



University
of Glasgow

Security

Networked Systems 3
Lecture 18

Lecture Outline

- Security considerations
 - Traffic monitoring, confidentiality and authentication
 - Validating input data
 - Buffer overflow attacks

Traffic Monitoring

- Possible to snoop on traffic on any network link
 - Wireless links – simply listen
 - Wired links – switches can be configured to forward a copy of all traffic to a particular link, for monitoring
- Ability to monitor traffic a legal requirement in many countries, for legal reasons
 - e.g., to enable authorised wiretaps by the police
 - Can also be exploited for malicious purposes

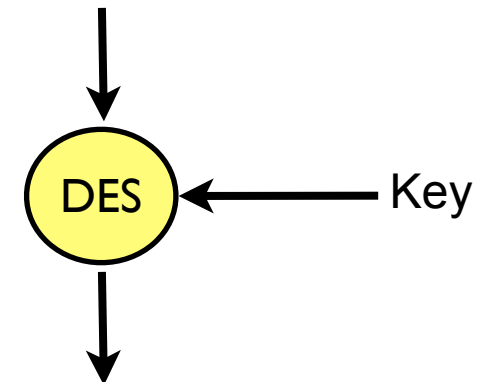
Confidentiality

- Must encrypt data to achieve confidentiality
- Two basic approaches
 - Symmetric cryptography
 - Advanced Encryption Standard (AES a.k.a. Rijndael), Triple-DES
 - Data Encryption Standard (DES) – broken; subject to brute-force attacks
 - Public key cryptography
 - The Diffie-Hellman algorithm
 - The Rivest-Shamir-Adleman (RSA) algorithm
 - Complex mathematics – will not attempt to describe

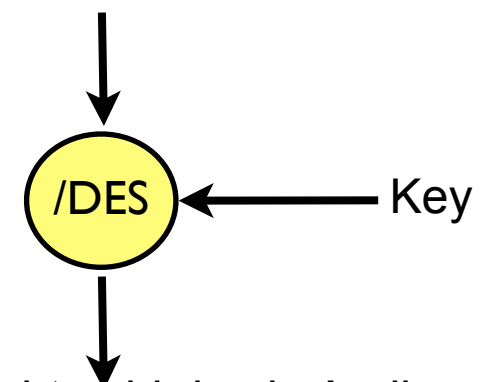
Symmetric Cryptography

- Function converts plain text into cipher-text
 - Fast – suitable for bulk encryption
 - Cipher-text is binary data, and may need base64 encoding
- Conversation is protected by a secret key
 - The same key is used to encrypt as is used to decrypt
 - Key must be kept secret, else security lost – a problem: how to distribute the key?

“It was a bright cold day in April,
and the clocks were striking
thirteen.”



