PRESENTATION BY MARK DOWD (@MDOWD)

RULES TO HACK BY

“HOW DO YOU ACTUALLY FIND BUGS?”

- EVERYONE

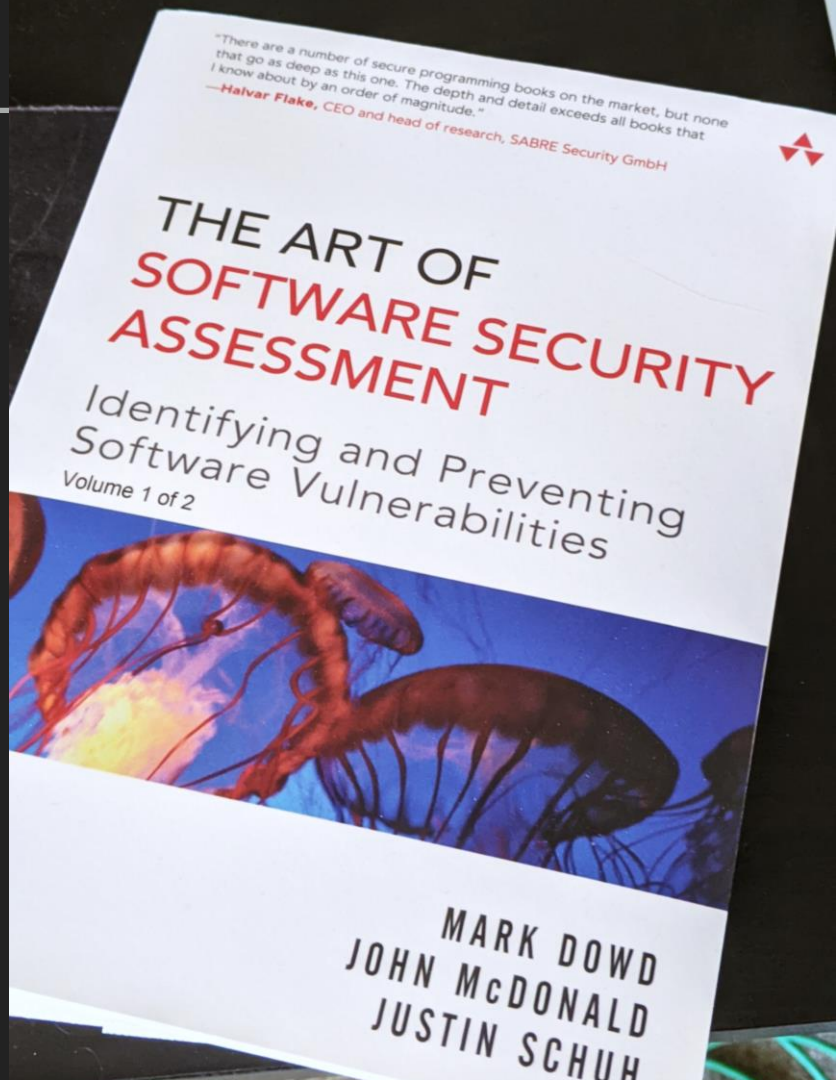
Process is seldom discussed

Presentations usually show "here is a cool bug I found,
and an even more cool exploit“

