Lecture 41 - OS Security Concepts

CprE 308

April 20, 2015

OS Security Concepts

What have we learned about Operating Systems so far?

- OS Goals
  - Resource Manager
  - User Interface
- Important concepts we have discussed
  - Multi-user, multi-process, multi-thread
  - Synchronization, Mutual Exclusion, Deadlocks
  - Scheduling
  - Memory Management
  - I/O Devices
  - Files and File Systems

What do we need in terms of security?

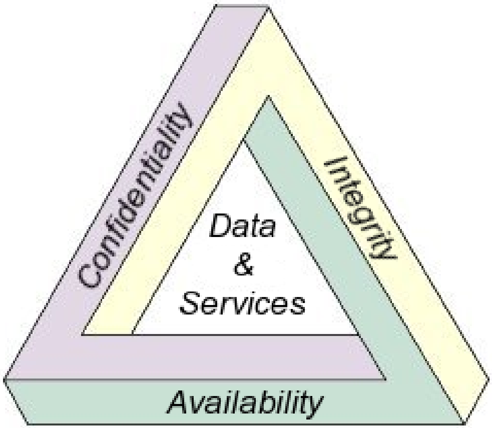


Figure 1: CIA Triad

What are the biggest problems?

Top 25 Most Dangerous Software Errors

`http://www.sans.org/top25-software-errors/`

Version 3.0, June 2011

General Security Goals

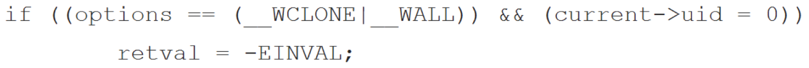
- Information Flow Secrecy
  - Denning's Lattice Model
  - Bell-LaPadula Model
- Information Flow Integrity
  - Brewer and Nash (Chinese Wall) Model
  - Biba Integrity Model
  - High/Low-water Mark Integrity
  - Clark-Wilson Integrity Model
- Covert Channels
  - The capability to transfer information between processes that are not supposed to be allowed to communicate by the computer security policy

Is open source more secure than proprietary?

- “Security through Obscurity” is no security at all
    - Kerckhoffs’s principle (assume the enemy knows the system)
- Open Source “potentially” gets more eyes
    - Is that a false sense of security?
    - Are the right people looking?
    - Is the project well funded/staffed?

Code snippet found in Linux Kernel

A bug or malware?

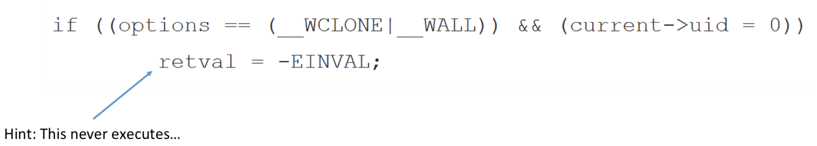


```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
        retval = -EINVAL;
```

Figure 2: Bug or Malware

Code snippet found in Linux Kernel

A bug or malware?



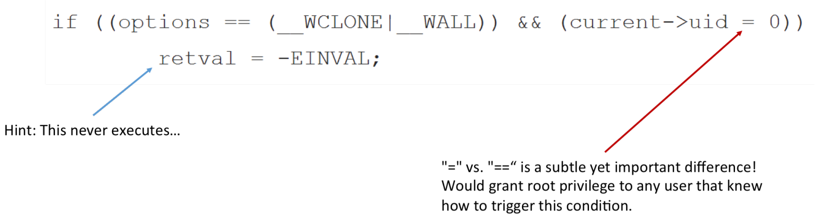
```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
        retval = -EINVAL;
```

Hint: This never executes...

