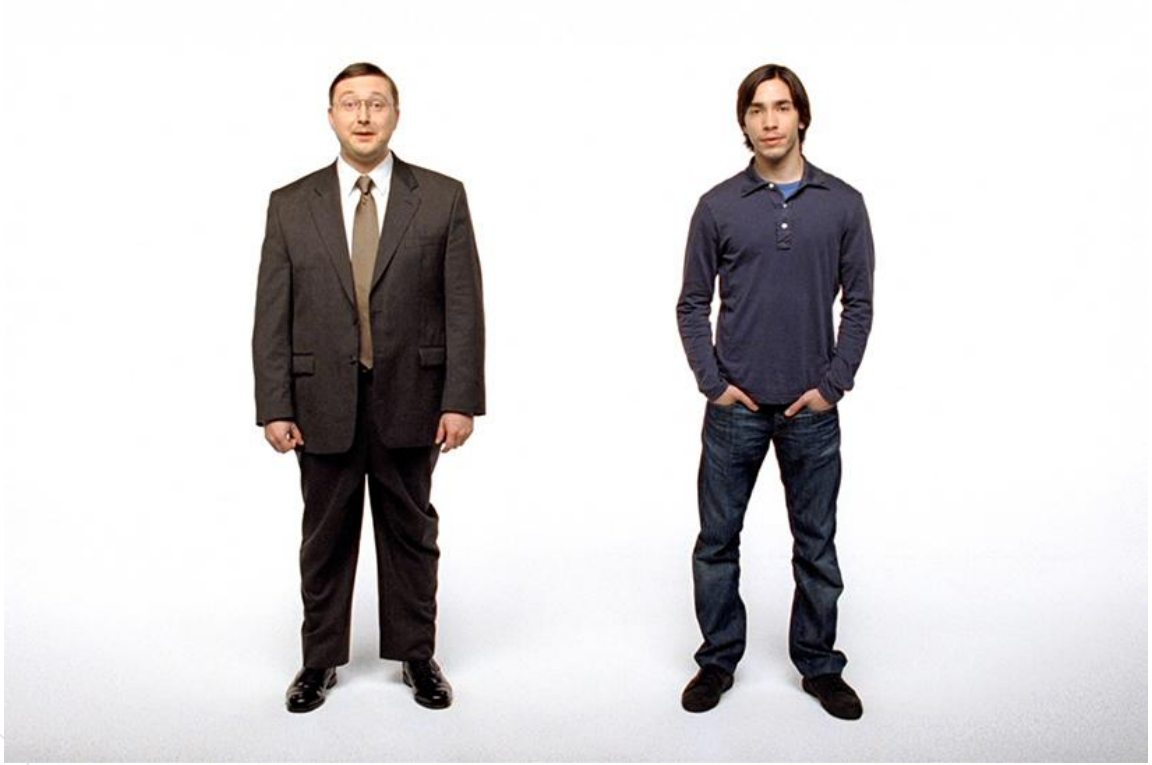
Beyond the PC: Emulating Closed Systems

EaaSI Training Module #8

During This Module

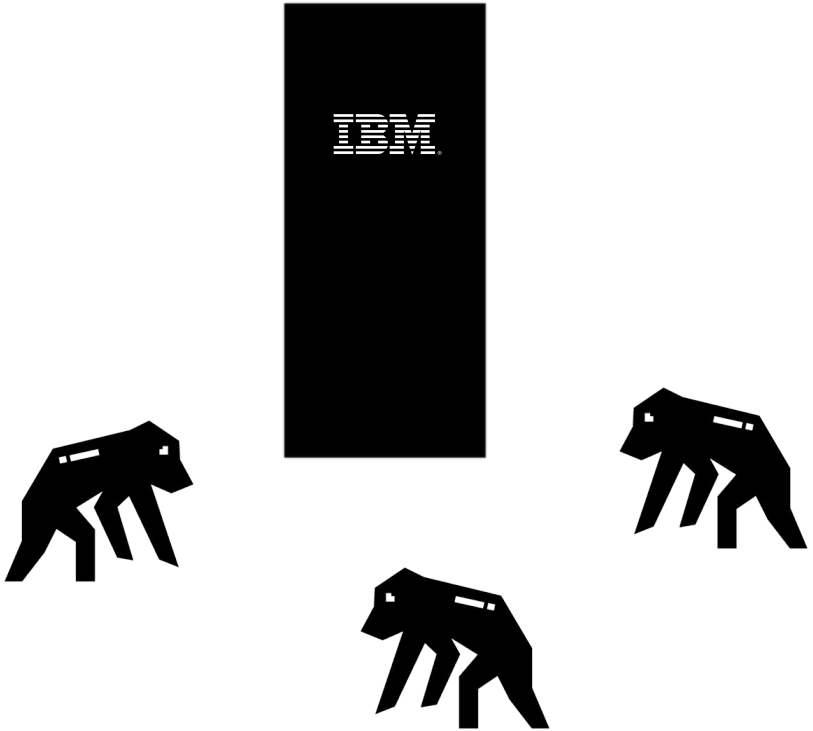
- What is a “closed system”, and how does it differ from an “open system”?
- What’s a “ROM” file?
- What factors limit the capabilities of open source emulators?



