Avoiding the Top Ten Software Security Flaws

@cigitalgem@sigmoid.social

JUNE 14, 2023

GARY MCGRAW, PH.D.
https://garymcgraw.com

Please live tweet this talk

For more see https://garymcgraw.com

where I'm coming from



Technology

Northern Virginia-Based Cigital to Synopsys (500 people)
Invented the field of software security (12 books)
alpha-geek who gives 20 talks a year
Light saber

Music

Carnegie Hall at 10 and 16. Suzuki.
The Bitter Liberals
Where's Aubrey
Funny faces while playing the violin

Life

Clarke County on the river near Berryville,
Living in the country
Fiction reader, Art collector, Craft cocktail maker, Cook
Solstice parties

work from the IEEE CSD

The nine organizations that spearheaded the IEEE CSD

Since the initial report

Building codes for IoT, Power Systems, Medical Devices

Security Design Analysis example WearFit

We need more talk about flaws and more examples of real flaws. If you are an architect, get involved!

what is a flaw?

two kinds of security defect

IMPLEMENTATION BUGS

- Buffer overflow
  - String format
  - One-stage attacks
- Race conditions
  - TOCTOU (time of check to time of use)
- Unsafe environment variables
- Unsafe system calls
  - System()
- Untrusted input problems

50%

ARCHITECTURE FLAWS

- Misuse of cryptography
- Compartmentalization problems in design
- Privileged block protection failure (DoPrivilege())
- Catastrophic security failure (fragility)
- Type safety confusion error
- Insecure auditing
- Broken or illogical access control (RBAC over tiers)
- Method over-riding problems (subclass issues)
- Signing too much code

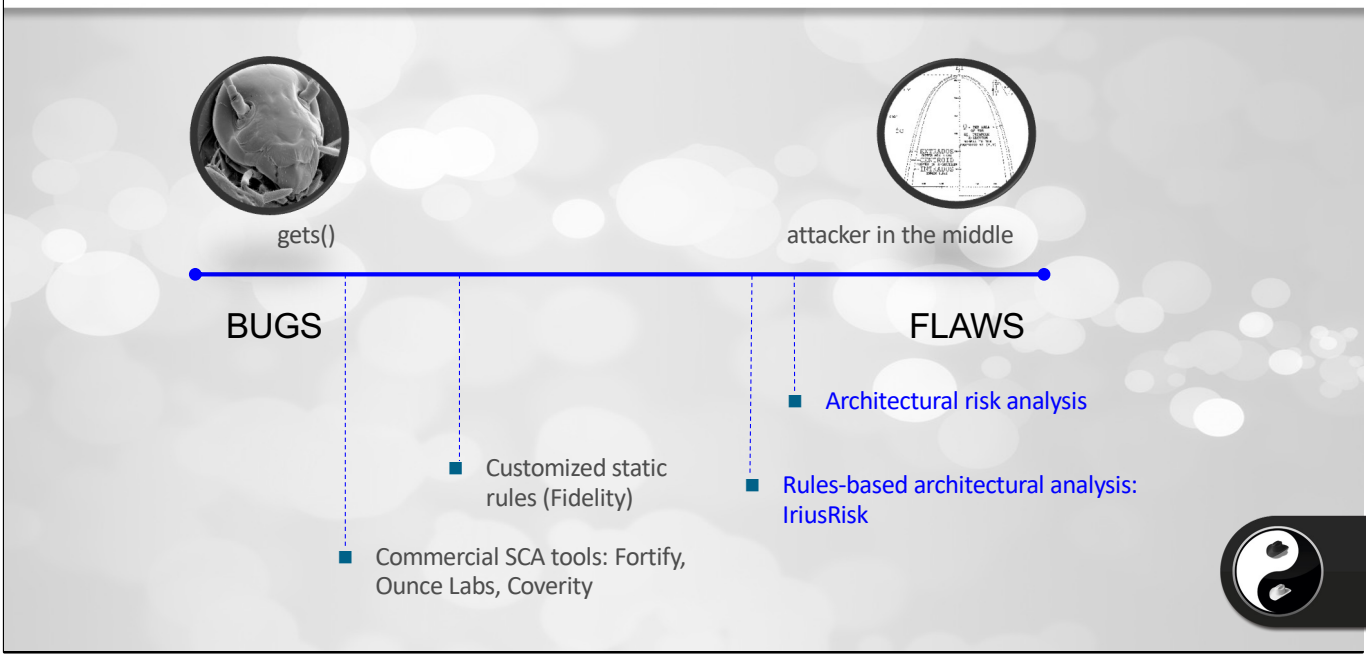
50%

Two kinds of software defect

Sometimes fixing the architecture (at Google for example) can eradicate jillions of FLAWS (XXS made much harder)

The easiest flaw in the world: “FORGOT TO AUTHENTICATE USER”

on defects: the bugs and flaws continuum



Dividing things into two piles is never very clean. This is a range of defects.

bugs versus flaws is a thing in software security

ARCHITECTURE ANALYSIS		
[AA1.1]	103	1
[AA1.2]	29	1
[AA1.3]	23	1
[AA1.4]	62	
[AA2.1]	18	
[AA2.2]	14	1
[AA3.1]	7	
[AA3.2]	1	
[AA3.3]	4	

- DevOps demands that we automate defect detection in the SDLC
- Automating bug finding is straightforward
  - Lots of great commercial technologies
  - Fixing is still a challenge
- Automating flaw finding has barely begun
  - Architectural risk analysis and threat modelling are still way too hard
  - Tooling that automates RA and provides consistency in results is just emerging (e.g., IriusRisk, SecuriCAD)
- BSIMM10 shows that we're (still) not paying enough attention to flaws
  - Only 12.5% of firms have a process

We've been talking about bugs versus flaws since 1999. But not enough progress has been made. You can see this in BSIMM10 results. http://bsimm.com

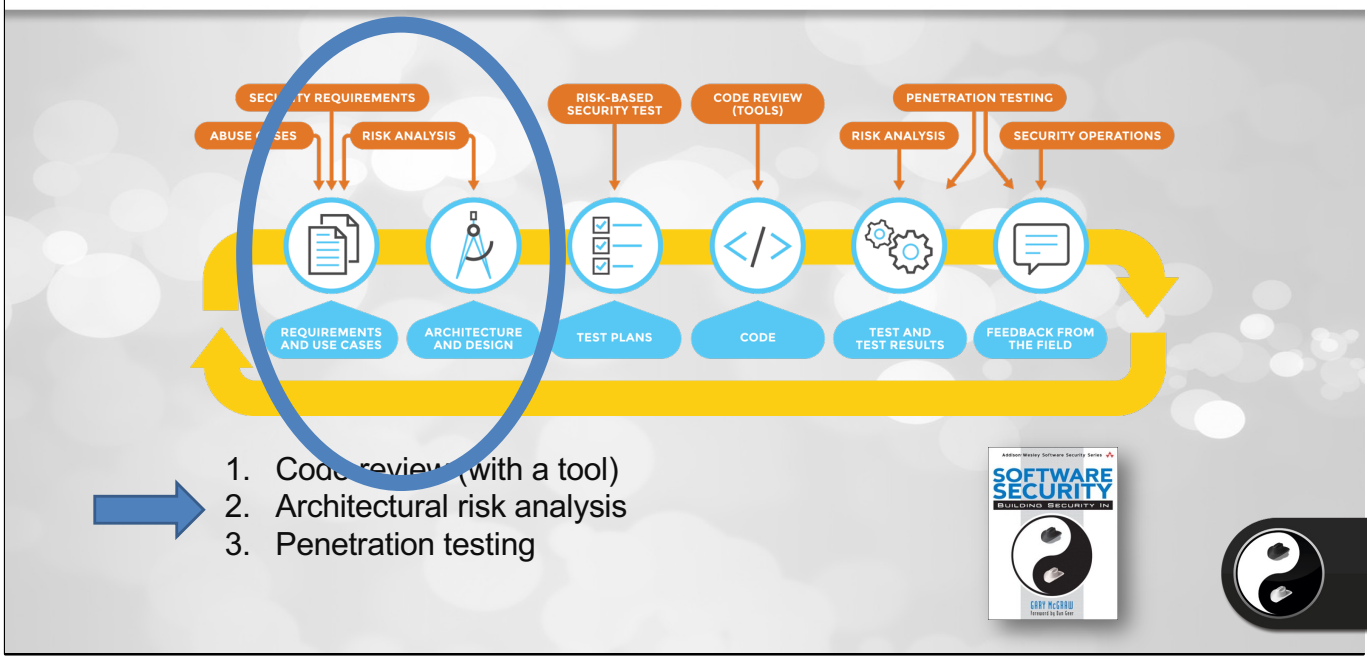
The most common approach is the "bunch of smart guys in a room" approach. Sporadic and inconsistent results.

