Robustness

• Security bugs are a fact of life

• How can we use access control to improve the security of software, so security bugs are less likely to be catastrophic?
Privilege separation

• How can we improve the security of software, so security bugs are less likely to be catastrophic?

• Answer: privilege separation. Architect the software so it has a separate, small TCB.
  – Then any bugs outside the TCB will not be catastrophic
To evaluate the security of Chromium's architecture, we took a three-pronged approach to evaluating the compatibility of Chromium's architecture. First, our implementation vulnerabilities that would have occurred in the rendering engine, even if the attacker's abilities and goals. The security architecture is designed to mitigate the most severe vulnerabilities that would have occurred in the rendering engine, even if the attacker's abilities and goals. The security architecture is designed to mitigate the most severe vulnerabilities that would have occurred in the rendering engine. Chromium's architecture with other browser architectures. For example, Chromium's architecture protects users from attack. We consider an attacker who knows an unpatched security vulnerability in the user's browser and sufficiently escapes a parameter to ShellExecute when handling the user's file system, helping protect the user from a drive-by malware: malicious web page exploits a browser bug to read/write local files or infect them with a virus.
The Chrome browser

Goal: prevent “drive-by malware”, where a malicious web page exploits a browser bug to read/write local files or infect them with a virus.

1. The attacker owns a domain name, say attacker.com.
2. The attacker is able to convince the user to visit his website by attracting a quarter of a million users for about $50 [1].
3. The attacker has a valid HTTPS certificate for the domain, and controls at least one host in the DNS blacklist [19]. The attacker has a valid HTTPS certificate for the domain, and controls at least one host in the DNS blacklist [19].
4. The attacker has a microdomain, such as a subdomain or a top-level domain, that the attacker can drive traffic to his site for convincing the user to visit attacker.com.
5. The attacker can send out spam e-mail, hosting popular content, or driving traffic to the user's site for convincing the user to visit attacker.com.
6. The attacker can purchase abilities to price himself at $5.

Typically, these abilities are sufficient validation of system calls and would be insufficient to mitigate the attacker's abilities and goals. The security architecture aims to prevent an attacker who compromises the browser from attacking other web sites.

The Chrome browser...
We took a three-pronged approach to evaluating the compatibility with existing web content. Typically, these abilities are suﬃcient for convincing the user to visit the user's web site. There are a number of techniques for convincing the user to visit attacker.com via advertising. It is diﬃcult to price this ability, but, in a previous study, we were able to attract a quarter of a million users for about $50 [1].

We consider an attacker who knows an attacker.com, such as driving traﬃc to attacker.com. For example, the attacker could send spam e-mail, hosting popular content, or promoting products. These abilities can be purchased for about $5. The attacker owns a domain name, say attacker.com, that has not yet been added to the browser's malware blacklist [19]. The attacker has a valid HTTPS cerﬁcate for the domain, and controls at least one host running attacker.com.

To evaluate the security of Chromium's architecture, we examine the disclosed browser vulnerabilities in Internet Explorer, Firefox, and Safari from the preceding year. For 70% of vulnerabilities that would have occurred in the browser without our implementation, we found security improvements due to architectural limitations. We deploy our implementation to millions of users world-wide. Third, we deploy our implementation to millions of users world-wide.

To conclude that the architecture extracts most of the security beneﬁts from the WebKit project. The tests our implementation of the architecture passes 99% of 1000K lines of code. The remaining vulnerabilities are due to architectural limitations. Second, we manually tested the implementation to verify that Chromium’s security architecture mitigates approximately 72% of vulnerabilities. These account for 700K lines of code.

We strictly limit the attacker to using the browser kernel interface. We ﬁnd that 38 of the 87 rendering engine vulnerabilities allowed an attacker to execute arbitrary code and would have been mitigated by Chromium’s architecture. These account for 700K lines of code. We determine which module would have emitted bitmaps of the rendered document. The rendering engine is designed to mitigate the most severe vulnerabilities that would have occurred in the rendering engine, but, Chromium’s architecture protects users from attack.

