Security Evaluation

Analyzing Systems

- When presented with a system, how do you know it's secure?
- Often, you're called upon to analyze a system you didn't design — application architects and programmers build it; security people get to pick up the pieces. . .
- It's better to build security in from the start, but that doesn't happen nearly as often as it should

When to Analyze

- The earlier, the better
- Some design decisions are very hard to correct later on
- Better yet, have frequent reviews
- Early reviews concentrate on the broad architecture; later reviews can look at the pieces

Types of Analysis

- Individual programs
- Overall system flow
- Usually, a faulty program means a faulty system, but sometimes faults are containable
- Let's look at system analysis

Analyzing Systems

- Both easier and harder
- Easier, because there are fewer components than lines of code
- Harder, because many of the details are abstracted away

The Usual Questions

- Who are the attackers?
- What might they want?

Overall Flow

- Identify the separate system elements
- Identify the data flows
- Look for security barriers
- Look for untrusted inputs

System Elements

- System elements are things like web servers, database engines, etc.
- Each of these is itself a complex system that needs to be analyzed
- Establish the properties of each element: where its inputs come from, what its outputs are, what can happen if something is corrupted

Protecting Elements

- What are the forms of access?
- What sorts of access controls are there?
- What is logged? To where? (Who looks at the logs?)

Data Flows

- Who talks to whom?
- How do they talk?
- Is the link exposed to the outside? Is it encrypted? Authenticated?
- Is the protocol otherwise safe?

Security Barriers

- Do they block all attack vectors?
- Are they strong enough?
- Are they flexible enough?

Input Filtering

- Where can enemy input enter the entire *system*?
- Is it properly checked?
- What about back channels, such as DNS?

System Management

- How will the elements be managed?
- Is more connectivity needed?
- Are other network services used?
- How do system management functions authenticate themselves?

Backups

- How are disks backed up?
- Again, is more connectivity needed?
- How are the backup media protected?

Drilling Down

- Is there other connectivity, such as to the organization?
- If there isn't now, might there be in the future? (The answer to that one is usually "yes". . .) What provisions are made for such connectivity?
- What parts of the design seem more vulnerable?

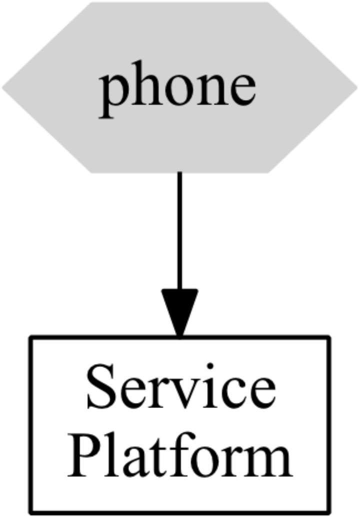
Weak Spots

- What parts of the design seem problematic?
- Some pieces are weaker than others
- Experience counts here — “trust your feelings, Luke”

Weak Spots: Web Server

- Web servers are quite complex
- CGI or ASP scripts are often locally written, and may have received less scrutiny
- How is the web server checked for intrusions?
- What are the consequences if it falls?

Simple Example: Mobile Phone Service



How Do We Analyze This?

- Three elements: a phone, a service platform (the mobile phone switch), and the radio link between them
- What can an attacker do?

☞ Hack the phone, hack the switch, eavesdrop on traffic, steal phone service

- What are our defenses?

Defenses

Stealing phone service	Strong authentication, these days via a SIM. (25 years ago, some phones (effectively) used a plaintext password—and yes, it was possible to steal phone service. More on that below.
Eavesdropping	Encrypt the radio link

Hacking the Phone

- **Locally** Not the phone company's problem! (That's another reason for SIMs)
- **SIMjacking** Typically a phone company problem—and often due to corrupt or poorly trained employees (but could happen via identity theft)
- **Over the Air** Who is the enemy? An outside party? Or the phone company?
- **Outsiders Hacking Phones** Encryption to protect the radio link—but must also protect the service platform. We're missing components!
- **Phone Companies Hacking Phones** The user's problem? The vendor's problem? Note: security analyses depend on viewpoint!
- **Other Customers Hacking Phones** The user's problem? The vendor's problem?

