CMPT 473
Software Quality Assurance

Security

Nick Sumner - Fall 2014

Basic Security

What are attributes of a secure program? CIA

Basic Security

What are attributes of a secure program? CIA

- (C)onfidentiality
  - No unauthorized information leaks

Basic Security

What are attributes of a secure program? CIA

- (C)onfidentiality
  - No unauthorized information leaks
- (I)ntegrity
  - No unauthorized data manipulation/corruption

Basic Security

What are attributes of a secure program? CIA

- (C)onfidentiality
  - No unauthorized information leaks
- (I)ntegrity
  - No unauthorized data manipulation/corruption
- (A)vailability
  - The system must be accessible as needed (no DOS)

Basic Security

What are attributes of a secure program? CIA

- (C)onfidentiality
  - No unauthorized information leaks
- (I)ntegrity
  - No unauthorized data manipulation/corruption
- (A)vailability
  - The system must be accessible as needed (no DOS)
- … (A_2)uthenticity
  - All actions are genuine, taken by the parties they claim/appear to be

Why Is This Special?

Poor security comes from unintended behavior.
→ Quality software shouldn't allow such actions anyway.

Why Is This Special?

Poor security comes from unintended behavior.
→ Quality software shouldn't allow such actions anyway.

- While our testing techniques so far find some security issues, many slip through! *Why?*

Why Is This Special?

Poor security comes from unintended behavior.
→ Quality software shouldn't allow such actions anyway.

- While our testing techniques so far find some security issues, many slip through! *Why?*
  - We cannot test everything

Why Is This Special?

Poor security comes from unintended behavior.
→ Quality software shouldn't allow such actions anyway.

- While our testing techniques so far find some security issues, many slip through! *Why?*
  - We cannot test everything
  - Concessions form part of an *attack surface*
    - Networks, Software, People

Why Is This Special?

Poor security comes from unintended behavior.
→ Quality software shouldn't allow such actions anyway.

- While our testing techniques so far find some security issues, many slip through! *Why?*
  - We cannot test everything
  - Concessions form part of an *attack surface*
    - Networks, Software, People
- Need additional policies & testing methods that specifically address security

What Could Possible Go Wrong?

- Many ways to attack different programs
- MITRE groups the most common into:

What Could Possible Go Wrong?

- Many ways to attack different programs
- MITRE groups the most common into:
  - Insecure Interaction
    - Data sent between components in an insecure fashion

What Could Possible Go Wrong?

- Many ways to attack different programs
- MITRE groups the most common into:
  - Insecure Interaction
    - Data sent between components in an insecure fashion
  - Risky Resource Management
    - Bad creation, use, transfer, & destruction of resources

What Could Possible Go Wrong?

- Many ways to attack different programs
- MITRE groups the most common into:
  - Insecure Interaction
    - Data sent between components in an insecure fashion
  - Risky Resource Management
    - Bad creation, use, transfer, & destruction of resources
  - Porous Defenses
    - Standard security practices that are missing or incorrect

[http://cwe.mitre.org/top25/#Categories]

Deeper Dive: Memory Safety

- Memory safety is a fundamental security issue for languages like C, C++, Assembly, ….

Deeper Dive: Memory Safety

- Memory safety is a fundamental security issue for languages like C, C++, Assembly, ….
- General idea:
  - Accessing memory when you shouldn't be able to might read, write, or execute inappropriately

Deeper Dive: Memory Safety

- Memory safety is a fundamental security issue for languages like C, C++, Assembly, ….
- General idea:
  - Accessing memory when you shouldn't be able to might read, write, or execute inappropriately
- Classic example:
  - stack buffer overflow attacks

Deeper Dive: Memory Safety

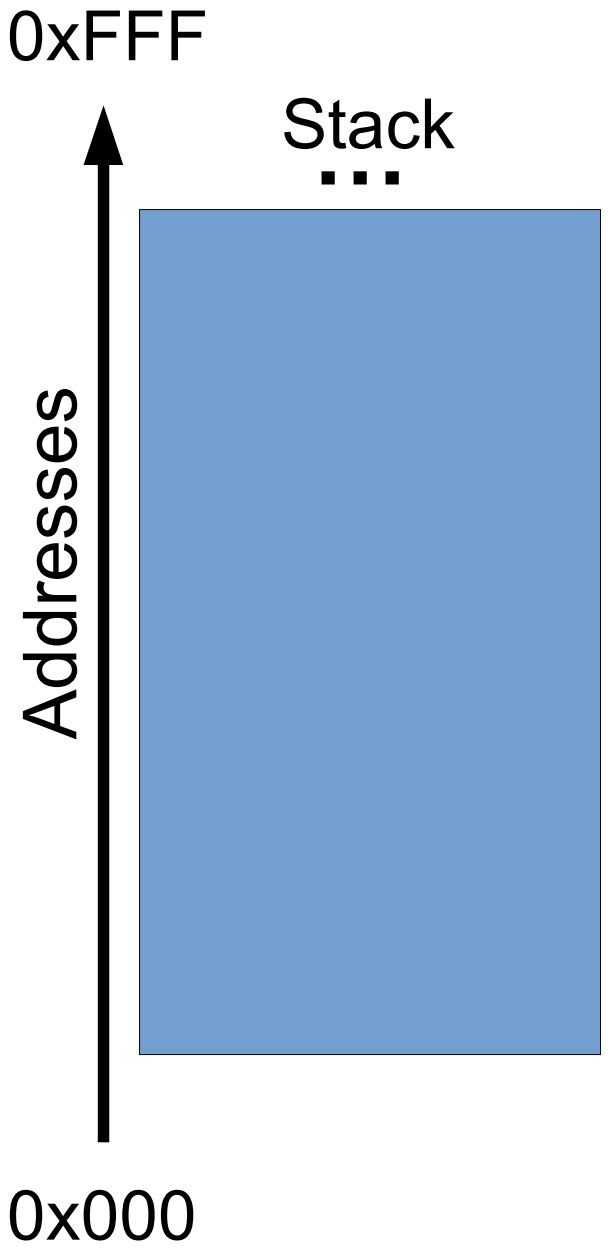
- Memory safety is a fundamental security issue for languages like C, C++, Assembly, ….
- General idea:
  - Accessing memory when you shouldn't be able to might read, write, or execute inappropriately
- Classic example:
  - stack buffer overflow attacks
  - Read more input into a buffer than the buffer can hold....

Deeper Dive: Memory Safety

- Memory safety is a fundamental security issue for languages like C, C++, Assembly, ….
- General idea:
  - Accessing memory when you shouldn't be able to might read, write, or execute inappropriately
- Classic example:
  - stack buffer overflow attacks
  - Read more input into a buffer than the buffer can hold....

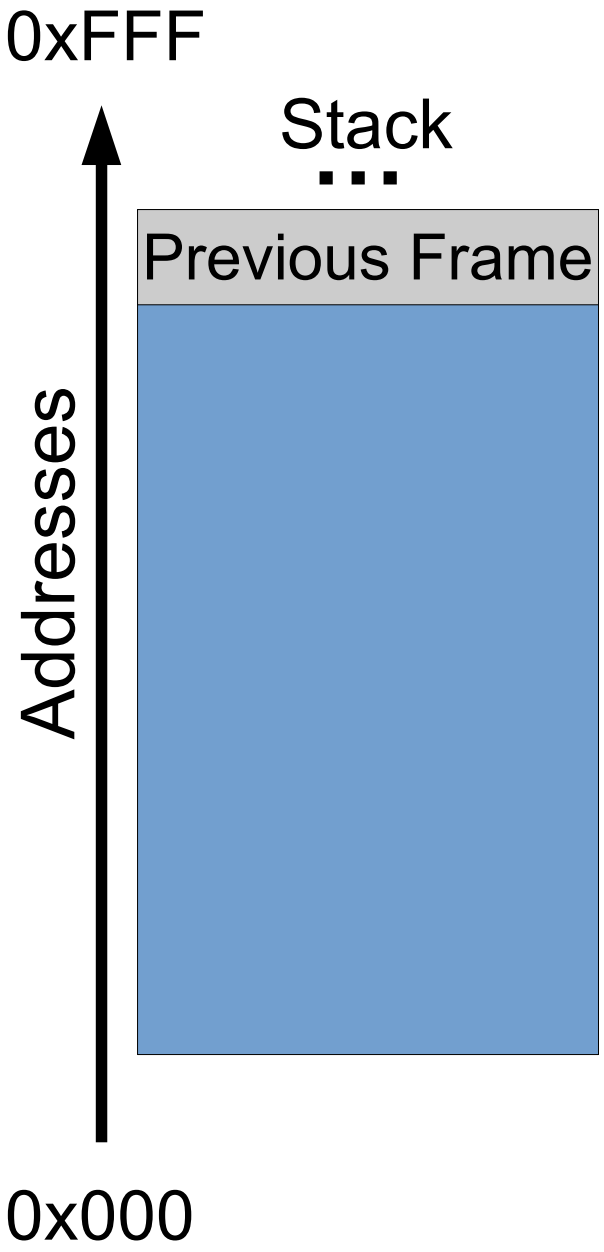
How many of you recall what a stack frame looks like?

Stack Buffer Overflows



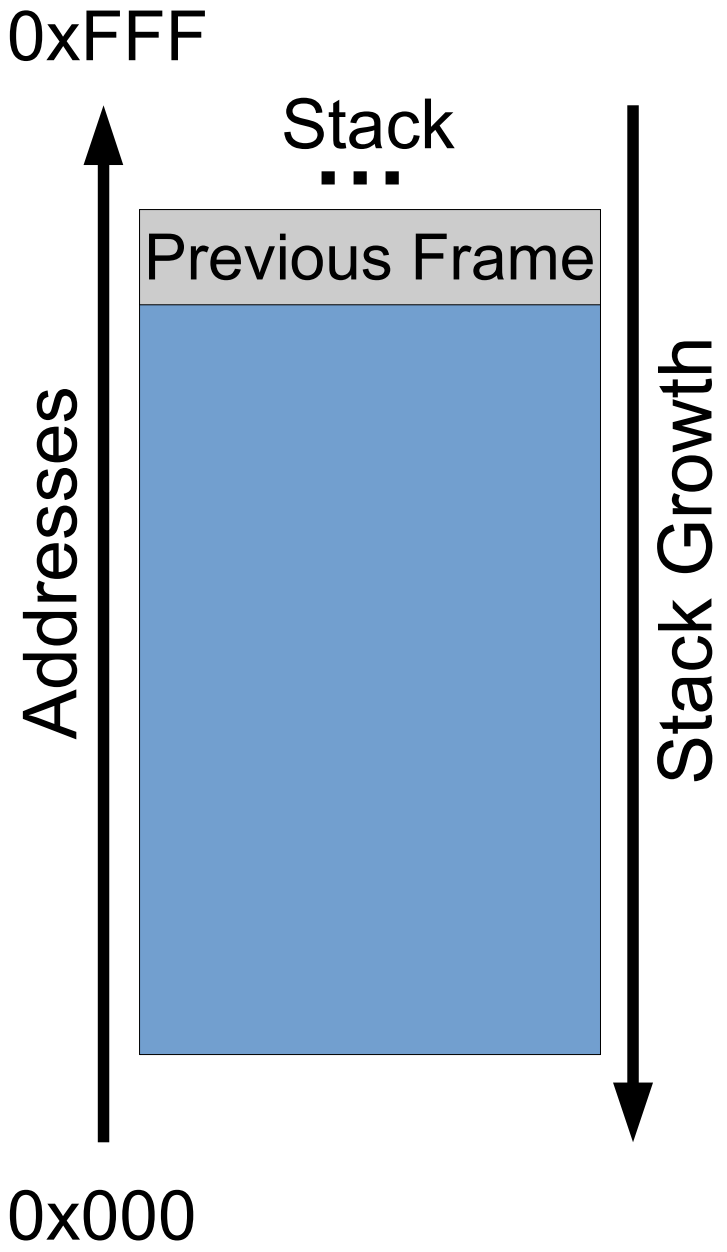

```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

Stack Buffer Overflows



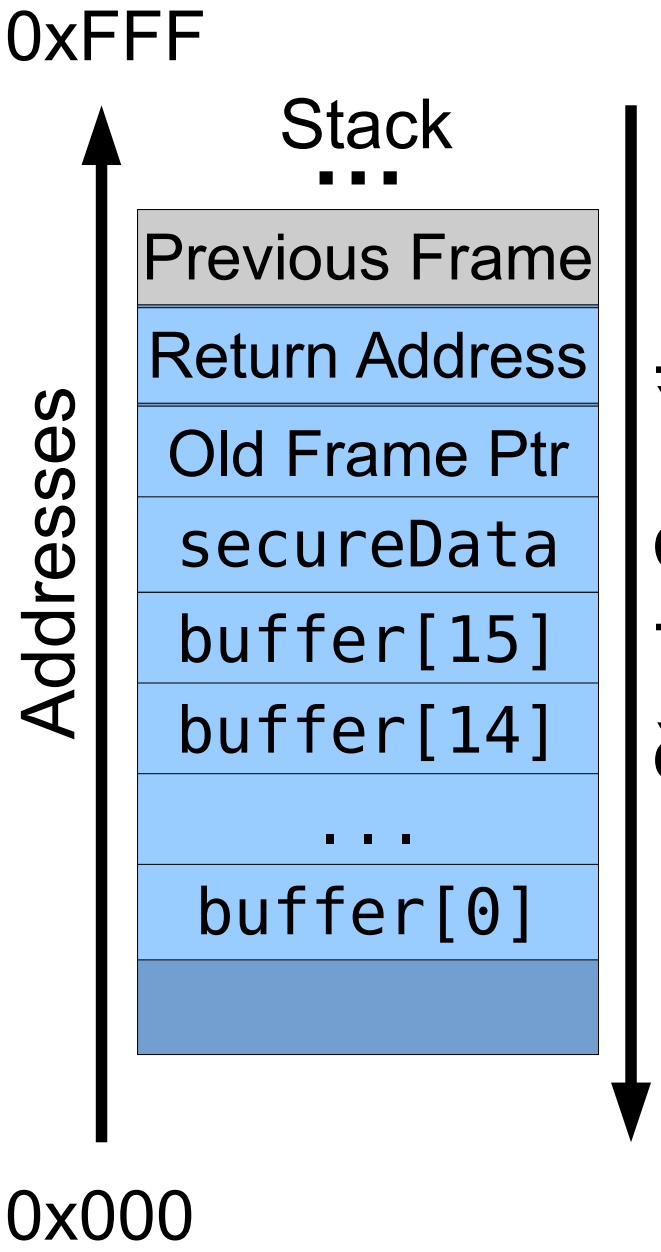

```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

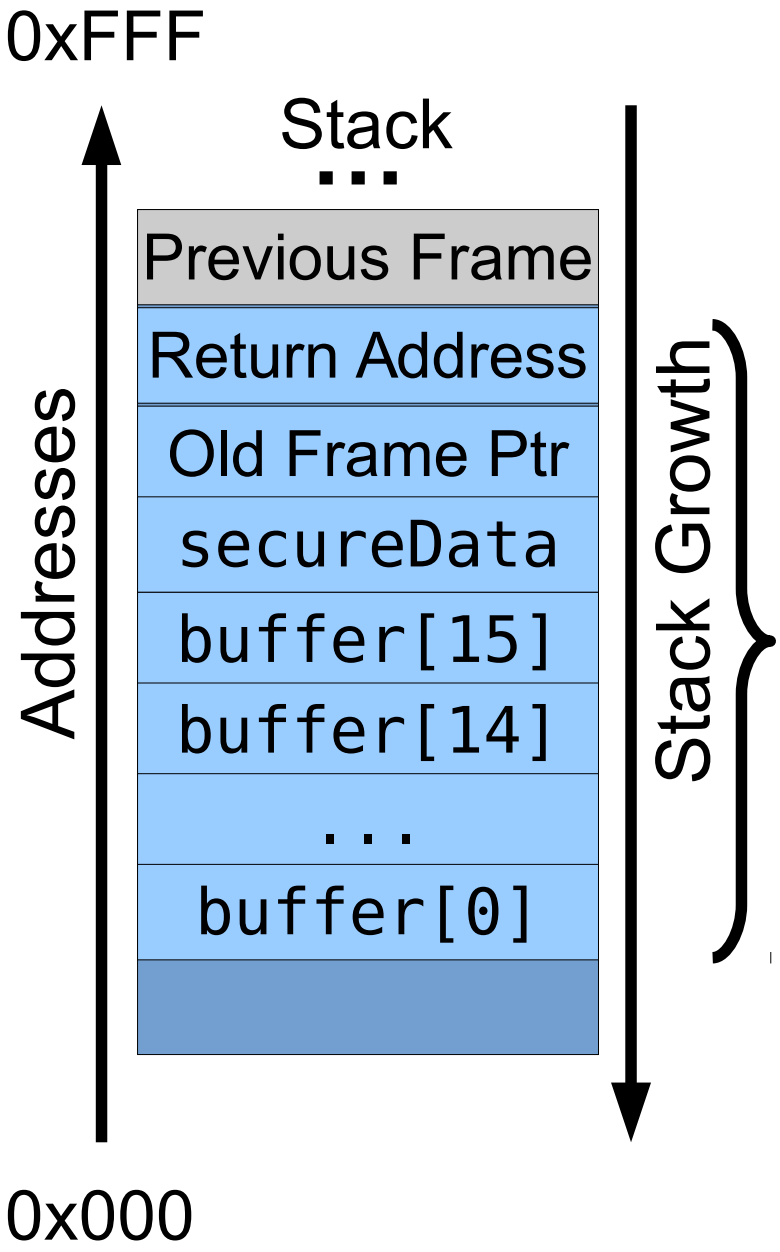
Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

