Assignment 2 Tutorial

Security Mechanisms

Papadogiannakis Manos  Spiridakis Konstantinos
papamano@csd.uoc.gr  spkostas@csd.uoc.gr

CS-457: Introduction to Information Security Systems
Computer Science Department
University of Crete
Outline

- Access Control
- Intrusion Detection
- Secret Sharing
Access Control System
Access Control System

• Create an access control system for system calls

• Restrict an executable from invoking a large number of system calls
Overview

Access Control System

(1) Read

Instructions File

(2) Launch

(3) Monitor

Tracee

System Call

Linux Kernel
Overview

./access_control instructions.txt suspicious.out

1. Read rules file
2. Launch executable
3. Monitor system calls
4. Kill when violation
Instructions File

- Given as **input** to your system
- Each line contains a **restriction** on system calls

```
instructions.txt

mmap 6
gettid 3
getpid 8
read 0
```

Not allowed to be called
Tracing

● Use `ptrace()` to monitor system calls

```c
#include <sys/ptrace.h>

long ptrace(enum __ptrace_request request, pid_t pid,
             void *addr, void *data);
```

**DESCRIPTION**

The `ptrace()` system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers. It is primarily used to implement breakpoint debugging and system call tracing.
Tracing

- The tracee might be **stopped** at both the entry and the exit of system calls
  - Need to carefully count calls
- When the tracee violates a rule, it should be **killed**
- There are some system calls that are called when a program starts
  - Need to carefully write instructions file
Implementation Details

- **Ptrace does not know system call symbolic names**
  - Works with system call numbers

- **Need to map names to numbers**

- **Important**: System call numbers and their symbolic names may be different across architectures or kernel versions
  - Your implementation should be compatible with CSD workstations!
You need to store:

1. **Monitored system calls**
   a. Their number
   b. The number of times they can be called
   c. *(Optional)* Their symbolic name

2. **The last X system calls made by the tracee**
Intrusion Detection System
Intrusion Detection System

- Create an intrusion detection system for network traffic
- Monitors network traffic, analyzes protocols and identifies suspicious activity
Overview

Intrusion Detection System

1. Read
2. Monitor
3. Print

Rules File

Network Traffic
Overview

./intrusion_detection rules.txt traffic.pcap

1. Read rules file
2. Read pcap file containing network traffic
3. Identify packets that match against any rule
4. Notify the user
Rules File

- Given as **input** to your system
- Each line contains a **pattern** for network traffic

rules.txt

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Port 1</th>
<th>Port 2</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.153.1 19987 192.168.153.130</td>
<td>80</td>
<td>ALERT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139.199.0.0/16 123 192.168.153.0/24</td>
<td>42244</td>
<td>WARN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rules Format

● **General format:**
  
  `<src IP address> <src port> <dst IP address> <dst port> <type>`

● **CIDR Notation**
  
  ○ Network Prefix and Host Identifier
  ○ 192.168.1.0/24 is equivalent to 192.168.1.* or 192.168.1.0 - 255
• Use the `pcap` library to process captured traffic

**PCAP(3PCAP) MAN PAGE**

Section: Misc. Reference Manual Pages (3PCAP)
Updated: 9 September 2020
Index Return to Main Contents

NAME

`pcap` - Packet Capture library

SYNOPSIS

```
#include <pcap/pcap.h>
```

DESCRIPTION

The Packet Capture library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a ``savefile``, and reading packets from a ``savefile``.
Monitoring

● You need to iterate over packets
  ○ e.g. pcap_loop()

● You need to parse protocol headers to get IP addresses and ports

● The headers you need to parse are:
  ○ Ethernet
  ○ IP
  ○ TCP
  ○ UDP
Monitoring

- You need to **iterate** over packets
  - e.g. `pcap_loop()`

- You need to parse **protocol headers** to get IP addresses and ports

- The headers you need to parse are:
  - Ethernet
  - IP
  - TCP
  - UDP
Implementation Details

You need to store:

