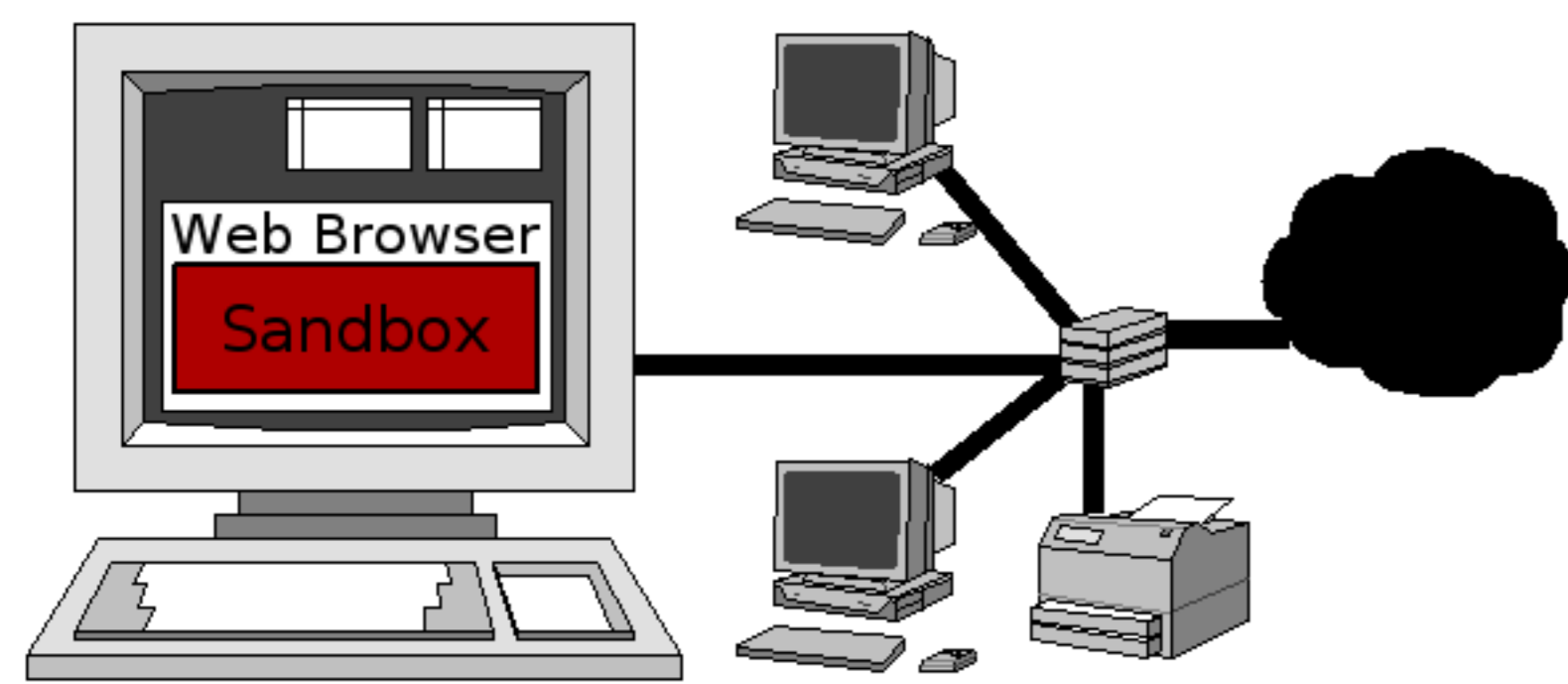


Existing Web Security

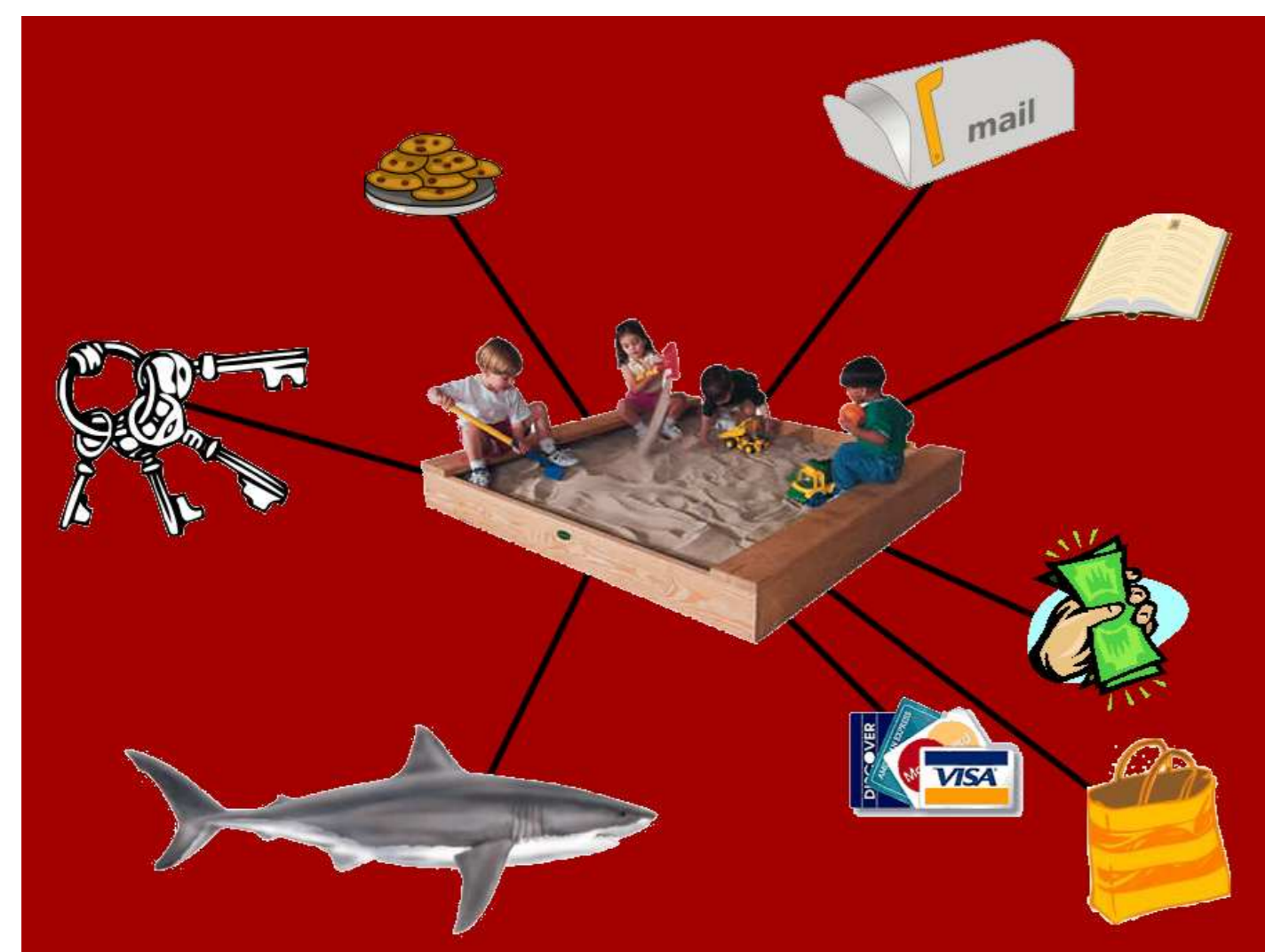
1 The Sandbox

A digital “sandbox” is a “safe” place to run untrusted code.

