Examining the Security of Local Inter-Process Communication

Brendan Leech '19 - CS 702 Spring 2019
Adviser: Peter C. Johnson

INTRODUCTION

Inter-process communication (IPC) is the way that any two running processes can send data or signals to each other. For example, using email is an example of IPC since one process sends the message to the recipient's process. Local IPC is a subset of this where both processes are on the same computer.

Even though local IPC is confined to a single machine, it has been shown to be insecure [1]. Bui et. al. impersonated a client or server for over a dozen commonly-used applications, including security-conscious applications like password managers and USB security tokens. A server impersonation attack occurs when the resource that a server uses is created before the server makes it, therefore clients connect to the fake version instead of the software's real server. These researchers found that developers often do not follow the same common security practices that are seen for networked communication. For example, these authors found examples of passwords and database contents being sent in plaintext, something that would never happen on the open Internet.

Because of this previous work showing local IPC security holes, I decided to look into local IPC security as well. In addition to attempting to impersonate a server process, I also investigated how well a process responds to randomized input. I wanted to see if input could get through an endpoint's parser to cause undesirable behavior, including possibly a crash. These attacks are called input-based attacks because the victim is made vulnerable by a particular input causing the attack to happen.

I looked at these two attacks---server impersonation and input-based attacks---for three forms of local IPC. I first looked at named pipes, which use a name in the file system to allow a process to join either as a reader or a writer. The next form is Internet sockets that use the loopback interface. These sockets use the same TCP and UDP that normal Internet sockets use, but send that data to an interface that represents the same local machine. Finally, I also looked at UNIX domain sockets. They can work similarly to a named pipe where a user can find the name in the file system and connect to the server. However, they can also be made anonymously using *socketpair* which returns two already-connected UNIX domain sockets. For a process to join a *socketpair* conversation, it must be explicitly given an endpoint by the process that owns them.

METHODS

To impersonate a server, I found the name of the local IPC resource acting as the server. If this name was always the same, then before starting the application I would create the resource with this unchanging name. However, if the name was randomly assigned for each instance of the application, then I would run the application, find the name of the resource, then delete it and make a new one.

To test the security of a communication endpoint against input-based attacks, I used a technique called fuzzing. Fuzzing is the act of sending random or semi-random data and seeing how a process responds. If invalid input is accepted, then the process may misbehave or even crash. To fuzz an endpoint, I first found genuine data that was sent between the real endpoints. I then sent the data through a process called radamsa which randomly modifies this input. Finally, I sent the data to the endpoint I was fuzzing and checked to see if the application was still running. To completely fuzz an endpoint, I did this process 500,000 times.

