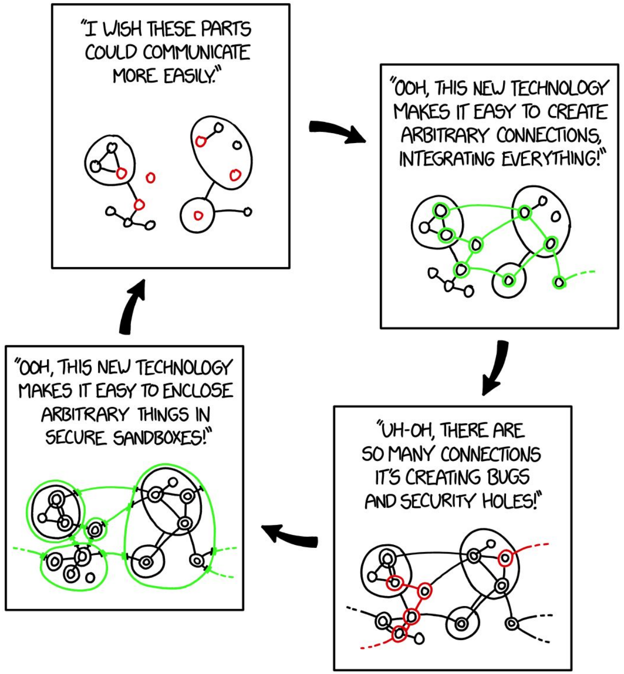
Module: Sandboxing

Into the Jail

Yan Shoshitaishvili
Arizona State University

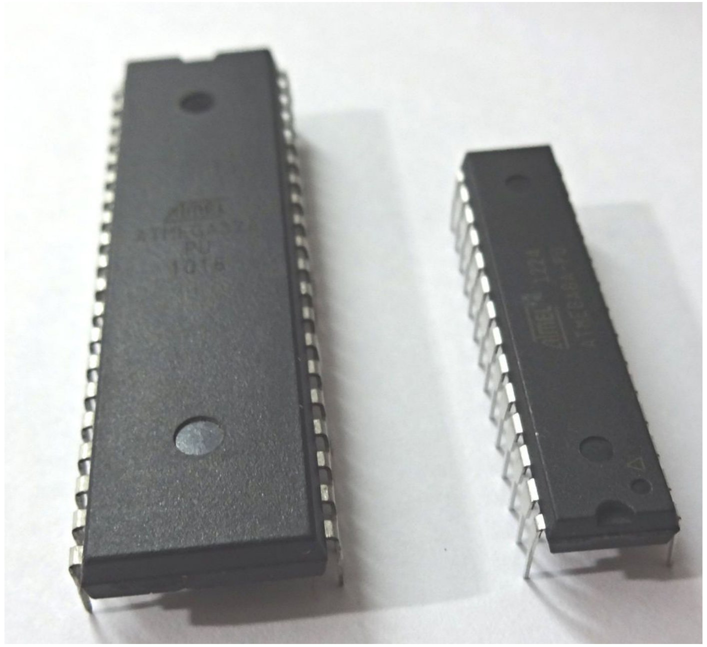

"I WISH THESE PARTS COULD COMMUNICATE MORE EASILY."
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO CREATE ARBITRARY CONNECTIONS, INTEGRATING EVERYTHING!"
"UH-OH, THERE ARE SO MANY CONNECTIONS IT'S CREATING BUGS AND SECURITY HOLES!"
"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO ENCLOSE ARBITRARY THINGS IN SECURE SANDBOXES!"

Sandboxing Cycle
https://xkcd.com/2044/

In the beginning (computing, circa 1950)...

First, everything ran on bare metal.

Problem: every process was omnipotent.



The split of OS and userspace (circa 1960)...

Hardware measures were developed to separate "system" and "process" code (1960s).

Problem: processes could still clobber each other.

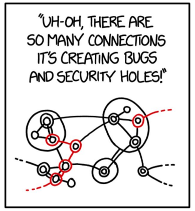
The rise of virtual memory (circa 1980)…

Hardware measures were developed to separate the memory space of different processes.

The rise of *in-process* separation (circa 1990)…

Separation between the interpreter and the interpreted code.

Browser hacking (circa 2000).

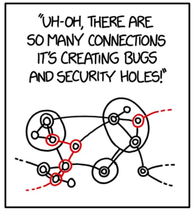

"UH-OH, THERE ARE SO MANY CONNECTIONS IT'S CREATING BUGS AND SECURITY HOLES!"

1. Link victim to malicious web page.
2. Trigger vulnerability.
3. Wreak havoc all over the victim machine.

Known as a "Drive By Download". Popular "traditional" targets:

- Adobe Flash
- ActiveX
- Java Applets

Browser hacking mitigations (circa 2010).


"UH-OH, THERE ARE SO MANY CONNECTIONS IT'S CREATING BUGS AND SECURITY HOLES!"

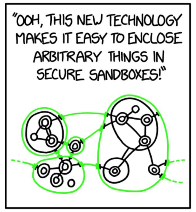
Original solution: eliminate traditional targets.

- Let's kill Adobe Flash!
- Why do we even need ActiveX?
- Java Applets are lame!

Turns out that this does not solve the problem. Hackers moved on to:

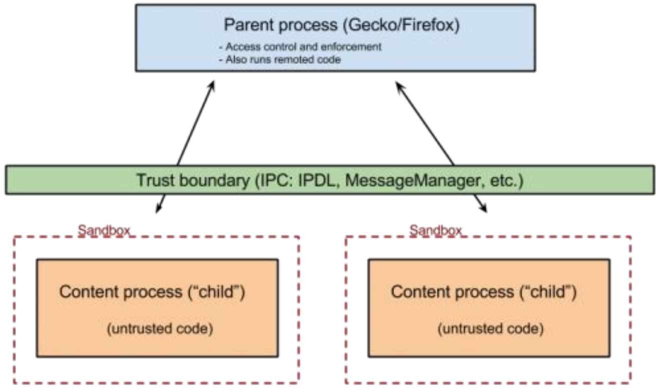
- JavaScript engine vulnerabilities.
- Media codec vulnerabilities.
- Imaging library vulnerabilities.

The rise of sandboxing (circa 2010)...


"OOH, THIS NEW TECHNOLOGY MAKES IT EASY TO ENCLOSE ARBITRARY THINGS IN SECURE SANDBOXES!"

Untrusted code/data (i.e., downloaded JavaScript, PNGs, PDFs, etc) should live in a process with *almost zero permissions*.

1. Spawn "privileged" parent process.
2. Spawn "sandboxed" child processes.
3. When a child needs to perform a privileged action, it asks the parent.


Parent process (Gecko/Firefox)
- Access control and enforcement
- Also runs remoted code
Trust boundary (IPC: IPDL, MessageManager, etc.)
Sandbox
Content process ("child")
(untrusted code)
Sandbox
Content process ("child")
(untrusted code)

In this module, we will learn about different sandboxing technologies and their weaknesses!

How well does this work?

Sandboxing is *extremely* effective.

In this class, we'll see several **strong mitigations** that are so effective that a second vulnerability is needed to bypass the mitigation and make the first vulnerability useful.

Sandboxes are a strong mitigation:

- need one set of vulnerabilities to exploit sandboxed process
- **need another set of vulnerabilities to "break out" of the sandbox.**