Closed, heterogeneous platforms and the defenders' dilemma

Looking back at the last 20 years of RE and looking ahead at the next few

SSTIC 2018 -- Thomas Dullien ("Halvar Flake")

Progress in Reverse Engineering

- 2010 zynamics blog post about “10 years of progress in RE” [link]
- Changes listed: Graphs, Dynamic Instrumentation, Python, Diffing, HexRays, Collaboration, and a future outlook to more sophisticated academic tools
- Perhaps it is time for another look - both backward and forward

Contents of the talk

- Personal experience with RE tools over the last few years (“Old man yells at cloud”)
- Some thoughts about the challenges (external AND cultural) that the RE community is facing
- Some changes I would like to see (both external AND cultural)
- Future outlook: Computing is changing radically, how will the RE community change?

Chapter 1:
Old man yells at cloud

Personal experience

- Lots of hands-on RE for both malware and vuln-dev from 1998 to summer 2010 (alternated with product development).
- Mostly development and big-corp middle-management from 2011-2015, sabbatical 2015-2016
- Return to concrete hands-on RE in 2017
- Total: 13 years of RE, approximately 6-7 years of “break”. What is it like to come back?

The 6-7 year pause

- Kept reading papers & published results
- Impressive list of “solved” problems published:
  - SMT solvers usable for automated input generation given a program path !
  - VSA with strided intervals and recency abstraction to resolve virtual method calls !
  - So many tools! BAP! Angr! Radare! McSema! Frida!
  - Even FOSS BinNavi!
- Surely reverse engineering must be a joy now.

Expectation set by the published results

- Given a large C++ binary for a mainstream architecture, the following should be “easy”:
    - Disassemble it accurately
    - Recover C++ classes reliably, recover class hierarchy reliably
    - Perform VSA with strided intervals & recency on the entire binary, resolving most virtual method calls statically
    - Perform coverage traces & perform set operations on execution traces
    - Given a path through the program, and a location I know I can hit, generate a big expression to throw to an SMT solver to see if a particular branch condition can be “flipped”
- All of this is surely solved, if we believe our own hype?

The great sobering

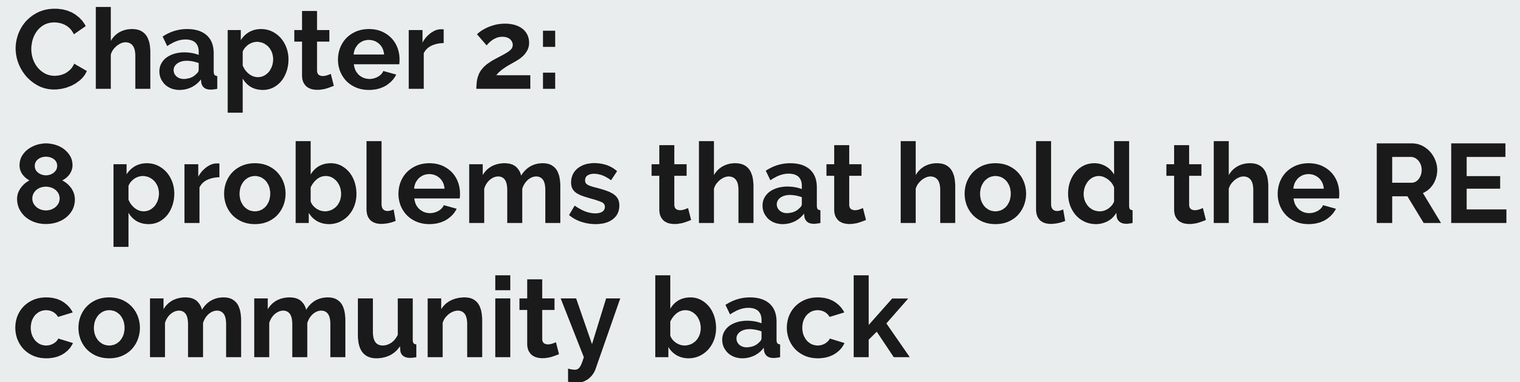
Almost everything I tried to use was broken, or did not work reliably (meaning failed when run on any significant real-world-software).

Examples:

- Getting coverage traces from MPENGINE.DLL - difficult because of privileged process
- Live debugging and obtaining traces from any mobile device - difficult because of closed platforms
- Finding a FOSS library that has reliably retrieves 80%+ of CFGs from a given binary - difficult because ... I do not know.
- Hooking malloc / free inside of a x64 process - difficult because no good FOSS libraries for runtime hooking x64 code (without causing allocations in target process) exist

Nostalgic feeling or reality?

- When weighed for the importance of the target platforms, average tooling was better in 2010 than it is in 2018 ? (feeling - hard to quantify)
- Windows had a mostly useful Debug API.
- Solaris had Dtrace, Linux had reasonably well-functioning debug facilities.
- PIN and DynamoRIO were well-maintained & working.
- Getting just the “basic” tooling to be productive took many months of work.

