manila,Philippines
…
x=1
for ctrentries=1 to a.AddressEntries.Count
  set male=out.CreateItem(0)
  male.Recipients.Add(a.AddressEntries(x))
  male.Body = "kindly check the attached LOVELETTER …"
  male.Attachments.Add(dirsystem
                 &"\LOVE-LETTER-FOR-YOU.TXT.vbs")
  male.Send
  x=x+1
next
```

Thoughtful message

Hid location

Creative speller

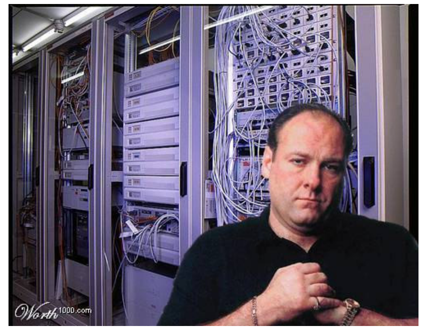
Good understanding of for loops

## Detecting "ILoveYou"

file.contains("@GRAMMERSoft Group")

- Signature Scanning
  - Database of strings that are found in known viruses
  - A/V scanner examines opened files (on-access) or stored files (on-demand) for that string





Picture by Tobic, http://www.worth1000.com/emailthis.asp?entry=31033

## Stereotypical Malwarist, 2007



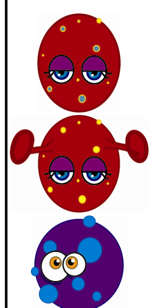
## The Organized Malware Industry

- Multi-million dollar industry
- Vulnerability black market
  - Zero-day exploits sell for ~$4000
- Virus "professionals"
  - Sell viruses, or use them to build botnets and rent spamming/phishing service

Bad news for society, but great news for security researchers!



## Modern Viruses



- Multi-threaded, stealthy, parasitic
- Self-encrypted: each infection is encrypted with a new key
  - No static strings to match except decryption code
- Metamorphic: the decryption code is modified with each infection
  - Modify instructions
- Below host level: rootkits



## Traditional Detection is Doomed

- **Reactive:** signatures only detect known viruses
- **Static:** code is easy to change and hard to analyze
- **Circumventable:** malware can get below the detector



## Our Goal

- Detect viruses:
  - At a level malware can't compromise
  - Without disrupting non-malicious applications
  - Without (overly) impacting performance
- Recognize the **fundamental behavior** of viruses, instead of relying on blacklists of known viruses

## Semi-Obvious Riddle

What is:

- Available on almost every computer
- Able to see all disk activity
- And has processing power and memory comparable to ~2000 Apple II's?

The disk processor.

200MHz ARM Processor, 16-32MB Cache



## Even More Obvious Riddle

What behavior do all file-infecting viruses have in common?

They infect files.



## Disk-Level Behavioral Detection

Executing Program

Program makes file requests to OS

Operating System

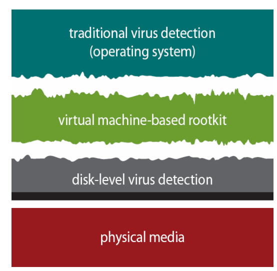
OS issues Read/Write requests to disk

Disk processor analyzes request stream for malicious behavior

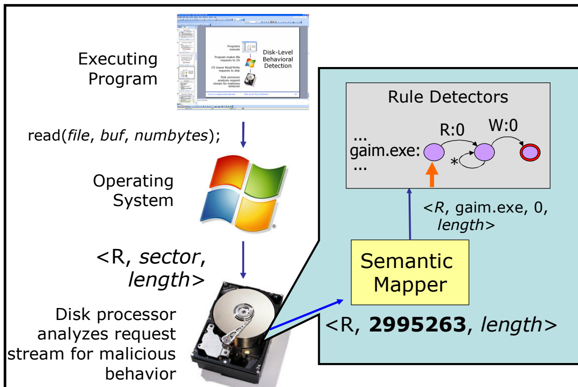


## Advantages

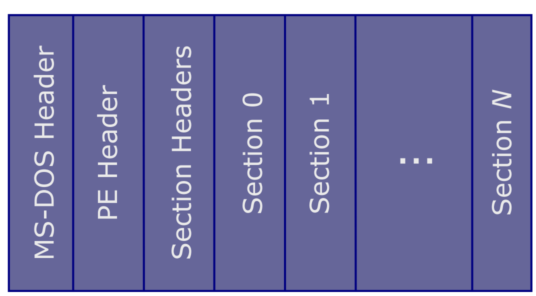
- **Proactive**
  - General techniques to detect new viruses
- **Difficult to Evade**
  - Can't hide disk events from disk
  - Dynamic: Hard to change disk-level behavior
- **Difficult to Circumvent**
  - Runs *below* host OS



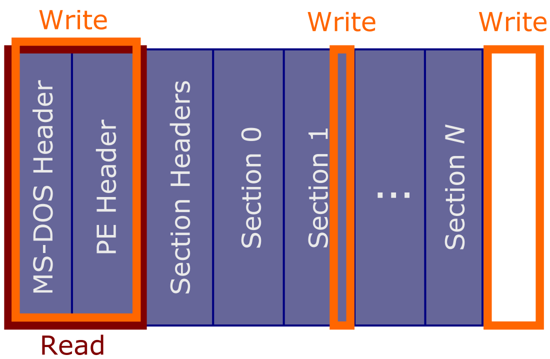





Windows PE File




## Infecting a Windows PE File



## RWW Rule

read [*name*@offset:0;
write [*name*@offset:0],
write [*name*@offset:∗]+

,-separated events in any order

;-separated groups are ordered

*name* is an executable file (starts with MZ or ZM)



## Detection Results

| Virus | *RRWW* | *RWW* | *RW* | *W* |
|---|---|---|---|---|
| Alcaul.o, Chiton.b, Detnat, Enerlam.b, Ganda, Harrier, Jetto, Magic.1590, Matrix.750, Maya.4108, NWU, Oroch.5420, Parite.b*, Resur.f, Sality.l*, Savior.1832, Seppuku.2764, Simile, Tuareg (19 viruses) | All infections detected | | | |
| Aliser.7825 | 70% | 83% | All infections detected | |
| Efish* | 87% | All infections detected | | |
| Evyl | 91% | All infections detected | | |



## False Positives

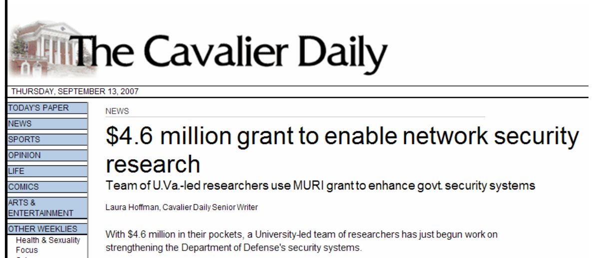
- Experiments with 8 users, 100 million events
  - *RRWW*: 3, *RWW*: 15, *RW*: 35, *W*: 118
- Few Causes: updates, system restores, program installs, software development
- Solutions – if we can change some hard to change things



## Helix Project: Security through Dynamic Diversity

with Jack Davidson, John Knight, Anh Nguyen-Tuong and University of New Mexico, UC Davis, UC Santa Barbara




The Cavalier Daily

THURSDAY, SEPTEMBER 13, 2007

TODAY'S PAPER
NEWS
SPORTS
OPINION
LIFE
COMICS
ARTS & ENTERTAINMENT
OTHER WEEKLIES
Health & Sexuality
Focus
Science

NEWS

$4.6 million grant to enable network security research

Team of U.Va.-led researchers use MURI grant to enhance govt. security systems

Laura Hoffman, Cavalier Daily Senior Writer

With $4.6 million in their pockets, a University-led team of researchers has just begun work on strengthening the Department of Defense's security systems.

## Security Through Diversity

- Today's Computing Monoculture
  - Exploit can compromise billions of machines since they are all running the same software
- Biological Diversity
- Computer security research: [Cohen 92], [Forrest$^+$ 97], [Cowan$^+$ 2003], [Barrantes$^+$ 2003], [Kc$^+$ 2003], [Bhatkar$^+$2003], [Just$^+$ 2004], [Bhatkar, Sekar, DuVarney 2005]

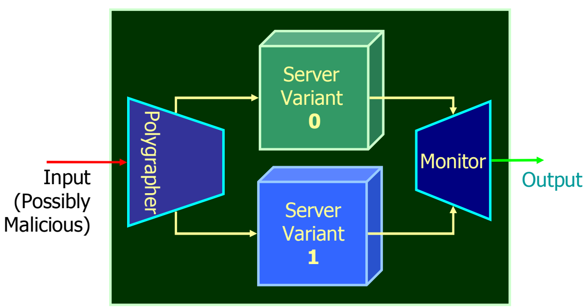


## N-Variant Systems

- Avoid secrets!
  - Keeping them is hard
  - They can be broken or stolen
- Prove security properties without relying on assumptions about secrets or probabilistic arguments
- Allows low-entropy variations



## 2-Variant System



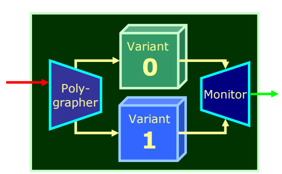



| N-Version Programming [Avizienis & Chen, 1977] | **N-Variant Systems** |
|---|---|
| Multiple teams of programmers implement same spec | Transformer automatically produces diverse variants |
| Voter compares results and selects most common | Monitor compares results and detects attack |
| No guarantees: teams may make same mistake | Guarantees: variants behave differently on particular input classes |



## N-Variant System Framework

- Polygrapher
  - Replicates input to all variants
- Variants
  - *N* processes that implement the same service
  - Vary property you hope attack depends on: memory locations, instruction set, system call numbers, scheduler, calling convention, …




- Monitor
  - Observes variants
  - Delays external effects until all variants agree
  - Initiates recovery if variants diverge



## Variants Requirements

- *Detection* Property
  Any attack that compromises Variant 0 causes Variant 1 to "crash" (behave in a way that is noticeably different to the monitor)
- *Normal Equivalence* Property
  Under normal inputs, the variants stay in equivalent states:

$$\mathcal{A}_0(S_0) \equiv \mathcal{A}_1(S_1)$$

Actual states are different, but abstract states are equivalent

## Memory Partitioning

- Variation
  - Variant 0: addresses all start with **0**
  - Variant 1: addresses all start with **1**
- Normal Equivalence
  - Map addresses to same address space
- Detection Property
  - Any *absolute* load/store is invalid on one of the variants



## Instruction Set Tagging

- Variation: add an extra bit to all opcodes
  - Variation 0: tag bit is a **0**
  - Variation 1: tag bit is a **1**
  - At run-time check bit and remove it
    - Low-overhead software dynamic translation using Strata [Scott, et al., CGO 2003]
- Normal Equivalence: Remove the tag bits
- Detection Property
  - Any (tagged) opcode is invalid on one variant
  - Injected code (identical on both) cannot run on both



## Ideal Implementation

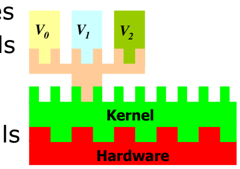
- Polygrapher
  - Identical inputs to variants at same time
- Monitor
  - Continually examine variants completely
- Variants
  - Fully isolated, behave identically on normal inputs

Too expensive for real systems



## Implementation

- Modified Linux 2.6.11 kernel
- Run variants as processes
- Create 2 new system calls
  - **n_variant_fork**
  - **n_variant_execve**
- Wrap existing system calls
  - Replicate input
  - Monitor system calls





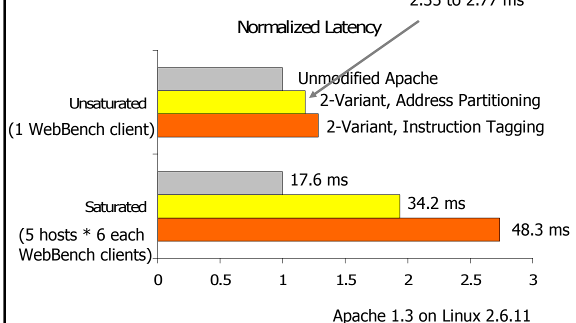

## Wrapping System Calls

- I/O system calls (process interacts with external state) (e.g., open, read, write)
  - Make call once, send same result to all variants
- Reflective system calls (e.g, fork, execve, wait)
  - Make call once per variant, adjusted accordingly
- Dangerous
  - Some calls break isolation (mmap) or escape framework (execve)
  - Disallow unsafe calls



## Results

Latency increase from 2.35 to 2.77 ms



## Big Research Challenges

- Useful variations: diversity effectiveness depends on adversary
  - *Change* some property important attack classes rely on
  - *Don't change* properties application relies on
- What do we do after detecting attack?
  - Recover state, generate signatures, fix vulnerabilities



## Summary

- Computer Security studies computing systems in the presence of adversaries
  - Cross-cuts all areas of CS
  - Projects involving disk drives, RFIDs, OS kernel, user-level applications, dynamic analysis
- Security Lunches (Wednesdays, 1pm)
  http://www.cs.virginia.edu/srg/
- Stop by my office Wednesday, 9:30-10:30am or email to set up a time