Stack Frame for foo

Stack Buffer Overflows

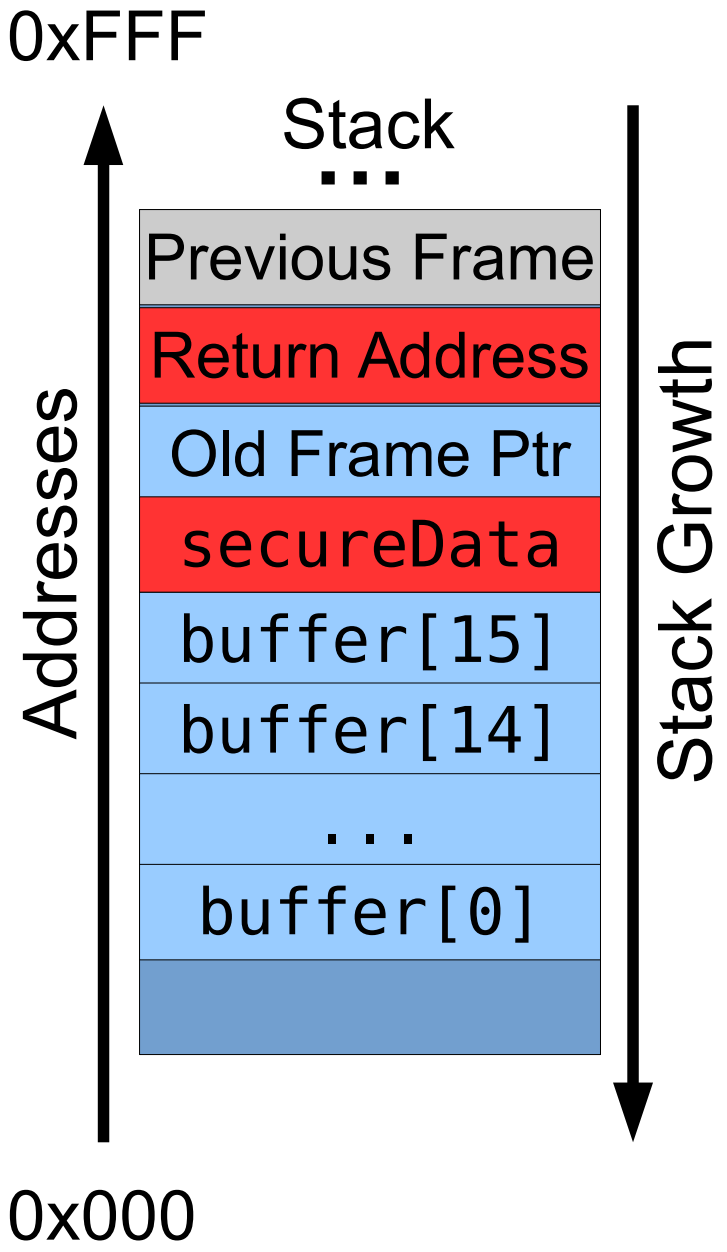



```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

Stack Frame for foo

What can go wrong?

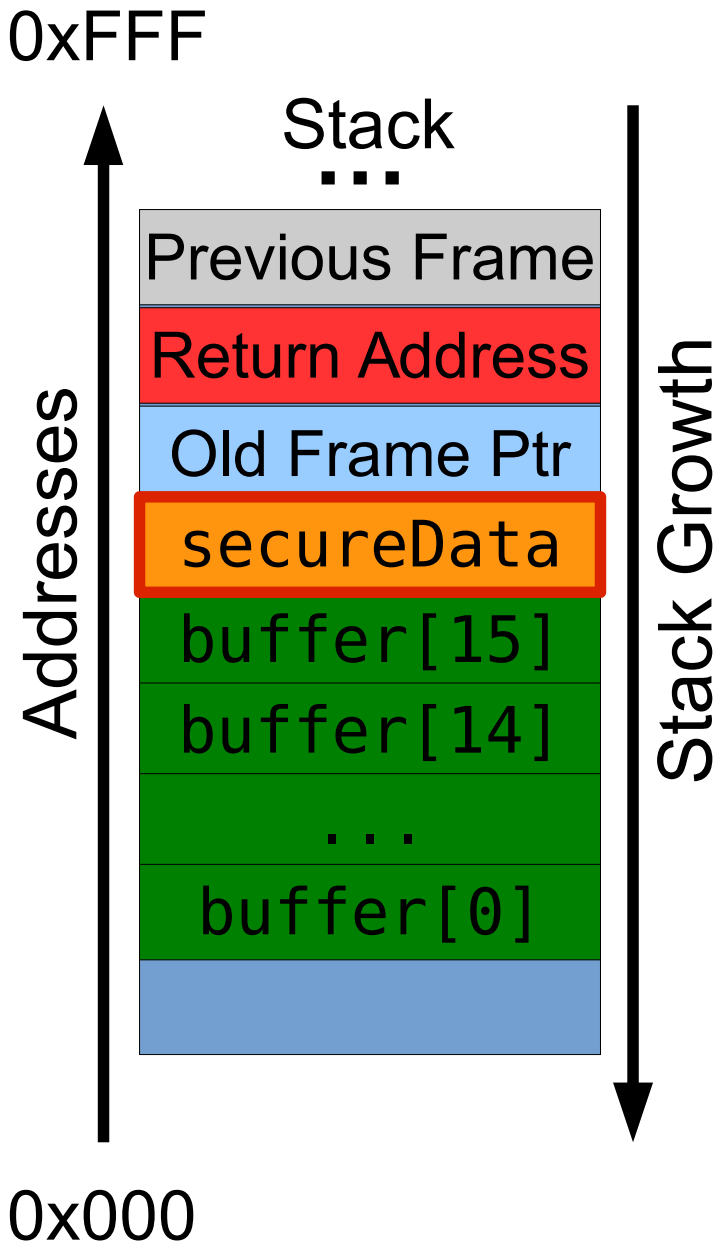
Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = “normal input”
+ “insecureData”

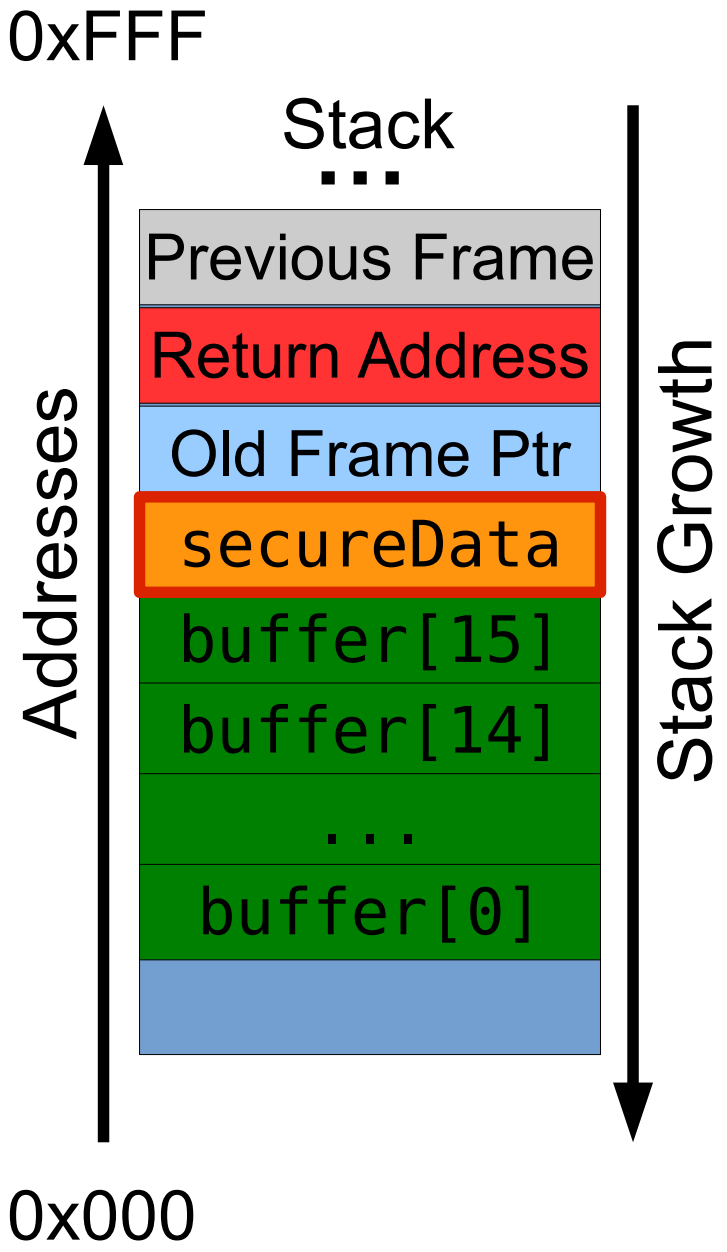
Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = “normal input”
+ “insecureData”

Stack Buffer Overflows

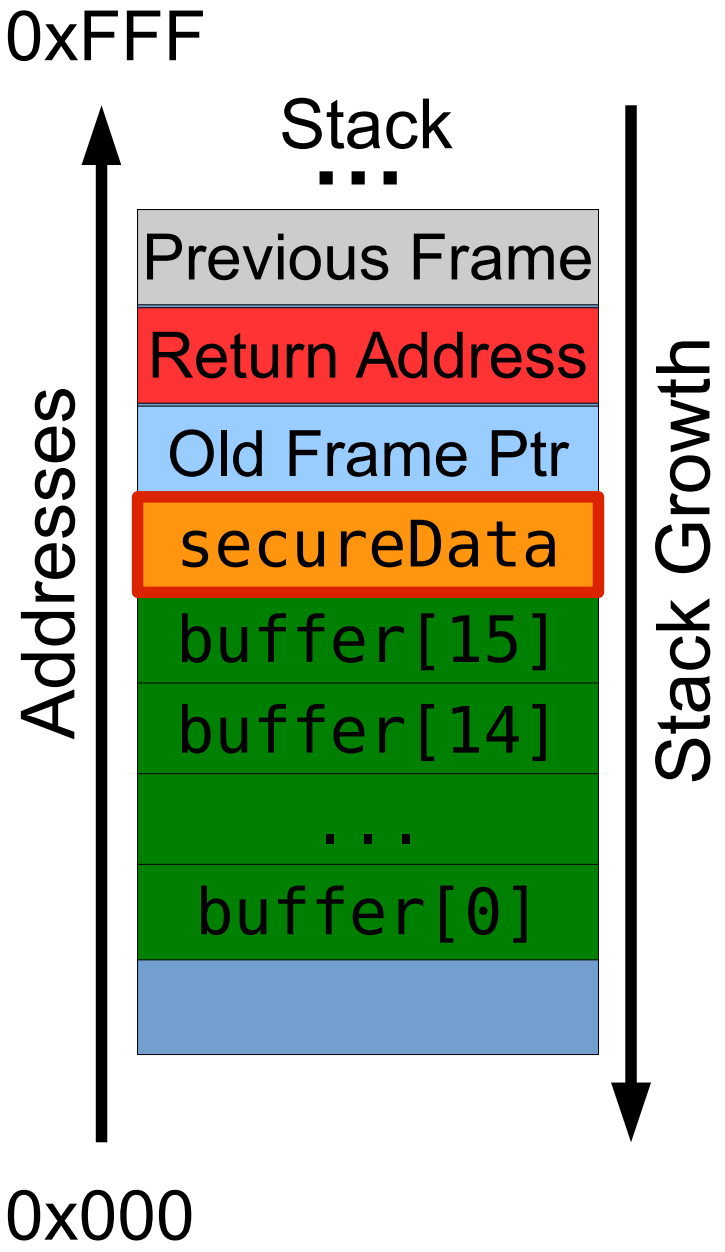



