Transforming Code to Drop Dead Privileges

Xiaoyu Hu
BitFusion.io Inc

Jie Zhou
University of Rochester

Spyridoula Gravani
University of Rochester

John Criswell
University of Rochester
Least Privilege Principle

“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”

An Example of An Over-Privileged Program

**ping**: send ICMP packets to a network host

privileges needed: open a raw socket

```c
socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

some other network related privileges in

```c
setsockopt(icmp_sock, SOL_SOCKET, SO_DEBUG, (char *)&hold, sizeof(hold));
setsockopt(icmp_sock, SOL_SOCKET, SO_MARK, &mark, sizeof(mark));
```

**However**

```
$ file /bin/ping
/bin/ping: setuid ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=c525a765d86db1bda61f56a497e6113871ef37b, stripped
jie@zimmer: ~
$ ls -l /bin/ping
-rwsr-xr-x 1 root root  44168 May  7 2014 /bin/ping
```
Privileged Programs

In Linux,

Privilege: the ability to override kernel’s access control rule

Privileged Program: program with one or more privileges
Linux Capabilities

- CAP_SETUID
- CAP_CHOWN
- CAP_DAC_OVERRIDE
- CAP_NET_RAW
- CAP_DAC_READ_SEARCH
- CAP_DAC_OVERRIDE
- CAP_NET_RAW
- CAP_CHOWN
- CAP_SETUID
- root
**Linux Capabilities**

**effective set**

- CAP_NET_RAW
- CAP_CHOWN
- CAP_SETUID

**permitted set**

- CAP_NET_RAW
- CAP_CHOWN
- CAP_SETUID
- CAP_FOWNER
- CAP_SYS_TIME

**priv_raise**: copy a privilege from permitted set to effective set

**priv_lower**: delete a privilege from effective set *temporarily*

**priv_remove**: remove privilege from permitted set *permanently*
**Linux Capabilities**

**ping**: send ICMP packets to a network host

prileges needed: open raw socket

```c
socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

some other network related privileges in

```c
setsockopt(icmp_sock, SOL_SOCKET, SO_DEBUG, (char *)&hold, sizeof(hold));
setsockopt(icmp_sock, SOL_SOCKET, SO_MARK, &mark, sizeof(mark));
```

**Problem Solved!?**

**two steps**
1. figure out which privileges a program requires
2. do priv_remove on a privilege when it’s no longer needed
int main(int argc, char *argv[]) {
    ......
    ...... foo();
    ......
}

void foo() {
    ...
    ...
    // use privilege P1
} priv_remove(P1)
It’s difficult to **manually** figure out when we can remove which capabilities permanently.

We need a tool!
AutoPriv

- LLVM-based compiler
- uses *data-flow analysis* techniques to analyze and transform programs
- drops privileges when they are no longer needed
Outline

• Design

• Implementation

• Performance Experiments

• Conclusion
Outline

• Design

• Implementation

• Performance Experiments

• Conclusion
**Live Privilege**: privilege that may be used along some path in the future

```c
int i = 0;
......
printf(...);
......

j = i + 10;
......

// require P1
open(...);
```
Design

AutoPriv Architecture
Design

Live Privilege Analysis

• iterative
• inter-procedural
• context-insensitive

• Propagate Privileges
  • within basic blocks (BB)
  • between successors BB to predecessors BB
  • callees to callers

• Remove Privileges
An Example Program

```
main
  A
    use P1
  B
    use P2
    call foo
  C
    call bar
  D

foo
  E
    use P3

bar
  F
    use P2
```
Local Privilege Analysis

main

A

In: P1

use P1

B

In: P2

use P2

C

In:

call foo

D

In:

call bar

E

foo

In: P3

use P3

F

bar

In: P2

use P2
Interprocedural Privilege Analysis

main requires

A

use P1

In: P1

B

use P2

In: P2

C

call foo

In:

D

call bar

In:

E

foo requires P3

In: P3

use P3

F

bar requires

In: P2

use P2
Interprocedural Live Privilege Analysis

**main** requires P1P2P3

- **A**: In: P1
  - use P1
  - Out:

- **B**: In: P2
  - use P2
  - Out:

- **C**: In:
  - call foo
  - Out:

- **D**: In:
  - call bar
  - Out:

---

**foo** requires P3

- **E**: In: P3
  - use P3
  - Out:

---

**bar** requires P2

- **F**: In: P2
  - use P2
  - Out:
Propagate: Successor to Predecessors

main requires P1P2P3

use P1

In: P1

Out:

In: P2

use P2

Out:

call foo

In:

Out:

call bar

In:

Out:

foo requires P3

use P3

In: P3

Out:

bar requires P2

use P2

In: P2

Out:
Propagate: Out to In

main requires P1 P2 P3

A
In: P1
use P1
Out: P2

B
In: P2
use P2
Out:

C
In: P3
Call foo
Out:

D
In: P3
Call bar
Out:

E
foo requires P3
In: P3
use P3
Out:

F
bar requires P2
In: P2
use P2
Out:
Propagate: Callee to Caller

main requires P1 P2 P3

A

In: P1 P2

use P1

Out: P2

B

In: P2

use P2

Out:

C

In: P2

call foo

Out:

D

call bar

Out:

E

foo requires P3

In: P3

use P3

Out:

F

bar requires P2

In: P2

use P2

Out:
Propagate Iteratively

main requires P1, P2, P3

A
In: P1, P2
use P1
Out: P2, P3

B
In: P2
use P2
Out:

C
In: P3
call foo
Out: P2

D
In: P2
call bar
Out:

E
foo requires P3
In: P3
use P3
Out:

F
bar requires P2
In: P2
use P2
Out:
Propagate: Caller to Callee’s Exit

main requires P1 P2 P3

A

In: P1 P2 P3

use P1

Out: P2 P3

B

In: P2

use P2

Out: P2

C

In: P3 P2

call foo

Out: P2

D

call bar

Out:

E

foo requires P3

In: P3

use P3

Out: P2

F

bar requires P2

In: P2

use P2

Out:
Remove Dead Privileges (In - Out)

main requires P1, P2, P3

A
  In: P1, P2, P3
  use P1
  Out: P2, P3
  priv_remove(P1)

B
  In: P2
  use P2
  Out: P2

C
  In: P3, P2
  call foo
  Out: P2
  priv_remove(P3)

D
  In: P2
  call bar
  Out: P2
  priv_remove(P2)

E
  In: P3, P2
  use P3
  Out: P2
  priv_remove(P3)

F
  In: P2
  use P2
  Out: P2
  priv_remove(P2)
Outline

• Design

• Implementation

• Performance Experiments

• Conclusion
Privilege Primitives

priv_raise(int cap_num, int capability, ...): copy a set of privileges from the permitted set to effective set. The permitted set remains unchanged.

priv_lower(int cap_num, int capability, ...): delete a set of privileges from the effective set temporarily.

priv_remove(int cap_num, int capability ...): remove a set of privileges from both the effective set and the permitted set permanently.
Implementation

**Manually** put a pair of `priv_raise()` and `priv_lower()` around a system call or library function call that uses some privilege(s), e.g.,

```c
priv_raise(1, CAP_NET_RAW);
icmp_sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
priv_lower(1, CAP_NET_RAW);
```

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>SLOC</th>
<th>Number of Privilege-Bracketing Function Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>4.1.5.1</td>
<td>51,371</td>
<td>29</td>
</tr>
<tr>
<td>su</td>
<td>4.1.5.1</td>
<td>51,371</td>
<td>34</td>
</tr>
<tr>
<td>ping</td>
<td>s20121221</td>
<td>12,001</td>
<td>6</td>
</tr>
<tr>
<td>sshd</td>
<td>6.61p</td>
<td>82,376</td>
<td>59</td>
</tr>
<tr>
<td>thttpd</td>
<td>2.26</td>
<td>8,367</td>
<td>8</td>
</tr>
</tbody>
</table>

Number of Required Privilege-Bracketing
5 LLVM IR passes

1. Split basic blocks
2. Local Privilege Analysis
3. Interprocedural Privilege Analysis
4. Interprocedural Live Privilege Analysis
5. Privilege Removal Instrumentation
Outline

- Design
- Implementation
- Performance Experiments
- Conclusion
• Compiler overhead induced by program analysis and instrumentation
• Program overhead induced by raising, lowering, and removing privileges

Experiment Setup

• LLVM: 3.7.0
• OS: 64-bit Ubuntu 16.04
• CPU: Intel Core i5–6600 3.30GHZ
• RAM: 8GB
• Disk: 256GB SSD
# Test Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>SLOC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>4.1.5.1</td>
<td>51,371</td>
<td>Password change utility</td>
</tr>
<tr>
<td>su</td>
<td>4.1.5.1</td>
<td>51,371</td>
<td>Run programs as another user</td>
</tr>
<tr>
<td>ping</td>
<td>s20121221</td>
<td>12,001</td>
<td>Send ICMP packets to a remote host</td>
</tr>
<tr>
<td>sshd</td>
<td>6.61p</td>
<td>82,376</td>
<td>Remote login server with encrypted connection</td>
</tr>
<tr>
<td>thttpd</td>
<td>2.26</td>
<td>8,367</td>
<td>a lightweight HTTP server</td>
</tr>
</tbody>
</table>

Test Programs
1. Use Clang to compile programs with -O2 to LLVM bitcode
2. Run LLVM’s opt with and without our passes
3. Run opt’s --time-passes to measure the user and system execution time

