CMPSC 497
Attack Surface

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

• … could be anywhere in a program
  ‣ Given the definition of a vulnerability, does that give us any insight into where we should look for vulnerabilities?
Security Problems

• … could be anywhere in a program
  ‣ Given the definition of a vulnerability, does that give us any insight into where we should look for vulnerabilities?
    • Software flaw
    • Accessible to an adversary
    • Who can exploit the vulnerability
Security Problems

• Could be anywhere in a program
  ‣ Given the definition of a vulnerability, does that give us any insight into where we should look for
    • Software flaw
    • Accessible to an adversary
    • Who can exploit the vulnerability

• Typically, we look for software flaws (static, symbolic, fuzzing), but today we will consider “adversary accessibility”
Attack Surface

- After Microsoft faced several large-scale vulnerability exploits in the early 2000s
- They began to consider how to prevent such vulnerabilities
After Microsoft faced several large-scale vulnerability exploits in the early 2000s

They began to consider how to prevent such vulnerabilities

Michael Howard of Microsoft defined the term “attack surface”

A program’s attack surface consists of the entry points that are accessible to an adversary

What does this mean?
Entry Points

• What’s an entry point?
Howard proposed the notion of a relative attack surface quotient (RASQ) metric

- The idea is that we can use the metric to compare systems to determine which has a larger relative attack surface

- The metric lists a set of entry points that you should be concerned about minimizing as a system distributor
  - Where entry points are approximated by unsafe system configuration options
Relative Attack Surface Metric

- Open (TCP/UDP) sockets - descriptors
- Open RPC endpoints - descriptors
- Open named pipes - descriptors
- Services - daemons
- Services running by default - daemons
- Services running as SYSTEM (or root) - daemons
- Active Web handlers – web server components
- Active ISAPI filters – web server add-ins
- Dynamic web pages – files
- Executable vdirs – directories for scripts
Relative Attack Surface Metric

- Enabled accounts – accounts
- Enabled accounts in admin group – accounts
- Null sessions to pipes and shares – anonymous connections allowed
- Guest account enabled – accounts (special)
- Weak ACLs in FS – files allowing “full control” to everybody
  - “Full control” is the moral equivalent of UNIX rwxrwxrwx permissions
- Weak ACLs in Registry – registry keys that allow “full control” to everybody
- Weak ACLS on shares – Directories that can be shared by remote users that allow “full control” to everybody
- VBScript, JavaScript, Active X enabled – applications enabled to execute Visual Basic Script, JavaScript or Active X controls
Relative Attack Surface Metric

- Essentially, you would **count the number of unsafe instances** of the above in your system to determine an estimate of the system’s attack surface
  - Also combined with **weights per item**, but numeric weights that are meaningful are often hard to predict effectively

- Windows systems saw a gradual reduction in attack surface metric values in the 2000s
  - But, attacks kept coming, exploiting new vulnerabilities

- Can we say something about programs individually with respect to their attack surfaces?
Can we say something about programs individually with respect to their attack surfaces?

What do we need to identify to determine the adversary-accessible entry points of a program?
Program Attack Surface

- Can we say something about programs individually with respect to their attack surfaces?

- What do we need to identify to determine the adversary-controlled entry points of a program?
  - Identify the relevant subset of system resources that can be used in an attack (are or could be controlled by an adversary)
  - Identify when such resources may be used by the program (program entry points)

- Is it possible to compute such information?
Program Entry Points

• What’s a program entry point?
• Programs obtain information from external sources (e.g., files and network sockets), and the program statements that access such external sources are entry points
  ▸ What’s an example of an entry point?
Program Entry Points

- What’s a program entry point?
- Programs obtain information from external sources (e.g., files and network sockets), and the program statements that access such external sources are entry points
  - What’s an example of an entry point?
    - System calls provide the sources for gaining most external information
    - But, for attack surfaces, we focus on the statements that a program makes to the individual library calls that lead to system calls
Library Calls as Attack Surface

• Why should we use library calls for the attack surface?
Library Calls as Attack Surface