But first…what's a "PC", really?

A Little History: Open Systems

- “PC” can refer broadly to **p**ersonal **c**omputers (of any brand, make, or model)
- IBM also used this term in the early 1980s for its IBM Personal Computer line
- IBM PC used an “open architecture” that allowed other hardware companies to release their own models that were “IBM PC compatible”
    - Meaning: they all used the same Intel x86-compatible processors and could usually use the same peripherals and software



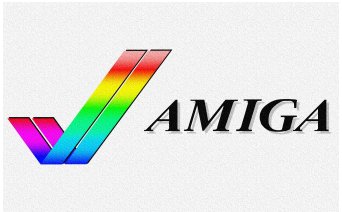
Building “the PC”

- The monster success of the IBM PC and its clones led the way for the varied ecosystem of hardware manufacturers creating interoperable x86 computers, components, and peripherals that still make up the desktop “PC” market today

- The fact that all these specific components share an open architecture makes it relatively simple to create virtual equivalents in emulation

Parallel History: ⟶
Closed Systems ⟶

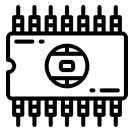
- There have also been many *closed* computing systems created over the years - machines that lock the user into a defined set of compatible components and software



What makes a closed system?

One or more of:

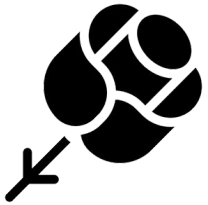
- Proprietary firmware
  - Embedded, device-specific software that makes it possible for a particular piece of hardware to boot an operating system



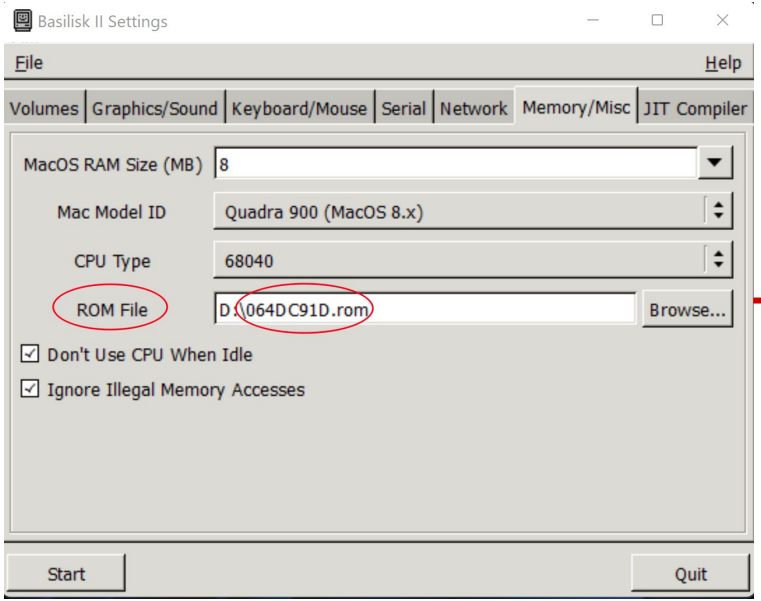
- Technical Protection Measures (TPM)
  - Digital locks preventing access to OS, firmware, or other components

- Inaccessible specifications
  - Trade secrets that prevent third-party integrations

What's in a "ROM"?

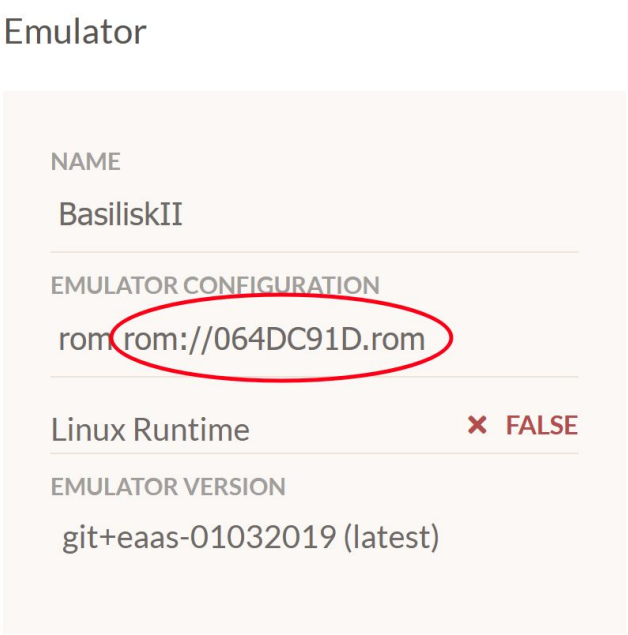


- Commonly-used term for a file containing firmware
- Copied from un-editable **R**ead-**O**nly **M**emory either directly embedded on a computer's motherboard or on a storage device
- On closed systems, obtaining a ROM file may:
  - Involve breaking TPM
  - Run contrary to a purchase agreement or Terms of Service