BUT WAIT, THERE"S MORE → DevOps is a shiny thing that may delay progress in architecture analysis even further.

As you rush off to adopt DevOps methods (even DevSecOps), don't forget the FLAWS

Irius Risk exists to build automation in finding, tracking, and fixing flaws

software security touchpoints circa 2006



The top three touchpoints are:

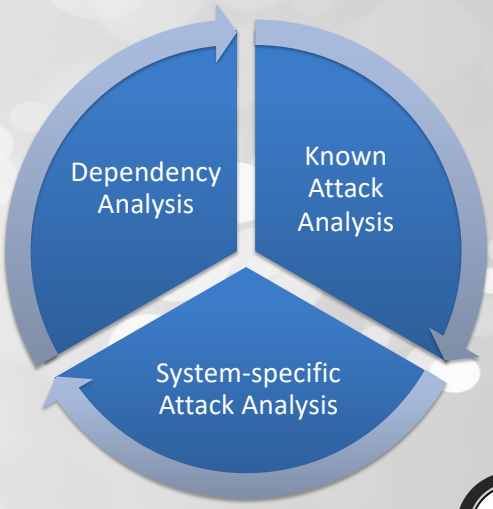
1. Code review with a tool
2. Architectural risk analysis
3. Pen testing

====

WHO DOES PEN TESTING? OK, WHO DOES ARCHITECTURE RISK ANALYSIS?

Three steps to architectural risk analysis (ara)

- Known Attack Analysis
  - Apply checklist of known attacks
  - Risk-based judgement of fitness
- System-specific attack analysis
  - Find attacks based on how the system works
  - Expose invalid assumptions
- Dependency analysis
  - Explore dependencies on frameworks and containers
  - How solid is the foundation?



One of these things is not like the other. You can automate dependency checking and known attack knowledge.
===
Story of VISA and ARA in 1997

Instead of simply listing ten flaws, we decided to show how to avoid flaws through better design.

I will try to include an example from Machine Learning security and my work at BIML for each flaw.

1) earn or give, but never assume trust

earn or give, but never assume trust

- ✓ Make sure all data from an untrusted client are validated
- ✓ Assume data are compromised

- Avoid authorization, access control, policy enforcement, and use of sensitive data in client code

ML** WHERE DID THOSE DATA COME FROM?

===

60% of machine learning risks are related to data issues. Public data can be biased and sometimes even intentionally poisoned.

MACHINE LEARNING SYSTEMS don't have a good answer to this set of risks yet

WHO IS CALLING YOUR API??

===

Most early android escalation of privilege (oh, sorry, "jailbreaking") flaws followed policy #1. System services assumed the information or messages they'd get were from authorized sources.

===

Delivery people being allowed inside. I even see this happen on accident during engagements when I'm in NYC. They have enough messengers there when I arrive security tends to just show me through as I use a messenger bag.