• Rules
  a. Source and destination IP addresses
  b. Source and destination Ports
  c. Message/Type
Secret Sharing System
Secret Sharing System

- Implement a secret sharing mechanism

- Distribute a secret among a group so that the secret cannot be revealed unless X people are present
Assume there are three friends: Alice, Bob and Carol

The three people will share a secret number “c” by each taking a piece of the number

Only when all three pieces are presented then all of them are able to reconstruct the secret number “c”
Implementation

- This will be achieved by constructing the polynomial
  \[ f(x) = a \cdot x^2 + b \cdot x + c \]

- Each person will take a point of the polynomial
  - Alice: \((1, f(1))\)
  - Bob: \((2, f(2))\)
  - Carol: \((3, f(3))\)

- When all 3 points are presented, they reconstruct the polynomial to retrieve secret number “c”
Simple Example

- Let's assume that Alice and Bob want to share the secret number 72
  - Since they are 2, the polynomial degree is 1

- They randomly choose “a” to be 14 and “b” is the secret number
  - \( f(x) = a \cdot x + b = 14 \cdot x + 72 \)

- They calculate \( f(1) = 86 \) and \( f(2) = 100 \)
  - Alice gets (1, 86)
  - Bob gets (2, 100)
Simple Example

• To **reconstruct** the secret, they present their points and reconstruct the polynomial

\[
\begin{align*}
86 &= a + b \\
100 &= 2a + b
\end{align*}
\]

\[
\begin{align*}
a &= 86 - b \\
100 &= 2(86 - b) + b
\end{align*}
\]

\[
\begin{align*}
100 &= 172 - 2b + b \\
-72 &= -b
\end{align*}
\]

\[
b = 72
\]
Implementation Details

- Develop a program called secret_sharing using C that does the following:
  - Using the `split` option and a secret number the program generates the 3 points
    - e.g. $ ./secret_sharing split 9
      > (1, 16), (2, 27), (3, 42)
  - Using the `join` option and the 3 points, the program reconstructs the secret number
    - e.g. $ ./secret_sharing join > (1, 16), (2, 27), (3, 42)
      > 72
  - Generalize your solution for N friends.
    - When any three present their points they are able to reconstruct the secret
Advanced Example

- Select a secret shared number “c”
  - E.g. c is 9

- $f(x) = 2 \cdot x^2 + 5 \cdot x + 9$ where $a, b$ were randomly generated

- When 3 shares $(x_1, x_2, x_3)$ are present:
  - $f(x_1) = a \cdot x_1^2 + b \cdot x_1 + c$  $f(1) = 16$  $16 = a + b + c$  
  - $f(x_2) = a \cdot x_2^2 + b \cdot x_2 + c$  $f(2) = 27$  $27 = 4 \cdot a + 2 \cdot b + c$  
  - $f(x_3) = a \cdot x_3^2 + b \cdot x_3 + c$  $f(3) = 42$  $42 = 9 \cdot a + 3 \cdot b + c$
Advanced Example

\[
\begin{align*}
16 &= a + b + c & (1) \\
27 &= 4a + 2b + c & (2) \\
42 &= 9a + 3b + c & (3) \\
42 &= 9\cdot a + 3\cdot b + c \\
-3\cdot16 &= 3\cdot a + 3\cdot b + 3\cdot c \\
-6 &= 6\cdot a + 0 - 2\cdot c & \iff a = (2\cdot c - 6)/6
\end{align*}
\]
Advanced Example

- We have computed that:
  - $a = \frac{2 \cdot c - 6}{6}$
  - $a = \frac{c - 5}{2}$

\[(2 \cdot c - 6)/6 = (c-5)/2 \Leftrightarrow c - 5 = \frac{2}{3} \cdot c - 2 \Leftrightarrow \frac{1}{3} \cdot c = 3 \Leftrightarrow c = 9\]
Thank You!

papamano@csd.uoc.gr
spkostas@csd.uoc.gr

Questions?

Credit

Icons from FlatIcon, made by Freepik: