The Case for Less Predictable Operating System Behavior

15th Workshop on Hot Topics in Operating Systems

Ruimin Sun, Donald Porter, Daniela Oliveira, Matt Bishop
The Monoculture Problem

- Any vulnerability is automatically widespread
  - W32/Blaster infected 1.4 million systems in a month.
  - Code-Red worm 2001
Malware Sizes are Small

- Lots of effort to get into a system
- Once malware on, easy to remain

How robust can 125 lines be?

Similar arguments in S. Forrest. (WEIS 2015)
We Blame Predictability

- After breaking into a system, malware almost always works as expected
  - Ex: Configuration files always in /etc
What if the OS behaved adversarially toward dodgy software?

- Ex.: Doing the wrong thing for some system calls

Robust malware would get a lot harder to write...
Good Software is Hard to Write

- Portable, robust software already includes a ton of error-handling code and end-to-end checks
  - E.g., lots of variation among UNIX variants
- And applications increasingly distrust the OS
  - Intel SGX, Haven, Inktag, Virtual Ghost, etc.
- Protecting against Iago attacks amounts to even more careful system call checking

Malware should have to do at least this much work
Toward An Unpredictable OS

- Our prototype: **Chameleon**
- Focus on unpredictable system call behavior
  - No bug-for-bug compatibility
  - Non-deterministic system call errors
- Applications run in one of three environments
  - Can move over time
Unpredictable: Unknown Software

- Example: Downloaded game, maybe *trojan*?
- Raise engineering effort for anything to run
  - Disproportionate harm to malware?
- Approach: Perturb system calls in kernel
  - Arguments, return values, drop call altogether
- Monitor software, possibly transition to other environments
Diverse: Known-Good Software

- Fewer instances of same exploit
  - Restrict unpredictability to specification
- Trend to push functionality out of kernel
  - Easier to implement N versions of each piece
- Randomly combine *implementations* at runtime
- More in paper
Deceptive: Known Malware

- Similar to a honeypot
  - Monitor and report to CERT

- System behaves consistently, but falsely
  - E.g., bogus files, fakes actions taken as root

- More in paper
Working Scenario

Bob, 78
living in a retirement community in Florida

• not computer savvy
• clicks in links from phishing email
• installs malware engage in later DDoS attacks
• Bob never notices as malware is active after 1am.
Chameleon Scenario

Signed by Mozilla

Who knows?

Matches malware signature

Diverse

Unpredictable

Deceptive

Looks ok

Looks malicious
Preliminary Study

- Hypothesis
  - Malware are more sensitive to unpredictable behavior

- Methodology [more in paper]
  - Interfere with common malware system calls

- Example Strategies
  1. Ignore the system call
  2. Change buffer bytes (read, write, send)
  3. Increase wait time
  4. Change file pointer (inject a seek)
Keylogger with Unpredictability

- Strategies on Keylogger
  - Change `write(fd, *buf, size) buffer;`
  - Change `lseek(fd, offset, whence) pointer;`
Botnet with Unpredictability

- **Strategies on Botnet**
  - Silence `read(fd, *buf, size);`
  - Silence or reduce `len in sendto(sockfd, *buf, len, ...)`;

![Comparison on Number of Bytes Sent](image1.png)

**Spam truncated with bytes lost**

![Comparison on Number of SYN Sent](image2.png)

**3 bots needed to do 1 bot’s job**
What About Benign Software?

- Firefox, Thunderbird and Skype work(ish)
  - Works normally most of the time
  - Occasional warnings (click “ok”)
  - Some functionality temporarily unavailable
    - Example: webpage or contacts won’t load
    - Usually fixed by retrying

Unpredictability can disproportionately harm malware!
Lots to Figure Out

- More precise strategies for unpredictability
  - Malware-specific system call sequences?
  - Other rules or policies for unpredictability
- Experiment with more software and malware
- User studies with real Bobs (and Alices)
- Abuse OS classes to implement 50+ lib. instances
Conclusion

- Unpredictability can thwart attacker with less effort than generating attacks

- Chameleon’s three environments:
  - Diverse: reduce monoculture of benign software
  - Unpredictable: raise engineering effort for malware to at least match other software
  - Deceptive: lures adversaries into revealing strategies

Contact me!
Ruimin Sun: gracesrm@ufl.edu