- Why should we use library calls for the attack surface?
  - A system call appears once in your process
    - In the library (libc)
  - There are several program statements that invoke each library call
    - Only a subset of these may be adversary accessible

- E.g., consider the “open” system call
  - May be invoked via “open” or “fopen” library call
    - fopen(input_pathname, …) vs. fopen(“/bin/sh”)

- How many library calls access adversary-controlled data?
Library Calls as Attack Surface

- Which library calls should constitute a program’s attack surface?
Library Calls as Attack Surface

• Which library calls should constitute a program’s attack surface?
  ‣ All of them
    • At some point, any call may access adversary-controlled data
    • So test them all
Library Calls as Attack Surface

• Which library calls should constitute a program’s attack surface?
  ‣ All of them
    • At some point, any call may access adversary-controlled data
    • So test them all
  ‣ Only ones that actually may access adversary-controlled resources
    • Only need to test a subset of such each program’s entry points to evaluate the attack surface
    • How do you determine which may access adversary-controlled resources?
Library Calls as Attack Surface

• Which library calls should constitute a program’s attack surface?
  ‣ All of them
    • At some point, any call may access adversary-controlled data
    • So test them all
  ‣ Only ones that actually may access adversary-accessible resources
    • Only need to test a subset of such each program’s entry points to evaluate the attack surface
    • How do you determine which may access adversary-controlled resources?
      ‣ An attack surface can be deployment-specific
Program Attack Surface

- Program system calls accessible to an adversary

Apache httpd

- Read Config File
- Read User HTML Page
- Load Shared Library
- Net Input (HTTP)

/etc/apache/httpd.conf
/home/user/www/*.html
/usr/lib/httpd/mod_ssl.so
TCP socket 1.2.3.4:80

Malicious Unprivileged User

Attack Surface!
System TCB Attack Surface

- Only 13.8% of total entry points for Linux system services were accessible to adversaries at all
  - Only 3.8% for read/write operations
  - Listing all entry points as attack surface would be a huge overapproximation

<table>
<thead>
<tr>
<th>Total Entrypoints</th>
<th>Accessible to Adversaries</th>
<th>Potentially Vulnerable (overt permissions)</th>
<th>Previously Known Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2138</td>
<td>295</td>
<td>81</td>
<td>35</td>
</tr>
</tbody>
</table>

- Found via runtime testing with Linux package test suites – lower bound
## System TCB Attack Surface

<table>
<thead>
<tr>
<th>TCB Type</th>
<th>Total Entry</th>
<th>Viol. Entry</th>
<th>Program</th>
<th>Component</th>
<th>Overt Violating Port</th>
<th>Object Type</th>
<th>Bug ID / Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>apcsp.t</code></td>
<td>3</td>
<td>3</td>
<td>apcsp</td>
<td>1 Unix socket</td>
<td>system dbus daemon</td>
<td>CVE-2009-0788</td>
<td></td>
</tr>
<tr>
<td><code>avahi.t</code></td>
<td>38</td>
<td>14</td>
<td>avahi-daemon</td>
<td>3 * Unix socket</td>
<td>avahi</td>
<td>CVE-2007-3472, CVE-2008-0461</td>
<td></td>
</tr>
<tr>
<td><code>consolekit.t</code></td>
<td>27</td>
<td>3</td>
<td>console-kit-daemon</td>
<td>1 * Unix socket</td>
<td>system dbus daemon</td>
<td>CVE-2010-4664</td>
<td></td>
</tr>
<tr>
<td><code>cupid.t</code></td>
<td>56</td>
<td>10</td>
<td>cupid</td>
<td>1 TCP socket</td>
<td>curl</td>
<td>CVE-2009-0540</td>
<td></td>
</tr>
<tr>
<td><code>devicekit.Librdt</code></td>
<td>72</td>
<td>6</td>
<td>udisks-daemon</td>
<td>1 * 4 Unix socket</td>
<td>system dbus daemon</td>
<td>CVE-2010-0746</td>
<td></td>
</tr>
<tr>
<td><code>devicekit-power.t</code></td>
<td>97</td>
<td>7</td>
<td>upower</td>
<td>1 * 2 Unix socket</td>
<td>system dbus daemon</td>
<td>CVE-2010-0746</td>
<td></td>
</tr>
<tr>
<td><code>dhept.t</code></td>
<td>18</td>
<td>2</td>
<td>dhept</td>
<td>1 raw socket</td>
<td>system dbus daemon</td>
<td>CVE-2009-0549</td>
<td></td>
</tr>
<tr>
<td><code>getty.t</code></td>
<td>188</td>
<td>28</td>
<td>getty</td>
<td>1 file read</td>
<td>inter-sys-run.t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>hald.t</code></td>
<td>188</td>
<td>28</td>
<td>hald</td>
<td>1 Unix socket</td>
<td>system dbus daemon</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>init.t</code></td>
<td>217</td>
<td>25</td>
<td>init</td>
<td>1 * 2 file read</td>
<td>inter-sys-run.t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>local-login.t</code></td>
<td>152</td>
<td>10</td>
<td>login</td>
<td>1 * 2 file read</td>
<td>inter-sys-run.t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>login.t</code></td>
<td>152</td>
<td>10</td>
<td>login</td>
<td>1 file read</td>
<td>user auth</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>logrotate.t</code></td>
<td>78</td>
<td>11</td>
<td>logrotate</td>
<td>1 * 2 file read</td>
<td>user home dir t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>NetworkManager.t</code></td>
<td>56</td>
<td>13</td>
<td>NetworkManager</td>
<td>1 netlink socket</td>
<td>system dbus daemon</td>
<td>CVE-2009-0778</td>
<td></td>
</tr>
<tr>
<td><code>netd.t</code></td>
<td>24</td>
<td>4</td>
<td>netd</td>
<td>1 udp socket</td>
<td>netd</td>
<td>CVE-2001-0414</td>
<td></td>
</tr>
<tr>
<td><code>resovrcd.t</code></td>
<td>17</td>
<td>9</td>
<td>resovrcd</td>
<td>1 * 3 file read</td>
<td>generic - all types</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>rkt-daimon.t</code></td>
<td>20</td>
<td>9</td>
<td>rkt-daimon</td>
<td>1 dir search</td>
<td>user home dir t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>sudo.t</code></td>
<td>78</td>
<td>11</td>
<td>sudo</td>
<td>1 * 2 file read</td>
<td>system dbus daemon</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>syslogd.t</code></td>
<td>29</td>
<td>1</td>
<td>syslogd</td>
<td>1 * 2 file read</td>
<td>generic - log files</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>system-dbus-daemon.t</code></td>
<td>62</td>
<td>15</td>
<td>system-dbus-daemon</td>
<td>1 netlink socket</td>
<td>system dbus daemon</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>udev.t</code></td>
<td>317</td>
<td>25</td>
<td>udev</td>
<td>1 * 2 file read</td>
<td>user home t</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>xdm.c</code></td>
<td>56</td>
<td>16</td>
<td>xdm-conn</td>
<td>1 * 2 file read</td>
<td>xdm</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>user.c</code></td>
<td>43</td>
<td>18</td>
<td>Xorg</td>
<td>1 * 3 file read</td>
<td>xdm</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total:** 2138 attacks on 296 unique programs, with 81 memory, 25 reached.
Case Study: Apache

- Launched in 1998
- In 1998, first vulnerability found in network parsing
  - Network input is an obvious attack surface point
- 1999 – log files
- 2001 – CGI script output
- 2002 – user-defined HTML files (~//public_html)
- 2004 – user-defined configuration files (.htaccess)
- Above 4 entry points are less obvious
  - Still an attack surface for user-defined configuration files in 2014
Program Attack Surface

- Program system calls accessible to an adversary

Apache httpd

- Read Config File: /etc/apache/httpd.conf
- Read User HTML Page: /home/user/www/*.html
- Load Shared Library: /usr/lib/httpd/mod_ssl.so
- Net Input (HTTP)
  - TCP socket 1.2.3.4:80

Malicious Unprivileged User

Attack Surface!
Which Are Adversary Controlled?