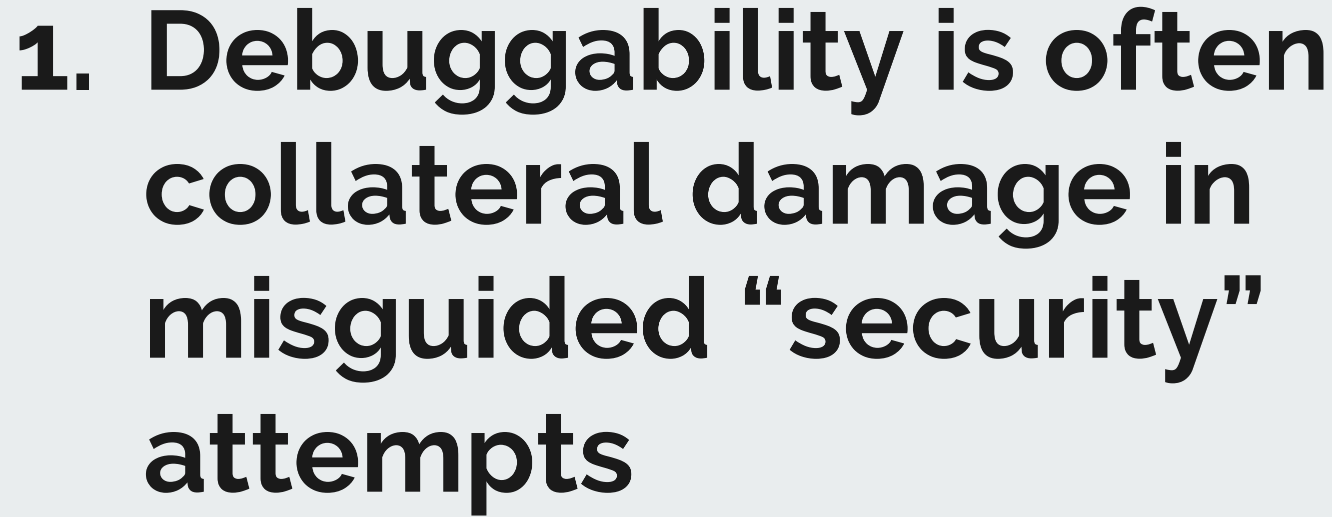


Chapter 2:
8 problems that hold the RE community back

Some are external (e.g. not the fault of the RE community).

Some are cultural (e.g. mostly our fault).

Please do not be angry with me. I mean no harm and do not wish to insult anyone.
Not everything I say applies equally to each project.



(Not the fault of the RE community. Partially the fault of the security community.)

False “security” by breaking inspectability

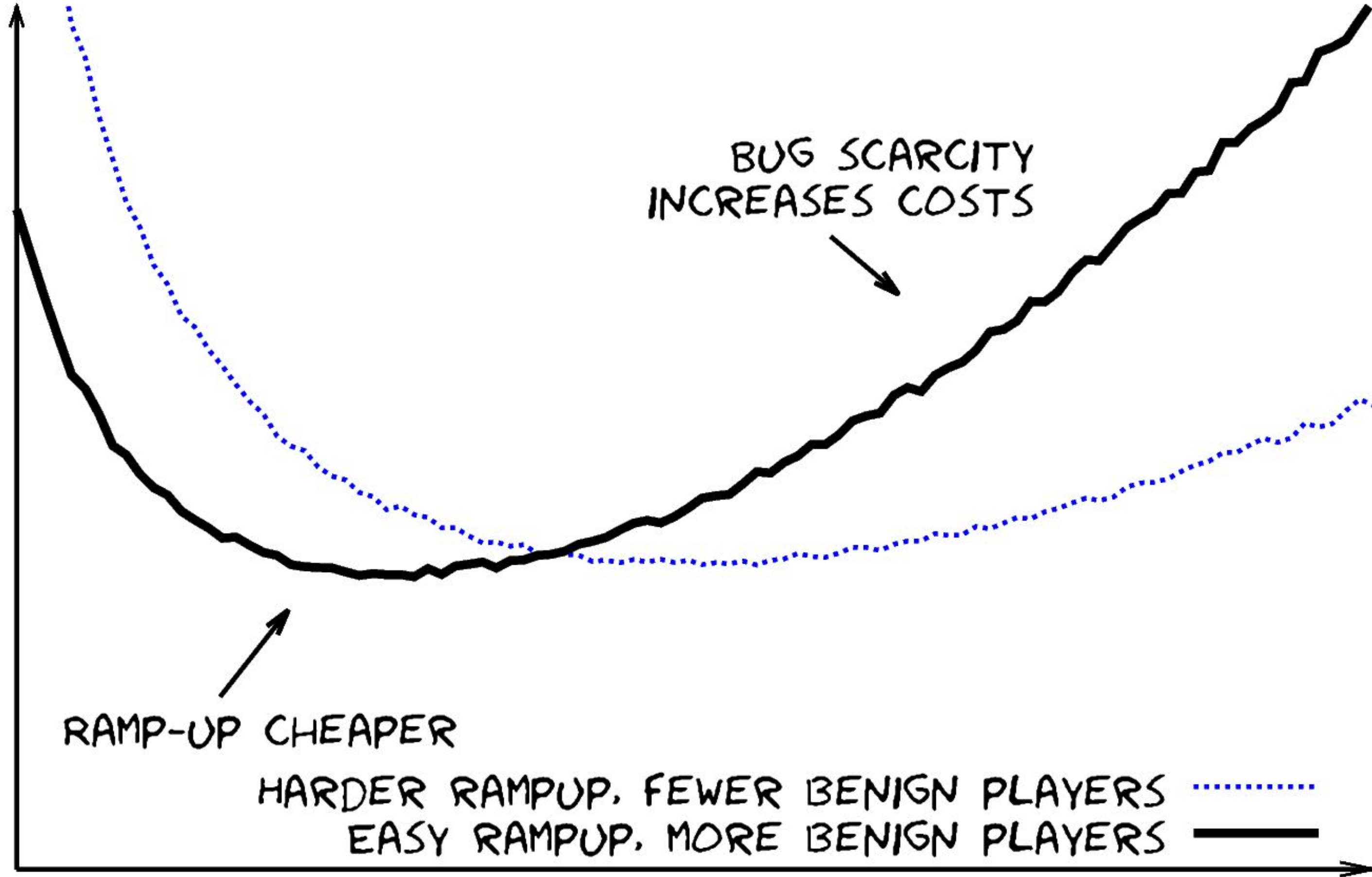
- Device and OS vendors misunderstand the iterated nature of security games
- Wrong assumption: Single-shot game vs. iterated game
- Create obstacles to debugging / inspectability to “raise the bar”
- Commercial attackers pay cost to build infrastructure once
- Defenders have to pay it again and again

