Defense Strategies

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

September 7, 2011
• Problem – Strategies to prevent attacks
• Programs: Prevent overflows
• Systems: Confine process interactions (MAC)
• Still may be some attacks – where?
• Assurance
Our Goal

• In this course, we want to develop techniques to detect vulnerabilities and fix them automatically

• What’s a vulnerability?

• How to fix them?

• We will examine the second question today
• How do you define computer ‘vulnerability’?
  ‣ Flaw
  ‣ Accessible to adversary
  ‣ Adversary has ability to exploit
Anatomy of Control Flow Attacks

- Two steps
- First, the attacker changes the control flow of the program
  - In buffer overflow, overwrite the return address on the stack
  - What are the ways that this can be done?
- Second, the attacker uses this change to run code of their choice
  - In buffer overflow, inject code on stack
  - What are the ways that this can be done?
Anatomy of Control Flow Attacks

- Two steps

- First, the attacker changes the control flow of the program
  - In buffer overflow, overwrite the return address on the stack
  - How can we prevent this?

- Second, the attacker uses this change to run code of their choice
  - In buffer overflow, inject code on stack
  - How can we prevent this? ROP conclusions
StackGuard

Figure 2: StackGuard Defense Against Stack Smashing Attack
StackGuard

- How do you think that Stackguard is implemented?
More Smashing

- Pincus and Baker, Beyond Stack Smashing, IEEE S&P, 2004

- Pointer modification
  - Function pointers and exception handlers
  - Data pointer – modify arbitrary memory location
  - Virtual functions – overwrite pointers to these functions

- Provide payload from earlier operation
  - Environment variables
  - Arc injection – provide exploit code on command line
StackGuard

- Related defenses
  - Reorder local variables on stack
  - Protect return address when set
  - Canaries to protect pointers

<table>
<thead>
<tr>
<th>Vulnerable Program</th>
<th>Result Without StackGuard</th>
<th>Result With Canary StackGuard</th>
<th>Result With MemGuard StackGuard</th>
</tr>
</thead>
<tbody>
<tr>
<td>dip 3.3.7n</td>
<td>root shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
<tr>
<td>elm 2.4 PL25</td>
<td>root shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
<tr>
<td>Perl 5.003</td>
<td>root shell</td>
<td>program halts irregularly</td>
<td>root shell</td>
</tr>
<tr>
<td>Samba</td>
<td>root shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
<tr>
<td>SuperProbe</td>
<td>root shell</td>
<td>program halts irregularly</td>
<td>program halts</td>
</tr>
<tr>
<td>umount 2.5k/libc 5.3.12</td>
<td>root shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
<tr>
<td>wwwcount v2.3</td>
<td>httpd shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
<tr>
<td>zgv 2.7</td>
<td>root shell</td>
<td>program halts</td>
<td>program halts</td>
</tr>
</tbody>
</table>

Table 1: Protecting Vulnerable Programs with StackGuard
Other Overflows

- Heap overflows
  - Overwrite data or metadata
  - Defend in manner similar to buffer overflows

- Integer overflows
  - No systematic defense

- Input filtering
  - No systematic defense
Confining Processes

- Mandatory Access Control
  - SELinux
Attack Surfaces

- Attack Surfaces
Assurance

• Problem: Prove to a third party that your system provides particular security protections

• Challenges
  ‣ What security protections are provided?
  ‣ How do we prove that such protections are designed/implemented correctly?

• Additionally
  ‣ How do we even know what security protections would be valuable to have?
Orange Book

• Part of Rainbow Series from NCSC
  ‣ Covers many facets of computer security

• AKA Trusted Computer System Evaluation Criteria
  ‣ To evaluate, classify, and select among computer systems

• Defines both
  ‣ Criteria for different categories of secure systems
  ‣ Evaluation requirements to satisfy those criteria
Orange Book

- Categories of Security Covered
- Access control
  - Mandatory and discretionary
- Accountability
  - Authentication and audit
- Assurance
  - Development and deployment
- Documentation
  - “Whoomp factor”
Orange Book

- Most important results were a set of security targets
- D – Minimal protection
- C – Discretionary protection
- B – Mandatory protection
- A – Verified Protection
Most important result were a set of security targets

- B – Mandatory protection
  - B1 – Labeled Security: MAC covers some exported
  - B2 – Structured Security: Comprehensive MAC and covert channels
  - B3 – Security Domains: Satisfies Reference Monitor

- A – Verified Protection
  - A1 – Verified Design: B3 Function with formal assurance
  - Beyond A1
Protection Requirements

- B2 – Structured Security (3.2, Pg. 27)
- Security policy (protections)
  - Object reuse – clean before reuse
  - Labels – TCB labels all subjects and objects
    - Label Integrity – Labels match levels
    - Export – Single level and Multi-level
  - MAC – Enforce over all resources
  - Accountability: Trusted Path and Audit
Assurance Requirements

• B2 – Structured Security (3.2, Pg. 27)

• Assurance
  ‣ Operational
    • TCB protected from tampering
    • Periodically validate integrity
    • Covert storage channels (detect and mitigate/eliminate)
  ‣ Lifecycle
    • Testing – to find if works as claimed
    • Formal model – of security policy (i.e., function) design and configuration
  ‣ Documentation
Common Criteria

• Problem with Orange Book was the binding of function (security policy) and assurance

• The Common Criteria separates these
  ‣ Security Targets
  ‣ Assurance Levels

• Although these are at least partially bound by protection profiles
Labeled Security Protection

• Essentially the B2 Security Policy

• Assurance
  ‣ Expected to EAL3

• Covering
  ‣ Configuration
  ‣ Delivery
  ‣ Development (High-level design)
  ‣ Guidance (Administration)
  ‣ Testing
  ‣ Vulnerability Assessment
Current Approach to Assurance

• Document from initial design
  ‣ Build system from formal models
  ‣ E.g., seL4 and VAX VMM

• Document existing system
  ‣ Collect design, config, admin, etc. from existing system
  ‣ E.g., Windows, Linux, Solaris, etc.

• Assurance level of existing systems are limited to EAL4 in practice
Current Approach to Assurance

- Document from initial design
  - Build system from formal models
  - E.g., seL4 and VAX VMM

- Document existing system
  - Collect design, config, admin, etc. from existing system
  - E.g., Windows, Linux, Solaris, etc.

- Assurance level of existing systems are limited to EAL4 in practice
Limited Impact on Systems

- **Old Claim:** Full assurance for existing systems is impractical

- **Old world**
  - Assurance is a design-time task
    - All deployments are proven secure
  - Few components are trusted to make security decisions
    - But trusted completely
  - Development is either done in a unified way or few guarantees are possible
    - Composition of modules or independent tasks (config and design) is non-trivial
Goal: Defend Existing Systems

- New Claim: Given a set of components, determine whether they defend themselves proactively

- New world
  - Can assurance be done at design and deployment?
    - All deployments are consistent with defenses
  - Can we work with layers of TCBs?
    - Trust monotonically decreased in a logical way
  - Can we compose a system from independent components?
    - Analysis of what is built
Summary

• We envision that program compromises are prevented in several ways
  ‣ Program integrity
  ‣ Mandatory access control
  ‣ Attack surfaces

• However, the results of these defensive efforts must be unified
  ‣ Assurance

• But, current assurance techniques do not match the practical challenges in software development