The architecture does not prevent an attacker who compromises the browser kernel. We ﬁnd that 72% of the disclosed vulnerabilities that allow an attacker to execute arbitrary code. If an attacker exploits such a vulnerability in the browser kernel, we determine which module would have emitted bitmaps of the rendered document. The rendering engine is designed to mitigate the most severe vulnerabilities that would have occurred in the rendering engine, but, Chromium’s architecture protects users from attack.

The rendering engine is designed to mitigate the most severe vulnerabilities that would have occurred in the rendering engine, but, Chromium’s architecture protects users from attack.
## Benefit of Secure Design

<table>
<thead>
<tr>
<th>Browser</th>
<th>Unknown unpatched vulnerabilities</th>
<th>SecurityFocus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Extremely critical (number / oldest)</td>
<td>Highly critical (number / oldest)</td>
</tr>
<tr>
<td>Internet Explorer 6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet Explorer 7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet Explorer 8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Internet Explorer 9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Firefox 3.6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Firefox 38</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Google Chrome 42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Opera 11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Safari 5</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
BE GOOD WITH YOUR MONEY
FROM THE BIG PICTURE
TO THE DETAILS THAT MATTER
Effortlessly manage your cash flow, budgets and bills from one place.

All-in-one? Done
From money and budgeting to customized tips and more—get a clear view of your total financial life.

Budgets? You betcha
Effortlessly create budgets that are easy to stick to. We even make a few for you.

Credit? Checked
Find out yours and learn how you can improve it. It's totally free.
Discuss with a partner

• How would you architect mint.com to reduce the likelihood of a catastrophic security breach?
  – E.g., where attacker steals all users’ stored passwords or empties out all their bank accounts overnight
Summary

• Access control is a key part of security.

• Privilege separation makes systems more robust: it helps reduce the impact of security bugs in your code.

• Architect your system to make the TCB unbypassable, tamper-resistant, and verifiable (small).
“Security is economics.”
What does this program do?
What can this program do? Can it delete all of your files? YES. Why?
“Least privilege.”
Touchstones for *Least Privilege*

- When assessing the security of a system’s design, identify the *Trusted Computing Base (TCB)*.
  - What components does security rely upon?
- Security requires that the TCB:
  - Is correct
  - Is complete (can’t be bypassed)
  - Is itself secure (can’t be tampered with)
- Best way to be assured of correctness and its security?
  - **KISS** = *Keep It Simple, Stupid!*
  - Generally, Simple = *Small*
- One powerful design approach: privilege separation
  - Isolate privileged operations to as small a component as possible
  - (See lecture notes for more discussion)
Check for Understanding

• We’ve seen that PC platforms grant applications a lot of privileges
• Quiz: Name a platform that does a better job of least privilege
“Ensure complete mediation.”
Ensuring Complete Mediation

• To secure access to some capability/resource, construct a *reference monitor*
• Single point through which all access must occur
  – E.g.: a network firewall
• Desired properties:
  – Un-bypassable (“complete mediation”)
  – Tamper-proof (is itself secure)
  – Verifiable (correct)
  – (Note, just restatements of what we want for TCBs)
• One subtle form of reference monitor flaw concerns *race conditions* …
procedure withdrawal(w)
    // contact central server to get balance
1. let b := balance

2. if b < w, abort

    // contact server to set balance
3. set balance := b - w

4. dispense $w to user

**TOCTTOU Vulnerability**

**TOCTTOU = Time of Check To Time of Use**
public void buyItem(Account buyer, Item item) {
    if (item.cost > buyer.balance)
        return;
    buyer.possessions.put(item);
    buyer.possessionsUpdated();
    buyer.balance -= item.cost;
    buyer.balanceUpdated();
}
NO LONE ZONE
SAC TWO MAN POLICY
MANDATORY

CAUTION

DO NOT KEY AT XX 10 L/D
EXCEPT IN CASE OF AN
EMERGENCY — MUST BE AT
LEAST 6 FT FROM NSL.
“Separation of responsibility.”
Coming Up …

• Homework 1 due Monday
• Project 1 is now available