

Developing Secure Software in 2024



Joshua "jduck" Drake
March 21st, 2024
CanSecWest

Agenda

- Introduction
- History Lesson
- Where we are
- Recent Events
- Call to Action
- Conclusions
- Q & A

About me

- Vulnerability research and exploit development career
- Most recently Sr. Security Engineer at Amazon Lab 126
- Bootstrapping a security startup
- Developing strategic Rust implementations

Previously:



amazon
security



Motivations

1. Improve global security maximally through computer science and engineering.
2. Raise awareness to build cross-industry momentum and standardization.
3. Evangelize safer development methods and technologies

Disclaimer

This presentation is my best effort to show the state of the art and share my experiences, thoughts, and opinions.

Please take time to think about the content and formulate your own ideas/response.

Trust and Guarantee

- Trust
 - "belief in reliability, truth, ability or strength of"
 - in this talk, of the security of software humans build
- Guarantee
 - "a formal promise or assurance that certain conditions will be fulfilled"
 - SoK Paper by Marcel Bohme is a must read.

Both of these concepts are crucial to understanding software security.

I. [SoK: Guarantees in Software Security](#)

History Lesson

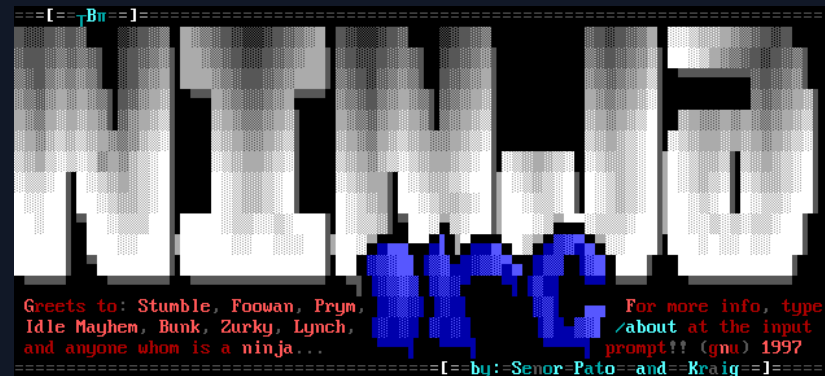
How did we get here?

My Journey

- Mischievous since birth
- Programming since age 11
- Internet user since age 13
- Learned C programming in 1995
- Studied Math + CS in university

My Journey

- Modified client based on on ircII
 - Abandoned! Do not use! (FreeBSD ports??)
- Found format string bugs after reading Teso paper.
- Realized securing software written in C is very hard.
- Decided to pursue security as a career path.



I. Exploiting Format String Vulnerabilities by scut

Microsoft's Example

Microsoft had big security problems in early 2000's.

- Windows was around 96% market share
- Several worms attacking various bugs
- Bill Gates took action
- Led the industry to many improvements

1. Bill Gates sends security memo to customers
2. Celebrating 20 Years of Trustworthy Computing

RCE in Microsoft ATL

CVE-2009-2493

- Research by Ryan Smith
 - Estimated industry cost: **10-100 million**
 - Incorrect characters in source code: **one**
1. [MS09-037: Microsoft Active Template Library \(ATL\) RCE](#)
 2. [MS09-060: Microsoft ATL ActiveX Controls for Microsoft Office RCE](#)

Undefined Behavior

- Intentional ambiguity in the C language specification
 - The C99 specification contains **193** mentions of UB.
 - Do newer standards have more or less?
 - Leaves compiler engineers without requirements
- Birthed many security issues, bugs, and even loss of human life 😞
- Ambiguity is the inverse of guarantee

1. [C99 List of Undefined Behavior](#)
2. [What every programmer should know and fear](#)
3. [Toyota Unintended Acceleration](#)

Assumptions, Reality, and Evolution

The reality is that most humans make assumptions when developing software.

For example, DJB assumed that type sizes wouldn't change in the future.

This eventually led to a security issue in his qmail software when 64-bit started to emerge.