2) use an authentication mechanism that can't be bypassed

use an authentication mechanism that can't be bypassed

- ✓ Prevent the user from changing identity without re-authentication, once authenticated.
- ✓ Consider the strength of the authentication a user has provided before taking action
- ✓ Make use of time outs

- Do not stray past the big three
  - Something you are
  - Something you have
  - Something you know
- Avoid shared resources like IP numbers and MAC addresses
- Avoid predictable tokens

DID YOU DISABLE YOUR TEST HARNESS? (blowing JTAG fuses on chip)
===
Old days story. Authentication worked fine, but database required GODpriv. So become GOD…

ML** In online situations, machine learning systems can be moved in a direction possibly unintended or unanticipated by designers

===
I had the source code for QA to assist with the work and found what looked like test scripts. They were simply called disable-host.jsp and enable-host.jsp. These were like 2-3 line JSP files and all it appeared to do was make a configuration change on the JVM. So I called disable-host.jsp in the QA environment (without authentication) and I get a response "all calls to the host have been disabled". I refresh the login page and get an error message saying it's down for maintenance. I call enable-host.jsp and then the app. is magically working again. That's the first fail which would fit nicely into your authN slides: presence of a test script which any unauthenticated attacker can call (by hitting a URL) and it brings down the app.
===

3) authorize after you authenticate

authorize after you authenticate

- ✓ Perform authorization as an explicit check
- ✓ Re-use common infrastructure for conducting authorization checks

- Authorization depends on a given set of privileges, *and* on the context of the request
- Failing to revoke authorization can result in authenticated users exercising out-of-date authorizations

Is being SOMEONE enough to do ANYTHING? Compartmentalize. Be stingy with privilege no matter who someone is.

===

ML** Can a user extract enough information to build a copy of your machine? How do you stop a malicious user from doing that?

===

Modern authorization systems may require stronger authentication to do more stuff (check balance versus transfer cash). Require more authentication to move up the PRIV chain.

===

How long should authorization last? Time it out.

===

Kerberos/PYKEK thing for “Authorize after Authenticate”. Oops, every enterprise serious about security now has to rebuild their entire domain.

4) strictly separate data and control instructions, and never process control instructions from untrusted sources

strictly separate data and control instructions, and never process control instructions from untrusted sources

- ✓ Utilize hardware capabilities to enforce separation of code and data
- ✓ Know and use appropriate compiler/linker security flags
- ✓ Expose methods or endpoints that consume structured types

- Co-mingling data and control instructions in a single entity is bad
- Beware of injection-prone APIs
  - XSS, SQL injection, shell injection
- Watch out for (eval)

The C sea of bits is a huge problem. Is it a pointer? A password? An integer? Who knows. TYPE SAFETY IS GOOD.

===

Non-executable stacks are good

===

SQL injection story

5) define an approach that ensures all data are explicitly validated

define an approach that ensures all data are explicitly validated

- ✓ Ensure that comprehensive data validation actually takes place
- ✓ Make security review of the validation scheme possible
- ✓ Use a centralized validation mechanism and canonical data forms (avoid strings)

- Watch out for assumptions about data
- Avoid blacklisting, use whitelisting

Data validation is super low-end, bottom line security
===
EVERYBODY SCREWS THIS UP
===
ML** know that data are more important than ever when it comes to ML. How do you avoid bias? How do you spot poisoning?
===
String functions in C were a notorious issue many years ago, but a SYMPTOM OF A FLAW
===
How big is the list of possible bad inputs?? (infinity)

6) use cryptography correctly

use cryptography correctly

- ✓ Use standard algorithms and libraries
- ✓ Centralize and re-use
- ✓ Design for crypto agility
- ✓ Get help from real experts

- Getting crypto right is VERY hard
- Do not roll your own
- Watch out for key management issues
- Avoid non-random "randomness"

ML** Turns out that the order in which you choose training examples really matters. So cryptographic randomness is a strength and a necessity.

SECURITY IS NOT CRYPTOGRAPHY

Show of hands: who has used crypto mechanisms in their code pile?
===
Textbook RSA. We teach it as an intro construction to RSA for most students but it doesn't meet the appropriate cryptographic security properties.
===
CRYPTO IS HARD

7) identify sensitive data and how they should be handled

identify sensitive data and how they should be handled

- ✓ Know where your sensitive data are
- ✓ Classify your data into categories
- ✓ Consider data controls
  - ✓ File, memory, database protection
- ✓ Plan for change over time

- Do not forget that data sensitivity is often context sensitive
- Confidentiality is not data protection
- Watch out for trust boundaries

GDPR has made the PII thing more obvious than ever. Does your system collect or CREATE sensitive data?

===

ML** When you train on confidential or sensitive information, it ends up IN your machine learning representation. Retrieving secrets is a well known attack on ML. GTP-3 and SSN completion attack.

===

Data can change its stripes according to context: People with AIDS list = critical for medicine, and really useful for blackmailers

8) always consider the users

always consider the users

- ✓ Think about: deployment, configuration, use, update
- ✓ Know that security is an emergent property of the system
- ✓ Consider user culture, experience, biases, …
- ✓ Make things secure by default

- Security is not a feature!
- Don't impose too much security
- Don't assume the users care about security
- Don't let the users make security decisions

Network security people say: Users are the worst! And you know who the most dangerous users are? Users with compilers.

===

Security decisions are hard to make. And they make a huge difference. Ever chmod something 777 just to get it to run? That.

===

ML** sometimes users can get more out of your ML system than you may think. Extraction attacks. Cloning attacks.

9) understand how integrating external components changes your attack surface

understand how integrating external components changes your attack surface

- ✓ Test your components for security
- ✓ Include external components and dependencies in review
- ✓ Isolate components
- ✓ Keep an eye out for public security information about components

- Composition is dangerous
- Security risk can be inherited
- Open source is not secure
- Don't trust until you have applied and reviewed controls
- Watch out for extra functionality

Anybody working on massively distributed or cloud architectures? LOL.

===

ML** Many ML models are open source and are used without integrity checks.

===

Who made that component? Who is keeping it up to date from a security perspective? How about that API? That micro-service?

===

Death by 1000 micro-service cuts

===

ABOUT THAT OPEN SOURCE…

10) be flexible when considering future changes to objects and actors

be flexible when considering future changes to objects and actors

- ✓ Design for change
- ✓ Consider security updates
- ✓ Make use of code signing and code protection
- ✓ Allow isolation and toggling
- ✓ Have a plan for “secret compromise” recovery

- Watch out for fragile and/or brittle security
- Be careful with code signing and system administration/operation
- Keeping secrets is hard
- Crypto breaks

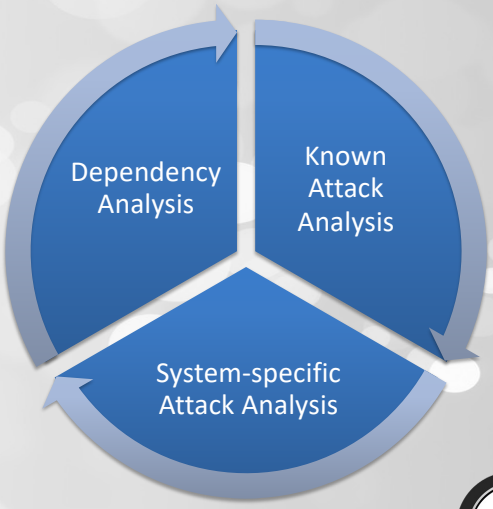
Things change. Software exists because things change. It is easier to update software than to ship an entirely new product (in theory).

===

PLAN FOR CHANGE

build a process to look for flaws (like ARA)

- Known Attack Analysis
  - Apply checklist of known attacks
  - Risk-based judgement of fitness
- System-specific attack analysis
  - Find attacks based on how the system works
  - Expose invalid assumptions
- Dependency analysis
  - Explore dependencies on frameworks and containers
  - How solid is the foundation?

