Your design is (you believe) secure
Your code is (you believe) correct
How do you operate your system securely?
Lots of things left to do...
Elements

- Code quality
- Patching
- System administration
- Process
- Logging (which we’ve already covered)
You didn’t write all of your own software
No, you really didn’t
You probably did not write the OS, the compiler, the libraries, the browsers your users rely on, your word processors, the firmware in all of your many devices, the software in BYOD gadgets, and more
How secure is all of that code?
You’re considering getting a software package—or perhaps you’re already running it—and you want to assess its security.

How do you do this?

Reputation, measurement, testing, code quality, reviews
Good code without good process is exceedingly rare

Does a good, secure reliable product imply the existence of a good process? Perhaps.

Does that mean that other packages from the same vendor are likely to be equally good? That’s a much harder call—different divisions in a company can have different processes

Even different programmers working on the same product can have different abilities

Overall, though, reputation is a decent starting point
Reputation of a product is a better predictor than the reputation of the company.

Sometimes, a company’s standards are the same throughout, but sometimes not.

There may be a legacy code base that takes a long time to clean up.

Very new products, even those developed properly, are more likely to harbor problems: “Never install .0 of anything.”
Sometimes, bad products get better—a lot better

In 1994, Bill Cheswick and I wrote

*The most common implementation of SMTP is contained in sendmail.*

...you get less than you pay for. Sendmail is a security nightmare.

No longer true—it’s been years since the last major hole in it
Microsoft as a Danger

- In the late 1990s, Microsoft products were a joke
- “Internet Exploder”
- “Microsoft Look Out!”
- “Using Internet-exposed IIS Web servers securely has a high cost of ownership. Nimda has again shown the high risk of using IIS and the effort involved in keeping up with Microsoft’s frequent security patches” —The Gartner Group, 2001
- (They canceled that warning in 2004.)
Bill Gates and Steve Ballmer realized that this sort of reputation for crapware was an existential threat to the company. They got religion on security and started a company-wide security effort. Result: Windows 11 is probably the most secure general-purpose operating system ever released.
You could build your own test suite
You could fuzz it
This is done, in high-risk situations, but it’s expensive and not commonly done by purchasers
Open Source

- With open source code, you can look at everything and (often) even download the test cases.
- Besides, “given enough eyeballs, all bugs are shallow”, right?
With open source code, you can look at everything and (often) even download the test cases.

Besides, “given enough eyeballs, all bugs are shallow”, right?

Well, no.

These eyes have to actually look, and they have to know what they’re doing.

These eyes probably need code-auditing tools and the knowledge and will to use them.

Very few open source projects have disciplined, high-quality development processes—which means that the code is often much worse.
Measurement and Code Quality Without Source

- It seems odd—how can one measure code quality without access to source code.
- It sounds improbable—but in fact it’s being done.
Assess the likely safety of software based on external characteristics

Not an absolute score, a relative one: the same application on different platforms, or different applications on one platform

*Not* a guarantee of security, but it does indicate relative risk
Many platforms support different safety measures, but programmers have to turn them on

Example: use “stack guard” to prevent some buffer overflows

Example: “data execution prevention” makes the stack and/or heap non-executable, to prevent the attacker from injecting code

Again, though, you have to enable them. CITL found that Zoom did not
It’s often possible to tell, externally, which functions were used.

Some functions are riskier than others. They *can* be used safely, but often are not; their presence suggests a lack of security consciousness, as well as potential danger in its own right.

- **Bad**: `strcat()`; **better**: `strncat()`; **best**: `strlcat()`
Complex code tends to be buggy and hence insecure

Things to look for: code size; number of conditional branches; size & number of stack adjusts; function size

Again: *not* a sign of insecurity, but (in the opinion of most software security people) correlated with it
Patching
No code base is perfect

I said that Microsoft code is very, very good—but this month’s “Patch Tuesday” fixed about 145 bugs, 10 rated “critical” and one in active use or about to be

Conclusion: you must patch

But—there are many issues
Issues with Patching

- Patches may themselves be buggy and create new problems
- The patch may not fix the problem
- Applications may be incompatible with patches
- Binary versus source patches
- Patch dependencies
Issues with Not Patching

- Some attackers reverse-engineer patches and learn new exploits
- Intelligence agencies certainly do that
- If you wait too long, it gets harder to upgrade; you often can’t skip versions
- Your software will eventually be EOLed
Patching Strategies

- Vendor push
- Mandatory or quasi-mandatory user downloads
- User pull
- None—software has been EOLed
Types of Security Holes

- Denial of service
- Local exploit—attacker must already be logged in
- Remote exploit—much more serious; the attacker doesn’t need to be on your system already
- Privilege escalation—a local user (or local malware) can gain more privileges
- Sandbox escape
- File disclosure
- Code execution
- No user interaction required
Threat Issues

- Are you being targeted? By whom?
- How difficult is the attack? Do likely enemies have that capability?
- Is the problem currently being exploited? By whom?
Patch Timing

- When is the patch available?
- When can you install it, relative to availability?
- How much of a window is there for attackers?
- Regular schedules are predictable for system administrators; they can schedule testing, downtime, etc.
- But what about urgent, off-schedule patches?
Patching Strategies

- First and foremost: be aware of patch availability
- (Remember that attackers pay attention)
- Second: assess your risk
- Third: schedule patch installation to minimize risk of patching versus risk of not patching
Most patches don’t break most things
History suggests that many people do not patch
But: most exploits are due to holes for which patches exist
Conclusion: force patch installation
Chrome: must patch; Windows: must patch; MacOS: autopatch encouraged but not mandatory
(Some platforms permit deferred patching—but not for too long)
Small Businesses

- Often treated the same as consumers
- But—are more likely to have mission-critical software that is incompatible with the patch
- Some delay is often advisable, to shake out bad patches—but few small businesses have IT expertise to track things like that
- Conclusion: often better to treat like consumers—but the IT consultant should have installed easy backup/restore (which is necessary anyway)
- If feasible, such businesses should have an IT service contract
Larger Businesses

- Patching for a large organization is *hard*
- There are many unusual or locally written applications
- There are many servers to patch
A well-run organization has a suitable test lab, equipped to test all local applications.