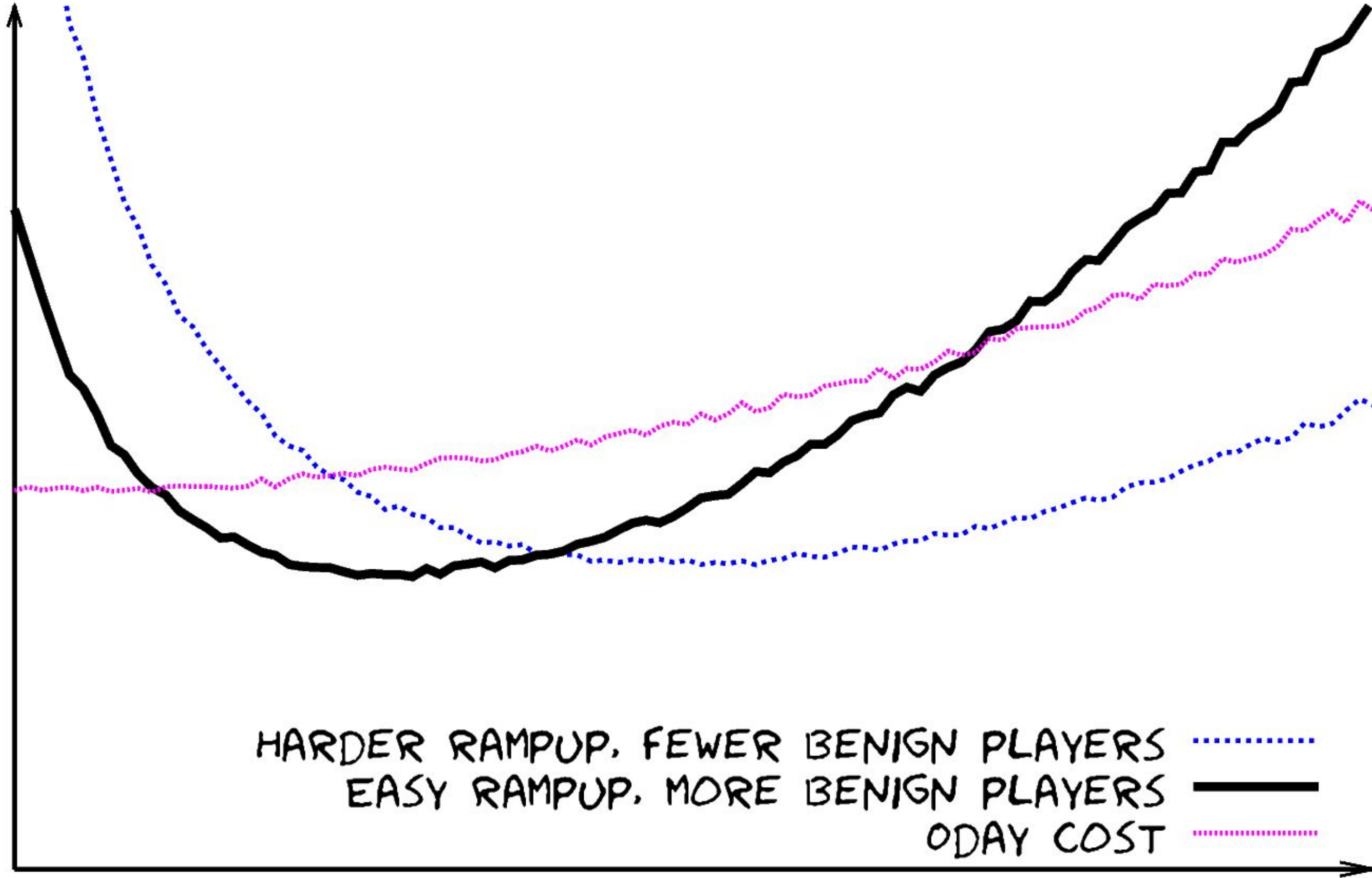
False “security”: Examples

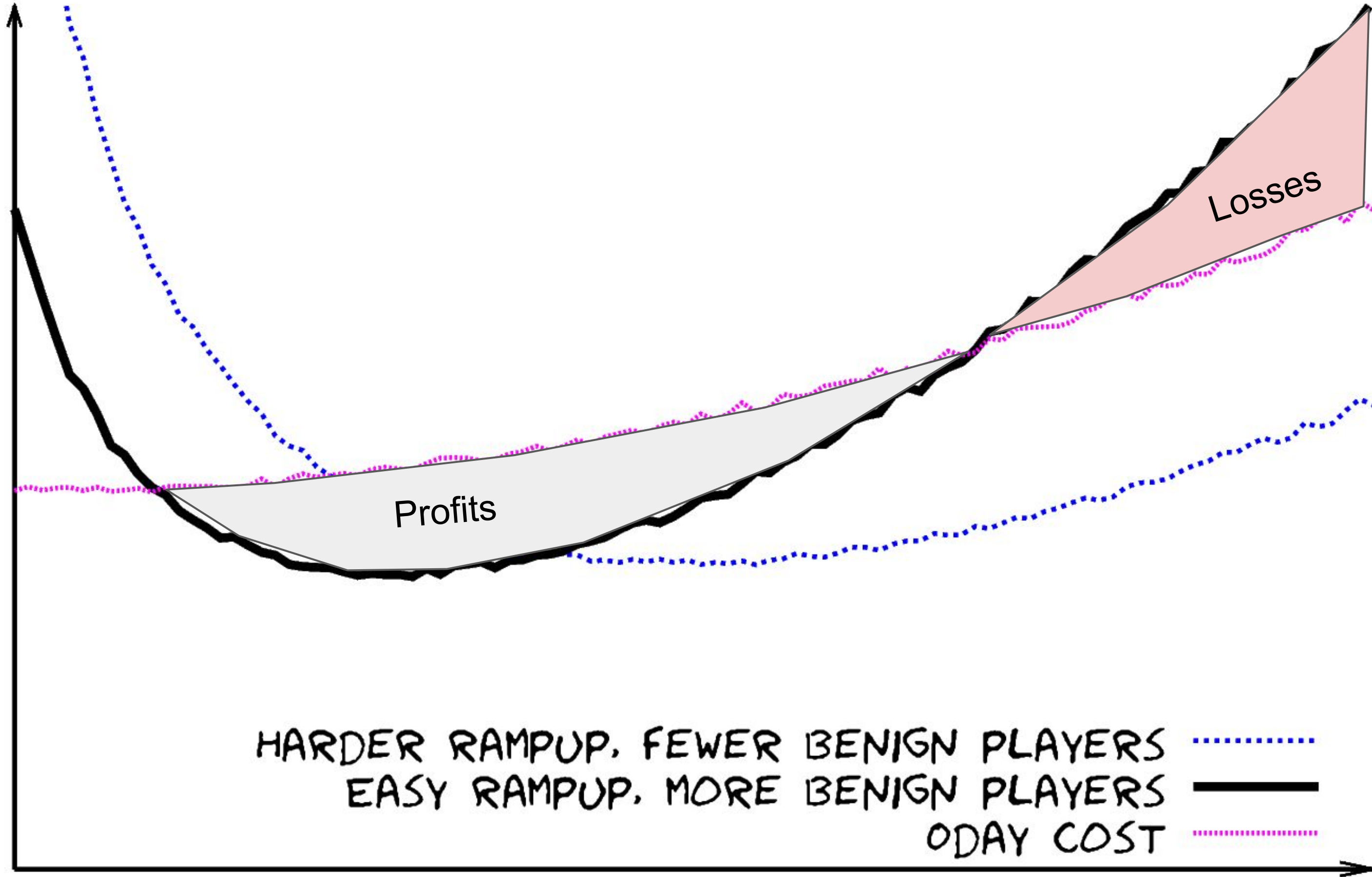
- Lack of debugging on iOS:
    - You need a jailbreak to perform kernel debugging.
    - Permanent tax on defenders: Disclose bug used for performing research, get it killed
    - Attackers leverage their “non-operational” bugs (e.g. not 100% stable, slow) to do debugging

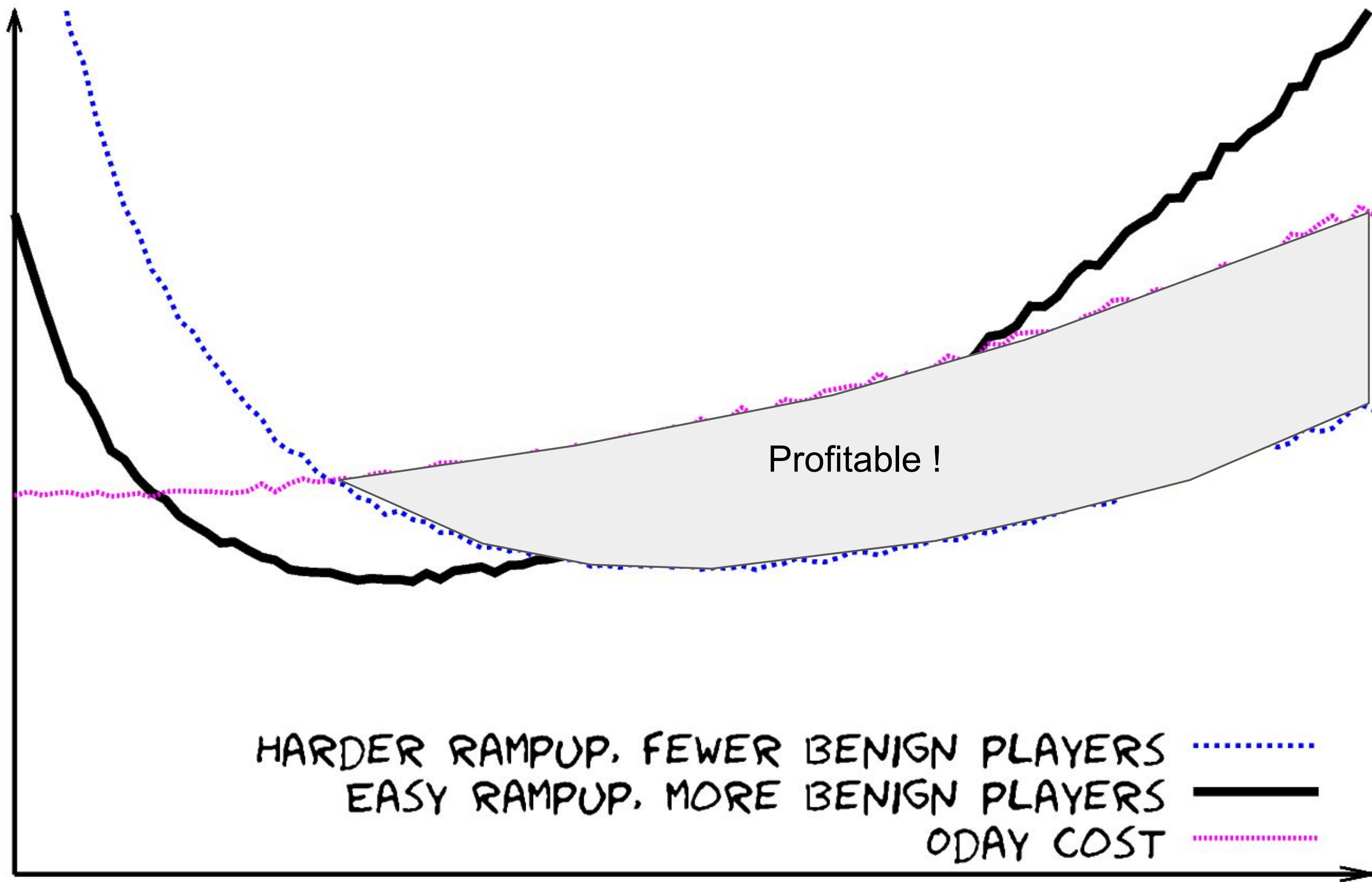
- Privileged processes in Windows:
    - Users cannot usefully debug a privileged process
    - Undocumented hack to disable it exists
    - Only net effect: Prevent benign researchers from doing their work

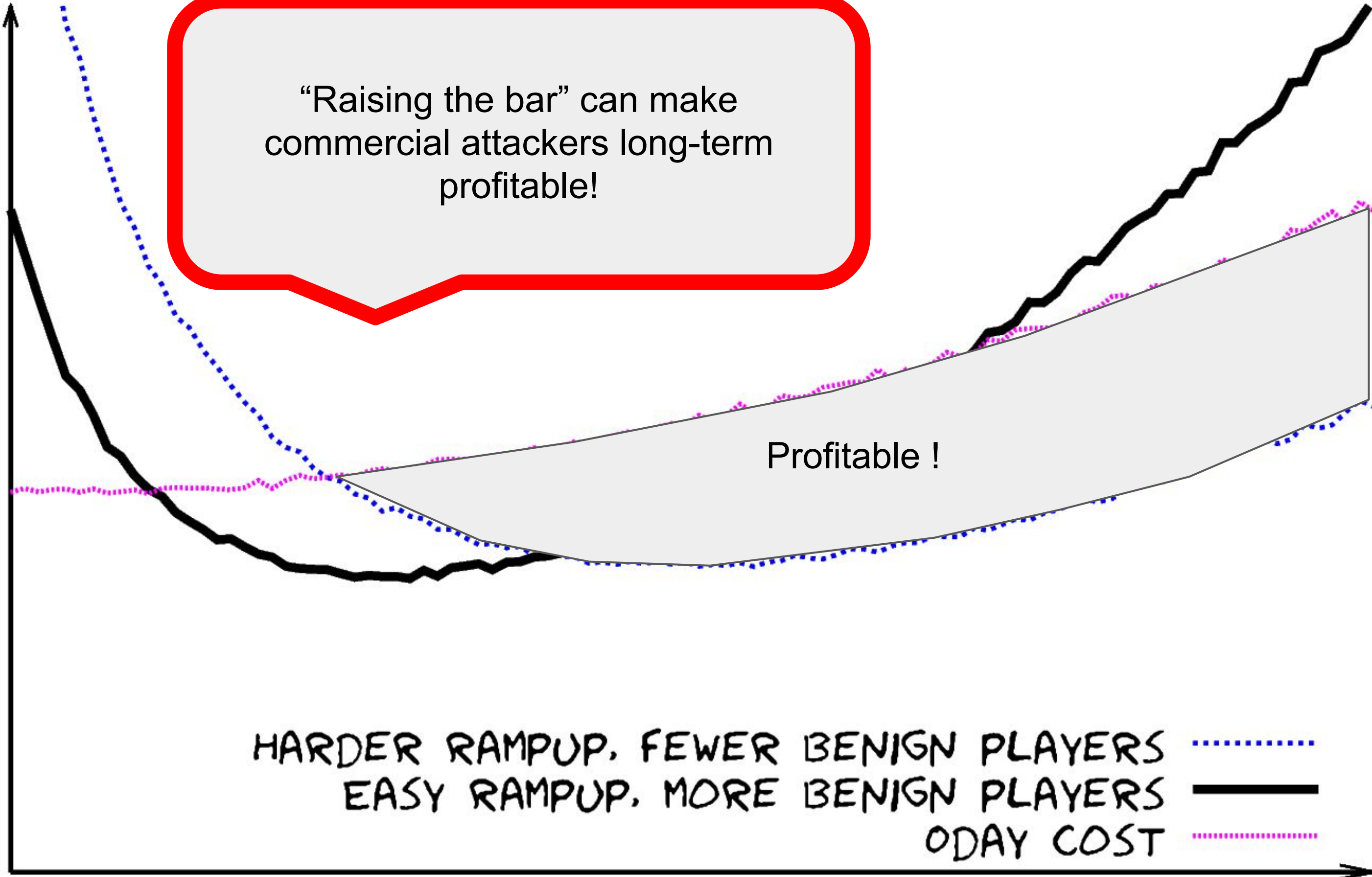
- Disabling JTAG when shipping routers and other devices
    - Heard this given as “advice” to vendors by security people
    - Will not deter serious attackers. Will deter the benign folks. Only sensible if physical access in threat model.


EFFECT OF HARDER RAMP-UP
BUG SCARCITY
INCREASES COSTS
RAMP-UP CHEAPER
HARDER RAMPUP, FEWER BENIGN PLAYERS
EASY RAMPUP, MORE BENIGN PLAYERS