For Example:
*The Macintosh 68k-series emulator **BasiliskII** requires a ROM file containing firmware from a real-world Macintosh in order to boot Mac OS. Here, the ROM file specified ("**064DC91D.rom**") was pulled from the motherboard of a **Performa 580***

EaaSI-ly Handling ROMs



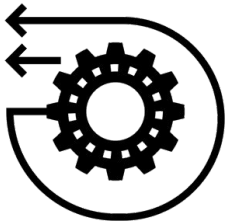
- In the EaaSI platform, ROMs have to be managed somewhat independently from emulator containers and Environments
- Have to balance:
  - **Decision-making:** EaaSI system and service administrators need the flexibility to make their own risk-assessments regarding how they manage and share firmware/ROMs
  - **Ease-of-use:** The EaaSI platform could make ROM use so easy that users don't need to think about them at all

Problem 1: Copyright Chilling Effect

- In the U.S. , Fair Use principles and DMCA exemptions help protect the *use* of emulators in preservation/heritage work
    - However, libraries/archives/museums are generally not leading or funding emulator *development*
    - Emulator development is often driven by individual/hobbyist/community projects that are more vulnerable to takedowns for using proprietary firmware AND can be more difficult to maintain at scale than similar projects led or funded by institutions

- Internationally, further investigation is needed to determine where and how preservation organizations could share emulation resources with pre-configured ROM files across jurisdictions

Problem 2: Reverse-Engineering



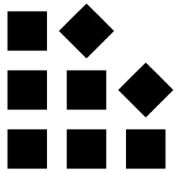
- EaaS* aims to provide a framework with consistent, reliable features on top of emulators that support both open and closed systems

- But the more closed a system is, the more developing an emulator for it requires reverse-engineering, work-arounds, and guesswork

- The differences between recreating an open system and reverse-engineering a closed one challenge the ability to build consistent features for a framework like EaaS

**** See Training Module #3 for discussion of the distinction between the EaaS framework and the EaaSI platform:***
https://www.softwarepreservationnetwork.org/eaasi-training-module-3-the-eaas-eaasi-stack/

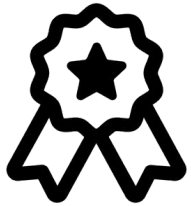
Toward a Generic Emulator Interface



- EaaSI currently uses custom, EaaS-specific “wrappers” to adapt the specific controls and requirements (like ROMs) of specific emulators to the generic preservation and access workflows offered by the framework (opening legacy files/data, running old applications and operating systems, etc.)
- The adaptation work necessary to create these “wrappers” takes significant effort and has kept EaaS development largely limited to the most common, high-demand, well-documented systems (like PCs)
- EaaS, EaaSI, and the preservation community at large would benefit from a consistent, abstract, non-EaaS-specific metadata schema to describe the functionality and features of emulators* - whether they are emulating open or closed systems

For more details and proposed implementation of such a schema, please read Rafael Gischke and Klaus Rechert’s 2022 iPRES paper, “A Generic Emulator Interface for Digital Preservation”

Standards (Redux)



- Computing history encompasses much more than PCs - it is full of wild, varied, and unique systems (some open, some closed) with different needs for emulation

- The heritage community can not retroactively standardize computing systems or emulators, but we can standardize the way we describe and talk about them

- Agreement on preservation metadata for emulators and software can't be tackled by EaaSI program alone

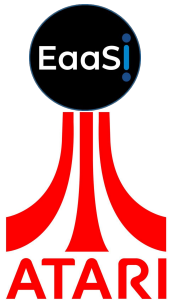
See Training Module #7 for related discussion of EaaSI's efforts to implement metadata standards:
https://www.softwarepreservationnetwork.org/eaasi-training-module-7-implementing-metadata-standards/

Dreaming Big For Emulator Development

- In the meantime, emulators themselves (not just platforms like EaaSI) could receive support and development to better conform to existing preservation standards

- More and better features could be made with preservation and remote access systems in mind rather than stand-alone desktop applications:
    - Broader compatibility with raw flux or forensic disk image formats
    - Exporting files or preservation metadata from emulated systems
    - Integration with common/open source repository management platforms

- Technical accuracy of emulators (particularly of closed systems) could be improved with the backing and protection of public-good/heritage orgs



Together, we can make sure even non-PC systems and emulators are still “EaaSI compatible”!

Credits

- Training Module written and designed by Ethan Gates, Software Preservation Analyst, Yale University Library
- Original photos, screenshots, and videos recorded by Ethan Gates
- Image of Macintosh Performa 580CD courtesy of National Library of New Zealand
- Icons sourced from The Noun Project
- EaaSI program of work sponsored by the Alfred P. Sloan Foundation and the Mellon Foundation, hosted by Yale University Library

Yale

Principle Partner

ALFRED P. SLOAN FOUNDATION

Sponsor

Mellon Foundation

Sponsor