Secure Software

Hypothesis - It is easier to protect software that is not buggy than software than is buggy

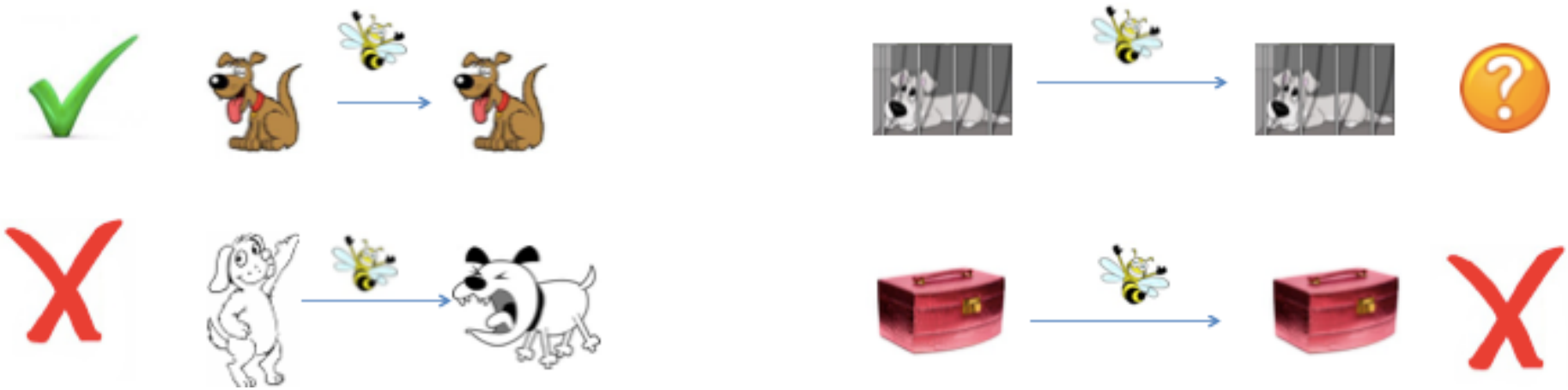
Corollary - it is easier to attack buggy code

"Software security is not security software"

https://www.synopsys.com/blogs/software-security/software-security/

Secure Software - protect software from attack

A software dog can be secured against wasp attacks



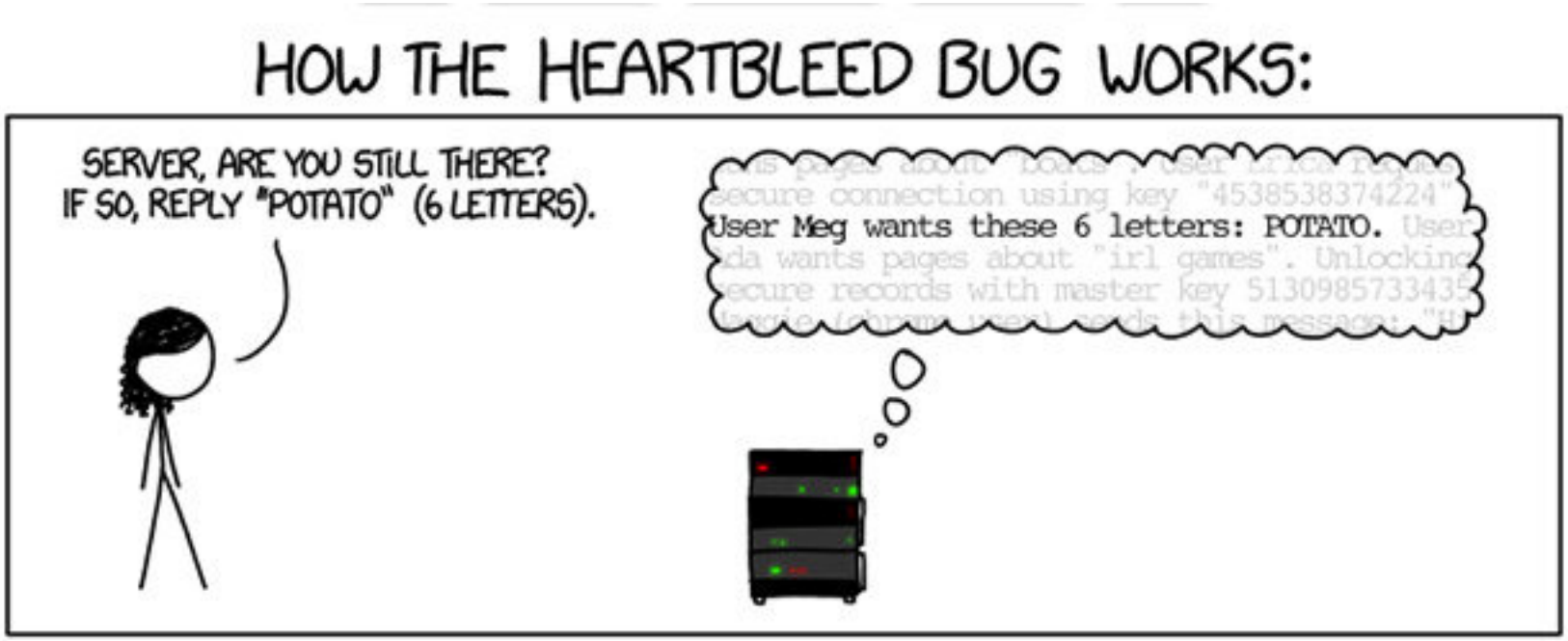
http://heartbleed.com

The Heartbleed Bug is/was a serious vulnerability in the popular OpenSSL cryptographic software library

The buggy code

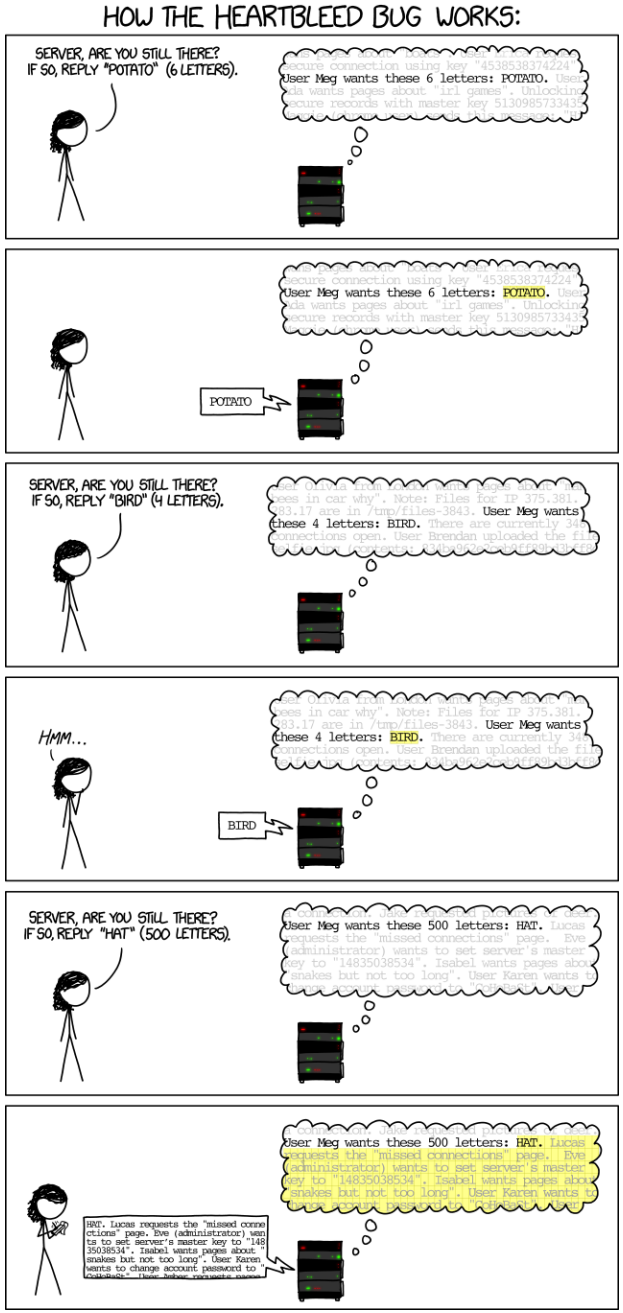
```
/* Read type and payload length first */
hbtype = *p++;
n2s(p, payload);
pl = p;
```

(In)Secure Software



https://www.engadget.com/2014/04/12/xkcd-heartbleed-bug-explainer/

(In)Secure Software



https://xkcd.com/1354/

(In)Secure Software

"goto fail" – Apple's SSL bug

```
. . .
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
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;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
. . .
```

if `err` is zero and there is actually no error to report.
The result is that the code leaps over the vital call to `sslRawVerify()`, and exits the function.
This causes an immediate "exit and report success", and the TLS connection succeeds, even though the verification process hasn't actually taken place.
A skilled attacker can easily exploit this

Intel's AMT Vulnerability

In April 2017, Intel announced a critical privilege escalation bug that was laying around its Active Management Technology (AMT) login page for the past seven years. The exploit allows a remote attacker to take control of vulnerable devices with ease.

The login code for the AMT web interface *incorrectly* used the strncmp function, which allowed users to gain access when inserting an empty password at the login screen.

Intel's AMT Vulnerability

The buggy code

```
int main () {
  string realpass = "secret";
  string userpass = "user-secret";
  int equal = strncmp(realpass.c_str(), userpass.c_str(), userpass.size());
  if (equal == 0) {
     printf ("'%s' equals to '%s'", realpass.c_str(), userpass.c_str());
  }
  return equal * equal; // make sure it's positive
}
```

Question: can you see the problem?

Intel's AMT Vulnerability

The fixed/*correct* code

```
  string realpass = "secret";
  string userpass = "user-secret";
  int equal = strncmp(realpass.c_str(),userpass.c_str(),realpass.size());
  if (equal == 0) {
     printf ("'%s' equals to '%s'", realpass.c_str(), userpass.c_str());
  }
  return equal * equal; // make sure it's positive
}
```

NOTE: we should add tests and documentation to distinguish between the correct and incorrect implementations

Intel's AMT Vulnerability

The Mega Conspiracy: using The Ken Thomson Hack

https://oded.ninja/2017/05/14/amt-n-ken-hack/

What if someone hacked into Intel's servers a few years ago, and updated their compiler to replace this:

```
strncmp(realpass.c_str(), userpass.c_str(), realpass.size())
C++Copy
```

with this:

```
strncmp(realpass.c_str(), userpass.c_str(), userpass.size())
C++Copy
```

Essentially adding a backdoor? What if the same attacker added code that turned off the attack when test runners were used? or when the compiler was running inside Intel's LAN?

This might sound crazy and far-fetched, but there are threat actors out there with the skill-set to pull this off. But hey, I'm not that paranoid. I do believe the vulnerability was introduced as a result of a human mistake... or not?

Facebook *View As* Security Breach

Which testers didn't do their job properly?

The "View As" feature is designed to allow users to experience how their privacy settings look to another person.

The first bug prompted Facebook's video upload tool to mistakenly show up on the "View As" page.

The second one caused the uploader to generate an access token—what allows you to remain logged into your Facebook account on a device, without having to sign in every time you visit—that had the same sign-in permissions as the Facebook mobile app.

Finally, when the video uploader did appear in "View As" mode, it triggered an access code for whoever the hacker was searching for.

This is a complex interaction of multiple bugs - the hackers likely required some level of sophistication.

(In)Secure Software

SQL injection

https://9gag.com/gag/a2mO5dd/sql-injection-in-a-nutshell

(In)Secure Software

SQL injection

https://9gag.com/gag/a2mO5dd/sql-injection-in-a-nutshell

(In)Secure Software

https://9gag.com/gag/a2mO5dd/sql-injection-in-a-nutshell

SQL injection

Code injection vulnerabilities (injection flaws) occur when an application sends untrusted data to an interpreter.

Prevention: Avoiding and testing for code injection bugs

Use APIs that, if used properly, are secure against all input characters. (Understand and use regular expressions)

Enforce language separation via a static type system

Use test services (like Burp, ZAP, …) to test for server side code injection vulnerabilities

DO NOT rely on testing alone - you need to inspect the code.

Avoidance is better than testing

Use code walkthroughs to identify security issues