Software vulnerabilities p.2

Moshe Kravchik
Credits: Mark Stamp, Tzachy Reinman, John Mitchell

https://flic.kr/p/kXT13r

1

Agenda

- Race conditions
- Integer overflow
- Input validation
- SQL Injection

https://flic.kr/p/6keSjR

2

Race conditions

- Arise when security-critical process occurs in stages
- Attacker makes change between stages
- Between authorization and use
- TOCTOU – time of check, time of use
- Security processes should be atomic

https://flic.kr/p/e4RZaS

3

TOCTOU example

```
function readFile($filename){
    $user = getCurrentUser();

    //resolve file if its a symbolic link
    if(is_link($filename)){
        $filename = readlink($filename);
    }

    if(fileowner($filename) == $user){
        echo file_get_contents($filename);
        return;
    }
    else{
        echo 'Access denied';
        return false;
    }
}
```

Time of Check

Change a file with link to /etc/shadow HERE

Time of Use

4

Integer overflow

- One of the most popular exploits
- Usually prepares grounds for memory exploits
- Programmers rarely think about rare overflow cases..
- Not only C/C++ - see exploits of overflow in Java

5

Integer overflow explained

- Standard unsigned integer types

Type	Bits	Range	x86-32
unsigned char	8	0 - 255	255
unsigned short int	16	0 - 65,535	65,535
unsigned int	16*	0 - 65,535	4,294,967,295
unsigned long int	32	0-4,294,967,295	4,294,967,295
unsigned long long int	64	$0 - 2^{64} - 1$	$0 - 2^{64} - 1$

*at least

- What is the result of:

```
unsigned char c = 255 + 1
```

6

Integer overflow explained

- Unsigned integers wraparound (modulo MAX+1)
- E.g., unsigned char:
  the result of 250+8 is 258 modulo 256, which is 2
- What is wrong with the following example:

```
for (unsigned int i = n; i >= 0; i--)
```

- Real-life example:
  - 1100 flights were grounded due to a crash of a flight-crew-scheduling software (Comair, 2004)
  - The SW used a 16-bit counter, limiting the number of changes to 32,768 a month
  - Storms -> too many changes -> system crash

7

Integer overflow exploitation

```
int* myfunction(int *array, unsigned int len)
{
      int *myarray; unsigned int i;
      myarray = malloc(len * sizeof(int));
      for(i = 0; i < len; i++)
            myarray[i] = array[i];
      return myarray;
}
```

- If len is large, len * sizeof(int) will wraparound
- The consequent copy will overwrite the heap!

8

Integer overflow explained

- Standard signed integer types

Type	Bits	Range	x86-32
char	8	-128- to 127	
short int	16	–32,768 to 32,767	
int	16*	–32,768 to 32,767	–2,147,483,648 to 2,147,483,647
long int	32	–2,147,483,648 to 2,147,483,647	–2,147,483,648 to 2,147,483,647
long long int	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

9

Signed Integer Type - Sign and Magnitude

- The sign bit represents whether the value is negative (sign bit set to 1)
  or positive (sign bit set to 0)
- The other value bits represent the magnitude of the value in pure binary notation
- For Example:

  00101011 = 43

  10101011 = -43

10

Signed Integer Type - Two's Complement

- Result of subtracting the number from 2^N
- To negate a two's complement value:
  - Toggle each bit, including the sign bit
  - Add 1 (with carries as required)
- For Example**:**

  00101011 = 43

  11010101 = -43

11

Overflowing signed integers

- According to C/C++ standard the signed overflow behavior is undefined
- Compiler can do what ever it wants!
  - Breaking the program silently in unpredictable ways, depending on optimization
  - Changing the behavior with compiler updates
- In practice overflowing will usually result in negative values

Value	Representation
127	01111111
…	….
1	00000001
0	00000000
-1	11111111
…	….
-127	10000001
-128	10000000

12

Signed overflow exploitation

```
int get_two_vars(int sock, char *out, int len)
{       char buf1[512], buf2[512];
        unsigned int size1, size2;
        int size;
        if(recv(sock, buf1, sizeof(buf1), 0) < 0)
                { return -1; }
        if(recv(sock, buf2, sizeof(buf2), 0) < 0)
                { return -1; }
        /* packet begins with length information */
        memcpy(&size1, buf1, sizeof(int));
        memcpy(&size2, buf2, sizeof(int));
        size = size1 + size2; /* [1] */
        if(size > len)
                { /* [2] */ return -1; }
        memcpy(out, buf1, size1);
        memcpy(out + size1, buf2, size2);
        return size;
```

13

Signed overflow exploitation

```
int get_two_vars(int sock, char *out, int len)
{
        char buf1[512], buf2[512];
        unsigned int size1, size2; int size;
        if(recv(sock, buf1, sizeof(buf1), 0) < 0)
                { return -1; }
        if(recv(sock, buf2, sizeof(buf2), 0) < 0)
                { return -1; }
        /* packet begins with length inf
        memcpy(&size1, buf1, sizeof(in
        memcpy(&size2, buf2, size
        size = size1 + size2; /* [1] */
        if(size > len)
                { /* [2] */ return -1; }
        memcpy(out, buf1, size1);
        memcpy(out + size1, buf2, size2);
        return size;
```

size1 = 0x7fffffff
size2 = 0x7fffffff
size1+ size2 = -2!

Arbitrary memory write!

14

Problematic conversions

- Mixing signed and unsigned – BAD idea (read here and here)
  - Rules for conversion are complex
  - Converted to signed if fits, otherwise to unsigned
  - Constants are always signed

```
int copy_something(char *buf, int len)
{
        char kbuf[800];
        if(len <= 800)
                return memcpy(kbuf, buf, len);
}
```

```
int main (void)
{
  long a = -1;
  unsigned b = 1;
  printf ("%d\n", a
> b);
  return 0;
}

Prints 0 in 64 bits
Prints 1 in 32 bits
```

memcpy expects unsigned int
Passing len = -2 will cause buffer overwrite

- Truncating

```
unsigned int ui = 300;
unsigned char uc = ui;
```

uc = 300 – 256 = 44

15

Overflow detection and mitigation

- Proper type selection (large enough, matches the operated types)

```
short total = strlen(argv[1] )+ 1;
size_t total = strlen(argv[1] )+ 1;
```

- Short is too short!
- Short is signed, losing half of values

- Range checking
  - Mostly forgotten
  - Causes code bloat
- Compiler features (GCC and clang –ftrapv flag)
- Automatic detection tools

16

Overflow detection and mitigation

- The CERT C Secure Coding Standard has several rules to prevent range errors:
  - INT30-C. Ensure that unsigned integer operation do not wrap
  - INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
  - INT32-C. Ensure that operations on signed integer do not result in overflow

17

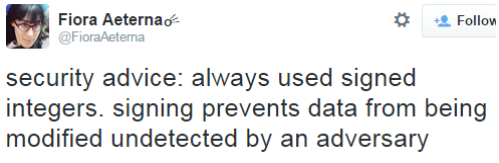
Range checks are not trivial

- Is this check enough?

```
unsigned int i,sum;
//set values to i and sum…
if (sum + i < UINT_MAX)
   sum += i
```

- No. It can wraparound too…
- The fix:

```
if (i <
UINT_MAX - sum)
```



18

Input validation

- Range checks are an example of input validation

Consider: strcpy(buffer, argv[1])

A buffer overflow occurs if

len(buffer) < len(argv[1])

Software must validate the input by checking the length of argv[1]

- Becomes hard if the language is complex

19

Server side validation

- Consider web form data
- Suppose input is validated on client

For example, the following is valid

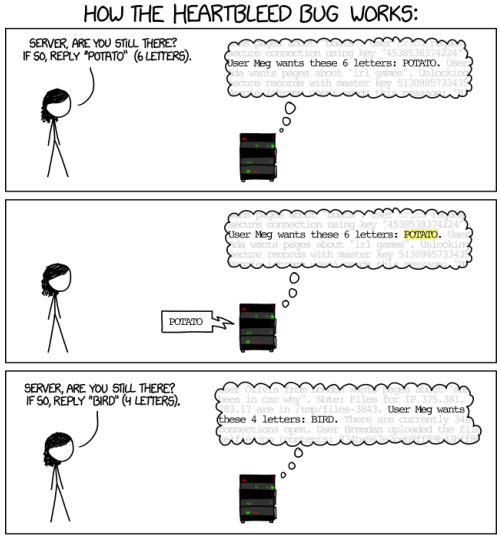
http://www.things.com/orders/final&custID=112&num=55A&qty=20&price=10&shipping=5&total=205

- Suppose input is not checked on server
- Why bother since input checked on client?
- Then attacker could send http message

http://www.things.com/orders/final&custID=112&num=55A&qty=20&price=10&shipping=5&total=25

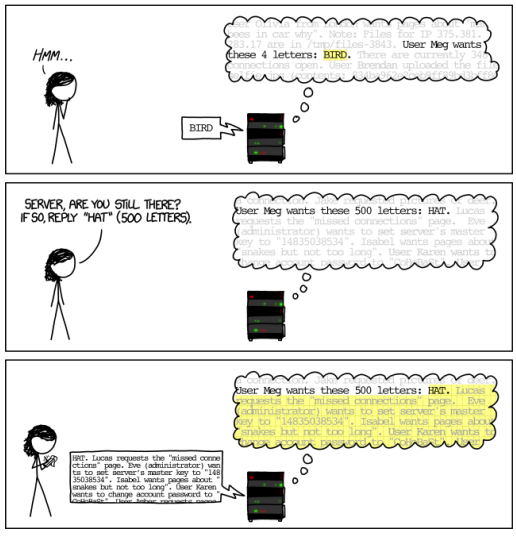
20

Input validation - Heartbleed



21

Input validation - Heartbleed




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

```
//receive request
packet = malloc(amt);
read(socket, packet, amt);
//build response
buffer = malloc(packet->len);
memcpy(buffer,
   packet->payload, packet->len);
write(s, buffer, packet->len);
```

22

Command injection

- Don't trust the user to do what he was supposed to!

```
filename = GetFileName()
system ("ls  -l " + filename + "> output")
print output;
```

```
➢ Enter file name
➢ file; cat /etc/shadow
```

Prints the passwords file

```
filename = GetFileName()
filename = CanonicalFileName(filename)
system ("ls  -l " + filename + "> output")
print output;
```

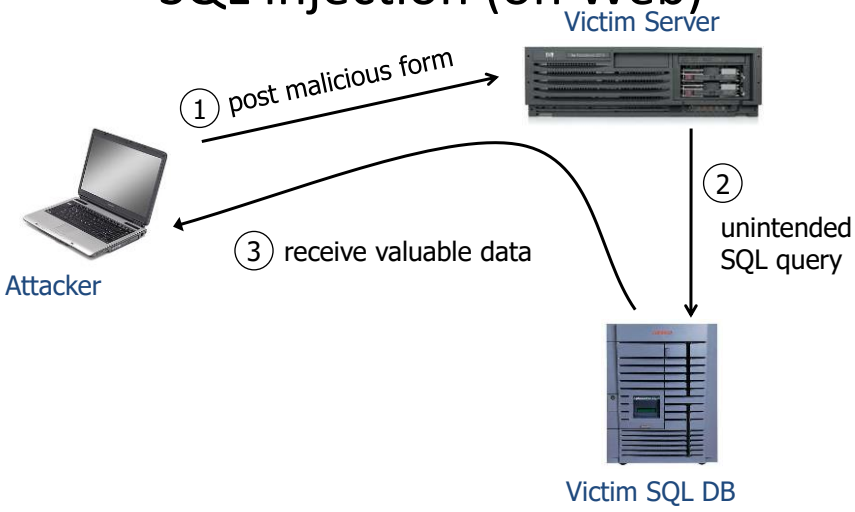
24

SQL injection

- Similar to command injection
- Uses SQL to abuse database command
- Exploits the lack of input validation or canonicalization
- One of the top web site vulnerabilities

26

SQL injection (on Web)



27

Sample Query

```
SELECT <columns> from <tbl> where <exp>
```

select * from comments
where user_id = 2;

2, 2, “I like sugar”
2, 3, “But not milk”

user_id	comment_id	comment
1	1	Test Comment
2	2	I like sugar
2	3	But not milk
3	4	Gordon is silly

comments

28

Tautologies

SELECT *<columns>* **from** *<tbl>* **where** *<exp>*

select * from comments
where user_id = 2
OR 1= 1;

1, 1, "Test Comment"
2, 2, "I like sugar"
2, 3, "But not milk"
3, 4, "Gordon is silly"

user_id	comment_id	comment
1	1	Test Comment
2	2	I like sugar
2	3	But not milk
3	4	Gordon is silly

comments

Tautologies are useful for attacks

29

Database queries with PHP

- Sample PHP

```
$recipient = $_POST['recipient'];
$sql = "SELECT PersonID FROM Person
WHERE    Username=".$recipient;
$rs = $db->executeQuery($sql);
```

- What if 'recipient' is a malicious string that changes the meaning of the query?

```
$sql = "SELECT PersonID FROM Person
WHERE    Username=x or 1=1";
```

Gets all IDs!

More SQL injection attacks (ASP)

```
set ok = execute( "SELECT * FROM Users
          WHERE user=' "  &  form("user")  & " '
   AND    pwd=' " & form("pwd") & " '" );

if not ok.EOF
            login success
else  fail;
```

```
ok = execute( SELECT …
                  WHERE user= '  ' or 1=1  -- … )
```

The "--" causes rest of line to be ignored.

```
ok = execute( SELECT …
            WHERE user= ' ' ; DROP TABLE Users  …  )
```

attacker can add users, reset pwds, etc.

31

More SQL injection attacks

```
set ok = execute( "SELECT * FROM Users
          WHERE user=' "  &  form("user")  & " '
   AND    pwd=' " & form("pwd") & " '" );

if not ok.EOF
            login success
else  fail;
```

user =

```
'; exec cmdshell
                  'net user badguy badpwd'/ADD --
```

Then script does:

```
ok = execute( SELECT …
                  WHERE username= '  ' ; exec …  )
```

If SQL server context runs as "sa", attacker gets account on DB server

32

CardSystems attack

- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business
- The Attack
  - 263,000 credit cards stolen from database
  - credit cards stored unencrypted
  - 43 million credit cards exposed

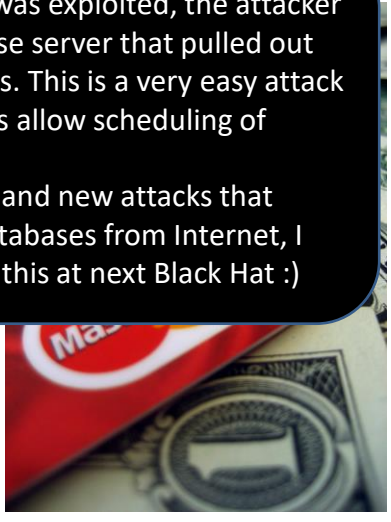
https://flic.kr/p/ayZf5K

33

CardSystems attack

What I have heard (from a trusted source) is that a SQL Injection vulnerability was exploited, the attacker created a Job in the database server that pulled out new records every 4 (?) days. This is a very easy attack since most database servers allow scheduling of actions as Jobs.
We have developed similar and new attacks that allows to steal complete databases from Internet, I hope we will be presenting this at next Black Hat :)
Cesar.

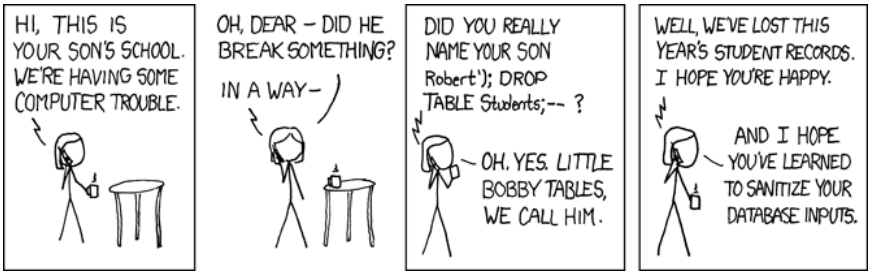
- CardS
  - cre
    com
  - SQL
  - put
- The At
  - 263,000 credi en from database
  - credit cards stored unencrypted
  - 43 million credit cards exposed



https://flic.kr/p/ayZf5K

34

Don't try this at home ☺



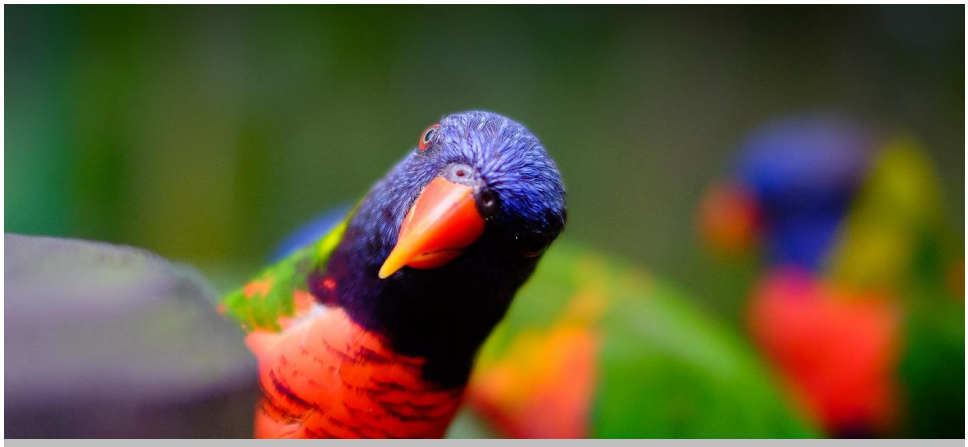
35

SQL injection mitigation

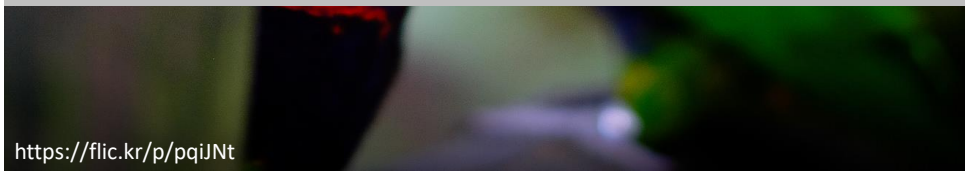
- Parameters escaping: ' → \'
- Use pre-built SQL queries (ensures arguments are converted to proper types)

```
SqlCommand cmd = new SqlCommand(
     "SELECT * FROM UserTable WHERE
     username = @User AND
     password = @Pwd", dbConnection);

     cmd.Parameters.Add("@User",
Request["user"] ); …

     cmd.ExecuteReader();
```

36



Questions?



37

Terms learnt

- Integer overflow
- Race conditions
- TOCTOU
- Input validation
- Server side validation
- Command injection
- SQL injection
- Canonicalization
- Escaping
- Prebuilt queries

38

38

Summary

- Pay attention to integer ranges and conversions
- Be aware of race conditions
- Don’t trust any input!
- Simple validation is not the best one

39