The reality is that most developers make far more assumptions.

I. [A remote code execution vulnerability in qmail \(LWN\)](#)

Where we are

Where does all that lead?

We Are In a Bad Placetm

Problems:

- People problems...
- Security industry fails
- Government fails
- Software vendor fails

Potential solutions coming later in the talk

NOTE: Please forgive the impending rant...

People Problems

- Perceived staffing crisis
 - Not enough qualified practitioners
- Insufficient developer and security education
 - Many developers don't know security
 - Many security people don't know development

We have a lot to learn from each other

Open your minds and seek to learn!

Security Industry Fails

- Security products shipping with trivial security bugs
- Lack of scientific approach / rigor
 - Tons of private reporting and silent fixes
 - Assessments time-boxed and point-in-time
 - Review scope is too large to deliver quality
- Unwilling to jump in to fix the code

I bet you can think of other things too!

NOTE: We should celebrate exceptions and seek to learn from them.

Government Fails

Lack of regulation on the software industry has led to foreseeable failures.

Zero-day markets removes many talented persons from the pool of qualified security practitioners.

Software Vendor Fails

- Market incentives undermine security
 - Good, Fast, Cheap -- Pick two.
- No Secure SDLC?
 - Microsoft SDL was born 20 years ago.
- C and C++ widely used, but error prone
- Ignore compiler warnings
- Ignore static analysis tool output
- Testing severely lacking
 - Fuzz much??

NOTE: We should celebrate exceptions and seek to learn from them.

Recent Events

What's going on?

All About Software Safety

People are getting serious about “safety” in coding.

- Temporal safety
- Spatial safety (bounds)
- Type safety
- Definite initialization
- Thread safety

Check out Saar Amar (Dec 2022) and David Teller (Feb 2023) writings.

1. [Survey of security mitigations and architectures, December 2022](#)
2. [Intro to Memory Unsafety for VPs](#)
3. [About Safety, Security and yes, C++ and Rust](#)

Government Activism

Governments turning Memory Safety into a MEME!

Several papers and Federal Register RFIs (and response comments on regulations.gov)

- 2022 - NSA Paper on Memory Safety
- 2023 - National Cybersecurity Strategy + RFI
- 2023 - Secure by Design (CISA + 13 other countries)
- 2024 - Back to the Building Blocks (ONCD) [[LINK](#)]
 - Results from 2023 RFI

1. [NSA Paper on Memory Safety, Nov 2022](#)
2. [National Cybersecurity Strategy \(107 RFI comments\)](#)
3. [Secure By Design \(83 RFI comments\)](#)

Industry Activity

Even before the recent federal push, several vendors are leading by example.

- Google - Rust in Android, donated \$1 mil
- Rust at Microsoft - win32k
- Apple investing in Swift in the kernel
- Rust in the Linux Kernel

Also, many startups are making smart choices.

Hardware: MTE

Memory Tagging Extension

64-bit ARM spatial/temporal safety enforcement

- Unfortunately, probabilistic (only N-tags)

Present in Google Pixel 8 and 8 Pro and hopefully others

- Disabled by default in Pixel OS, but easy to enable
- Enabled by default in [GrapheneOS](#)
- Exposes poor software quality!

1. [Arm Memory Tagging Extension AOSP Docs](#)
2. [Collection of MTE related crashes](#)

Hardware: CHERI

Capability Hardware Enhanced RISC Instructions (CHERI)

- "joint research project of SRI International and the University of Cambridge to revisit fundamental design choices in hardware and software to dramatically improve system security"

Capabilities are bounded pointers / cannot be forged

Expect silicon in fall 2024 (fingers crossed)

For now: Morello project

1. <https://www.cl.cam.ac.uk/research/security/ctsrtd/cheri/>

2. <https://www.morello-project.org/>

OpenSSF

Open Source Security Foundation

- "seeks to make it easier to sustainably secure the development, maintenance, and consumption of the open source software (OSS) we all depend on."

