

SECURE CODING GUIDE



What is Secure Coding Guide?

Guides and Sample Code

Secure Coding Guide

Table of Contents

Introduction

Types of Security Vulnerabilities

- Buffer Overflows
- Unvalidated Input
- Race Conditions
- Interprocess Communication
- Insecure File Operations
- Access Control Problems
- Secure Storage and Encryption
- Social Engineering

Avoiding Buffer Overflows and Underflows

- Stack Overflows
- Heap Overflows
- String Handling
- Calculating Buffer Sizes
- Avoiding Integer Overflows and Underflows
- Detecting Buffer Overflows
- Avoiding Buffer Underflows

Security Features that Can Help

Validating Input and Interprocess Communication

- Risks of Unvalidated Input
- Fuzzing
- Interprocess Communication and Networking

Race Conditions and Secure File Operations

Elevating Privileges Safely

- Circumstances Requiring Elevated Privileges
- The Hostile Environment and the Principle of Least Privilege
- Avoiding Elevated Privileges
- Running with Elevated Privileges
- Calls to Change Privilege Level

Next

Introduction to Secure Coding Guide

Secure coding is the practice of writing programs that are resistant to attack by malicious or mischievous people or programs. Secure coding helps protect a user's data from theft or corruption. In addition, an insecure program can provide access for an attacker to take control of a server or a user's computer, resulting in anything from a denial of service to a single user to the compromise of secrets, loss of service, or damage to the systems of thousands of users.

Secure coding is important for all software; if you write any code that runs on Macintosh computers or on iOS devices, from scripts for your own use to commercial software applications, you should be familiar with the information in this document.

At a Glance

Every program is a potential target. Attackers will try to find security vulnerabilities in your applications or servers. They will then try to use these vulnerabilities to steal secrets, corrupt programs and data, and gain control of computer systems and networks. Your customers' property and your reputation are at stake.

Security is not something that can be added to software as an afterthought; just as a shed made out of cardboard cannot be made secure by adding a padlock to the door, an insecure tool or application may require extensive redesign to secure it. You must identify the nature of the threats to your software and incorporate secure coding practices throughout the planning and development of your product. This chapter explains the types of threats that your software may face. Other chapters in this document describe specific types of vulnerabilities and give guidance on code hardening techniques to fix them.

Hackers, Crackers, and Attackers

Contrary to the usage by most news media, within the computer industry the term *hacker* refers to an expert programmer—one who enjoys learning about the intricacies of code or an operating system. In general, hackers are not malicious. When most hackers find security vulnerabilities in code, they inform the company or organization that's responsible for the code so that they can fix the problem. Some hackers—especially if they feel their warnings are being ignored—publish the vulnerabilities or even devise and publish *exploits* (code that takes advantage of the vulnerability).

The malicious individuals who break into programs and systems in order to do damage or to steal something are referred to as *crackers*, *attackers*, or *black hats*. Most attackers are not highly skilled, but take advantage of published exploit code and known techniques to do their damage.

Attackers may be motivated by a desire to steal money, identities, and other secrets for personal gain; corporate secrets for their employer's or their own use; or state secrets for use by hostile governments or terrorist organizations. Some crackers break into applications or operating systems just to show that they can do it; nevertheless, they can cause considerable damage. Because attacks can be automated and replicated, any weakness, no matter how slight, can be exploited.

The large number of insiders who are attacking systems is of importance to security design because, whereas malicious hackers are most likely to rely on remote access to computers to do their dirty work, insiders might have physical access to the computer being attacked. Your software must be resistant to both attacks over a network and attacks by people sitting at the computer keyboard—you cannot rely on firewalls and server passwords to protect you.

No Platform Is Immune

So far, OS X has not fallen prey to any major, widespread attack like the Meltdown virus. There are several reasons for this. One is that OS X is based on open source software such as

Feedback



Hacker vs Attacker



Types of Vulnerabilities

Buffer overflows / underflows

Unvalidated input

Social Engineering

etc.



Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```




Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```




Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```



Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf); ←
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```



Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```



Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```



Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");
    return 0;
}
```



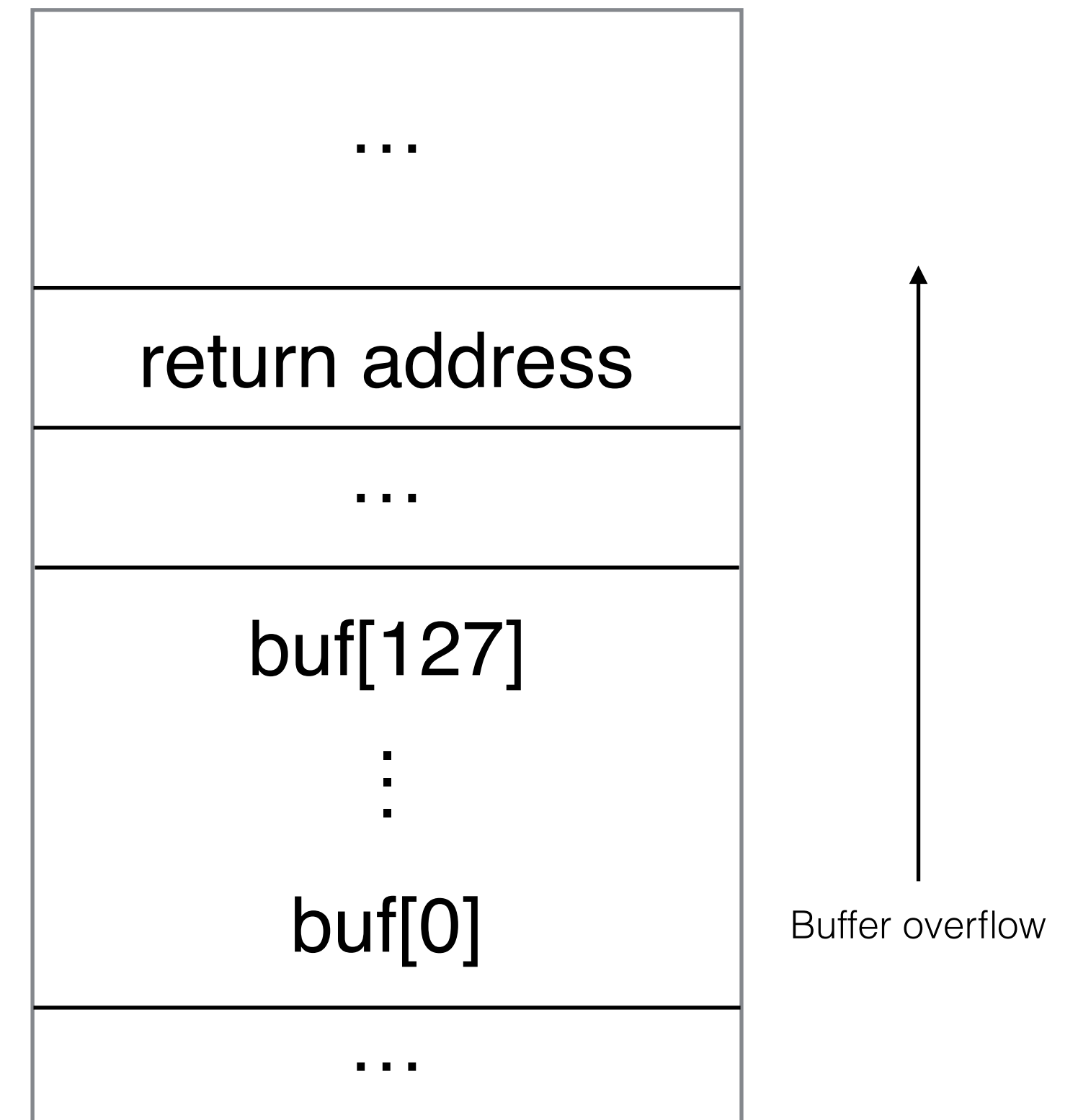
Buffer overflows / underflows

```
#include <stdio.h>
#include <string.h>

void doit(void) {
    char buf[128];
    gets(buf);
    printf("%s\n", buf);
}

int main(void) {
    printf("So... The End...\n");
    doit();
    printf("or... maybe not?\n");