Some of this can be done with virtual machines, some can’t— but it has to be done.

Ordinary desktops are easier to patch; they run similar software (though not everyone will use every application).

Servers are *hard*. 
Every server is different

Servers often talk to databases—and what if a database is corrupted?

Servers tend to have more complex software
Actually, what software is running on which machines?
What versions of the software?
What versions of the libraries?
By the way—who runs each server? In a large company,
You have to track all of this!
Software Bill of Materials

- A machine-readable list of all components that go into a software product
- Must show dependencies, versions, etc.
- Lets you assess your risk from newly announced bugs
- Note well: the machine-readable piece is *important*, since it permits creation of databases and automatic identification of vulnerable packages
You *will* have to patch things

Design your infrastructure for this

Example: run your servers on VMs, snapshot the VM before patching, and then push the patch

If the patch proves problematic, revert the snapshot
What if You Can’t Patch?

- Sometimes, you can’t patch—EOLed software, incompatibilities, buggy patch, etc.
- Mitigate and monitor
  - Add temporary firewall rules, blocking some sites or filetypes
  - Temporarily block email with certain attachments
  - User training, though that’s always difficult
  - Extra monitoring for signs of this exploit
- None of this is ideal, but it may beat the alternative
When code you use has been EOLed, you’re in trouble
Generally, that means that you have not been updating regularly
You probably have old code that won’t run with newer versions
For operating systems, you may not even be able to buy replacement hardware
You will not get any more security patches—but attackers will keep looking for holes
You’re suddenly forced to upgrade everything, at once
Good luck...
System Administration
Sysadmins: your front-line soldiers in keeping systems running well
This especially includes security—if you don’t have good system administration, your site *will* be insecure
But you have to *let* them do their job
Challenges to System Administration

- Little conceptual unity to the problem—hence little academic study on simplifying the problem
- Much of the work is invisible until something goes wrong—and then the sysadmins get blamed
- Low pay, low status, interrupt-driven—and high stress
- Too few resources to do the job properly
- But it’s an utterly vital role
They configure computers and infrastructure
They install patches
They help set security policies
They enforce these policies
They monitor logs
They do the initial investigation into incidents, and are generally the ones who sound the alarm when something has happened
Building Tools

- The only way to do scalable, secure system administration is to build tools ahead of time
- Databases: what devices you have, what they run, etc.
- Database-driven configuration tools
- Patching tools—you can buy them; make sure you use them
- Ticketing systems
- Tracking tools
- Etc.
GUls are good for novices—but they don’t scale

Maybe you can do powerful operations, but only if you’d thought of every possible operation ahead of time

Example: “The following employees, plus people in location X, are being transferred to another company; their access to Project Y must be restricted”

Example: configure all 117 border routers to block port 514

Example: which servers are running Apache 2.4.41 on Ubuntu 20.04.4 with IPv6 enabled?

Then: take the answer to one of those queries, and push a specific patch to them and only them
The Dark Side is Powerful

- What if your system administrator has turned evil?
- If you’re targeted by an intelligence agency, your sysadmins might be targeted
- (Btw, underappreciated employees are relatively easy recruits)
- Sysadmins have root privileges and can override file permissions
- They’re the ones monitoring the logs to see if someone is doing something nasty
- How do you stop this?
The Dark Side is Very Powerful

- Protection against a rogue sysadmin is somewhere between very hard and impossible.
- Personnel background checks are expensive, intrusive, and of questionable efficacy.
- Best solution: logging.
- Log everything, including the trouble tickets that led to any actions.
- Note: sysadmins should create their own tickets before taking actions on their own initiative.
- These logs should be audited by someone outside the sysadmin’s organization.
- It’s not a perfect solution, but there aren’t any great ones.
Your Customers
If you’re a vendor, your code will almost certainly require patches
Depending on what you sell, some of those bugs will be security bugs
You have to fix things
You’re still likely to have bugs and security holes
Those bugs are far more likely to be exploitable
Again, you have to fix things
For how long will you support old versions of your code?
- Longer is more customer-friendly, but also more expensive
- What platform versions will you continue to support?
- Have you communicated this clearly to your customers?
Generating Patches

- Version control systems are your friend, even if they’re hard to use
- You need to keep old build environments available, too—virtual machines are your friend
- Older hardware platforms are more problematic—and you have to be sure that they’re patched to the right level
- Note that some software versions will no longer run on newer platform patch levels
- How long do you expect it to take to generate—and test—a patch?

https://xkcd.com/1597
Reporting Bugs

- How do your customers report bugs, including security bugs?
- How do non-customers report security problems?
- Do you make it clear that you won’t threaten them for doing so?
- Do you pay “bug bounties”? 

Secure Operations
Alerting Customers

- How do you alert customers to potential security problems?
- What is the timing of your detailed description of the security hole relative to when patches are available?
- How do you distribute patches?
- Can you track how many customers have installed the fixes?
- How easy is it for customers to install the patches?
- How easy is it for customers to install the patches at scale?
None of these questions have easy answers

Piggybacking on a platform’s update mechanism, e.g., an app store answers some of the questions, but it may be too expensive

But you have to have answers for all of these questions and more
Questions?

(Scarlet tanager, Riverside Park, May 8, 2019)