. 28 (2 4)
1:
, (web.py, Sinatra,).
, .

http: localhost: 000 test file=foo signature=46b4ec 8611 1 4dacd4 d664e d6 fdc88efb 1

HMAC (HMAC
) foo signature.
, , .

insecure_compare,
(. . False).
insecure_compare 0 (,
time.sleep(0.0) Python).

insecure_compare .

, ,
(!).

.

CTF . , 0m ctf (https: a xchapman.github.io security 201 0 26
h1 02 ctf 201 .html) HackerOne .

```
package set_four
import (
        "crypto hmac"
        "crypto sha1"
        "encoding hex"
        "net http"
        "strings"
        "time"
)

const COMPARE_DELAY = 20

const LISTEN_ADDR = ":8    1"

type timedResponse struct {
        r *http.Response
        id string
        elapsed int64
}

func StartServer() {
        http.HandleFunc(" test", ValidationServer)
        http.HandleFunc(" test  2", FasterValidationServer)
        go http.ListenAndServe(LISTEN_ADDR, nil)
}

func ValidationServer(w http.ResponseWriter, req *http.Request) {
        status :=   00
```

```
	message := req.FormValue("file")
	sig := req.FormValue("signature")

	if InsecureValidateHMAC(message, sig) {
		status = 200
	}

	http.Error(w, http.StatusText(status), status)
}

func InsecureValidateHMAC(message, signature string) bool {
	goodSig := HMACSHA1(cryptopals.RANDOM_KEY,[]byte(message))
	return InsecureCompare([]byte(signature), []byte(goodSig), COMPARE_DELAY)
}

func HMACSHA1(key, message []byte) string {
	mac := hmac.New(sha1.New, key)
	mac.Write(message)
	return hex.EncodeToString(mac.Sum(nil))
}

func InsecureCompare(a, b []byte, delay uint8) bool {
	if len(a) != len(b) {
	return false
}

	for i := 0; i < len(a); i++ {
		time.Sleep(time.Duration(delay) * time.Millisecond)
		if a[i] != b[i] {
			return false
		}
	}

return true
}

func findSlowestRequest(requests map[string]int64) string {
	slowest := int64(0)
	slowestKey := ""

	for key, value := range requests {
		if value > slowest {
			slowest = value
			slowestKey = key
		}
	}

	return slowestKey

func ExploitTimingAttack(url string, length int) string {
	var known string
	results := make(chan timedResponse)

	chars := "0123456789abcdef"

	for i := 0; i < length; i++ {
		requests := make(map[string]int64)

		filler := strings.Repeat("_", length-(i+1))
		for j := 0; j < len(chars); j++ {
			signature := strings.Join([]string{known, string(chars[j]), filler}, "")
```

```
                urlWithSig := strings.Join([]string{url, signature}, "")
                go TimeHTTPRequest(urlWithSig, string(chars[j]), results)
            }

            for j := 0; j < len(chars); j++ {
                res := <-results
                if res.r.StatusCode == http.StatusOK {
                    return strings.Join([]string{known, res.id}, "")
                }
                requests[res.id] = res.elapsed
            }

            bestGuess := findSlowestRequest(requests)
            known = strings.Join([]string{known, bestGuess}, "")
        }
    return ""
}

func TimeHTTPRequest(url, id string, results chan timedResponse) {
    start := time.Now()
    resp, err := http.Get(url)
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()
    elapsed := time.Since(start).Nanoseconds()
    results <- timedResponse{resp, id, elapsed}
}
```

Задание 2: Найдите реализацию SHA-1 на вашем языке программирования (например, можно использовать https://github.com/ajalt/python-sha1 для Python). Примечание: это задание является подготовкой к атаке Hash Length Extension, поэтому нужна именно чистая реализация SHA-1, а не библиотечная.

Напишите функцию, которая будет реализовывать MAC вида SHA1(key || message), где || - конкатенация.
Убедитесь, что вы не можете подделать сообщение, не изменив при этом MAC.
http://cryptopals.com/sets/4/challenges/28

```
package set_four

import (
    "encoding/hex"
    "fmt"
    "math/rand"
)

var SecretPrefix = []byte("\x00\x01Super Secret Prefix\x02\x03")

func ValidateSecretPrefixSHA1(message []byte, mac string) bool {
    sha := sha1.New()
    sha.Write(SecretPrefix)
    sha.Write(message)
    h := sha.Sum(nil)
    return hex.EncodeToString(h) == mac
}
```

```
func TamperMessage(message []byte, mac string) error {
	for i := 0; i < len(message); i++ {
		for j := 0; j < 8; j++ {
			message[i] ^= (1 << uint(j))
			if ValidateSecretPrefixSHA1(message, mac) {
				return fmt.Errorf("Tampered message matches MAC. Message: %v",
message)

			}
			message[i] ^= (1 << uint(j))
		}
	}
	return nil
}

func RandomBytesDontMatch(mac string, iterations int) error {
	maxBytes := 1024
	for i := 0; i < iterations; i++ {
		randBytes, _ := cryptopals.GenerateRandomBytes(rand.Intn(maxBytes))
		if ValidateSecretPrefixSHA1(randBytes, mac) {
			return fmt.Errorf("Random message matches MAC. Message: %v", randBytes)
		}
	}
	return nil
}
```