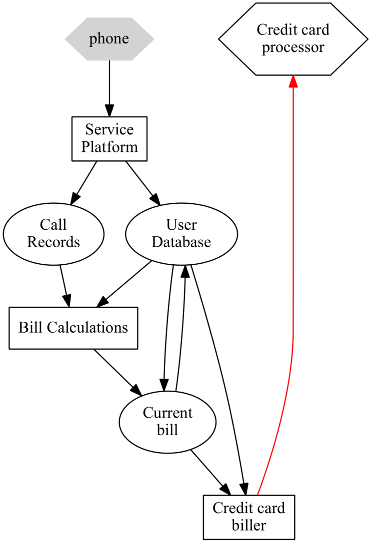
Hacking the Switch

- Can the phone switch be hacked over the air?
- What is the attack surface?
- High—it has to be available to phones, and the protocol is very complex

Can We Firewall the Switch?

- No—its essential function, talking to mobile phones, is the most vulnerable point
- We have to harden it—and make sure there is a lot of intrusion detection

More Complex Example: Mobile Phone Service



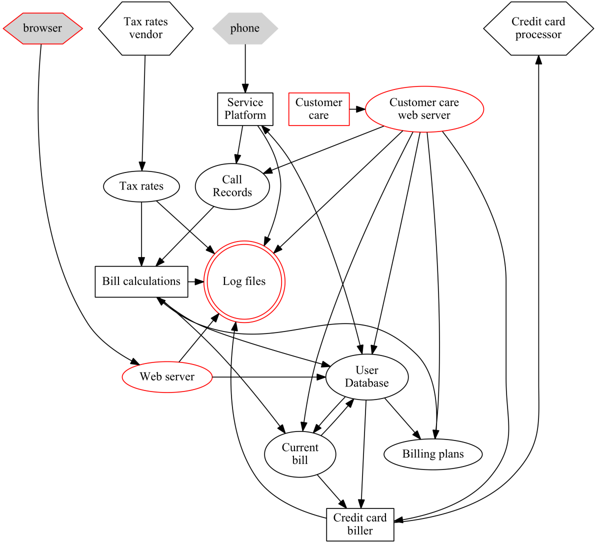
We’ve Added Billing

- Many new elements and links
- We need to look at internal hacking and internal links, e.g., what is the risk if the links aren’t encrypted?
- And: we have a new external link: to the credit card processor

How Risky is a Credit Card Processor?

- You'd think it was pretty safe—they're handling lots of valuable financial data
- But: credit card processors have been hacked
- But: there is actually an input channel from the processors, to notify them of, e.g., chargeback problems

Quasi-Realistic Example: Mobile Phone *Billing*



New Important Elements

- Customer's browser—talks to web server to create and review account; update data
- Customer care—touches many places
- External vendor for tax rates
- A log file database

Tax Rate Vendor?

- There are many, many jurisdictions in the US alone
- Many will tax phone service, but at different rates
- In the US, cities can have their own taxes on top of the state tax rate
- A phone company shouldn't track this, so it buys database updates from an external vendor

Browser Activity

- Customer browsers are utterly untrustworthy—but they have to touch the user database
- Can we trust the web server?
- What is the attack surface of a web server?
- High!
- We can firewall it from the rest of the billing complex—but it has to touch a vital database
- Needed: an application-level filter between the web server and the database

Customer Care

- Vital role—helps people; corrects errors from buggy code *or* from customer misperception
- A major risk, but one that's unavoidable
- Also: what about a dishonest customer care agent?

Securing Customer Care

- The customer care web server is a vital filter—it processes potentially dangerous inputs
- But—it's a web server, with all that implies for its attack surface and hence its security
- We need application-level filters between it and any database it touches
- Plus: we need *logging and auditing*

Logging

- Utterly vital—you need logs to know what happened
- But: hackers *love* to mess with log files
- So: we need to put log files on a separate, secure machine

Outputs of a Review

- Description of the threat model: resources, enemies, and their powers
- *Prioritized* list of weak points
- *Prioritized* list of improvements
- Go/Fix/No-go recommendation

(Abbreviated) Threat Model: Resources

- Service availability
- Billing integrity (accuracy, no theft of service, etc.)
- Conversation confidentiality
- System integrity
- Out of scope: phone integrity, unless it's a telco element that was corrupted to permit attacks on the phones

(Abbreviated) Threat Model: Attackers

- Ordinary consumers: bill integrity
- Hackers: system integrity and availability, maybe conversation confidentiality
- Intelligence agencies: service availability, conversation confidentiality, system integrity

Billing: Once a Bad Threat Model

- Despite no encryption, telcos thought that account spoofing was very hard
- But: the rate of “password”-stealing and phone cloning was *much* higher than expected
- Why?

