CMPSC 447
Privilege Separation

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University
Our Goal

• In this course, we want to develop techniques to **prevent** vulnerabilities from being created
  ‣ Prevent flaws
  ‣ Prevent access or exploitation of flaws
    • **Privilege separation** prevents access and exploitation, but moving sensitive data to another address space
OpenSSH

- Secure remote login software
- Client and server architecture

- Client and server establish secure channel using private key stored on server
- Enabling client to login using password without fear of password sniffing
OpenSSH

• Is security-critical software
  ‣ Runs as root – needs to be able to login users
  ‣ Stores and uses a private key that if lost could enable machine spoofing
  ‣ Has access to user passwords that may apply to any machine in the domain
  ‣ Launches user processes under the authenticated user ID, which requires root privilege

• That is OK, OpenSSH is written in C, so I am sure there are no problems
OpenSSH

• Is secure-critical software
  ‣ Runs as root – needs to be login users
  ‣ Stores and uses a private key that if lost could enable machine spoofing
  ‣ Has access to user passwords that may apply to any machine in the domain
    ‣ Launches user processes under the authenticated user ID, which requires root privilege

• That is OK, OpenSSH is written in C, so I am sure there are no problems
  ‣ That was a joke…
OpenSSH Vulnerabilities

• Circa 2002

  ‣ CVE-2000-0525 – does not properly drop privileges, allowing local users to execute arbitrary commands

  ‣ CVE-2001-0872 – does not properly cleanse critical environment variables, allowing local users to gain root

  ‣ CVE-2001-1029 – does not drop privileges before reading the copyright files, allows local users to read arbitrary files

  ‣ CVE-2002-0059 – releases certain memory more than once ("double free"), allowing remote attackers to execute arbitrary code

  ‣ CVE-2002-0083 – Off-by-one error allows remote malicious servers to gain privileges.
Retroactive Security

• “Penetrate and patch” as flaws are exposed as vulnerabilities
OpenSSH

• After patching enough of these and other vulnerabilities, what is the impact on?
  ‣ Preventing privilege escalation (to root)
  ‣ Protecting program secrets
OpenSSH

- After patching enough of these and other vulnerabilities, what is the impact on?
  - Preventing privilege escalation (to root)
  - Protecting program secrets

- Not sure whether there are other latent flaws that can be exploited (vulnerabilities)?

- Can we make some change to the design to make such flaws much more difficult to access or exploit?
Retrofit Security Mechanisms

• Several codebases have been retrofit with security mechanisms
  ▸ X Server, postgres, Apache, OpenSSH, Linux Kernel, browsers, etc.

• With a variety of security mechanisms:
  ▸ Privilege separation, Authentication, Auditing, Authorization, etc.
Privilege Separation

• Isolate parts of a program into separate protection domains each with
  ‣ Access to a subset of the program data
  ‣ Different system privileges (access rights)

• Goals
  ‣ Small amount of code with sensitive data and privileges
  ‣ Rest of code can run with basic (low) data and privileges

• What parts of code need access to sensitive data and privileges in OpenSSH?
Privilege Separation

• What parts of code need access to sensitive data and privileges in OpenSSH?
  ‣ Code that needs access to root privileges
    • to change UID of child process (integrity)
  ‣ Code that needs access to critical secrets
    • For setting up secure channels and password authentication (secrecy)
Privilege Separation

• How do we take a monolithic program and create one or more privilege-separated components?
Privilege Separation

• How do we take a monolithic program and create one or more privilege-separated components?
  ‣ Need to identify privileged data in your program
  ‣ Integrity
    • Must not be impacted by adversary inputs
    • E.g., Data used in operations that require ‘root’ privileges
  ‣ Secrecy
    • Must never be leaked to adversaries
    • SSH private keys

• Then, you need to determine code (functions) that operate on such data
Privilege Separation

• How do we take a monolithic program and create one or more privilege separated components?

Untrusted (receive client input)

Integrity (root ops)

Secrecy (authenticate)
Information Flow

• One security property for evaluating programs is information flow

• Use information flow to control
  ‣ Secrecy
  ‣ Integrity
Information Flow Secrecy

- One security property for evaluating programs is information flow

- Information Flow Secrecy
  - Subjects – Subject Level $L_S$
  - Objects – Object Level $L_O$
  - $L_S \geq L_O$ for Subject to read an object
  - $L_S \leq L_O$ for Subject to write an object
Information Flow Secrecy

• One security property for evaluating programs is information flow

• Information Flow Secrecy
  ‣ Subjects – Subject Level $L_S$
  ‣ Objects – Object Level $L_O$
  ‣ $L_S \geq L_O$ for Subject to read an object
  ‣ $L_S \leq L_O$ for Subject to write an object
One security property for evaluating programs is information flow.

**Information Flow Integrity**

- Subjects – Subject Level $I_S$
- Objects – Object Level $I_O$
- $I_S \leq I_O$ for Subject to read an object
- $I_S \geq I_O$ for Subject to write an object
Privilege Separation

- How do we take a monolithic program and create one or more privilege separated components?