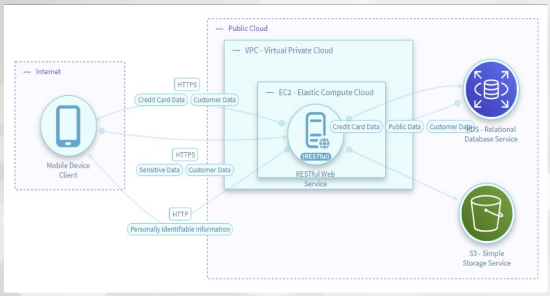


One of these things is not like the other. You can automate dependency checking and known attack knowledge.

===

Story of VISA and ARA in 1997

IriusRisk is automating ARA

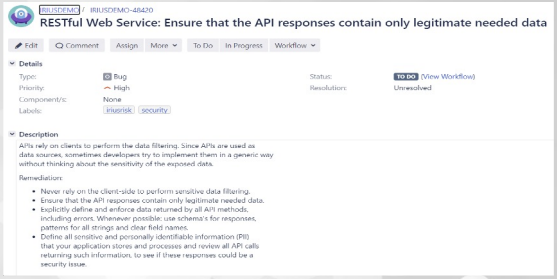


1. Rules engine parses diagram

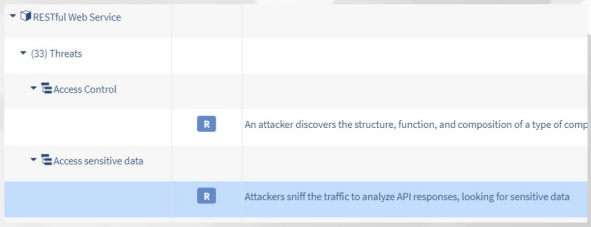
Architectural Security Rule

If **data flow** has **tag** 'HTTP' and **trust zone** == Internet
and **data** == PII
then create **threat** and **countermeasure** in model

2. Rules generate threats & countermeasures



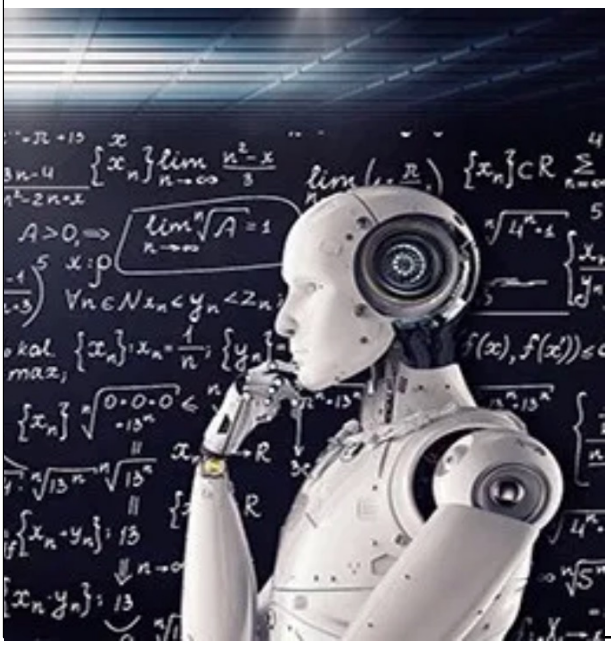
3. Countermeasure uploaded to Jira



Full disclosure: I chair the Technical Advisory Board of IriusRisk

where to learn more

This is the future…

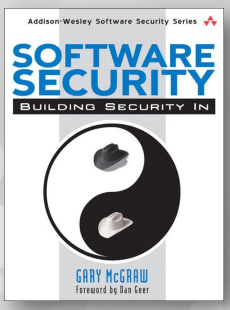


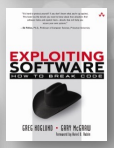
- Machine learning is showing up everywhere
- Secure your own AI/ML using the BIML-78 risks as a guide

BIML

build security in

- Writings, Blogs, Music
  https://garymcgraw.com
- BIML: Security of Machine Learning
  https://berryvilleiml.com
- Send e-mail:
  gem@garymcgraw.com





@cigitalgem@sigmoid.social