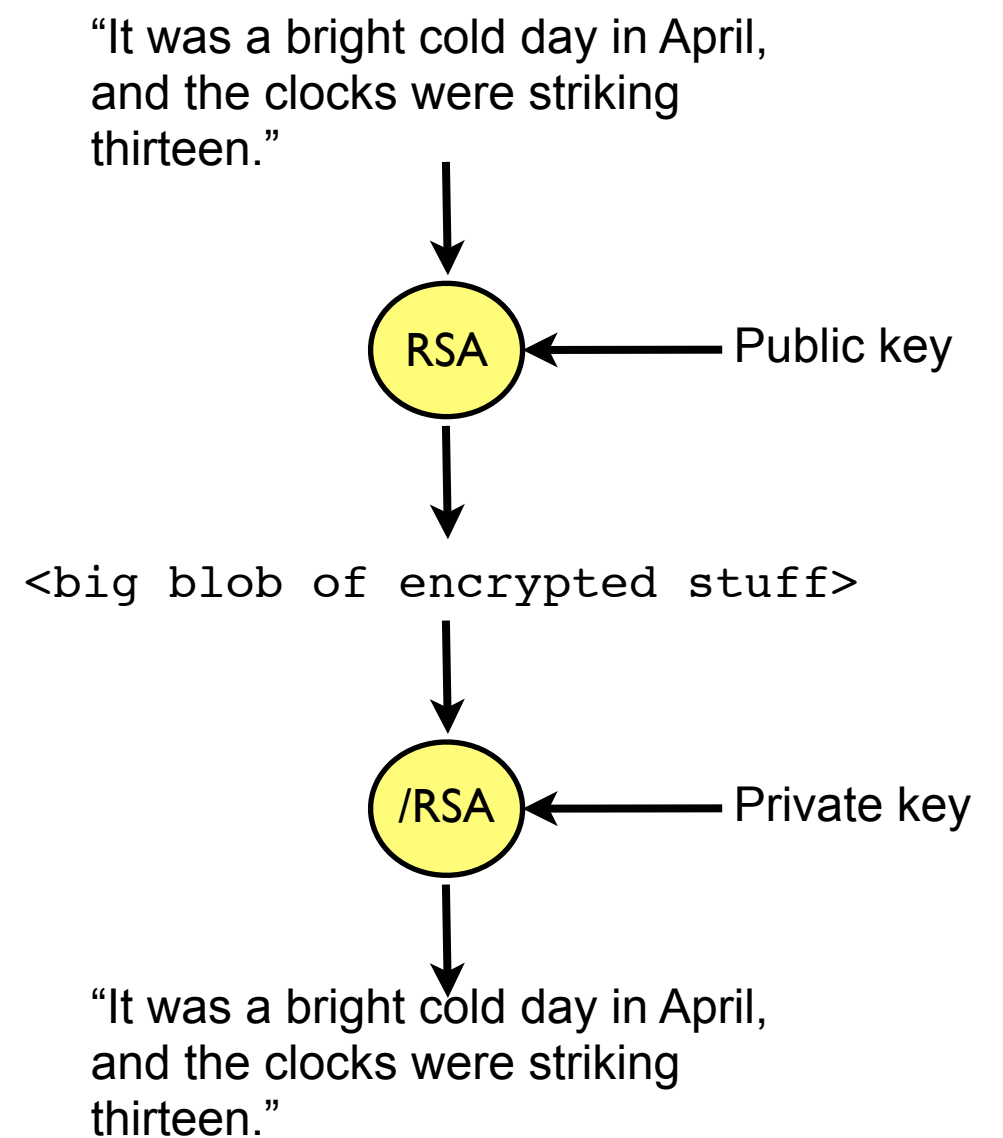
rX27qrhlM/Pd5UnkpqTuXnJBZecF1
bP5Xd8ouyAWgCLxZJUD951SaxusX5
bj002P9XkVGGHmM0qByZxu2pU+cCl
sERzuHKxc



“It was a bright cold day in April,
and the clocks were striking
thirteen.”

Public Key Cryptography

- Key split into two parts:
 - Public key – is widely distributed
 - Private key – must be kept secret
- Encrypt using public key
→ private key is needed to decrypt
 - Public keys are published in a well known directory → solves the key distribution problem
 - Problem: *very* slow to encrypt and decrypt



Hybrid Cryptography

- Use combination of public-key and symmetric cryptography for security and performance
 - Generate a random, ephemeral, *session* key that can be used with symmetric cryptography
 - Use a public-key system to securely distribute this session key – relatively fast, since session key is small
 - Encrypt the data using symmetric cryptography, keyed by the session key
 - Examples: PGP for email, SSL for web pages

Authentication

- Encryption can ensure confidentiality – but how to tell if a message has been tampered with?
 - Use combination of a *cryptographic hash* and public key cryptography to produce a *digital signature*
 - Gives some confidence that there is no *man-in-the-middle attack* in progress
- Can also be used to prove origin of data

Cryptographic Hash Functions

- Generate a fixed length (e.g., 160 bit) hash code of an arbitrary length input value
 - Should not be feasible to derive input value from hash
 - Should not be feasible to generate a message with the same hash as another
- Examples: MD5 and SHA-1
 - *Note: weaknesses found in both – care required!*

MD5("It was a bright cold day in April, and the clocks were striking thirteen") = 2c794fa2698f4b1bc5aa4e290abdf3a5

Digital Signature Algorithms

- Generating a digital signature:
 - Generate a cryptographic hash of the data
 - Encrypt the hash with your *private key* to give a *digital signature*
- Verifying a digital signature:
 - Re-calculate the cryptographic hash of the data
 - Decrypt the signature using the public key, compare with the calculated hash value → should match

