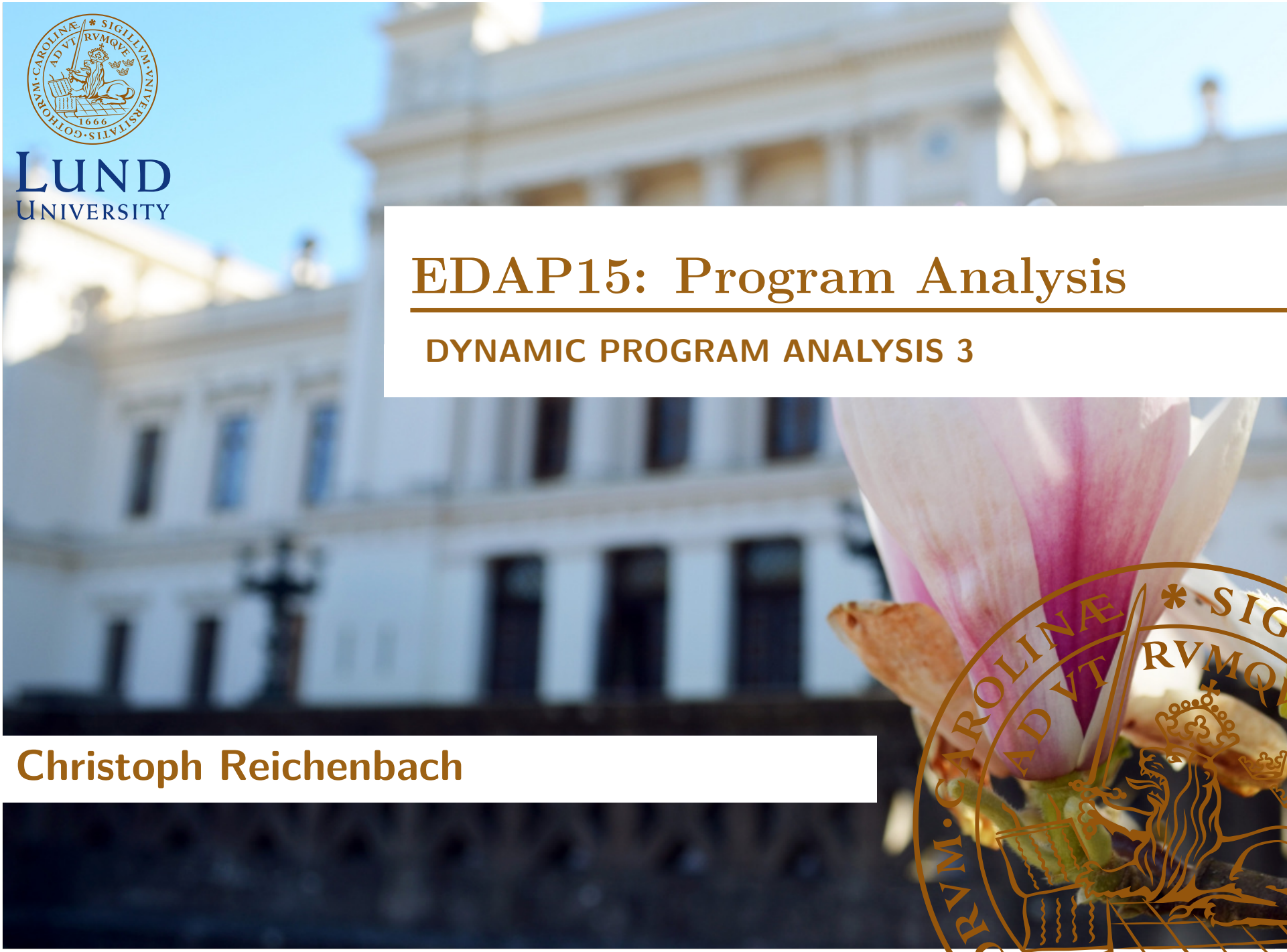


LUND UNIVERSITY

EDAP15: Program Analysis

DYNAMIC PROGRAM ANALYSIS 3

Christoph Reichenbach

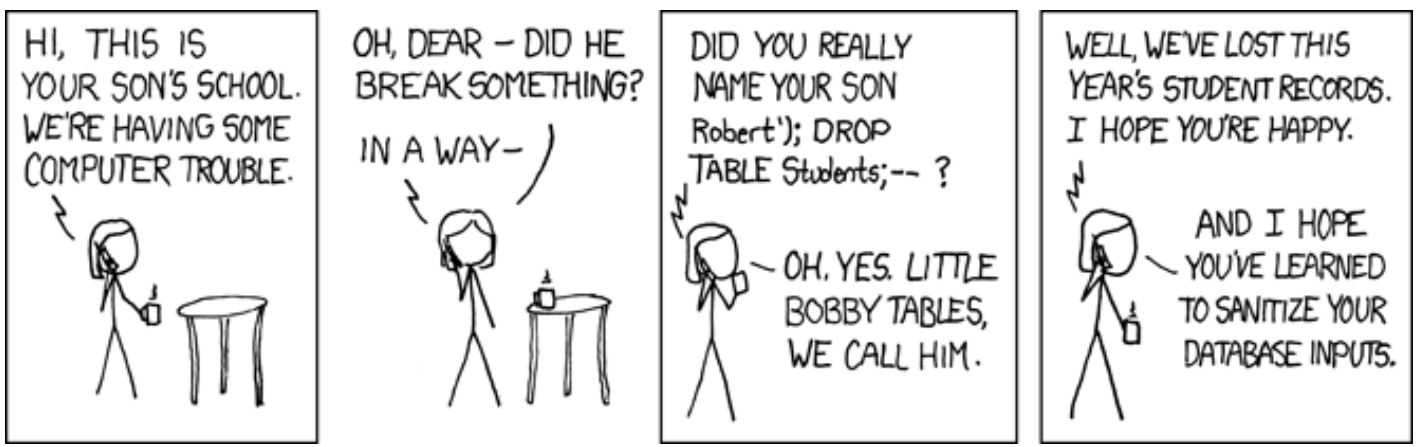
Welcome back!

Questions?

Tainted Values (1/2)

Python

```
username = request.GET['user']
...
q = sql.query("SELECT * from Users WHERE name='"
              + username + "'")
user_data = q.run
```

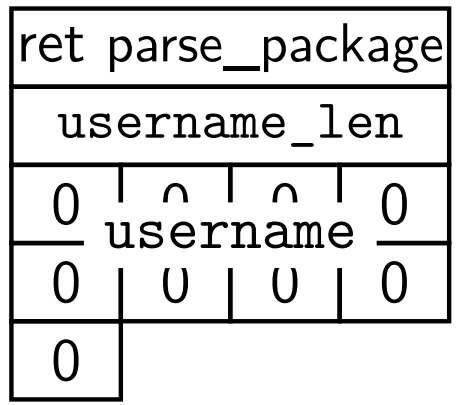


Tainted Values (2/2)

C

```
int parse_package(s* out, uint8* data) {
  char username[9] = { 0 };
  int username_len = data[0];
  // spec says:  length <= 8
  memcpy(username, data+1, username_len);
  ...
}
```

Stack

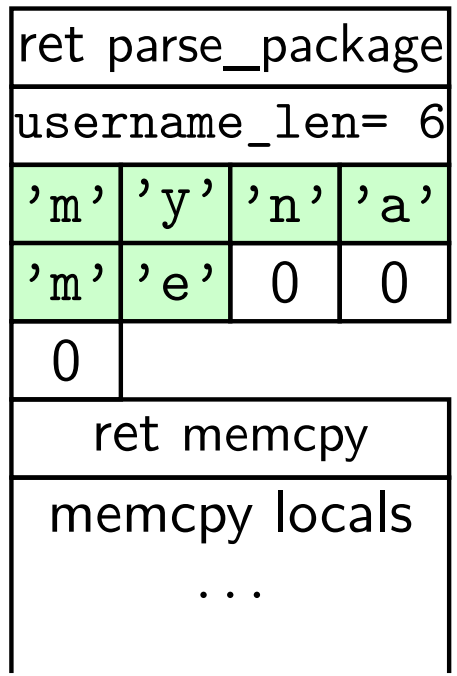


Tainted Values (2/2)

C

```
int parse_package(s* out, uint8* data) {
  char username[9] = { 0 };
  int username_len = data[0];
  // spec says:  length <= 8
  memcpy(username, data+1, username_len);
  ...
}
```