Doesn't mention all the other bits

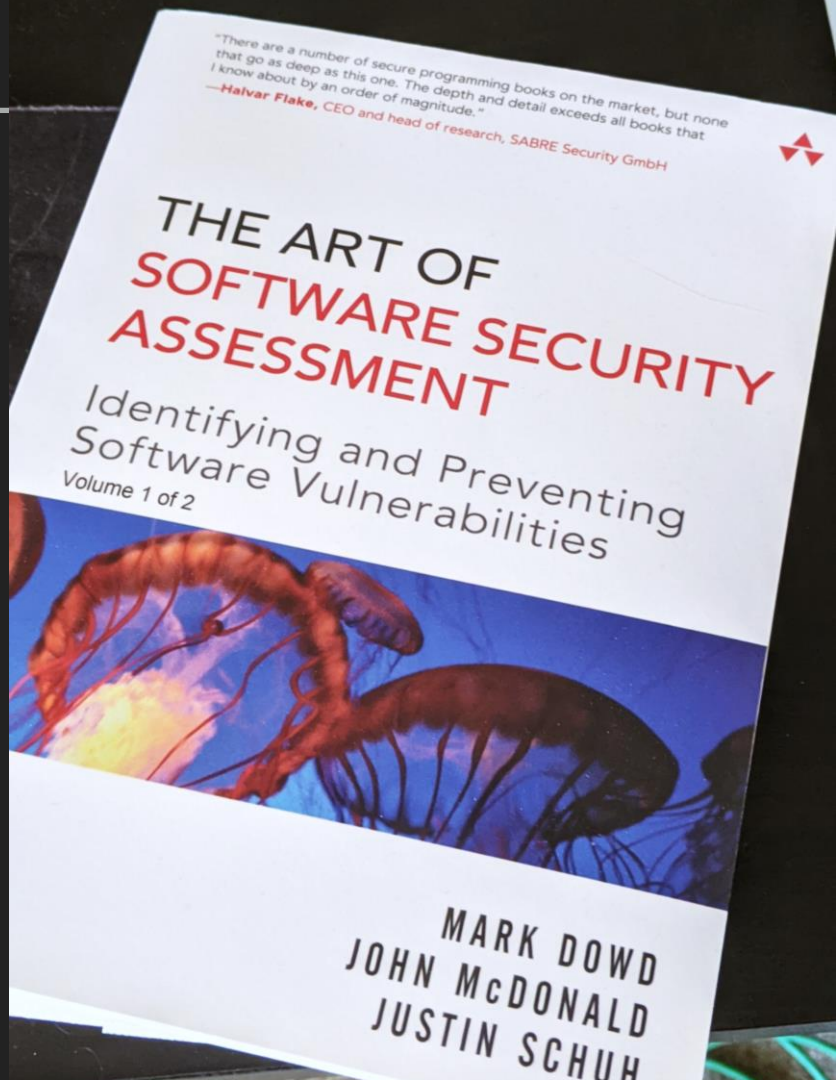
SCOPE

- TAOSSA described mostly technical details
  - Limited methodology
- These are my rules
- Your style and mileage may vary



TAKEAWAYS

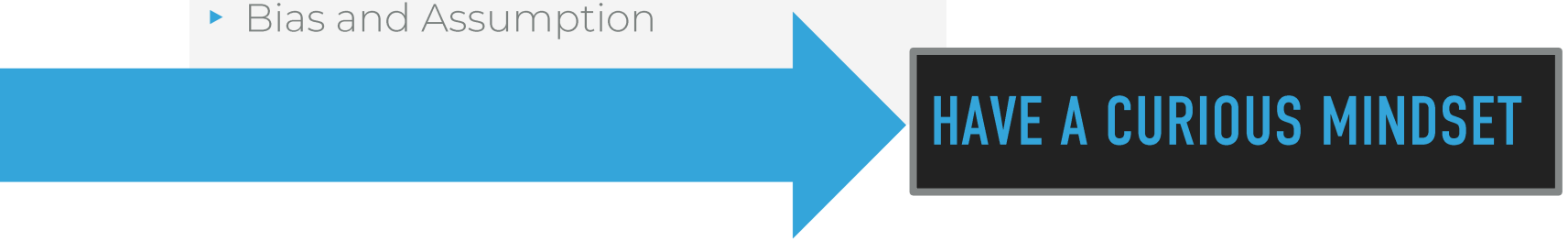
- Non-practitioners: Exposure to what vulnerability research is like
- Juniors: Actionable advice
- Seniors: Compare and contrast techniques, and encourage sharing of their own experiences



MINDSET

MINDSET

- Temperament
- Dealing with Failure
- Moving On
- Motivation
- Confidence
- Bias and Assumption



TEMPERAMENT

- Defining traits of a successful security researcher:
  - Curiosity
  - Detail-oriented
  - Ability to deal with failure and continual evidence that you're wrong
- This doesn't have to be as bleak as it sounds
  - On the contrary, it can be quite rewarding!