Figure 3: Bug or Malware

Backdoor attempt found in Linux Kernel

Source: `https://freedom-to-tinker.com/blog/felten/the-linux-backdoor-attempt-of-2003`



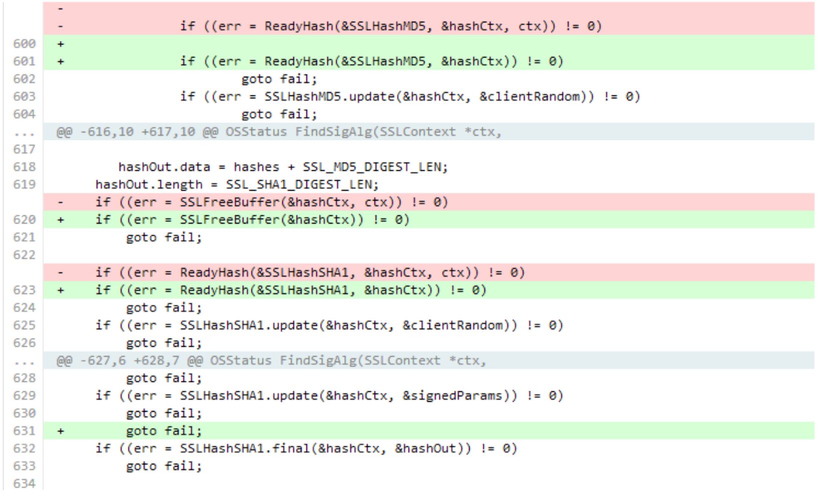
```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
        retval = -EINVAL;
```

Hint: This never executes...

"=" vs. "==“ is a subtle yet important difference!
Would grant root privilege to any user that knew how to trigger this condition.

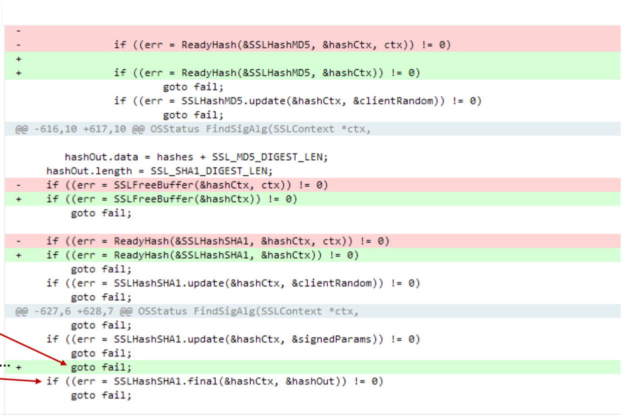
Figure 4: Bug or Malware

Where’s the problem?



```
-
-               if ((err = ReadyHash(&SSLHashMD5, &hashCtx, ctx)) != 0)
+
+               if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
                       goto fail;
               if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
                       goto fail;
@@ -616,10 +617,10 @@ OSStatus FindSigAlg(SSLContext *ctx,

       hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
-   if ((err = SSLFreeBuffer(&hashCtx, ctx)) != 0)
+   if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

-   if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
+   if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
@@ -627,6 +628,7 @@ OSStatus FindSigAlg(SSLContext *ctx,
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
+       goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

```

Apple SSL CVE-2014-1266 (GOTO Fail Bug)



```
-
-               if ((err = ReadyHash(&SSLHashMD5, &hashCtx, ctx)) != 0)
+
+               if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
                        goto fail;
                if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
                        goto fail;
@@ -616,10 +617,10 @@ OSStatus FindSigAlg(SSLContext *ctx,

            hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
        hashOut.length = SSL_SHA1_DIGEST_LEN;
-       if ((err = SSLFreeBuffer(&hashCtx, ctx)) != 0)
+       if ((err = SSLFreeBuffer(&hashCtx)) != 0)
            goto fail;

-       if ((err = ReadyHash(&SSLHashSHA1, &hashCtx, ctx)) != 0)
+       if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
            goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
            goto fail;
@@ -627,6 +628,7 @@ OSStatus FindSigAlg(SSLContext *ctx,
            goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
            goto fail;
+           goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
            goto fail;
```

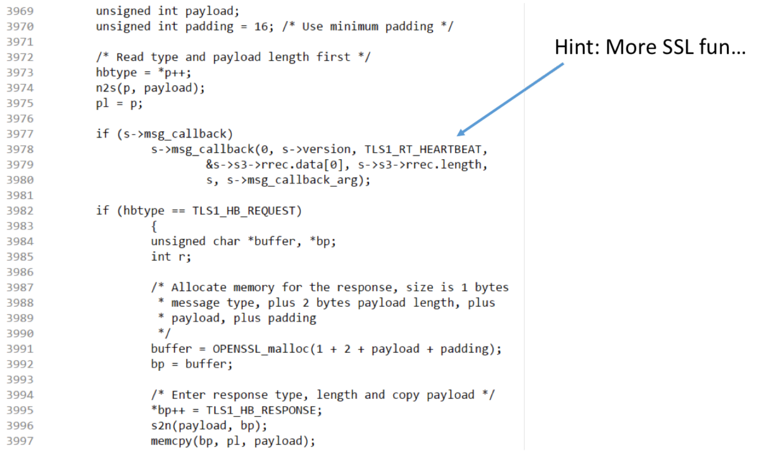
Always goto fail

Never does the check to verify server authenticity...

Apple SSL CVE-2014-1266 (GOTO Fail Bug)

- Should have been caught by automated tools
- Survived almost a year
- Affected OSX and iOS (because of shared code branches)

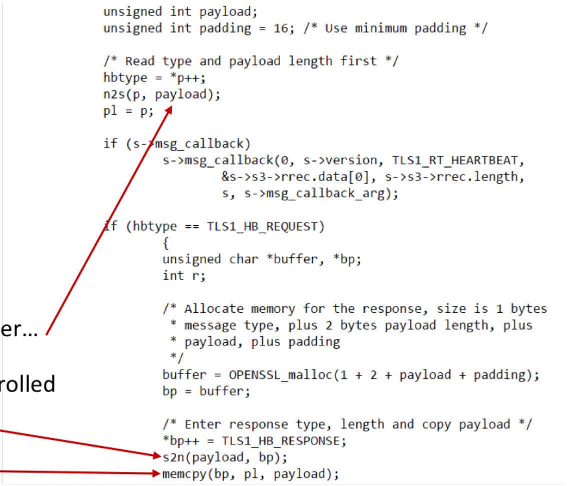
Where’s the problem?



```
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

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

        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
                int r;

                /* Allocate memory for the response, size is 1 bytes
                 * message type, plus 2 bytes payload length, plus
                 * payload, plus padding
                 */
                buffer = OPENSSL_malloc(1 + 2 + payload + padding);
                bp = buffer;

                /* Enter response type, length and copy payload */
                *bp++ = TLS1_HB_RESPONSE;
                s2n(payload, bp);
                memcpy(bp, pl, payload);
```

Figure 7: Bug or Malware

Heartbleed



```
unsigned int payload;
unsigned int padding = 16; /* Use minimum padding */

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

if (s->msg_callback)
        s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                &s->s3->rrec.data[0], s->s3->rrec.length,
                s, s->msg_callback_arg);

if (hbtype == TLS1_HB_REQUEST)
        {
        unsigned char *buffer, *bp;
        int r;

        /* Allocate memory for the response, size is 1 bytes
         * message type, plus 2 bytes payload length, plus
         * payload, plus padding
         */
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;

        /* Enter response type, length and copy payload */
        *bp++ = TLS1_HB_RESPONSE;
        s2n(payload, bp);
        memcpy(bp, pl, payload);
```

Heartbeat message size controlled by the attacker...

Response size also controlled by the attacker...

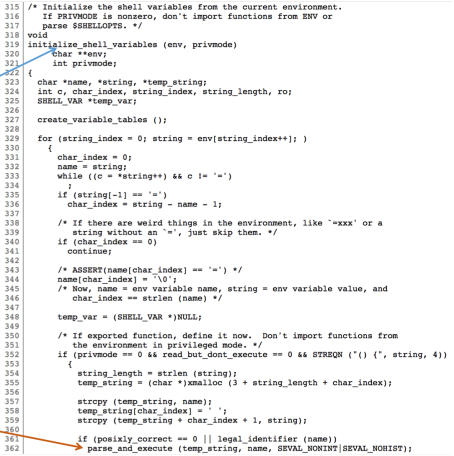
Reads too much data!

Heartbleed

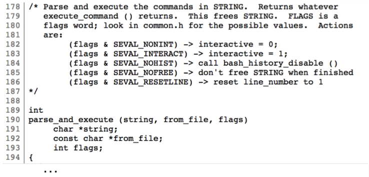
- Much less obvious error
- Survived several professional code audits (for ∼2 years)
- "Catastrophic" is the right word. On the scale of 1 to 10, this is an 11. ∼Bruce Schneier

Where's the problem?

Hint…

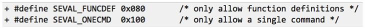


```
/* Initialize the shell variables from the current environment.
   If PRIVMODE is nonzero, don't import functions from ENV or
   parse $SHELLOPTS. */
void
initialize_shell_variables (env, privmode)
     char **env;
     int privmode;
{
  char *name, *string, *temp_string;
  int c, char_index, string_index, string_length, ro;
  SHELL_VAR *temp_var;

  create_variable_tables ();

  for (string_index = 0; string = env[string_index++]; )
    {
      char_index = 0;
      name = string;
      while ((c = *string++) && c != '=')
        ;
      if (string[-1] == '=')
        char_index = string - name - 1;

      /* If there are weird things in the environment, like `=xxx' or a
         string without an `=', just skip them. */
      if (char_index == 0)
        continue;

      /* ASSERT(name[char_index] == '=') */
      name[char_index] = '\0';
      /* Now, name = env variable name, string = env variable value, and
         char_index == strlen (name) */

      temp_var = (SHELL_VAR *)NULL;

      /* If exported function, define it now.  Don't import functions from
         the environment in privileged mode. */
      if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
        {
          string_length = strlen (string);
          temp_string = (char *)xmalloc (3 + string_length + char_index);

          strcpy (temp_string, name);
          temp_string[char_index] = ' ';
          strcpy (temp_string + char_index + 1, string);

          if (posixly_correct == 0 || legal_identifier (name))
            parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
```



```
/* Parse and execute the commands in STRING.  Returns whatever
   execute_command () returns.  This frees STRING.  FLAGS is a
   flags word; look in common.h for the possible values.  Actions
   are:
        (flags & SEVAL_NONINT) -> interactive = 0;
        (flags & SEVAL_INTERACT) -> interactive = 1;
        (flags & SEVAL_NOHIST) -> call bash_history_disable ()
        (flags & SEVAL_NOFREE) -> don't free STRING when finished
        (flags & SEVAL_RESETLINE) -> reset line_number to 1
*/

int
parse_and_execute (string, from_file, flags)
     char *string;
     const char *from_file;
     int flags;
{
    …
```

Fix adds:



```
+ #define SEVAL_FUNCDEF 0x080       /* only allow function definitions */
+ #define SEVAL_ONECMD  0x100       /* only allow a single command */
```

Missing some input validation checks…

Figure 10: Bug or Malware

Shellshock

- Bug is the due to the absence of code (validation checks)
- Present for 25 years!?
- Even more complicated to find
- Still learning the extent of this bug

Shellshock

Figure 11: Bug or Malware

Passwords

Does your computer “store” your password?

Should it?

Password Hashing

Goal: Don't store passwords!

Ideal Goal: Don't even "encrypt" passwords

- `hash(x) == hash(x)`
- `hash(x) != hash(y)`
- if `hash(x) == hash(x')` then `x` has not changed
- given `hash(x)`, `x` cannot be recovered
- it is infeasible to find a collision such that `hash(m1) == hash(m2)`

Trusting Trust

- In 1984 Ken Thompson was presented with the ACM Turing Award
  - Famous acceptance speech “Reflections On Trusting Trust”
  - Highly encourage to read the speech
  - Link: 3 Page PDF

Misconfigurations

Probably the biggest contributor to insecure operating systems

- Running a web server as root
- Default usernames and password
- Anonymous read/write FTP access
- Storing passwords in cleartext
- Improper permissions on files/executables
- Unpatched/old software, disabling system updates

Resources

- CprE 431/531 Information System Security
- CprE 532 Information Warefare
- CprE 533 Cryptography
- CprE 536 Computer and Network Forensics
- Security Technical Implementation Guides (STIGs)

Compete to win free security training invitation

US Cyber Challenge (http://uscc.cyberquests.org)