- Untrusted (receive client input)
- Integrity (root ops)
- Secrecy (authenticate)
OpenSSH Privilege Separation

• What parts of code need access to sensitive data and privileges in OpenSSH?
  ‣ Code that needs access to root privileges
    • to change UID of child process (integrity)
  ‣ Code that needs access to critical secrets
    • For setting up secure channels and password authentication (secrecy)

• How would you privilege separate these functionalities from the rest of OpenSSH?
OpenSSH Privilege Separation

- How OpenSSH looks after privilege separation

Figure 4: Overview of privilege separation in OpenSSH. An unprivileged slave processes all network communication. It must ask the monitor to perform any operation that requires privileges.
Separation Issues

• Information Flow Issues
  ‣ Secrecy
    • Secret component must return authentication result
    • Filter secrets from the response (declassify)
  ‣ Integrity
    • High integrity component must receive input
    • Validate integrity of untrusted inputs (endorsement)
  ‣ Both
    • In many cases the secret data is also high integrity
    • What then?
Separation Issues

• Information Flow Issues
  ‣ Secrecy
    • Secret component must return authentication result
    • Filter secrets from the response (declassify)
  ‣ Integrity
    • High integrity component must receive input
    • Validate integrity of untrusted inputs (endorsement)
  ‣ Both
    • In many cases the secret data is also high integrity
    • What then? Both declassification and endorsement
Declassification and Endorsement

• Declassification
  ‣ Remove as much impact from the secret as possible
  ‣ **Example:** Password checking
  ‣ What is the minimal impact of password value of checking result?

• Endorsement
  ‣ Remove influence of untrusted input as much as possible
  ‣ **Example:** Untrusted request
  ‣ What is the minimal influence of an untrusted input on request processing?
Implementing Privilege Separation

• Getting privilege separation to work correctly is non-trivial
  ‣ Need to turn a function call
  ‣ Into a remote procedure call

• One challenge
  ‣ Data in caller and callee are no longer in the same protection domains
  ‣ **Example**: `int check(char *passwd)`
    ‣ Normally, pass as a pointer to a memory location “passwd”
    ‣ Now, need to copy memory from caller to callee
Implementing Privilege Separation

• Complex task for programmers

• Simplify by specifying as a remote procedure call (RPC)
  ‣ RPC in terms of interface description language (IDL)
    • Marshalling (on caller) and unmarshalling (on callee) input arguments
    • Reverse on return
  ‣ Performance impact
    • What if there are many RPCs to the privilege separated domain?
Implementing Privilege Separation

• Some Issues

• Synchronization cost
  ‣ Suppose the original function call passes a reference to a large structure
    • int fn(struct t1 *t);
  ‣ But, only uses one field – do we need to copy it all?

• Multithreading
  ‣ What if the two domains (caller and callee) have concurrent access to the same data?
Privilege Separation

• Complex task for programmers
  ‣ We would like to automate this task (next time)
Privilege Separation In Use

• Browsers

Fig. 1. Browser Blueprint. *It shows typical interactions between browser components in processing a web page.*
Some Browser Goals

• Isolate web page processing from network processing
• Isolate browser components that need filesystem access from those that do not
• Isolate the processing of one web page from another
• Isolate the execution of browser processing from the JavaScript engine
• Isolate the execution of browser processing from browser extensions
Browser Separation

- Firefox has 20+ components

<table>
<thead>
<tr>
<th>Comp#</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>136</td>
<td>367</td>
<td>74</td>
<td>155</td>
<td>32</td>
<td>3</td>
<td>131</td>
<td>21</td>
<td>77</td>
<td>366</td>
<td>10</td>
<td>269</td>
<td>763</td>
<td>7</td>
<td>223</td>
<td>24</td>
<td>137</td>
<td>478</td>
<td>24</td>
<td>188</td>
<td>53</td>
</tr>
</tbody>
</table>

Table 2. Kilo-lines of Source Code in Firefox Components. In our experiments, we consider the following components: 0. NETWORK, 1. JS, 2. PARSER, 3. DOM, 4. BROWSER, 5. CHROME, 6. DB, 7. DOC SHELL, 8. EDITOR, 9. LAYOUT, 10. MEMORY, 11. MODULES, 12. SECURITY, 13. STORAGE, 14. TOOLKIT, 15. URIO LOADER, 16. WIDGET, 17. GFX, 18. SPELLCHECKER, 19. NSPR, 20. XPCONNECT, and 21. OTHERS.

- That “security” is the largest is not entirely a good sign
- Browsers are as complex as operating systems
Take Away

• Programs may have lots of ad hoc bugs that prevent it from running securely
  ‣ However, there are certain security goals we may want to achieve
    • Focusing on the goals may make the program easier to protect through security mechanisms targeted for those goals
  ‣ One such security mechanism
    • Privilege separation: Isolate code with extra privileges or sensitive resources from rest of the program – call via small API