It Wasn’t the Cost!

- Drug dealers were happy to pay for phones not linked to them (they used one-way pagers for alerts, and then made outgoing calls to clients and suppliers)
- There was test gear on the market that could pick up “passwords” and reprogram phones with it and the associated phone number
- Pattern: buy a phone, pay an electronics tech to give you a new number, use it for a week, pay again
- Result: by the time police found the number and got a wiretap warrant, they were using a different number and maybe even a different phone
- Telcos got the threat model wrong. . .

Recommendations

1. Add an auditing function
2. Add firewalls as indicated
3. Review internal sysadmin connectivity and security
4. Consider encryption for internal links

Rationale

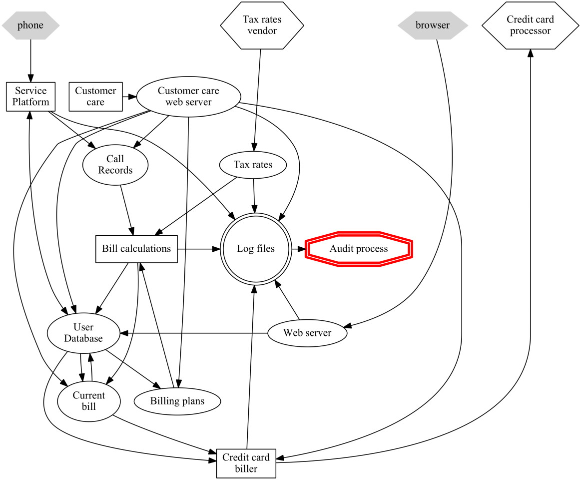
Auditing There are very important but unavoidable very risky elements

Firewalls Add a layer of defense to cope with some of these at-risk elements

Sysadmin Review There are crucial, privileged functions that have not yet been audited. But this is lower priority than the other two, because those are *known* problems

Encryption Internal links may be safe enough; certainly, they're less of a risk than the other items

Revised Network Diagram



(Recommended firewalls not shown)

Auditing

- Log files don't do any good if you never look at them
- We *must* have an audit process
- Automated log file analysis, to spot problems or attacks
- Manual auditing: a good database, good query languages, etc.
- Manaul, routine checking of a subset of entries, to validate the logging

Outcomes of a Review

- All is cool (don't be afraid to say so, but it rarely happens. . .)
- A few fixable flaws
- Serious, unfixable problems
- Not deployable

Serious, Unfixable Problems

- There may be flaws that can't easily be fixed
- Example: a piece of vital third-party software that does stupid things
- Can you layer on something else to provide necessary protection?
- Example: to protect a vendor product that sends plaintext passwords oer the network, you could add a VPN

Not Deployable

- Sometimes, that's the right answer
- However — how important is the project?
- What is the *business* cost of not deploying it?
- It's important to be both honest and realistic — and that's a delicate balancing act

Software Engineering Code of Ethics

1. PUBLIC—Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER—Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT—Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT—Software engineers shall maintain integrity and independence in their professional judgment.

. . .

(See https://ethics.acm.org/code-of-ethics/software-engineering-code/ for the rest.)

Making “No” Stick

- Be prepared to back up your assessment
- Demonstrate *exactly* how an enemy could get in
- Estimate the likelihood of the attack
- Estimate the *business* loss if it happens
- If you can’t do that, it’s more likely the previous category

Bad Excuses You'll Hear

- It's closed source; no one knows how it works
  - ☞ It's a lot easier to figure such things out than it appears to those who have never done it
  - ☞ What about corrupt insiders?
- Who'd attack us?
  - ☞ Some people will attack anything
- No one would try that
  - ☞ Some people will try anything

Making Recommendations

- This is often a political process
- Concrete suggestions for improvement are better than “this is awful!”
- Suggestions should be realistic in terms of cost, benefit, and business situation
- Security is *engineering*; it’s not an absolute goal to be pursued at any cost
- There are always legacy systems you can’t touch

“No” Can Win

- We showed the strong possibility of a devastating outcome
- ☞ Management backed up the security team’s evaluation
- We all agreed that a small-scale beta trial could find functionality problems and did not present serious risks
- Compromise: do the beta trial during the six months it would take to rearchitect and rewrite the offending subsystem

Questions?

(Cedar waxwing Morningside Park, March 20, 2022)