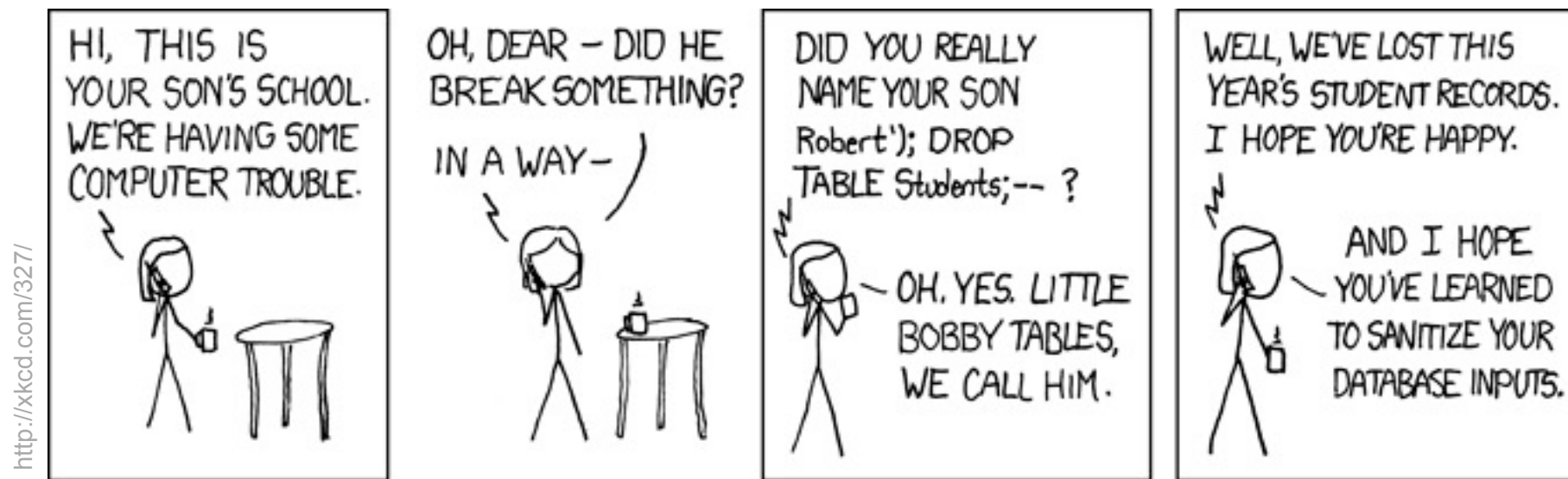
Existing Secure Protocols

- Wide range of existing security protocols give confidentiality and authentication:
 - IPsec
 - Transport Layer Security (TLS)
 - An enhancement to the Secure Sockets Layer (SSL)
 - Datagram TLS
 - Secure shell (ssh)
- Use them – don't try to invent your own!

Validating Input Data

- Networked applications fundamentally dealing with data supplied by un-trusted third parties
 - Data read from the network may not conform to the protocol specification
 - Due to ignorance and/or bugs
 - Due to malice, and a desire to disrupt services
- Must carefully validate all data before use

Malicious User Input



- Beware escape characters in user-supplied data!
- Must sanitise all user-supplied data before use
 - Stop malicious users including control characters that might disrupt operation of any scripting language inside your application

Buffer Overflow Attacks

- The C programming language doesn't check array bounds
 - Responsibility of the programmer to ensure bounds are not violated
 - Easy to get wrong – typically results in a “core dump”
 - What actually happens here?

Function Calls and the Stack

```
// overflow.c
#include <string.h>
#include <stdio.h>

static void
foo(char *src)
{
    char dst[12];

    strcpy(dst, src);
}

int
main(int argc, char *argv[])
{
    char hello[] = "Hello, world\n";

    foo(argv[1]);
    printf("%s", hello);
    return 0;
}
```

```
$ gcc overflow.c -o overflow
$ ./overflow 123456789012
Hello, world
$ ./overflow 1234567890123
Abort trap (core dumped)
$
```

What happens when `argv[1]` is longer than 12 bytes?

