Defence Against the Dark Arts

Rolfe Bozier

16-Apr-2014

When software bugs manifest…

- Conventional software bugs
  - May be triggered by certain data, a specific use case etc.
  - If they occur:
    - Try a workaround
    - Get a patch
    - Worst case, software is unusable
- Software security bugs
  - People *look* for them
  - May not occur in normal usage
  - If triggered, the consequences can be *really bad*
  - Attackers are very creative; outwitting them is really difficult

Consequences of security bugs

- Type of security breach
  - Uncontrolled memory read
  - Uncontrolled memory write
  - Arbitrary code execution
- Consequences
  - They get to read data you'd rather keep private
  - They get to bypass your security restrictions
  - They compromise your PC, server, database, …

Surely they need to know about your system?

- They probably know more than you think
  - They could be targeting an OSS component in your system (e.g. OpenSSL, libpng/libtiff/libjpeg, zlib…)
  - They could be targeting your use of runtime libraries
  - You might be deploying a standard application (e.g. WordPress, PHP-Nuke or some other CMS)
  - If they have a copy of the executable or device, they could easily reverse engineer it
- It makes sense for attackers to invest a lot of time into analyzing systems – if successful they can attack many systems

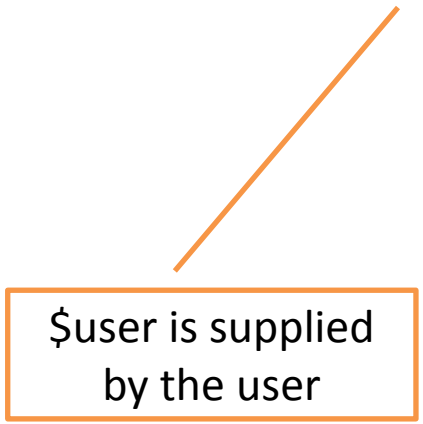
What can go wrong?

- You can partition issues into 2 groups:
  1. Known problems – you don't want to make these mistakes [again]
  2. Unforeseen issues – by definition these are hard to prevent
- Let's look at some examples

SQL injection attacks

- Many websites and applications have an SQL database as a backend
  - Usually there is a high level language on top of the database making access easy
- Here is a very typical piece of PHP code:

```
$query = ”select * from users where login = ’$user’”
$stmt = $db->prepare($query);
$stmt->execute();
if ($row = $stmt->fetch_row())
    …
```



SQL injection attacks

- But what if some specifies something unexpected, such as: `’ OR 1 --`
  - Now the SQL statement becomes something unexpected:

```
select * from users where login = ’’ OR 1 --’
```

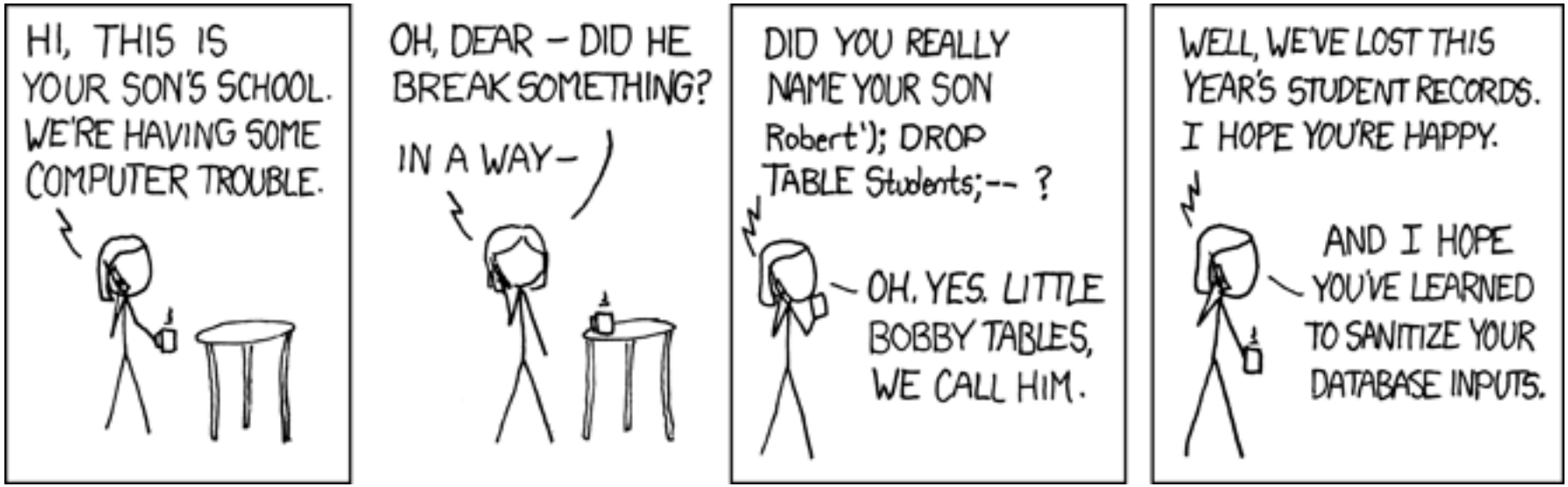
- The attacker can now alter the SQL statement that is executed…

SQL injection attacks

- How to prevent this?
  - Option 1: Sanitize all user-supplied input
    - This is always a good idea, but can be hard to always get it right
  - Option 2: Use a more secure method (bind variables):

```
$query = ”select * from users where login = ?”
$stmt = $db->prepare($query);
$stmt->bind(”s”, $user);
$stmt->execute();
if ($row = $stmt->fetch_row())
    …
```

SQL injection attacks



Buffer overflows

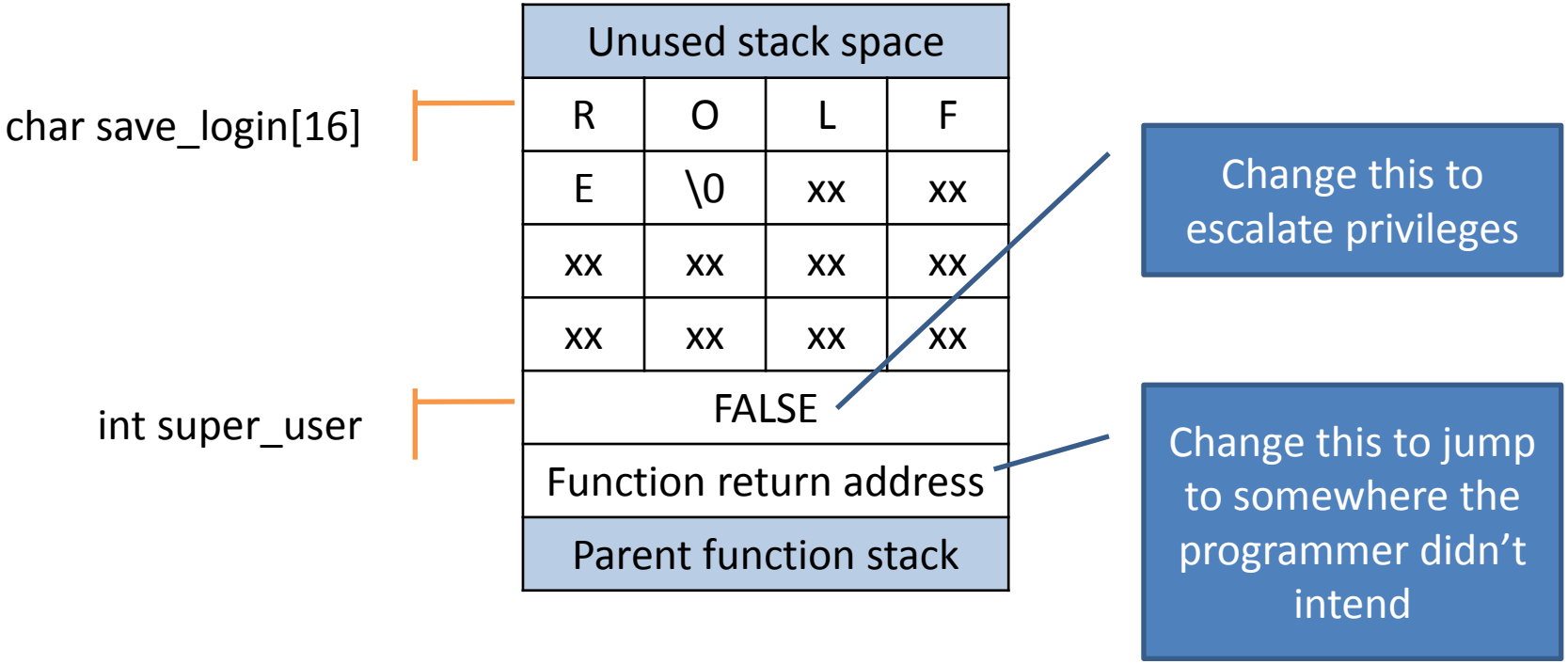
- Consider the following code:

```
int process_user(char *login)
{
    int super_user;
    char save_login[16];

    super_user = is_user_privileged(login);
    strcpy(save_login, login);

    ...
```

- What happens if “login” is longer than expected?

Buffer overflows

- What interesting things could we overwrite?
- Here is the stack layout:



Buffer overflows

- It’s not always this obvious…
- Buffer overflows are a classic attack on Windows PCs
  - Lots of tasks run with high privileges
  - Microsoft spend millions doing fuzz testing to try to mitigate this
  - The payoff for a valid attack is huge

Apple SSL bug

- Here's a recent bug from Apple:

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    err = sslRawVerify(...);

    if(err) {
           sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                "returned %d\n", (int)err);
           goto fail;
    }

fail:
    return err;
}
```

Apple SSL bug

- Impact:
  - A malicious user can set up a man-in-the-middle attack on a secure website
  - You think you're talking to your bank using a secured connection, but the bad guy is relaying every packet

Heartbleed bug – this week's security issue

- OpenSSL is a library providing secure connections to web servers via the SSL protocol
- SSL supports keep-alive messages:
  - A client sends a packet containing some [arbitrary] data
  - The server sends a reply-packet back with the data included
- Here's the pseudo-code:

Heartbleed bug

```
struct
{
    unsigned short len;
    char payload[];
} *packet;

/* read the keep-alive request packet from the socket */
packet = malloc(amt);
read(s, packet, amt);

/* allocate a buffer for the reply */
buffer = malloc(packet->len);

/* copy the payload from the request to the reply */
memcpy(buffer, packet->payload, packet->len);

/* send the reply back down the socket */
write(s, buffer, packet->len);
```

Heartbleed bug

```
struct
{
    unsigned short len;
    char payload[];
} *packet;

/* read the keep-alive request packet from the socket */
packet = malloc(amt);
read(s, packet, amt);

/* allocate a buffer for the reply */
buffer = malloc(packet->len);

/* copy the payload from the request to the reply */
memcpy(buffer, packet->payload, packet->len);

/* send the reply back down the socket */
write(s, buffer, packet->len);
```

What if the payload was only 1 byte, but they said it was 10k?

Heartbleed bug

- The client gets back their 1 byte of data followed by another 10k of random server memory
  - Probably decrypted by this point
  - Maybe user-names & passwords, account numbers, system passwords…
  - Maybe even the private key for the SSL encryption certificate
- Repeat until you get something interesting

This all sounds too hard!

- It's all about risk
  - If you're small and uninteresting, maybe you are safe…
  - If you are Apple or Microsoft or Google, you are a huge target
  - *Where does Canon fit?*
- Maybe it's not important…
  - Really, who wants to spend time trying to compromise a printer?
  - But what about a printer *driver*?

What can you do?

- Stop thinking like a programmer or user and start thinking like the bad guy
  - What's the worst thing that could happen to my product?
  - How could I break my own software?
  - What are *all* the interactions between the attacker and my software?
    - Do not trust any user-supplied data
    - Do not trust an external caller of your functions
    - Do not trust the packets you receive
- It's *really hard*
  - Unfortunately hindsight is always 20/20