```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = “normal input”
+ “insecureData”

The integrity of the secure data is corrupted.

Stack Buffer Overflows



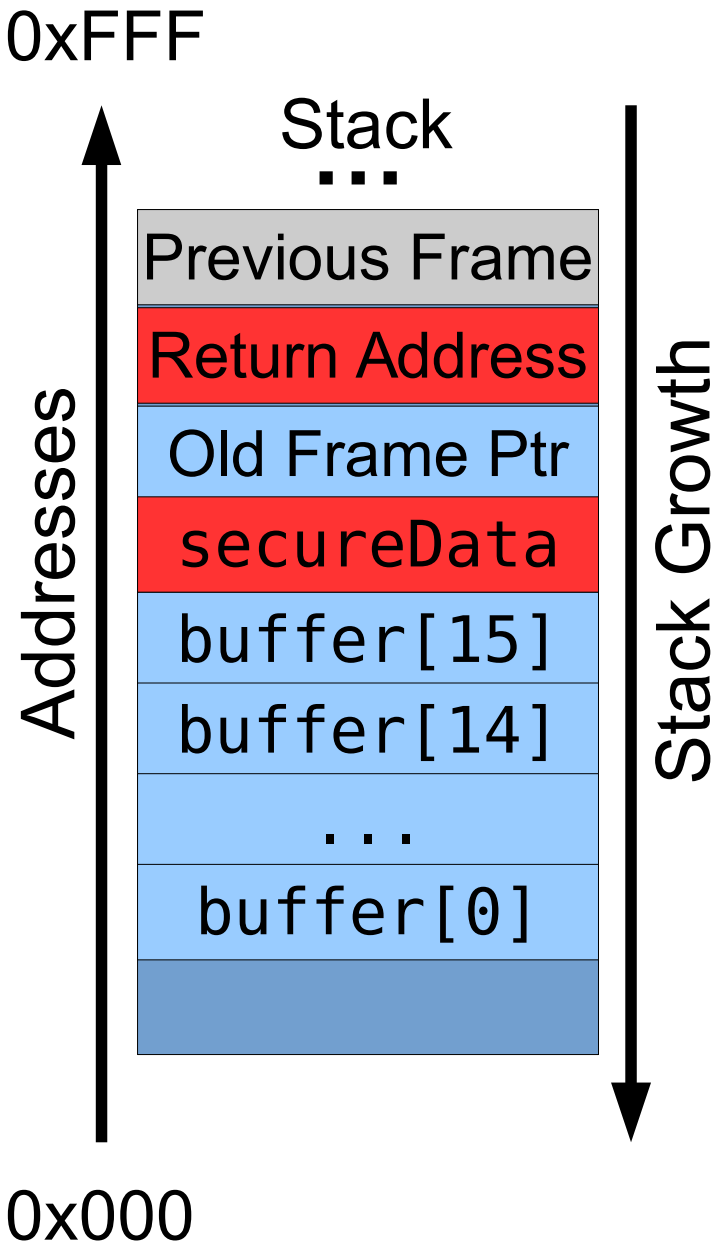

```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = "normal input"
+ "insecureData"

The integrity of the secure data is corrupted.

What else can go wrong?

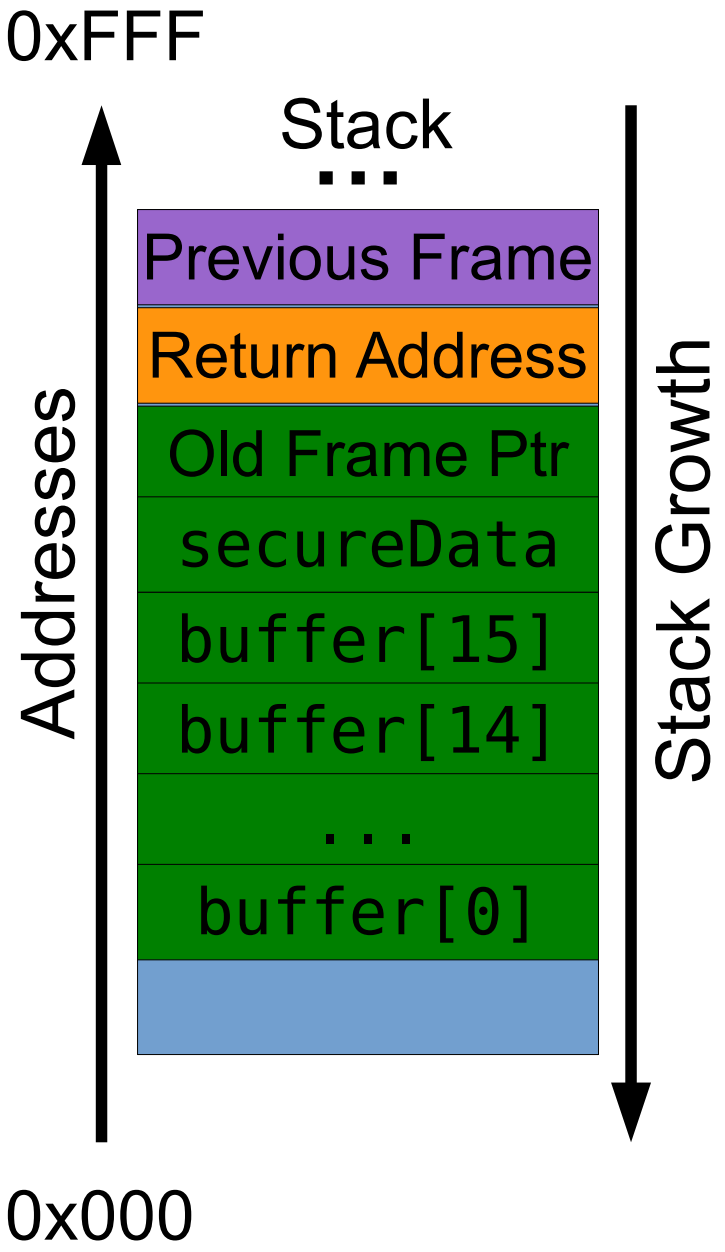
Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = “input”
+ “payload address”
+ “payload”

Stack Buffer Overflows