- How do we identify the relevant subset of system resources that can be used in an attack?
  - These are presumably controlled by an adversary of the program
  - Who are the program’s adversaries and which resources might they control?
Which Are Adversary Accessible?

• How do we identify the relevant subset of system resources that can be used in an attack?
  ‣ These are presumably controlled by an adversary of the program
  ‣ Who are the program’s adversaries and which resources might they control?
    • Adversaries – who is your adversary?
    • Or perhaps, who do you not need to trust?
What Must a Program Trust?

- What software on a system must your program trust?
  - Kernel (and below, such as BIOS and bootloader)
  - Some user space programs

- What makes you have to trust another user space process?
For Linux Services

- What software on a system must your program trust?
  - Kernel (and below, such as BIOS and bootloader)
    - Modify kernel resources – kernel code/data, drivers and sysfs, kernel resources (memory)
  - Some user space programs
    - Modify your executable file
    - May modify other files you depend on, but that is ad hoc

- So how do you identify programs that may modify kernel resources and executable files?
Access Control – DAC, MAC

- Discretionary Access Control (DAC)
  - `rwx` bits

- Mandatory Access Control (MAC)
  - Labels for processes (subjects), resources (objects)
Access Control for Control

• Suppose only root is trusted
  ‣ Permissions to files owned by target
    • F1: rwx------
    • F2: rwxr-xr-x
    • F3: rwxrwxrwx
  ‣ Which files can be sources of possible attacks?
    • Are adversary-controlled files
  ‣ How does this determine the attack surface?
Access Control for Accessibility

- Suppose only root is trusted
  - Permissions to files owned by target
    - F1: rwx------
    - F2: rwxr-xr-x
    - F3: rwxrwxrwx
  - Which files can be sources of possible attacks?
    - Are adversary-controlled files
  - How does this determine the attack surface?
    - Entry points (library calls) that use files like F3
Access Control for Accessibility

• Given an access control policy and a target program
  ‣ Find the other programs that can modify a critical resource
    • For Linux services, kernel resource or executable file of target program
  ‣ Programs that can modify those are trusted programs
  ‣ All others may be adversaries

• Can compute a tighter (more accurate) attack surface using mandatory access control

• Is that too conservative a view of adversaries?
Access Control for Accessibility

• Given an access control policy and a target program
  ‣ Find the other programs that can modify a critical resource
    • For Linux services, kernel resource or executable file of target program
      ‣ Those a trusted programs
      ‣ All others may be adversaries
  • Is that too conservative a view of adversaries?
    ‣ Actually, no. Data presented previously was based on that view
Access Control for Accessibility

- Given an access control policy and a target program
  - Find the other programs that can modify a critical resource
    - For Linux services, kernel resource or executable file of target program
      - Those a trusted programs
      - All others may be adversaries
    - Is that too conservative a view of adversaries?
      - Actually, no. *In addition, other resources that the target program creates may be adversary controlled.*
Access Control for Accessibility

• Given an access control policy and a target program
  ‣ Find the other programs that can modify a critical resource
    • For Linux services, kernel resource or executable file of target program
      ‣ Those a trusted programs
      ‣ All others may be adversaries

• Is that too conservative a view of adversaries?
  ‣ Actually, no. In addition, other resources that the target program creates may be adversary controlled. E.g., log files and environment variables (e.g., ShellShock)
Access Control for Accessibility

• How might we leverage attack surfaces?
Access Control for Accessibility

• How might we leverage attack surfaces?
  ‣ Defend the attack surface systematically
  ‣ Focus fuzz testing
  ‣ Focus information flow testing
  ‣ Detect unexpected attack surfaces at runtime
  ‣ Verify defenses at attack surfaces
  ‣ …
Take Away

- Finding vulnerabilities is hard
  - Entails finding a software flaw that is exploitable
  - And also accessible to adversaries

- Attack surfaces examine understanding the accessibility of programs to adversaries detect vulnerabilities
  - Identify adversaries of programs
  - Identify resources that are controlled by adversaries
  - Identify entry points where such resources are used

- Improve software testing and defenses