- Protects: the computer, the file system, applications

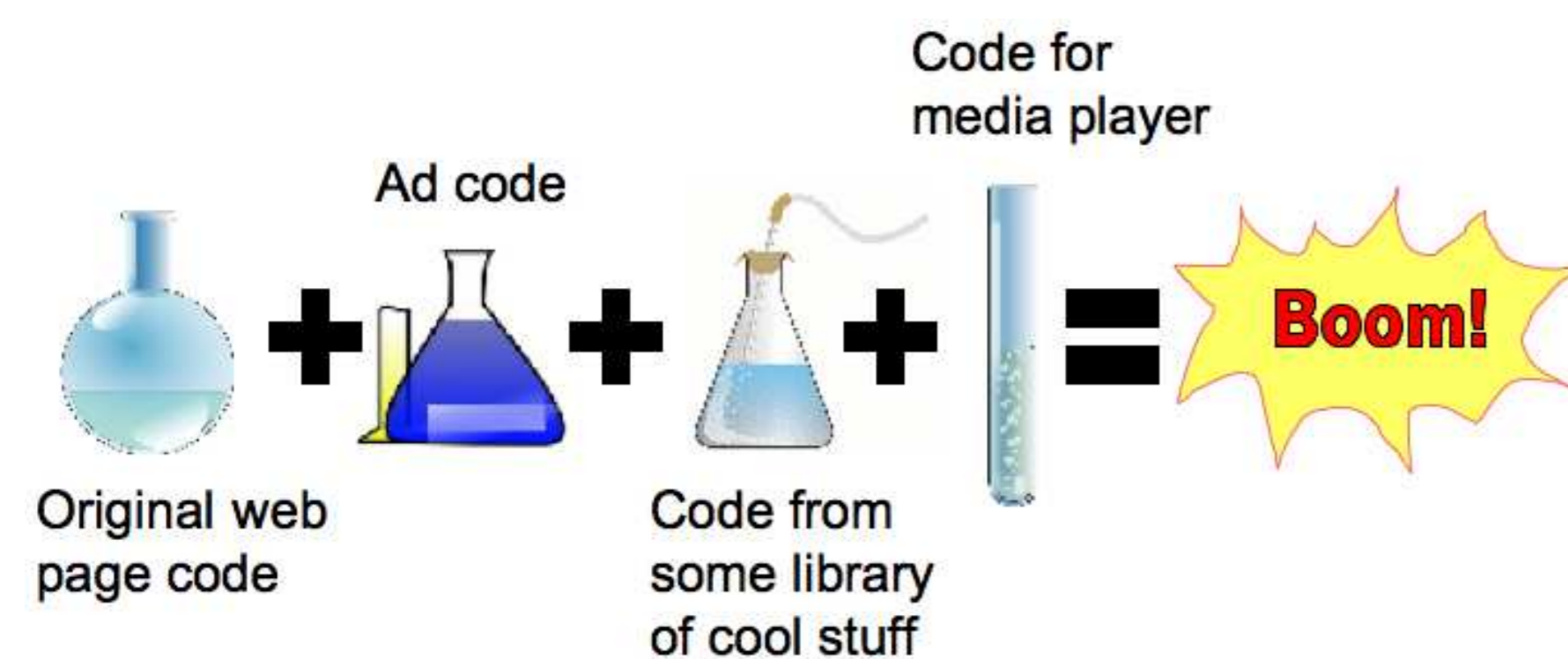


- Does not protect: the web page itself, anything that passes through the browser



2 The Same Origin Policy

The same origin policy states that scripts can only modify pages that share the same origin (domain, protocol, port)



- The origin of the script is the page that loads it, not the script's location
 - by including an ad, you've given the ad server access to anything on your domain
- Pages load a lot of code from external sources (ads, videos, counters, music, etc.)
- Users are often unaware of powers granted to every script
 - assume ad sever scripts can only display the ad

3 Security Policies for JavaScript

- Typically all-or-nothing: JavaScript is on or off in the browser
- Some settings for web annoyances such as popups
- Disabling JavaScript breaks many sites

Abstract

In recent years, it has become possible to create entire applications within the web browser. Technologies such as JavaScript, Flash, and DHTML have allowed us to build interfaces previously only seen in desktop applications, but they have also left us vulnerable to web-based attacks such as cross-site scripting (XSS). Existing security mechanisms such as JavaScript's same-origin policy and “sandbox” are no longer sufficient to protect users and data. This poster discusses some of the threats we now face as well as some potential ways we can defend against attacks.

The Problems

1 Cross Site Scripting

A security exploit where an attacker inserts malicious code into a page.

- Often done through an HTML form, directly or indirectly
- User may not need to click anything, simply viewing a malicious page is enough
- Can be persistent or non-persistent
- Enabled by insufficient data checking in web pages

2 Code Injection

- Used all the time for web advertisements
 - Avoided in other applications, but necessary for web economics!
- Browsers don't distinguish between sources for code
- One included script can include many others
- Scripts can modify webpages, other scripts, control plugins, etc.
- The same thing that makes it easy for us to include ads and other content is also making us more vulnerable to XSS

How do we ensure security when we need to run external code?

3 No Way to Run “Some” JavaScript

- User cannot choose what they want to run.
- Web developer may not realise that scripts are including extra code
 - There is no way to ensure that only known code is included
 - Previously safe scripts can become malicious without any warning

Future Work

- Distinguish code from different sources in the browser and choose what to run
- What is normal JavaScript code?
- Can safe and dangerous code be distinguished without breaking web pages?
- Automatic generation of appropriate security policies
- Assertions of security for code

Current Solutions

1 Fix the Code

Do careful input-checking to avoid XSS bugs.

- Replace `<>` & ' and " to be fully safe
- If you want some HTML, you can't just blindly strip “bad” tags:
 - Be careful of constructions like `<scr<script>ipt>`
 - Watch attributes such as `<p onmouseover=“misbehave()”>`
- Need to verify any code that you include too

And fixing the code has been so effective thus far with other security flaws, hasn't it?

2 Sign the Code

Code can be digitally signed by an authority.

- This ensures that code hasn't been changed since it was last seen
- But anyone can get an SSL certificate, so anyone can probably sign code
- Does not ensure it came from the correct source
- Does not ensure code is safe

3 Allow Finer-Grained Access Control

Allow users to choose exactly what JavaScript they want to allow and deny.

- Hard to design a good interface
- Requires fairly high-level knowledge to create an access control policy
- Users will turn off access control rather than figure out how to set decent policy
- Good policies have not been defined

4 Check for Known Bad Code

Can check all web pages for patterns known to be bad.

- Protects users from a number of known attacks, like a virus scanner
- Can't use many signatures without affecting performance
- New exploits or even variations may be missed
- Very hard to cover either vulnerabilities or legitimate code with a small range of signatures

5 Data Tainting

Idea is to mark data to indicate where it came from and control how it can be used.

- Option 1: secure values are “tainted” and cannot be passed to un-trusted code
- Option 2: User input is “tainted” and code here is constrained/not allowed
- Tainting is hard to do: marking all input is equivalent to validating all input
- May need cooperation on both client and server side