EFFECT OF HARDER RAMP-UP
HARDER RAMPUP, FEWER BENIGN PLAYERS
EASY RAMPUP, MORE BENIGN PLAYERS
0DAY COST


EFFECT OF HARDER RAMP-UP
Losses
Profits
HARDER RAMPUP, FEWER BENIGN PLAYERS
EASY RAMPUP, MORE BENIGN PLAYERS
0DAY COST


EFFECT OF HARDER RAMP-UP
Profitable !
HARDER RAMPUP, FEWER BENIGN PLAYERS
EASY RAMPUP, MORE BENIGN PLAYERS
0DAY COST

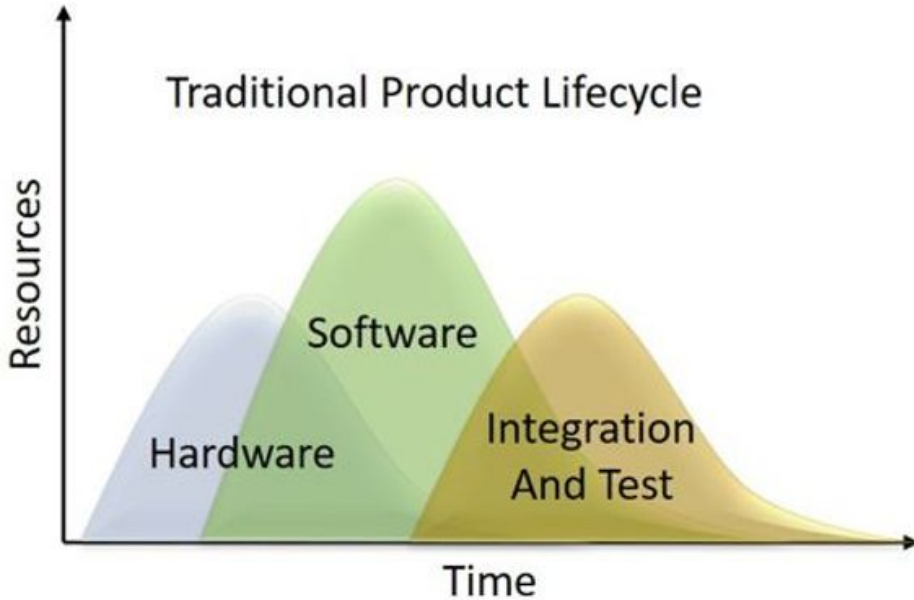

EFFECT OF HARDER RAMP-UP
“Raising the bar” can make commercial attackers long-term profitable!
Profitable !
HARDER RAMPUP, FEWER BENIGN PLAYERS
EASY RAMPUP, MORE BENIGN PLAYERS
0DAY COST

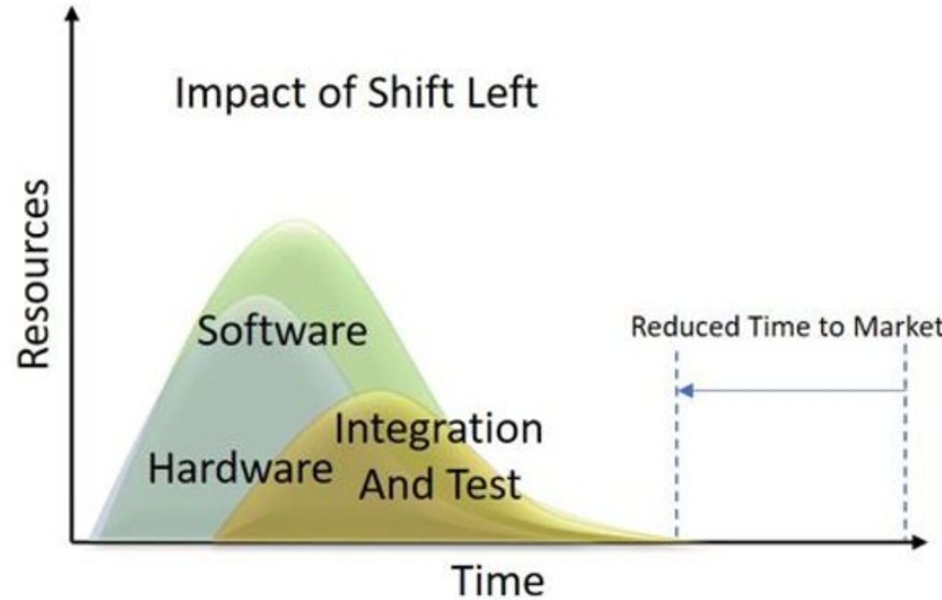
Debuggability of platforms

- Only platform where debugging is significantly better today than in 2007 is Linux.
- All other platforms have gotten harder to debug, harder to introspect etc.
- These “security” measures have become like DRM: Primarily an inconvenience to the good guys.
- NET LOSS FOR OVERALL SECURITY UNDER ANY REASONABLE SET OF ASSUMPTIONS

(Not the fault of the RE community.)

“Left shift”: Compressing development stages





Net effect of “left shift”

- Device vendors do not have reliable methods of debugging
- Device vendors can barely ship a working device, much less introspect & instrument
- If the organisation that has all the spec on their desk cannot do it, how would RE’s go about it?

(Not the fault of the RE community.)

Economic climate sucks “air” out of RE tools

- Internet giants are hugely profitable, per-engineer:
  - Google: Approx. 1.3m$ in revenue per employee
  - Apple: Approx 1.85m$
  - Facebook: 1.62m$
- Extremely hard to compete, if you have to compete with them
- Total addressable market for RE tools: Somewhere between 20m$ and 100m$?
- That would mean a total of 12-60 engineering positions worldwide for RE tooling
- National security & fragmentation makes things more complicated.

Difficulties of the RE tools market

- zynamics considered raising VC at one point.
- Working through the business plan, it became clear that our proposal was close to:
  - “Let’s solve a bunch of really hard CS problems, for which we will need top-notch engineering talent, and then we will sell the results at 1000$ per pop to about 1000 customers worldwide.
- Not the greatest pitch. There is a good reason why VCs ask for total addressable market.
- I am not saying RE companies cannot be profitable and healthy. I am saying that the technical sophistication required to build an RE company is not necessarily in healthy proportion to the business opportunity.

(Somewhat the fault of the RE community.)

Most RE’s are not good developers

- If you want proper tooling, you need both RE and development expertise:
  - An RE that can effectively communicate practical needs
  - A developer that can build things to address these needs
- Easiest if one person can do both. Extremely difficult to find.
- RE’s tend to build throwaway scripts, only rarely invest in proper infrastructure work.
- Even “better” codebases are littered with “paper deadline is tomorrow” hacks.
- Cooperation between RE practitioners and tooling development is rare, and needs to be cherished.

Some RE's derive pride from poor tooling

- Some undercurrents in our community are actively harmful
- We sometimes applaud people for pushing through in spite of poor tooling
- Yes, it is impressive to see someone dig a 100-metre-trench using chopsticks. I was fine doing that as a teenager, but in my late 30s, I want a shovel, by age 50's, a bulldozer.
- Strong culture of "get the job done" vs. "invest in future". Understandable given the economic incentives (consulting gigs etc.), but is this long-term sustainable?

Other blind spots

- Unreasonable fascination with single-core Python scripts and dynamically typed languages
- Memory efficiency be damned (but real-world problems need to scale to 1m functions)
- What is multicore computing?
- What is distributed computing?
- What is stringent unit or integration testing?

(Fault of the RE community.)

Way too many frameworks, duplication

- We are worse than the Javascript community when it comes to frameworks: IDA/HexRays. BAP. PaiMei. BinNavi. Radare. BinaryNinja. Miasm. Angr. DynInst. Pharos. Rev.ng. Insight. BARF. BINCOA. **And there are even more.**

- Each framework implements a subset of the functionality that is needed. Few even implement the basics right.

- Data exchange between tools is painful to impossible.

- Nobody can tell you what tool implements what functionality, and to what degree of reliability.

- It seems everybody looks at the existing frameworks, decides they don’t work, and builds a new one.
- Extremely tempting.
- Funny for a community that should read code: We never do, instead rewrite ambitiously, then abandon halfway through.

6. Paperware

(partial fault of the RE community, partial fault of the academic community, fault of the incentive system)

Tools are written for a paper / presentation

- A significant chunk of our tools are written with the purpose of either publishing a paper or speaking at a con.
- The tool usually works on precisely the examples in the talk.
- Sometimes no working implementation (even in binary form) is published.
- When the implementation is published, it often fails to generalize to random unseen binaries.
- Only tools that get to be “mature” are those that are used by people on an ongoing basis

Tools are written for a paper / presentation

- Academic incentive structure punishes those that write honestly about what a tool can not do.
- Many academics never use their research on real-world problems to validate. Some practitioners do not, either!
- **Misperception about what problems are solved and what problems are not solved abound.**
- Any working implementation of VSA-with-recency? Any working implementation of input crafting from patches? Even CFG recovery is empirically shaky with most public tools!
- Researchers that realize this and tackle unsolved problems get rejected with “but paper XYZ claimed to have done this already”.

Tools are written for a paper / presentation

- Remember Feynman:

> "For a successful technology, reality must take precedence over public relations, for nature cannot be fooled."

- We would do good to remember this when writing papers, and when reviewing papers.

Non-reproducibility of results

- Papers should make available artifacts that allow reproducibility. VMs, Docker images, statically linked binaries, or whatever.

- Usual excuses why this does not happen are not good: “Perhaps we want to commercialize”.
    - Publish binary-only implementation
    - Publish source code under very restrictive license

- Current state: I no longer know what is true and what is hyperbole when reading papers. Hurts people that honestly state limitations, hurts the entire research field long-run.

(partial fault of the RE community)

- Given that no tool does it all, what is a reasonable collection of tools that interact well?
- Disassembly, coverage, dynamic instrumentation, VSA, etc.
- For most problems there are several implementations, but which one works?
- Other open-source communities with complex stacks of specialized tools develop “distributions”:
    - Linux: Debian, Redhat, Kali etc.
    - LaTeX: MikTeX, TeTeX etc.
    - Data Science: Anaconda etc.
- Everybody on his own - try a tool, fail, try the next, add it to the quiver.

(partial fault of the RE community)

We are too focused on one layer (code)

- Reverse engineering is the process of recovering engineering process intermediate artifacts
- Engineering is forward: Requirements, specs, architecture, code, compiling, linking
- Very few RE tools make effort to recover anything above code
- C++ class hierarchies, larger-scale-modules, etc. -- all of this needs to be recovered
- Generating C/C++ pseudocode is not the end!

End of yelling.

Apologies for the long rant.

I do not mean harm.

One request to platform vendors.

Four requests for us.

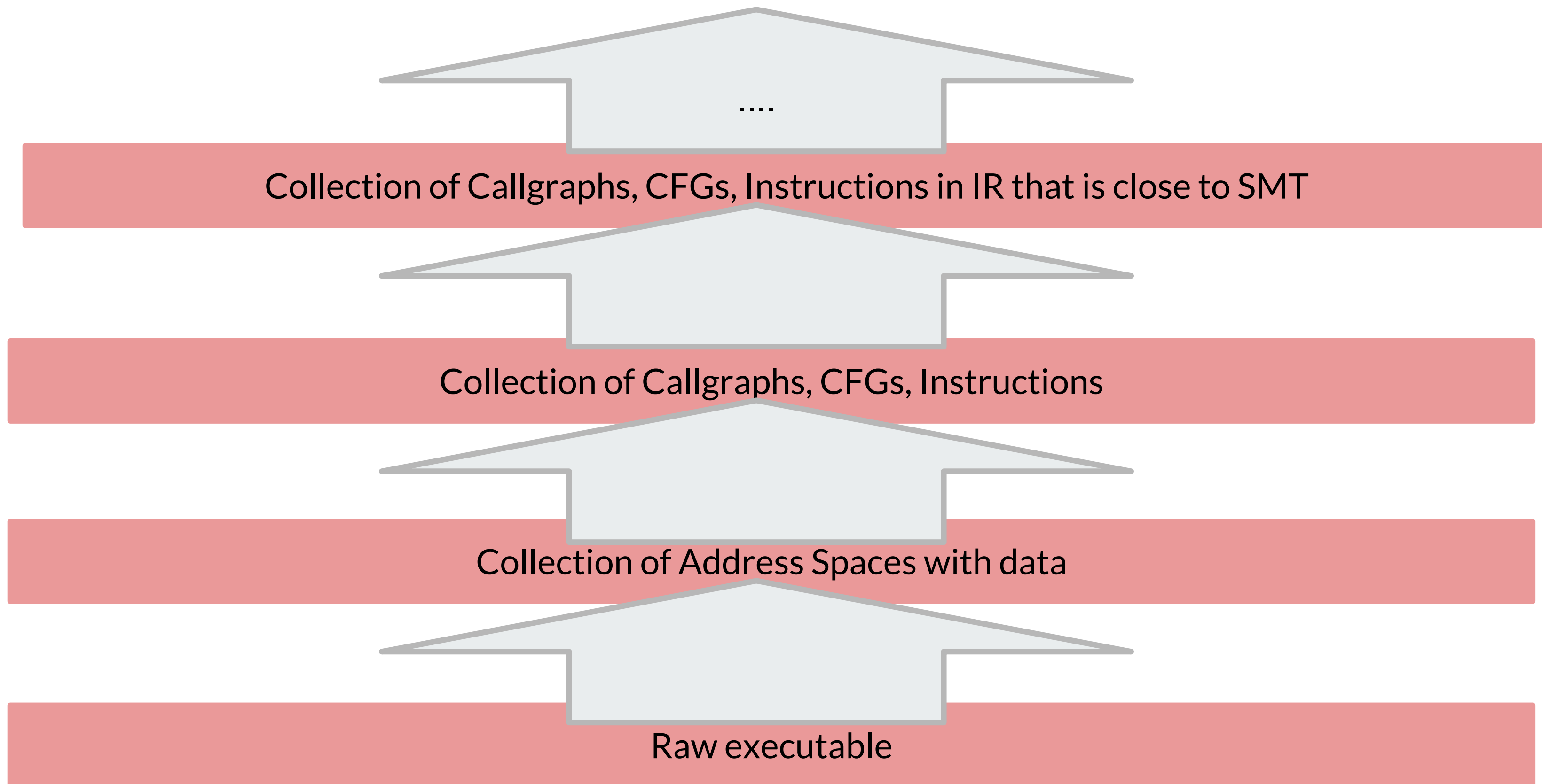
Request to the platform vendors: **Please stop the nonsense that people that do not have a privesc-bug cannot meaningfully debug / inspect.**