Function Calls and the Stack

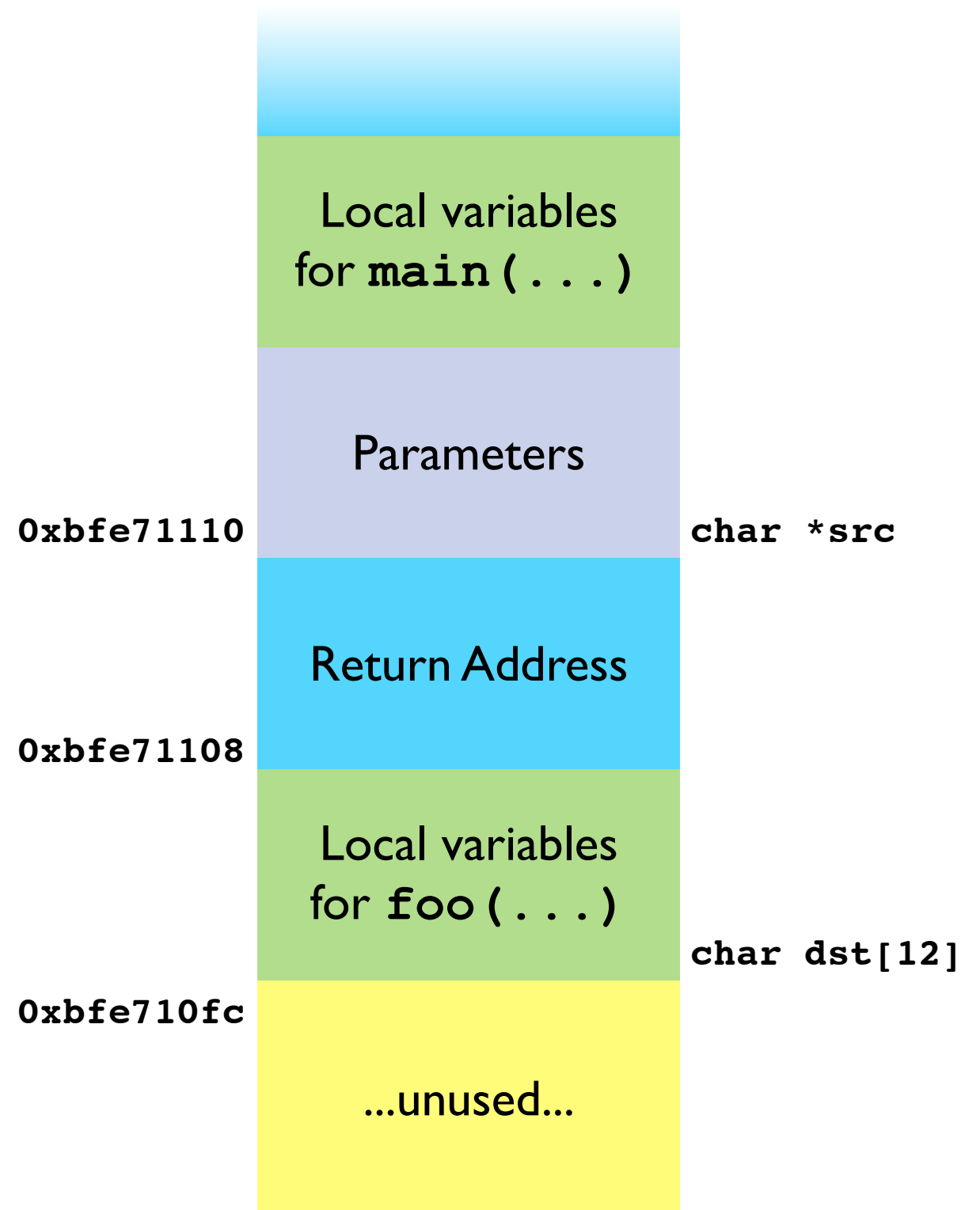
```
// overflow.c
#include <string.h>
#include <stdio.h>

static void
foo(char *src)
{
    char dst[12];

    strcpy(dst, src);
}

int
main(int argc, char *argv[])
{
    char hello[] = "Hello, world\n";

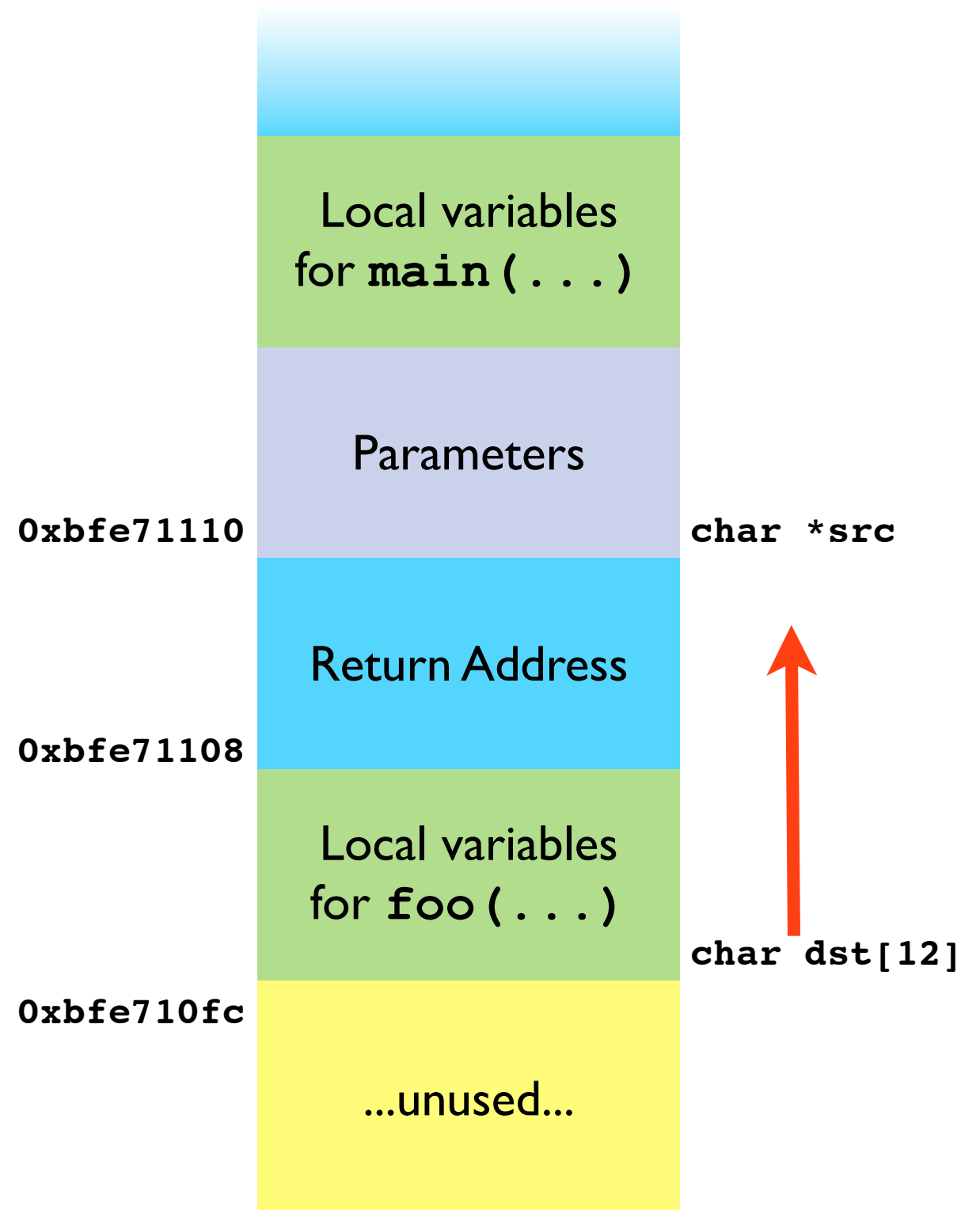
    foo(argv[1]);
    printf("%s", hello);
    return 0;
}
```



Example of call stack within
the call to the function `foo()`

Function Calls and the Stack

- The `strcpy ()` call doesn't check array bounds
- Overwrites the function return address on stack, along with the following memory locations
- If malicious, we can write executable code into this space, set return address to jump into our code...



Example of call stack within the call to the function `foo ()`

Arbitrary Code Execution

- Buffer overflows in network code are the primary source of security problems
 - If you write network code in C, but *very* careful to check all array bounds
 - If your code can be crashed by network traffic, it probably has an exploitable buffer overflow
- <http://insecure.org/stf/smashstack.html>

Questions?