    return 0;
}
```



Other types of overflows

Heap overflows

Integer overflow

Pointer arithmetic



How to avoid these overflow?

Use functions that check the bounds (i.e. fgets instead of gets)



How to avoid these overflow?

Use functions that check the bounds (i.e. fgets instead of gets)

Some things are done by OS



How to avoid these overflow?

Use functions that check the bounds (i.e. fgets instead of gets)

Some things are done by OS

DON'T USE C



Unvalidated input

Format string vulnerabilities

URL commands

Code insertion

Social engineering



Format string vulnerability

Let's Code!



Social engineering attacks

Tricking people into normal security procedures.

An example: Phishing



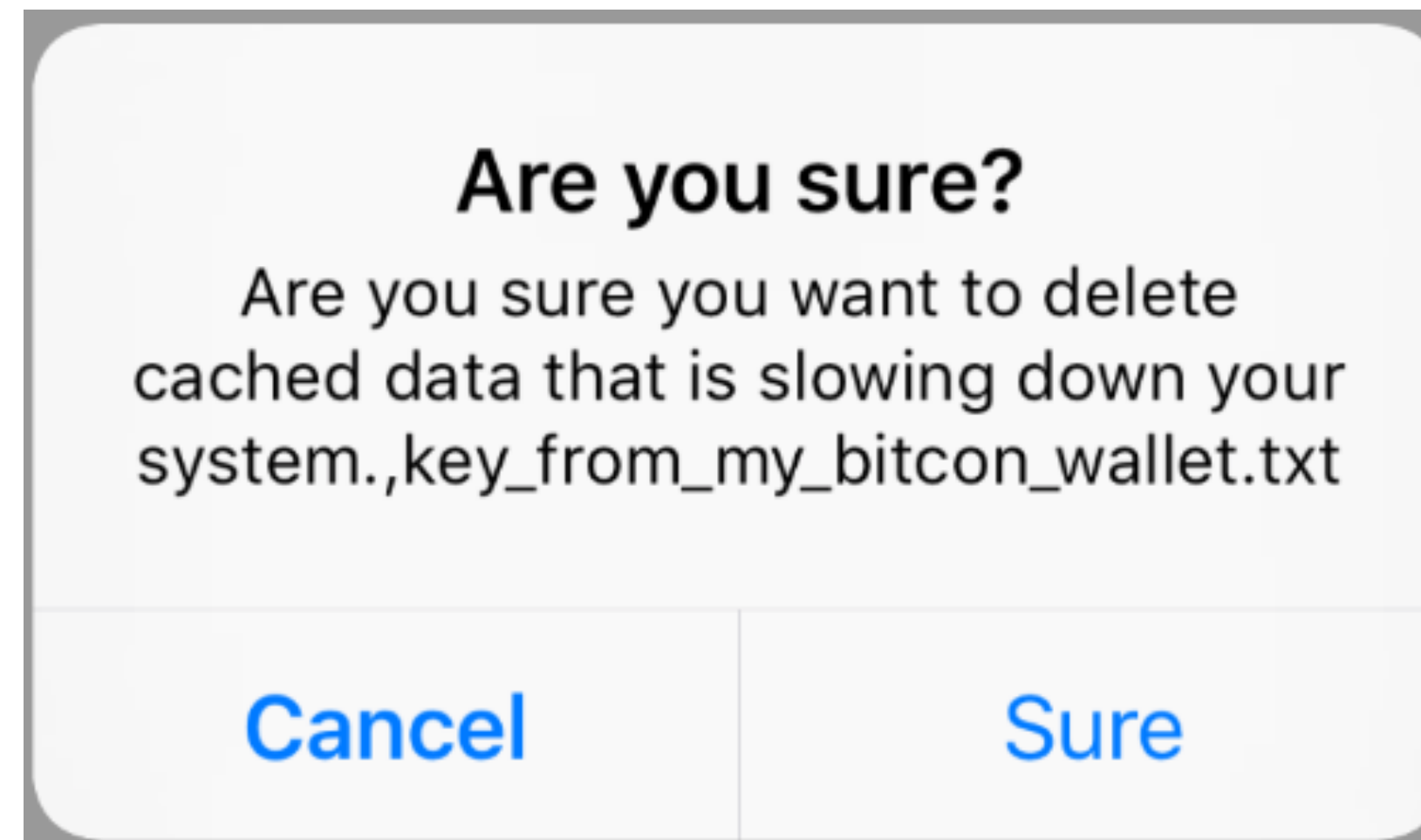
Social engineering attacks

myapp://cmd/delete?file=cached data that is slowing down your
system.,key_from_my_bitcon_wallet.txt



Social engineering attacks

myapp://cmd/delete?file= cached data that is slowing down your
system.,key_from_my_bitcon_wallet.txt



Other types of vulnerabilities

Insecure file operations



Other types of vulnerabilities

Insecure file operations

Race conditions



Other types of vulnerabilities

Insecure file operations

Race conditions



And finally...