Stack

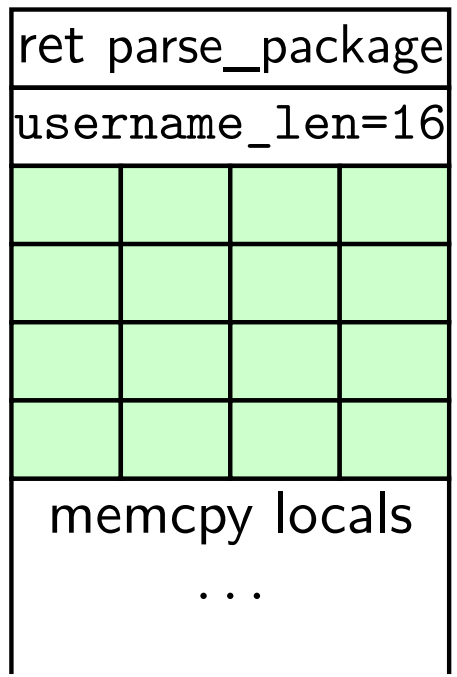


Tainted Values (2/2)

C

```
int parse_package(s* out, uint8* data) {
  char username[9] = { 0 };
  int username_len = data[0];
  // spec says:  length <= 8
  memcpy(username, data+1, username_len);
  ...
}
```

Stack



Tracing 'Tainted' Values

Taint Analysis:

- Track *tainted* values
- Remove taint if values are *sanitised*
- Detect if they reach sensitive *sinks*
- NB: Static taint analysis may also be possible

Unsafe input

- **Taint source**: Network ops
- **Sanitiser**: SQL string escape
- **Taint sink**: SQL query string

Leaking secrets

- **Taint source**: Plaintext passwd.
- **Sanitiser**: cryptographic hash
- **Taint sink**: Network ops

Dynamic Taint Analysis

```
query_l = "SELECT ...'"
query_r = "'"
username = request.GET['user']
...
query_str = query_l + username
query_str = query_str + query_r
q = sql.query(query_str)
```

```
query_l = "SELECT ..."ϵ
query_r = "'"ϵ
username = "..."t

query_str = "..."t
query_str = "..."t
Fault!
```

Dynamic Taint Analysis

Strategy:

- Annotate tainted values with *taint tags* or *shadow values*

```
s = read_network() // string in s will be tainted
t = "foo" + "bar"  // string in t will be untainted
```

- Extend operators to propagate taint:

$\oplus$	ϵ	**t**
ϵ	ϵ	t
t	t	t

$$\texttt{"foo"}^{v}[1] = \texttt{"o"}^{v}$$

$$\texttt{"foo"}^{v}\texttt{+"bar"}^{w} = \texttt{"foobar"}^{v \oplus w}$$

- Check taint sinks for tainted input
- Needs instrumentation (shadow values) or explicit support by runtime (e.g., Perl, Ruby)

Conditionals

- Should conditionals propagate taint?
- Usually such *control dependencies* don't propagate taint

Python

```
if secret_password == '':
    network_send('Account disabled, cannot log in');
```

Attackers vs. Taint Ananlysis

Is taint analysis 'sound enough' to detect attempts to expose sensitive data?

- *Attackers can subvert this analysis via control dependencies:*

C

```
for (i = 0; i < 16; ++i) {
  for (k = 0; k < 8; ++k) {
    if (secret_password[i] & 1 << k) {
      network_send("Meaninless Message");
    } else {
      network_send("Something Else");
    }
} }
```

System Command Attack

C

```
char d_secret[1024];
strcpy(d_secret, "/tmp/");
strcat(d_secret, secret); // taint d_secret

int iopipes[2];
pipe(iopipes);
...
if (fork()) { // create child process
  // connect pipes
  execv("/bin/rm", d_secret); // call external 'rm'
}
char[1024] buf; // untained!
read(iopipes[0], ...); // read output from 'rm'
```

System call will print e.g.:

```
rm:  cannot remove '/tmp/mysecretstring':  No such file or
directory
```

Side Channel Attacks

Many more attacks possible:

- Timing attacks:
  - Two threads
  - One sends signal to other, with delays
  - Delay loop length dependent on secret
- File length attack:
  - Write dummy file
  - File length (or other metadata) encodes secret
- Graphics buffer attack:
  - Write to screen
  - Read back with OCR
  - Or adjust widget position / font size to encode secret

Summary

- Dynamic taint analysis tracks **tainted** values (from **taint sources**)
- Tags also referred to as **shadow values**
- Removes taint if values are **sanitised**
- Detects attempts to use tainted values in **taint sinks**
- Still many weaknesses in analysis:
  - Control-dependence attacks
  - System command attacks
  - Side-channel attacks
- Can be strengthened with *symbolic* techniques

Outlook

- Lecture Wednesday at **08:00** (usual room)
- Review session: Bring your questions!

`http://cs.lth.se/EDAP15`