It's super harmful. And the perceived security gain is not real. Thank you.

Facilitate interoperability, testability, reproducibility.

If we can’t agree on the right tools ...

- Let’s at least try to make them swappable and comparable
- If we define clean data interchange standards between abstraction layers, we can plug & swap
- Will require effort
- Layers defined as JSON or ProtoBuf or whatever else


....
Collection of Callgraphs, CFGs, Instructions in IR that is close to SMT
Collection of Callgraphs, CFGs, Instructions
Collection of Address Spaces with data
Raw executable

- We can swap out individual tools when they do not work
- We can compare performance of tools
- We can compare impact of low-level improvements cascading up the stack
- We can do better real-world testing
- Issues such as porting BinDiff between IDA and BinaryNinja etc. would go away

Download one set of packages that “just work”.

A RE-tools distribution

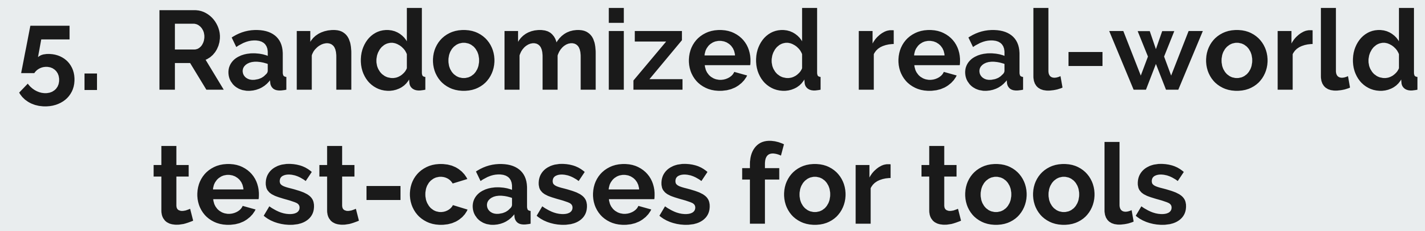
- Simple installation & build for a full suite of tools.
- Ships with QEMU (with deterministic replay) for full-system time-travel debugging
- Coverage analysis, dynamic resolution of indirect calls
- Disassembly
- Type recovery etc.

Probably requires (2) to work

Do one thing. Do it well. Do it predictably, and reliably, and reproducibly.

Do one thing well

- Stop the framework-itis
- Build libraries that take clean input, provide clean output
- Make it easy to interoperate with other tools
- Try to remain agnostic to who the previous layer and next layer in the abstraction is



If analysts cannot rely on the tool working on real code, it may as well not exist.

Reliably functioning tools

- Few of our tools work reliably.
- IDA and Hex-Rays are solid, dependable, reliable. For productive use, P(works | new_binary) needs to be high, more than 9 out of 10. Unreliability is deadly.
- Extensive real-world test-cases are needed.
- Automated runs against these test-cases are needed.
- When new compilers, platforms, or OS versions arise, the test-cases need to be updated.

The coming storm in computing.

The coming changes

- Computing will change more in the next 20 years than it did in the last 20 years
- Biggest changes are:
  - Move to cloud
  - Heterogeneous computing prompted by end of Moore
  - The race “down the stack”
- RE community will have to adapt as well.

Impact of cloud

- Datacenter-as-a-computer
- OS needed for that computer
- Back to the 60’s / 70’s:
    - Only a handful of computers globally
    - Custom OS
    - “Closed binary”, not only “closed source”

Impact of heterogeneous compute

- Moore’s Law & single-core scaling gave us ABI stability (40 years x86 with minor changes)
- This was good for RE - tooling could slowly catch up, RE tools are always behind dev tools
- Future are heterogeneous computing systems:

CPU | GPU | Linear Algebra / ML engine

Wifi Baseband | GSM Baseband | FPGA (?) | ???

Impact of heterogeneous compute

- Hardware / Software co-design is increasing
- More components will be poorly documented or undocumented
- How does one disassemble the code sent to a GPU?

Visible race down the stack

- 1997-2017, RE’s could mostly do their job without reading datasheets
- Increasingly more of a job requirement
- Datasheets for obscure CPUs, difficult-to-access documentation
- FPGA expertise to dump firmwares etc. increasingly common, and increasingly commonly needed
- “Real” attackers can probably steal the internal documentation … (defender dilemma again -- “real” attackers have more information available than defenders)

Summary

- Coming technological change will have massive impact on computing
- Extremely hard to forecast where things will lead
- RE community is always a small boat in the greater currents of the industry
- Always undertooled, always underfunded, and always one generation behind the dev tools
- Need to use our resources wisely

Several scenarios are possible. Questions?