02/24

Table of Contents

Administrivia

Course Projects:

- Goal: do some *new* research
    - Work in groups to make it more substantial (2 or 3)
    - Flexible about topic (what kind of problem)
        - Anything that interests you that is related to security
        - Rule of thumb: If there is an adversary, then you are good.
    - Must have some novel aspect (more than reading a previous paper and understanding/reproducing what they did)
    - Don't recycle research, but you can build on your own existing research.
    - Don't need something that is as complete as a conference paper
- Deadline for 1 page project proposal: a week from Monday

Organized Office Hours: Thursday 12:30 and after each class.

Lecture

Prof: "[Capabilities] essentially had no impact" so they are a kind of a crazy idea that is exciting but has struggled with adoption. What do you think the barriers to adoption are?

- Rachel: It would require rewriting code (getting legacy code rewritten is hard not just technically but also because of bureaucracy)
    - Prof: Yes and can't incrementally adopt by writing new modules in Joe-E
- Orr: seems to contradict Java's principles, e.g. by not allowing reflection
    - Reflection used a lot for testing
- Prof: Lack of libraries
    - You can only use a subset.
    - You may need a security expert to review each library before allowing it.
    - Benefit of ecosystem missing.
- Prof: Need to design your system differently (to make it a good match for this paradigm) so you're now imposing a different mindset on developers. This is asking a lot!

Now we look through an example that may help you "discover" capabilities.

In ancient history (~1960s) lots of users would use a main frame to compile their code. Let's look at a vulnerability in that compiler setup.

Suppose we had the following access control policy:

- Compiler has read/write access to entire filesystem (must be able to access source code, output, and internal stuff like a billing log file log.txt)
- Users have read/write access to their own files (e.g main.c, a.out, …)

The user uses the compiler by calling something like the following, where main.c is the source code, and a.out is where the compiler will generate to.

```
Compiler.compile("main.c", "a.out")
```

Internally, the compiler would follow steps like the following (in the second step the compiler is keeping track of who to bill for this compile job):

- Compiler calls Filesystem.read("main.c") – approved (OS allows that)
- Compiler calls Filesystem.append("log.txt", "… log entry …") – approved
- Compiler calls Filesystem.write(a.out, "… compiled code …") – approved

Given this setup, what is the vulnerability?

- Users can use the compiler for free!
    - Users don't have access to the log file
    - but the compiler has write access….
    - So they can trick the compiler into overriding previous log entries!
- Attack: call Compiler.compile("main.c", "log.txt")

    - Will write to log.txt instead of a.out so it overrides (trashes) the log.
    - So you can use the compiler normally first to get your desired a.out, and then use the compiler in this malicious way to erase your costs.

This kind of attack is called “confused deputy.”

- Compiler is the deputy
- Attacker confuses the deputy
- OS sees that compiler is trying to access the log file and OKs the override
    - The developer probably didn’t intend for this to be possible, and
    - the user couldn’t do this on their own (they don’t have permissions to write to the log file).

Cross site request forgery (CSRF) is a confused deputy attack.

- Web based vulnerability
- Let’s say that Dave has an account at schwab investment bank
    - Buttons in the online interface do things like “buy shares”, “sell all my shares” …
    - Clicking the button means my browser goes to a link.
    - When the bank’s server receives a request, it says “Oh Dave wants to sell all its stocks so let’s sell them”
    - Website uses cookies to make sure it really is Dave
        - Just a random 128bit unique value that is unguessable
        - Browser remembers it
        - Every subsequent action that Dave’s browser makes will include this cookie
            - “Hey here’s the value you previously gave me, you must be you”
        - Nobody else could guess what that value is, and the server gave it to Dave when he logged in
    - When we click to sell all stocks, the server checks the cookie to make sure Dave is really Dave.
    - The attack: someone sends Dave an email “to get your vaccine appointment click here”
        - The link is to “bank.com/sell_all_my_stocks”
        - Cookies get sent along if Dave was already logged in
        - Server thinks Dave wants to sell all his stocks
        - Dave clicking the link in this email has inadvertently made him sell all his stocks

Why is this an instance of the confused deputy problem?

- Rikhav: the browser is the deputy who is taking actions on your behalf even though you don’t want your cookies attached to this request. (Tricking the browser into doing something that was not intended)

Abhishek: the attack works because the link is public knowledge---could randomize link/address/unique secret value included in URL.

- Great! That's a standard defence (don't make it a publicly known page)
- Usually implemented with a hidden form parameter: effectively the same thing.
- So, the button to sell all stocks links to a secret URL that the attacker can't know about and therefore can't use in their malicious email.

Diagnosis: What allows confused deputy attacks?

- Ambient authority
    - E.g. the compiler in original example had ambient authority to write to anything
        - The OS gave the compiler a global set of rights that apply to all its writes
            - Doesn't require compiler to designate its intent

One way to think about it is that the secret value is a capability!

- If I wanted someone else to have the ability to sell my stocks, I could do that by giving them my secret URL, passing that capability to them.

Capabilities solution:

- No ambient authority
- Bundle designation (the resource) with authority (permission to that resource)
    - When the compiler from the example indicates that it wants to read a file, it needs to pass one thing that both identifies the file and the purpose/permission for executing the write.

Capabilities combine designation (file) with authority (permission to write to it). A file object can bundle this info together, while a primitive string cannot. So, we would want, for the compiler example, code like this:

```
Compiler.compile(file src, file dest) {
        … = read(src)
        Append(logcap, logentry)
        …
        Write(dest, output)
}
```

Instead of BAD code like this (using designation (string) and privilege separately):

```
Compiler.compile(String src, Privledge srcpriv, String dest, Privledge destpriv) {
        … = read(src, srcpriv)
        Append(log.txt, logpriv, logentry)
        …
        Write(dest, destpriv, output)
}
```

(Post Student Lead Discussion)

Issues tend to come up when client calling service that has some permissions and we need to figure out what permission the service has. Client → Server → Resource

Quick comparison of capabilities style access control to others that have been proposed:

- IPC/RPC/microservices
    - access control policy based on server's permission (callee not caller)
    - (today most of the world acts on this)
- Unix program execution
    - access control policy based on caller (client's) permissions.
    - When that is not sufficient, you can use Setuid
        - (like when I want to run something as root)
        - kind of like IPC/RPC---based on callee permissions---except it can request to use caller permissions (make explicit request).
- Java stack inspection
    - Looks at call stack and takes intersection of permissions of entire call stack (intersection of caller and callee).
    - Would actually stop confused deputy for compiler.
    - But problem: compiler won't be able to write to log file since user doesn't have permission to it.
    - By explicit request, use the callee's permissions and ignore call stack.

Student Lead: Arun leads a discussion on The Problem with Threads.

Programming in a non-concurrent setting is easy, as soon as you introduce threads things become complicated. Opinion: avoid threads.

Example code: Value holder with interface for listeners.

- Issue could be
    - One thread calls “setValue”
    - Another thread calls “addListener”
        - Adding to iterator while iterating (Collection changed) will throw an exception.
        - This exception wouldn’t happen in single threaded case.
- Java would handle this with synchronized methods
    - But this introduces other issues:
        - Lock cycles (set value called by one thread and another thread calls method) so we get deadlock!
- OK maybe instead of synchronizing an entire method we synchronize a part of it
    - Copy the iterator in a sync block and then iterate over local copy that can’t be modified by some other thread
    - Might still give weird behaviour, not necessarily a bug
        - Two different threads call setValue, could give results out of (expected) order
            - Depending on interleaving, first setValue call could be the one that happens second which could be surprising, lead to an error down the road.
- Many attempts and subtle behaviour!
    - “Deterministic ends should be accomplished with deterministic means”
    - Threads are bad, everything is going to be buggy! (authors opinion)

Arun: Why aren’t things is awful today?

- Orr: Maybe things aren’t that bad today because usability of threads is now better?
- Mayank: things like reducing shared memory have helped?

Prof: Can we interpret problem with threads as a capability issue?

- Capability folks would say “hard to reason when you have shared mutable state”
- People pushing capabilities seem to also be interested in alternate models of concurrency.