There are few jobs where it is more apparent that your understanding of technology is wrong

LEARN HOW TO DEAL WITH FAILURE

- Two projects (can be unrelated, or different parts of the same)
  - Learn to recognize when you have hit a wall and have become unproductive
  - Switch to your secondary project
- Consider having a development project as your secondary project
  - Do an achievable, measurable task
  - Regain a sense of achievement

MOVING ON - KNOWING WHEN TO QUIT

- Perseverance is important, but at some point you are just wasting time
  - Drains motivation and confidence
- Sunk cost fallacy makes it hard to quit
- Why your project was NOT a waste of time
  - You will return to it in future
  - You have paid the startup cost of understanding this code base

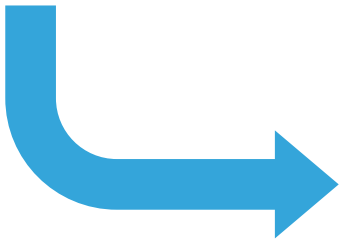
MOTIVATION – REMAINING ENGAGED

The more you're curious about how a technology works,
or how an algorithm achieves its goal,
the less monotonous code review is

MOTIVATION – BUG PATCHING

Bugs being patched is frustrating…

- … but they are evidence that you were on the right track!

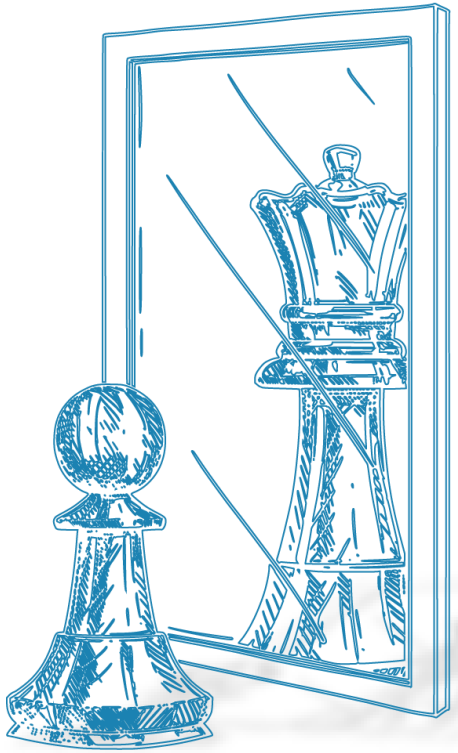


Try to see patches as a double-edge sword

- Inspiration for ideas for variations or new patterns

CONFIDENCE

- Research is a daunting field to enter
- “This is impossible!”
- Some security researchers you respect had the same self-doubt coming in, and have recurrences from time to time



* Sanity not guaranteed

MY STAGES OF GRIEF

There's no way I'm ever going to find anything in this

.. Oh!

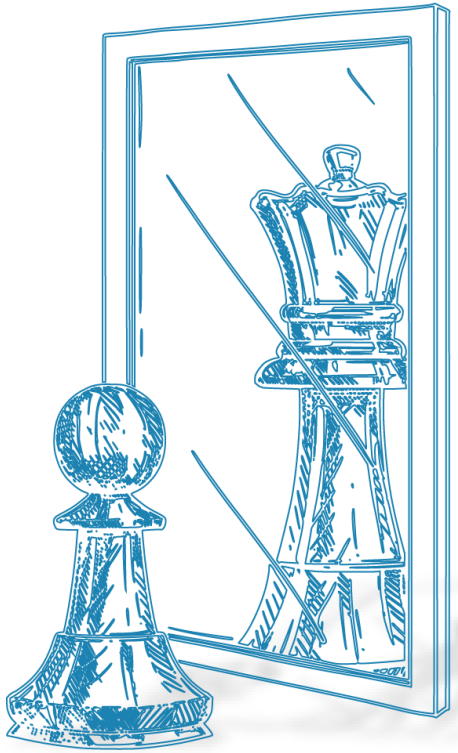
Do I even know C?