```
void foo(char *input) {
  unsigned secureData;
  char buffer[16];
  strcpy(buffer, input);
}
```

input = “input”
+ “payload address”
+ “payload”

What does this mean?

Memory Safety

- Vulnerabilities come from reading/writing/freeing
  - Out of bounds pointers
  - Dangling pointers

Memory Safety

- Vulnerabilities come from reading/writing/freeing
  - Out of bounds pointers
  - Dangling pointers
- Why doesn't Java face this issue?

Memory Safety

- Vulnerabilities come from reading/writing/freeing
  - Out of bounds pointers
  - Dangling pointers
- Why doesn't Java face this issue?
- Is this intrinsic to languages like C++?
  - Why/Why not?

Memory Safety

- Vulnerabilities come from reading/writing/freeing
  - Out of bounds pointers
  - Dangling pointers
- Why doesn't Java face this issue?
- Is this intrinsic to languages like C++?
  - Why/Why not?
- Are these still a real issue?

Memory Safety

- Vulnerabilities come from reading/writing/freeing
  - Out of bounds pointers
  - Dangling pointers
- Why doesn't Java face this issue?
- Is this intrinsic to languages like C++?
  - Why/Why not?
- Are these still a real issue?
  - http://www.symantec.com/security_response/vulnerability.jsp?bid=70332

Another Case: SQL Injection

SQL – a query language for databases

- Queries like:
  “SELECT grade,id FROM students
  WHERE name=” + username;

ID	Name	Grade
0	Alice	92
1	Bob	87
2	Mallory	75

Another Case: SQL Injection

SQL – a query language for databases

- Queries like:
  “SELECT grade,id FROM students
  WHERE name=” + username;

ID	Name	Grade
0	Alice	92
1	Bob	87
2	Mallory	75

- Values for name, grade often come from user input.

Another Case: SQL Injection

SQL – a query language for databases

- Queries like:
  “SELECT grade,id FROM students
  WHERE name=” + username;

ID	Name	Grade
0	Alice	92
1	Bob	87
2	Mallory	75

- Values for name, grade often come from user input.

Why is this a problem?

Another Case: SQL Injection

username = “'bob'; DROP TABLE students”

- What happens?

SQL Injection

[http://xkcd.com/327/] [http://bobby-tables.com/]

- The user may include commands in their input!

SQL Injection

[http://xkcd.com/327/] [http://bobby-tables.com/]

- The user may include commands in their input!
- Need to *sanitize* the input before use

SQL Injection

[http://xkcd.com/327/] [http://bobby-tables.com/]

- The user may include commands in their input!
- Need to *sanitize* the input before use

How would you prevent this problem?

A Subtle Problem in General

- The problems may be much more subtle:

  User A can read files X,Y,Z and write to S,T
  User B can read files X,Y,S and write to Z,T

  How can we ensure that no information
  from A is ever written to Z?

A Subtle Problem in General

- The problems may be much more subtle:

  User A can read files X,Y,Z and write to S,T
  User B can read files X,Y,S and write to Z,T

  How can we ensure that no information from A is ever written to Z?

  Can you envision a scenario that creates this problem?

A Subtle Problem in General

- The problems may be much more subtle:

  User A can read files X,Y,Z and write to S,T
  User B can read files X,Y,S and write to Z,T

  How can we ensure that no information
  from A is ever written to Z?

- Care may be required to enforce *access control policies*

A Subtle Problem in General

- The problems may be much more subtle:

  User A can read files X,Y,Z and write to S,T
  User B can read files X,Y,S and write to Z,T

  How can we ensure that no information
  from A is ever written to Z?

- Care may be required to enforce *access control policies*
  - Discretionary access control – owner determines access

A Subtle Problem in General

- The problems may be much more subtle:

  User A can read files X,Y,Z and write to S,T
  User B can read files X,Y,S and write to Z,T

  How can we ensure that no information
  from A is ever written to Z?

- Care may be required to enforce *access control policies*
  - Discretionary access control – owner determines access
  - Mandatory access control – clearance determines access

Assuring Security

- Make risky operations someone else's job
  - e.g. Google Checkout, PayPal, Amazon, etc.

Assuring Security

- Make risky operations someone else's job
  - e.g. Google Checkout, PayPal, Amazon, etc.
- Define rigorous security policies
  - What are your CIAA security criteria?

Assuring Security

- Make risky operations someone else's job
  - e.g. Google Checkout, PayPal, Amazon, etc.
- Define rigorous security policies
  - What are your CIAA security criteria?
- Follow secure design & coding policies
  - And include them in your review criteria

Assuring Security

- Make risky operations someone else's job
  - e.g. Google Checkout, PayPal, Amazon, etc.
- Define rigorous security policies
  - What are your CIAA security criteria?
- Follow secure design & coding policies
  - And include them in your review criteria
  - Apple secure coding policies
  - CERT Top 10 Practices
  - Mitre Mitigation Strategies

Assuring Security

- Make risky operations someone else's job
  - e.g. Google Checkout, PayPal, Amazon, etc.
- Define rigorous security policies
  - What are your CIAA security criteria?
- Follow secure design & coding policies
  - And include them in your review criteria
- Formal certification

Common Proactive Approaches

How are these techniques applied?

Common Proactive Approaches

How are these techniques applied?

- Security must be part of design
  - Prepared Statements, Safe Arrays, etc.

Common Proactive Approaches

How are these techniques applied?

- Security must be part of design
  - Prepared Statements, Safe Arrays, etc.
- Regular security audits
  - Retrospective analysis & suggestions

Common Proactive Approaches

How are these techniques applied?

- Security must be part of design
  - Prepared Statements, Safe Arrays, etc.
- Regular security audits
  - Retrospective analysis & suggestions
- Penetration testing
  - Can someone skilled break it?

Security Overall

- Security is now a pressing concern for all software

Security Overall

- Security is now a pressing concern for all software
  - Old software was designed in an era of naiveté and is often vulnerable/broken

Security Overall

- Security is now a pressing concern for all software
  - Old software was designed in an era of naiveté and is often vulnerable/broken
  - New software is built to perform sensitive operations in a multiuser and networked environment.

Security Overall

- Security is now a pressing concern for all software
  - Old software was designed in an era of naiveté and is often vulnerable/broken
  - New software is built to perform sensitive operations in a multiuser and networked environment.

Not planning for security concerns from the beginning is a broken approach to development