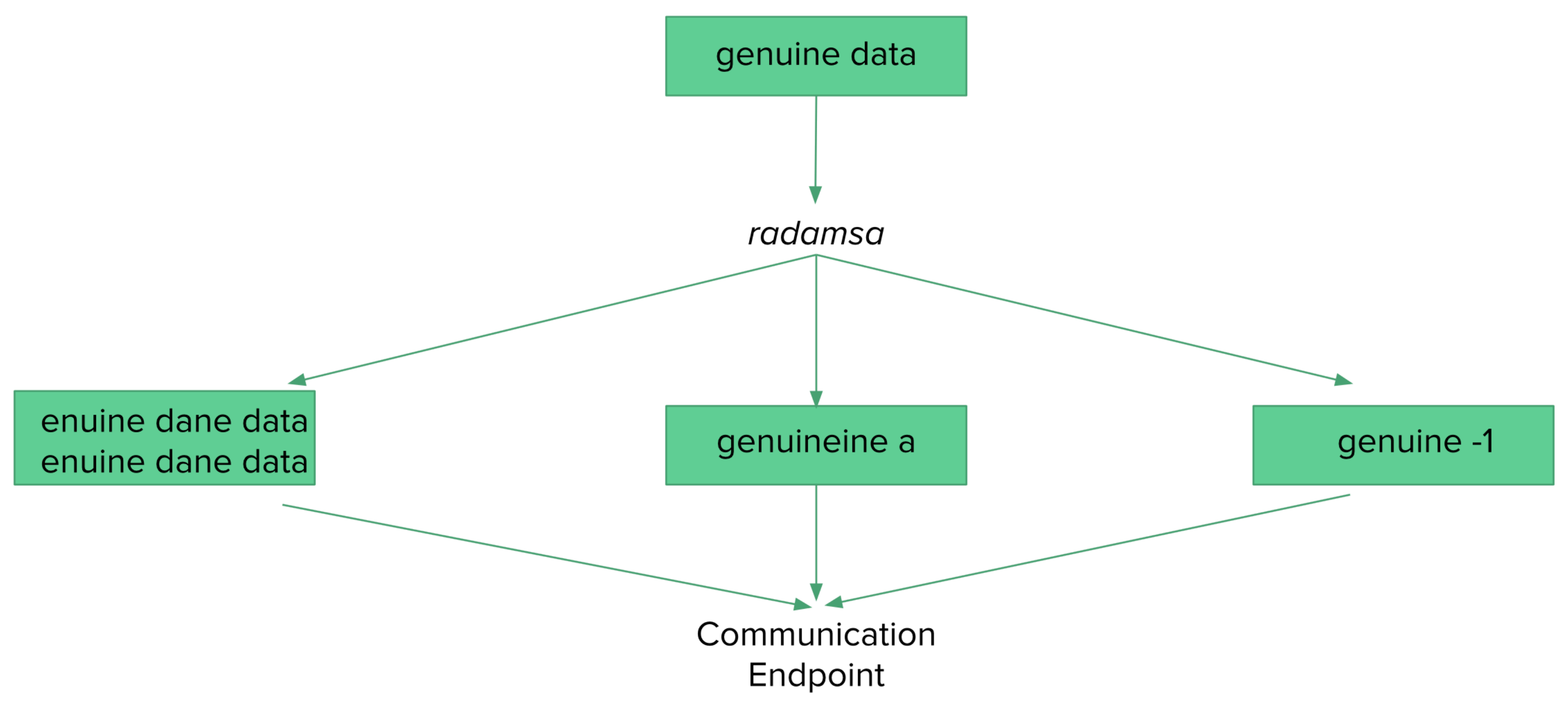

Figure 1: This graphic shows how the phrase "genuine data" can result in different strings after being sent through a fuzzing software called *radamsa*.

REFERENCES

[1] 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.

RESULTS

Spotify: A music streaming service

Form of Local IPC	Socket Type	Attack Vector	Result
Internet Domain Socket	1 TCP Socket	Input-based	Fuzzed with no crash
Internet Domain Socket	3 UDP Sockets	Input-based	Fuzzed with no crash
UNIX Domain Socket	3 named connections	Input-based	Owned by others, could not fuzz
UNIX Domain Socket	8 *socketpair* endpoints	Input-based	Cannot fuzz

VS Code: A text editor with a built-in shell and many add-ons

Form of Local IPC	Socket Type	Attack Vector	Result
Internet Domain Socket	2 TCP Sockets	Input-based	Fuzzing cause VIM extension to crash
UNIX Domain Socket	Named "main" socket	Input-based	Fuzzed with no crash
UNIX Domain Socket	Named "main" socket	Server impersonation	Denial-of-Service Attack
UNIX Domain Socket	Named "git" socket	Input-based	Possible security feature
UNIX Domain Socket	Named "ipc" socket	Input-based	Cannot fuzz
UNIX Domain Socket	Named "shared" socket	Input-based	Fuzzed with no crash
UNIX Domain Socket	18 *socketpair* endpoints	Input-based	Cannot fuzz

launchd: The process responsible for booting a computer

Form of Local IPC	Socket Type	Attack Vector	Result
Internet Domain Socket	4 TCP Sockets	Input-based	Could not reproduce
Internet Domain Socket	2 UDP Sockets	Input-based	Fuzzed with no crash
UNIX Domain Socket	3 named connections	Input-based	Fuzzed with no crash
UNIX Domain Socket	3 named connections	Input-based	Possible security feature

mDNSResponder: Responsible for networking, Airplay/Airdrop

Form of Local IPC	Socket Type	Attack Vector	Result
Internet Domain Socket	1 TCP Socket	Input-based	Always CLOSED
Internet Domain Socket	62 UDP Sockets	Input-based	Fuzzed with no crash
UNIX Domain Socket	1 named socket	Input-based	Fuzzed with no crash
UNIX Domain Socket	1 named socket	Server impersonation	Denial-of-Service Attack
UNIX Domain Socket	4 *socketpair* endpoints	Input-based	Cannot fuzz

Figure 2: The tables above show the results of my research for each endpoint of each application. Green means secure, orange is possibly secure but not sure, and red means insecure. Grey is N/A.

DISCUSSION

Using the tables above, we can make conclusions about the security of each of the four applications above. Based on my research, Spotify seems secure against input-based attacks targeting its local IPC endpoints.

VS Code is not secure since I found a bug by fuzzing a local TCP socket. A bug could lead an attacker to a vulnerability, which could be exploitable, meaning that it may be possible to create an attack that makes use of this bug. Additionally, using server impersonation, I could develop a Denial-of-Service attack that prevented a user from opening VS Code at all. Finally, there is a possible security feature where a connection is closed then reopened when receiving more than two messages, but it is impossible to tell if this is a bug or a security prevention tool.

launchd has a similar possible security feature as VS Code does, but for many sockets. Outside of this, it looks to be secure against local input-based attacks.

I found mDNSResponder to be secure against input-based attacks for all of its open resources. However, I could perpetrate a Denial-of-Service attack through server impersonation that denied access to most of the Internet, including webpages.

ACKNOWLEDGEMENTS

I'd like to thank my advisor, Pete, for all of his help this semester. This project would not have been a success without his advice and guidance throughout the semester. I'd also like to thank the people who took my survey, which helped inform my decision of what applications to investigate. Finally, I want to thank my friends and family who supported me throughout the semester.