Interesting design choices...

CONFIDENCE

- Remember, there are no penalties for trying*
- If you look far enough back at someone's achievements, you will probably find something you are capable of now, or is within your grasp

GROWTH MINDSET: "I CAN'T DO THAT... YET"



* Sanity not guaranteed

BIAS AND ASSUMPTIONS

Common code-reviewer biases

- Everyone has looked at this already ("many eyes make all bugs shallow")
- Even if I found something, it will be unexploitable (server-side)
- The X attack surface is not interesting now (eg. media parsing in browsers)
- There are no more bugs in this
- The protocol doesn't allow you to do X

Secret to my early career: most major applications were rarely audited in real depth!

BIAS AND ASSUMPTIONS

Common code-reviewer biases

- Everyone has looked at this already ("many eyes make all bugs shallow")
- Even if I found something, it will be unexploitable (server-side)
- The X attack surface is not interesting now (eg. media parsing in browsers)
- There are no more bugs in this
- The protocol doesn't allow you to do X

Example: array.concat() and array.join() are the most vulnerable functions of all time

CVE-2017-5030, CVE-2016-1646, CVE-2021-21225, CVE-2017-0134, CVE-2017-8634, CVE-2014-3176, CVE-2016-2464, CVE-2017-11903, CVE-2016-7189, CVE-2012-0464, CVE-2016-1857

BIAS AND ASSUMPTIONS

Common code-reviewer biases

- Everyone has looked at this already ("many eyes make all bugs shallow")
- Even if I found something, it will be unexploitable (server-side)
- The X attack surface is not interesting now (eg. media parsing in browsers)
- There are no more bugs in this
- The protocol doesn't allow you to do X

Sometimes I don't read documentation about the protocol until halfway through specifically to avoid bias on it's potential vulnerabilities

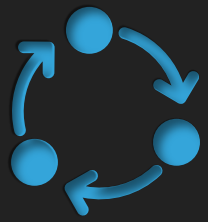
RULES

Learn How to Deal with Failure

Adjust and Adapt

No Practice is Wasted

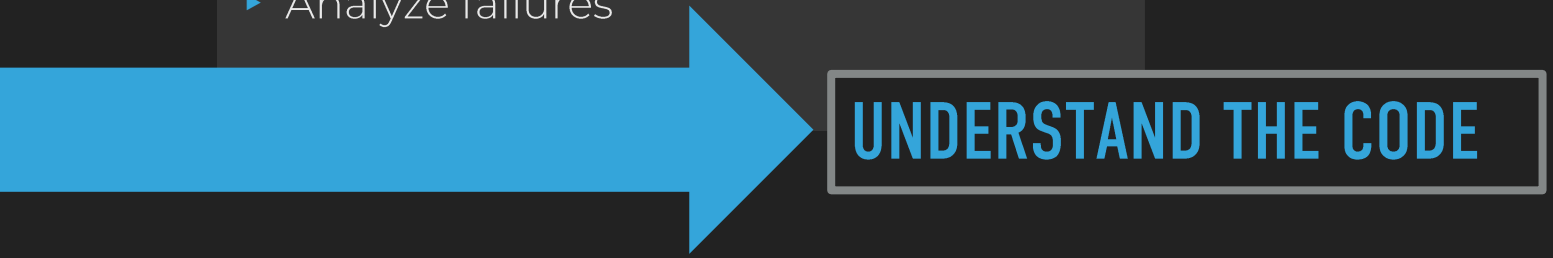
Question Assumptions



AUDITING PROCESS

AUDITING PROCESS

- Understanding the code
- Documenting your findings
- Identifying bias
- Tooling
- Revisit code base
- Analyze failures



ATTEMPT TO UNDERSTAND THE CODE

- A lot of people try to short-circuit this process
  - Reliance on tools
  - Fuzzers/static analyzers/taxonomies are a guide, not the whole process
- Many of today's vulnerabilities are complex, and require in-depth understanding of the codebase
  - Small idiosyncrasies
  - Quirky APIs that are prone to misuse
  - Minor inconsistencies

ADVANTAGE IS IN THE ABILITY TO UNDERSTAND CONTEXT

ATTEMPT TO UNDERSTAND THE CODE

Example 1: Natalie Silvanovich's Signal (state machine) bug

- Requires understanding the semantics of a state machine

Example 2: Every mitigation bypass

- Mitigation bypasses are contextualized vulnerabilities in integrity-validation algorithms

ATTEMPT TO UNDERSTAND THE CODE

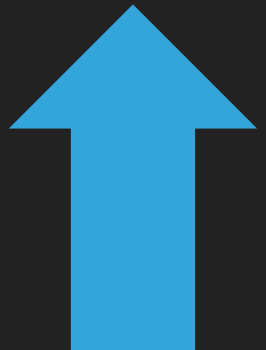
- You tend to observe inconsistencies that don't amount to anything individually, but are invaluable when combined
  - Example: mach_msg UAF by Ian Beer CVE-2021-30949

  (https://bugs.chromium.org/p/project-zero/issues/detail?id=2232)

- Exploitation (mechanics of weird machines), other interesting effects
  - Example: Flash

  (https://www.cs.utexas.edu/~shmat/courses/cs6431/dowd.pdf)

THE MORE YOU UNDERSTAND ABOUT HOW THE PROGRAM WORKS, THE BETTER EQUIPPED YOU ARE TO FIND BUGS (AND EXPLOIT THEM)

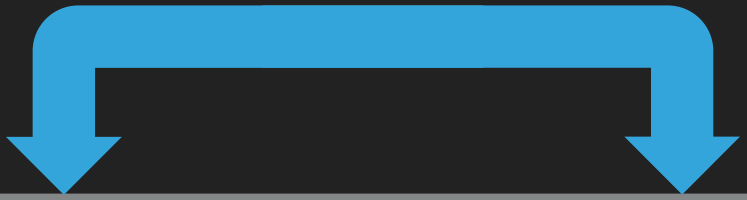


UNDERSTAND THE CODE: HOW I DO IT

THE BEST WAY TO UNDERSTAND HOW SOMETHING WORKS IS TO EXPLAIN IT TO SOMEONE ELSE

Explaining signal handling resulted in me finding a remote Sendmail vulnerability!

WHAT I'M LOOKING FOR



SOFTWARE RISK = AVAILABLE ATTACK SURFACE * COMPLEXITY

Either of these are the bug hunter's best friend

WHAT I'M LOOKING FOR – ATTACK SURFACE

- Attack surface can be indirect
  - Even mitigations are attack surface
- Often your initial perception of attack surface is naïve
  - Hidden/non-obvious attack surfaces are the best

The attack surface is the vulnerability – finding a bug there is just a detail

WHAT I'M LOOKING FOR - COMPLEXITY

- Complexity is plentiful:
  - Feature-driven (thanks W3C!!)
  - Legacy support
  - Often avoidable: The anomaly of cheap complexity

(http://rule11.tech/papers/2018-complexitysecuritysec-dullien.pdf)

EXAMPLE: LOG4SHELL (CVE-2021-44228)

- Over-powered component for a relatively simple job
- Ostensibly logs error messages
- Bonus formatting functionality: download and run code
- Most consumers didn't need this

BORROWING IDEAS

- Bugtrackers / Diffs
  - Can show where a bug is, but not what it is and why it's a security bug
  - Invaluable source of interesting bugs, why a particular piece of code exists, etc
  - Can inspire new ideas: variants, same bugs in other code bases etc
  - Mean but viable: track commits by error-prone developers
- Comments
  - Describes things I'd never thought of

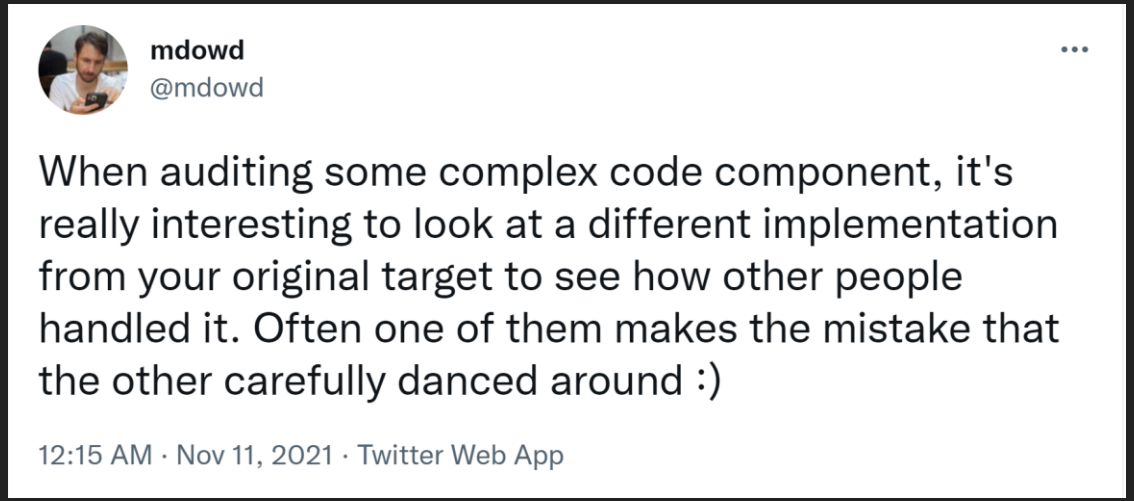
DOCUMENT YOUR FINDINGS

- Get in to the habit of documenting
  - Ideas, bug candidates, idiosyncrasies, data structures, algorithms
- Documenting failed ideas is as important as documenting successful ones!
  - Avoids repeating the idea sometime later
- Long term view: I'm going to revisit this later

I HAVE NOTES ON CODEBASES THAT SPAN A DECADE!

APPLY YOUR KNOWLEDGE BROADLY

- Look at other software attempting to achieve the same task
- If one implementation makes a mistake...



APPLY YOUR KNOWLEDGE BROADLY

- Example: Revisiting Natalie's state maching bugs
  Project Zero: The State of State Machines (googleprojectzero.blogspot.com)
  - Startup cost: understanding WebRTC
  - Payoff: Signal, JioChat, Mocha, Duo, Facebook Messenger

REVISIT CODE BASES AND FAILED BUGS

- Code bases are not static
  - Code rewritten
  - Features added / changed

AND

- Environment is not static

ANALYZE YOUR FAILURES

- If someone succeeds where you didn't, have a look at what they found
- Try to figure out: why did I miss it?
- Is this a one-off or teachable trick/blind spot?
- Can you improve on that?
- Is there a pattern in those failures?

FAILURE IS AN OPPORTUNITY TO LEARN

TOOLING

- Tooling and interactive testing can save a lot of time and energy, build / steal where appropriate
- Mistake: I nearly always avoid doing this

IF YOU DON'T LIKE FUZZING, TEAM UP WITH SOMEONE WHO DOES

Richard Johnson
@richinseattle

World class bug hunters always begrudgingly accept the power of fuzzing. It’s inevitable.

11:50 AM · Dec 16, 2021 · Twitter for iPhone

4 Retweets **1** Quote Tweet **55** Likes

TOOLING

- Careful, it's also a huge rabbit hole
  - Tooling allows people to deal with less ambiguity - writing vs reading
  - It is more comfortable and familiar for most
- The most effective tools are made when a specific new outcome is required
- Continuous feedback to keep you on-task

"IT'S AMAZING HOW MUCH WORK PEOPLE WILL DO TO NOT UNDERSTAND SOMETHING - MARK" – HALVAR

RULES:

Understand the Code
Document Your Analysis
Revisit Code Bases
Analyze Failures

SUMMARY

- Vulnerability research is difficult, but it is a learnable skill
- Dealing with frustration and new information constructively is a key differentiator to success
- Reading code is a skill distinct from writing code, and takes practice
- Say happy birthday to @runasand today