Examining the Security of Local Inter-Process Communication

Brendan Leech

Advised by Prof. Peter C. Johnson

I. Abstract

- Many applications have more than one process running
- They communicate over local IPC (completely within the machine)
- This communication has been shown to have vulnerabilities
- This thesis will look at commonly-used applications to find what, if any, security issues can be found.

II. Acknowledgements

III. Introduction

- Many modern applications are split into multiple processes
- This helps to separate functionality, so that each individual process can do its specific job
- However, for these processes to work together, they need to be able to communicate
- This communication is called inter-process communication
  - Definition of IPC: any form of communication between two processes
  - This is a broad definition that captures everything from email to shared memory to semaphores.
  - Talk about how IPC works
    - RFCs
    - How network stack allows communication between computers
    - How local IPC works just within a computer
- This communication has been shown to be insecure
  - How? How can this be related to networking?
  - Brief survey on what insecure means: information disclosure and execution hijacking

- With local IPC, the communication never leaves the physical machine, so an attacker would need to have access to the computer
- Many computers have guest accounts, and there are ways to keep programs running after a user has logged out (nohup and fast-user switching)
- There are also all public terminals, like the public computers at Middlebury. These would be prime targets, since they have many users accessing them throughout the day.
- Therefore, an attacker could start a malicious program and logout, without the victim having any idea that their machine is compromised
- Plan of the Thesis
  - Create survey to find commonly-used applications and their local IPC footprints
  - Using the results, find "interesting" programs to study. Interesting will be defined once the survey has been completed. I want to use applications that are both widely-used and use different forms of local IPC.
  - Use fuzzing software and tools to examine the communication to try and find vulnerabilities
  - Fuzzing can find crashes
  - Studying the communication can find ways to steal information or hijack execution

IV. Background

A. Inter-Process Communication

1. What is IPC
   - IPC is the way that any two programs communicate with each other
   - Broadly speaking, everything from email, to webpages, to Spotify, uses IPC
   - The most common form people know is internet communication, which allows separate computers to connect
   - Talk about the kinds of problems well-solved by multi-process applications
     - Web servers (to handle concurrent connections)
     - Password managers
     - XWindows (server to do processing, client shows the pixels)
2. Local IPC
   - This communication occurs entirely within one computer

- Since it stays within the computer, many times the overhead required to communicate between computers can be avoided
- This can make the communication significantly faster
- Because applications are split into multiple processes, they also must communicate sensitive data
- Since it's within one machine, some security experts believe it is not worth trying to encrypt
- However, many programs make an effort to encrypt it, but do a worse job than with their networked communication

3. Forms of Local IPC
   a. Communication to localhost
      (1) How does this work (explain full network stack)
      (2) Why is localhost useful
   b. UNIX Domain Sockets
      (1) How does this work
      (2) How is this different from localhost
      (3) What are the tradeoffs between localhost and these
   c. Named Pipes
      (1) How do these work
      (2) What are the benefits over localhost/UNIX domain sockets
      (3) Differences between these and anonymous pipes
4. How to see local IPC
   a. For localhost - Wireshark
   b. For UNIX domain sockets - sniff the packets as they are sent or modify kernel functions to send messages to another location as well
   c. For named pipes - join the pipe as a reader

B. Attack Vectors against Host-Only Applications

- Memory leaks: not flushing memory values and leaving confidential information
- Communication channels

C. Input Management/Parsing

- Input management/parsing is the process to make sure that input follows the expectations of the programmer
- Unexpected input could follow execution paths that were unintended. Could create bugs or other vulnerabilities
- Call all input a language. Expected input is the accepted language of the program
- If the language is regular or context-free, then we can prove whether or not a parser only accepts valid input. Can put this parser at the beginning of program

- However, if the language accepted is recursively-enumerable, then being able to prove that a parser only accepts this language is undecidable
- Talk about language theoretic security
- If programmers designed their program/protocol so that it accepted a regular/context-free language, then we can always guarantee whether a parser is correct or not
- However, many protocols use formats that are neither regular nor context-free
- Having a parser at the front of the program is easier to ensure it works correctly, as opposed to spreading the parsing output throughout the program

D. Input-Based Vulnerabilities

- What they look like
- What are their effects?
- Survey of some that have happened: heartbleed, buffer overflow in libpng 1.2.5, SQL injection
- If there is any issue in a system call, then since those occur in kernel mode, that would be a huge vulnerability

E. Fuzzing

1. What is fuzzing
   - Sending random or semi-random input to a process to try and induce a crash
   - Try to cause undesired behavior by the program
   - This is one good way to go about finding input vulnerabilities
   - This can be done by the authors, or by third parties
   - Why either side would want to do it
2. The fuzzers I am using
   - Radamsa
   - Sulley
   - afl
   - What makes them different from each other

V. Results

A. Application 1

- How does IPC work for this application?
- What did we learn from studying communication?
- What did we learn from fuzzing?

B. Application 2

- How does IPC work for this application?

- What did we learn from studying communication?
- What did we learn from fuzzing?

C. Application 3

- How does IPC work for this application?
- What did we learn from studying communication?
- What did we learn from fuzzing?

VI. Future Work

- What can be done to avoid these security issues in the future?
- Would these changes be difficult to complete?

VII. Discussion

- We have found X bugs/vulnerabilities in these applications that use local IPC
- These bugs are due to: careless programming and unknown attack vectors
- They can be fixed in the future with two-way authentication or an air gap

References

[1] Hristo Bojinov, Elie Bursztein, Xavier Boyen, and Dan Boneh. Kamouflage: Loss-resistant password management. In *European symposium on research in computer security*, pages 286–302. Springer, 2010.

[2] Thanh Bui, Siddharth Prakash Rao, Markku Antikainen, Viswanathan Manihatty Bojan, and Tuomas Aura. Man-in-the-machine: Exploiting ill-secured communication inside the computer. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1511–1525, Baltimore, MD, 2018. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/bui.

[3] Patrice Godefroid, Michael Y Levin, and David Molnar. Sage: whitebox fuzzing for security testing. *Queue*, 10(1):20, 2012.

[4] Ivan Homoliak, Dominik Breitenbacher, Alexander Binder, and Pawel Szalachowski. An air-gapped 2-factor authentication for smart-contract wallets.

[5] Zoran Spasov and Ana Madevska-Bogdanova. Inter-process communication, analysis, guidelines and its impact on computer security. 2010.

[6] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[7] Blake Watts. Discovering and exploiting named pipe security flaws for fun and profit, 2002. URL: http://www.blakewatts.com/namedpipepaper.html.