Ran experiments 20 times for each program

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Average</th>
<th>Standard Deviation</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>Original</td>
<td>136.17 ms</td>
<td>0.43 ms</td>
<td>27.18%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>173.19 ms</td>
<td>0.57 ms</td>
<td></td>
</tr>
<tr>
<td>su</td>
<td>Original</td>
<td>205.96 ms</td>
<td>1.00 ms</td>
<td>15.15%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>237.16 ms</td>
<td>1.09 ms</td>
<td></td>
</tr>
<tr>
<td>ping</td>
<td>Original</td>
<td>159.02 ms</td>
<td>0.46 ms</td>
<td>7.87%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>171.54 ms</td>
<td>0.40 ms</td>
<td></td>
</tr>
<tr>
<td>sshd</td>
<td>Original</td>
<td>4,090.87 ms</td>
<td>21.16 ms</td>
<td>27.19%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>5,203.24 ms</td>
<td>29.56 ms</td>
<td></td>
</tr>
<tr>
<td>thttpd</td>
<td>Original</td>
<td>317.95 ms</td>
<td>0.77 ms</td>
<td>17.12%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>372.38 ms</td>
<td>0.66 ms</td>
<td></td>
</tr>
</tbody>
</table>
## Application Overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Configuration</th>
<th>Repetition</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>change current user’s password</td>
<td>200</td>
</tr>
<tr>
<td>su</td>
<td>ran <code>echo</code> as another user</td>
<td>200</td>
</tr>
<tr>
<td>ping</td>
<td><code>ping -c 10 localhost</code></td>
<td>50</td>
</tr>
<tr>
<td>sshd</td>
<td>ran <code>scp</code> to fetch files from 16 KB to 16 MB</td>
<td>500</td>
</tr>
<tr>
<td>thttpd</td>
<td><code>ab -c 32 -n 10000</code></td>
<td>60</td>
</tr>
</tbody>
</table>

How We Ran Each Test Program
## Application Overhead

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Average</th>
<th>Standard Deviation</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>passwd</td>
<td>Original</td>
<td>36.29 ms</td>
<td>3.75 ms</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>36.66 ms</td>
<td>2.94 ms</td>
<td></td>
</tr>
<tr>
<td>su</td>
<td>Original</td>
<td>7.74 ms</td>
<td>0.05 ms</td>
<td>0.01%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>7.78 ms</td>
<td>0.04 ms</td>
<td></td>
</tr>
<tr>
<td>ping</td>
<td>Original</td>
<td>9,211.49 ms</td>
<td>0.82 ms</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>AutoPriv</td>
<td>9,211.42 ms</td>
<td>0.79 ms</td>
<td></td>
</tr>
</tbody>
</table>

Performance of passwd, su, and ping
Application Overhead

![Graph showing sshd performance with file size vs average bandwidth. The graph compares Original_sshd and Priv_sshd with different file sizes ranging from 16 KB to 16384 KB. The x-axis represents file size (KB), and the y-axis represents average bandwidth (KB/s). The graph indicates a trend where average bandwidth increases with file size for both Original_sshd and Priv_sshd.](image-url)
## Application Overhead

<table>
<thead>
<tr>
<th>File Size (KB)</th>
<th>Original_thttpd</th>
<th>Priv_thttpd</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
</tr>
<tr>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1024</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16384</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Average Bandwidth (MB/s)**

![Graph showing thttpd performance](image-url)
Outline

• Design

• Implementation

• Performance Experiments

• Conclusion
**Conclusion**

- **AutoPriv** - an LLVM-based compiler that transforms code to drop dead privileges
- on average 19% overhead during optimization
- no overhead in the programs that AutoPriv transforms

Open-source: [https://github.com/jtcriswell/AutoPriv/tree/AutoPriv](https://github.com/jtcriswell/AutoPriv/tree/AutoPriv)
How More Secure Is AutoPriv’ed Program?

We measured how many dynamic instructions are executed with each privilege available in the permitted set.

<table>
<thead>
<tr>
<th>Program</th>
<th>CAP_CHOWN</th>
<th>CAP_SETUID</th>
<th>CAP_SYS_CHROOT</th>
<th>CAP_NET_BIND_SERVICE</th>
<th>CAP_SETGID</th>
</tr>
</thead>
<tbody>
<tr>
<td>httpd</td>
<td>323 (0.00%)</td>
<td>323(0.00%)</td>
<td>4,686,266 (9.82%)</td>
<td>4,686,627 (9.82%)</td>
<td>4,693,826 (9.84%)</td>
</tr>
<tr>
<td>passwd</td>
<td>2,654(3.81%)</td>
<td>43,952(63.02)</td>
<td>43,952(63.02)</td>
<td>69,582(99.77)</td>
<td>69,582(99.77)</td>
</tr>
</tbody>
</table>
How Did You Handle Function Pointers?

Our call graph is conservative.

- LLVM’s built-in call graph
- Data Structure Analysis (DSA)
I’m Still Concerned With Manual Bracketing …

It’s very difficult for a compiler to figure out which function call needs which privileges.

For instance,

open(…)