Задание выполнил студент группы 2345 Романенко Кирилл

Задание 1

Напишите небольшое веб-приложение, используя готовый фреймворк (web.py, Sinatra, …). В этом приложении будет один эндпойнт, который принимает на вход имя файла и подпись.

`http://localhost:9000/test?file=foo&signature=46b4ec586117154dacd49d664e5d63fdc88efb51`

Сервер при получении подобного запроса будет брать HMAC (функцию HMAC реализуйте самостоятельно) от файла `foo` и сверять полученное значение со значением `signature`. Если значения совпадут, то сервер покажет файл, иначе сервер вернет ошибку.

Теперь напишите функцию `insecure_compare`, которая побайтово сравнивает строки и реализует “ранний выход” (т.е. возвращает False на первой паре не совпавших байт). В цикле `insecure_compare` добавьте искусственную задержку в 50 мс (например, функцией time.sleep(0.05) в Python).

Для проверки подписи используйте `insecure_compare` вместо обычной функции сравнения.

Используя факт задержек и раннего выхода, напишите программу, которая будет подбирать валидную подпись для любого файла (без знания ключа!).

В реальном мире подобные уязвимости тяжело эксплуатировать из-за задержек в сети. Однако это хороший пример атаки по сторонним каналам и похожие уязвимости время от времени встречаются в CTF-ах. Например, в 50m-ctf (https://ajxchapman.github.io/security/2019/03/26/h1-702-ctf-2019.html) от HackerOne была очень похожая уязвимость.

http://cryptopals.com/sets/4/challenges/31

```
package set_four

import (
	"crypto/hmac"
	"crypto/sha1"
	"encoding/hex"
	"net/http"
	"strings"
	"time"
)

const COMPARE_DELAY = 20

const LISTEN_ADDR = ":8771"
```

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

func StartServer() {
     http.HandleFunc("/test", ValidationServer)
     http.HandleFunc("/test32", FasterValidationServer)
     go http.ListenAndServe(LISTEN_ADDR, nil)
}

func ValidationServer(w http.ResponseWriter, req *http.Request) {
     status := 500

     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
}
```