Programovanie v oeračných systémoch

10 - Services, Security

Jozef Šiška

Department of Applied Informatics
Comenius University in Bratislava

2016/2017

Services

- Detach
- stdio
- Controlling tty and SIGHUP
- Privileges
- pidfile

`https://github.com/jirihnidek/daemon/blob/master/src/daemon.c`

Restricting privileges

- Dropping privileges
  - a dameon is (usually) run as root
  - sets up any resources it needs root access for (opening privileged ports, reading sensitive files)
  - "drops" root permissions: user, group and additional groups (`https://github.com/wertarbyte/coreutils/blob/master/src/setuidgid.c`)
- Capabilities
  - split permissions
  - program / daemon starts as normal user
  - is granted capabilities (i.e. `CAP_NET_BIND_SERVICE` – bind ports <1024)

```
man 6 capabilities
```

Security

- Authentication
- Authorization / access control
- Attacks
  - Privilege escalation
  - Code injection
- Problems
  - Bad design, allowing user too much freedom:
    - not dropping privileges correctly
    - specifying applications to run / files to write to...
    - using user's env (`$EDITOR`, `$PAGER`)
  - Data races
  - Bugs: buffer overflows
  - Apple: not checking return values correctly: `https://objective-see.com/blog/blog_0x24.html`

Buffer Overflow

Buffer overflows are bugs that happen when you copy code from `http://bufferoverflow/` without thinking about it and understanding it completely.

Or not...?

Buffer Overflow

Buffer overflows are bugs that happen when you copy code from `http://bufferoverflow/` without thinking about it and understanding it completely.

Or not...?

Buffer Overflow

```c
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[8];
    gets(buf);
    printf("%s\n", buf);
    return 0;
}
```

- gets is deprecated anyway (manpages itself says DO NOT USE)
- C only problem?
- language with a sane, dynamic string class should not have this problem...
- ... but we probably want to limit the size of loaded data anyway

Buffer Overflow

```
#include <stdio.h>
int main(int argc, char **argv)
{
    char buf[8];
    gets(buf);
    printf("%s\n", buf);
    return 0;
}
```

- `gets` is deprecated anyway (manpages itself says DO NOT USE)
- C only problem?
- language with a sane, dynamic string class should not have this problem...
- ... but we probably want to limit the size of loaded data anyway

Buffer Overflow

```
char src[32];
char dst[16];

strcpy(dst, src);
```

```
char src[32];
char dst[32];

char *p = dst;
*p++ = '/';
strcpy(p, src);
```

again, sane string class should avoid this...

Buffer Overflow

```
char src[32];
char dst[16];

strcpy(dst, src);
```

```
char src[32];
char dst[32];

char *p = dst;
*p++ = '/';
strcpy(p, src);
```

again, sane string class should avoid this...

Buffer Overflow istream

```
#include <iostream>
int main(int argc, char **argv)
{
    char buf[8];

    std::cin >> buf;
    std::cout << buf;
    return 0;
}
```

This is really C++ trying to be backwards compatible with C strings...
So does it have the same problem as `gets`?

Buffer Overflow istream (fixed)

```
#include <iostream>
int main(int argc, char **argv)
{
    char buf[8];
    std::cin.width(sizeof(buf));
    std::cin >> buf;
    std::cout << buf;
    return 0;
}
```

Stack Buffer Overflow

```
#include <stdio.h>
void foo()
{
    char buf[8];
    gets(buf);
    printf("%s\n", buf);
}

int  main(int argc, char **argv)
{
    printf("start\n");
    foo();
    printf("end\n");
    return 0;
}
```

Buffer Overflow

- Not passing available size
- Passing wrong size
  - storing 8 byte string in an 8 byte array (what about the terminating NUL?!?)
  - integer owerflow, `int n = strlen(s)` (stores unsigned `size_t` in a signed `int`), though not really practically exploitable unless it's a a `char` or `short` (*"usernames are sure to be shorter than 255 chars. . . "*)
  - just bugs when calculating sizes
  - data size mismatch
  - copy paste, duplication (of numbers)

Exploiting Buffer Overflows

- Crash program (not that interesting)
- Change data (not that interesting)
- Execute injected code
- Gain root access (trick a setuid/setgid program to execute shell)

Nice writeups:
`https://sploitfun.wordpress.com/2015/06/26/linux-x86-exploit-development-tutorial-series/`

Mitigation

- Programmer
  - don’t use / design bad APIs
  - take great care when there’s no other way
  - avoid explicit numbers, use constants / `sizeof`
  - reviews / analysis
- System
  - non-executable memory pages (for stack, data...)
  - Stack canary / protector
  - ASLR (address space layout randomization) - randomize addresses where "stuff gets loaded"