Funded by the Linux Foundation

Many efforts: Training, Scorecard, Alpha-Omega, Memory Safety WG, etc

1. <https://openssf.org/>

C++ Community

The C++ community is taking safety seriously, but unclear what will be standardized and when.

Maybe C++26?

- Bjarne Stroustrup - C++ Safety Profiles etc
- Herb Sutter - cppfront, ISO C++ Chair, blogging
- Sean Baxter (@seanbax) - "Memory-safe C++"

1. [Bjarne's Profiles repo on GitHub](#)
2. [Herb's CPPFront repo on GitHub](#)
3. [C++ safety, in context by Herb](#)

Defining Undefined Behavior

Shafik Yaghmour (Intel) is working to drive change around "undefined behavior".

<https://wg21.link/P1705>

I emailed him with encouragement, but no response :-/

Document behavior in the spec based on empirical observations

1. P1705R1: Enumerating Core Undefined Behavior in C++
2. Why do we need a Undefined Behavior Annex for the C++ standard?

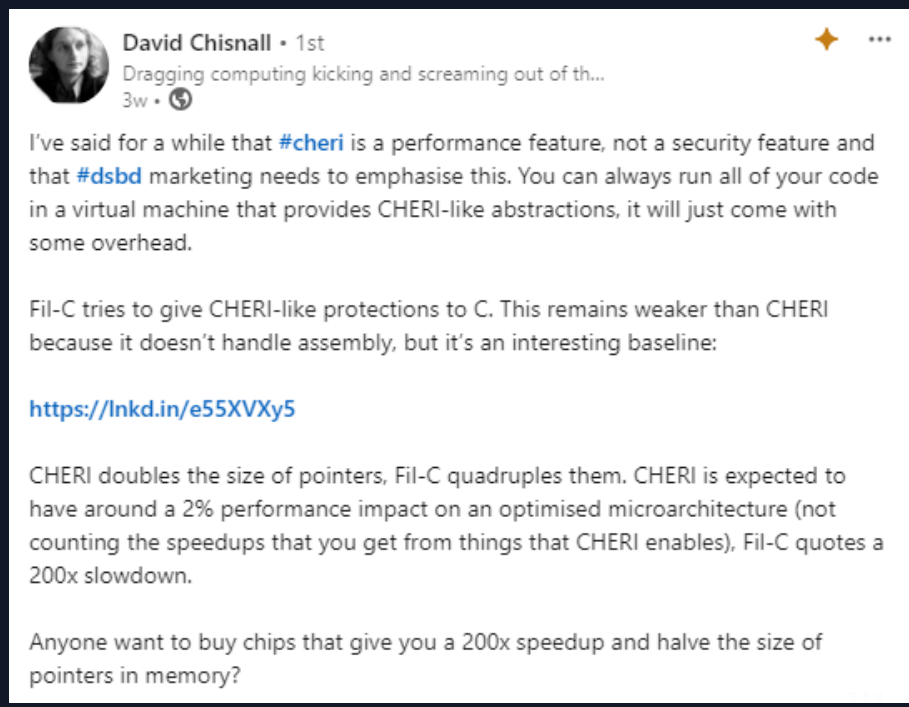
Other Interesting Stuff - Fil-C

Fil-C is a modified compiler to produce memory safe programs from C source code.

”The Fil-C Manifesto: Garbage In, Memory Safety Out!”

by Filip Jerzy Pizlo (@filpizlo)

I. Fil-C on GitHub



David Chisnall • 1st
Dragging computing kicking and screaming out of th...
3w • 🌐

I've said for a while that #cheri is a performance feature, not a security feature and that #dsbd marketing needs to emphasise this. You can always run all of your code in a virtual machine that provides CHERI-like abstractions, it will just come with some overhead.

Fil-C tries to give CHERI-like protections to C. This remains weaker than CHERI because it doesn't handle assembly, but it's an interesting baseline:

<https://lnkd.in/e55XVXy5>

CHERI doubles the size of pointers, Fil-C quadruples them. CHERI is expected to have around a 2% performance impact on an optimised microarchitecture (not counting the speedups that you get from things that CHERI enables), Fil-C quotes a 200x slowdown.

Anyone want to buy chips that give you a 200x speedup and halve the size of pointers in memory?

Call to Action

What can we do?

Hot Buzzwords

You should know what these mean:

- Shift left
 - Move testing earlier in the process
 - Developers lose context over time -- tighter feedback loops are more efficient
- DevSecOps
 - Integrate security processes into development pipelines
 - To block or not to block? That is the question

About CWE

Common Weaknesses Enumeration

- CVEs lesser-known little brother
- Huge catalog of ways developers have introduced security issues into software
- Includes many real-world code excerpts
- Supports multiple languages and views
- **Should be studied by every developer**
- See also Top 25 etc

I. [CWE About Page - MITRE](#)

Implement SSDLC

Secure Software Development Life Cycle

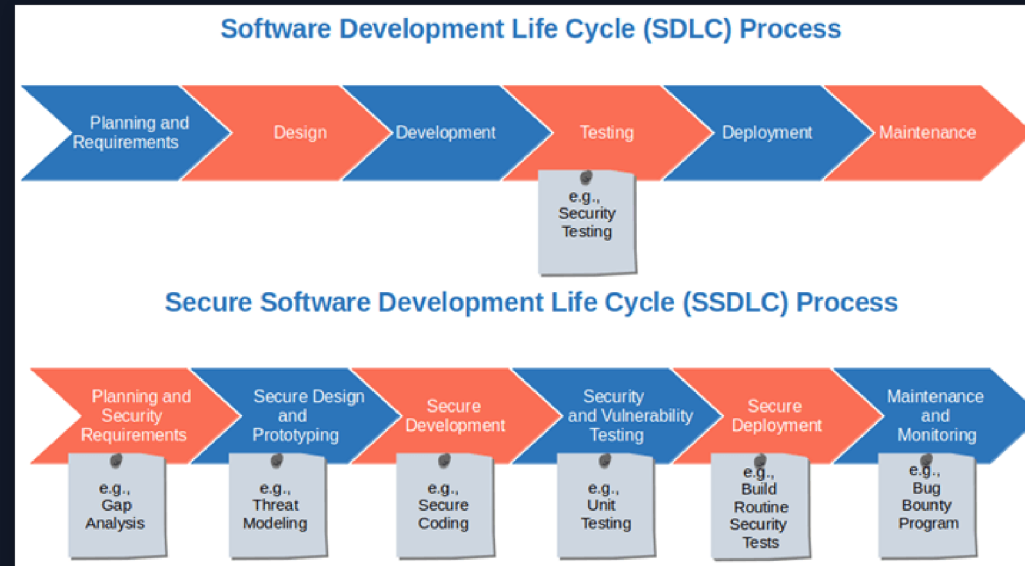


Image provided as an example only

1. [Secure SDLC by CodeSigningStore \(digicert\)](#)
2. [NIST Secure Software Development Framework](#)

Design Phase

Early choices have long-lasting impact.

- On security baselines
- On maintenance burden

Every feature is a potential attack surface.

- Design with response in mind

Programming Language choices inherit technical debt

Programming Language Choice

Often viewed as a religious topic with many factors to consider. Some (subjective) properties:

lang	perf.	get compiled	weaknesses	debug
C	high	medium	high	high
C++	high	medium	high	high
Rust	high	high	low	low
Go	high	medium	low	low
Python	low	n/a	low	low

I use and recommend modern languages

Thoughts on C

- A language for computer wizards
 - Literal minefield of inconsistency
 - Immense burden put on the developer
- Little to no improvements for a long, long time
 - Why not improve string APIs??

Thoughts on C++

- Many improvements but often with new footguns
 - Example: move semantics in C++ 11
- Backward compatibility is a primary goal
 - Leads to "C/C++" code
 - Guarantees require breaking changes
- Adoption rate is *VERY* slow
 - Most C++ teams use C++ 17 at best

Thoughts on Rust/Go

Ideal languages for the modern world.

- Benefit from decades of learning about software construction.
- Make it easier to write high quality code
- Comparable performance to C and C++
- Cost is mainly learning curve and memory
 - Computers have never been faster and memory has never been cheaper

Obligatory Rust Slide

Bias warning: I ♥ Rust

- Safety focus
- Fast & Efficient
- Multi platform support
- Ecosystem
- **Empowering**

But this is not a talk on Rust.

Threat Modeling

Threat Modeling is crucial

- *Work with experienced security personnel*
- Who uses the software? How?
- What assets are involved? How are they protected?
- Identify mitigations and implement them

Your threat model is a living document

Consider publishing it

1. [OpenThreatModel on GitHub](#)
2. [Threat Modeling on OWASP](#)

Implementation Phase

While constructing, be diligent and keep learning.

1. Hopefully you picked a modern language
2. Create and adhere to coding guidelines / standards
3. Conduct code reviews with qualified colleagues
4. Pay close attention to compiler warnings
5. Use any and all static analysis tools

Testing

Write lots of tests

- Unit tests, Integration tests
- **Leverage fuzz testing!**

USE SANITIZERS: Address, Thread, UB

- Especially if you're using C or C++

Test your code changes yourself

- Observe your software's behavior first hand
- Who better to know if it's working correctly?

Verification

Formal verification is a desirable property.

- Coq, Frama-C, CBMC, Kani (for Rust), etc

Widely considered the ultimate in trust and guarantees

But it's not a panacea either

- Verifiers are software too

I. [The C bounded model checker: criminally underused](#)

Post-Deployment

1. Periodically re-assessing security
 - The environment changes, remember gmail?
2. Invite review - open source / bug bounties / etc
 - Marcel wrote about this in his paper, and I agree.
 - This kind of adversarial relationship is virtuous.

Security is a process.

Conclusions

What I hope you learned

General Takeaways I

Transparency -- we need more

- How is your team investing in security?
- What is your software made of? (SBOM)
- Who audited what?
 - Tracked sometimes, but indirectly (ie. CVE credits)
- What had little or no review?
- What tests run in CI?

General Takeaways II

Required security classes

- Computer security 101
 - Core concepts
 - History of attacks and defenses
 - Modern best practices

Modern tools incorporate learnings from previous failures

- Feed failures back into security testing pipeline

Takeaways for Management

Management must set direction appropriately

1. Invest in improving software quality
2. Security bugs are bugs
 - Less bugs means less security bugs

Evidence shows that better quality leads to better efficiency AND cost savings

I. Top reasons why businesses should invest in software quality

Takeaways for Developers

1. More configurable features flags
 - Allow users to toggle features
2. Choose modern languages and tools/toolchains!
3. Learn more about security issues (CWE)
4. Test, test, fuzz test, use sanitizers.
5. Talk to decision makers about prioritizing quality

Takeaways for Security Practitioners

Teach your skills! Take an apprentice!

Let's get scientific!

- Consider your projects as collections of experiments
- Document them using Hypothesis / Experiment Design / Notes / Results format
- Increase rigor, especially in security testing.

Learn more about development

- Don't be afraid to contribute code

All the safety is not enough

Bugs will still happen

- Command injection, SQL injection, XSS, all the injections
- Logic errors, crypto fails
- Authentication missing or poorly implemented
- ...and so on

Memory safety issues are reportedly 70% of the problems that Microsoft and Google encounter and fix.

Conclusion

Developing secure software is hard work.

Development orgs must take ownership and make wise decisions.

Modern tools and techniques can help to reduce the burden.

Please take these concepts back to your development orgs and push for improvements.

- Remember: Security is a process and a team sport.

Go read Marcel's paper!!

Thanks for your time!

Any questions or comments?

Feel free to reach out later:

Joshua J. Drake

jduck @ Twitter/Discord/Mastadon/etc

About these slides

Slides were created in [markdown](#) with [nreveal.js](#)

You can export by printing the [PDF](#)

the real end. really.