Universidade Federal de Minas Gerais

Cibersegurança - Projeto de programação #3

Junho 2023

1 Introdução

Neste projeto, investigaremos as cifras de Vigenère e, em seguida, analisaremos as vulnerabilidades em funções hash criptográficas amplamente usadas, incluindo as vulnerabilidades de colisão.

2 Cifras de Vigenère

Para esta parte 1, resolva manualmente ou escreva um programa (talvez em Python). Você pode ler sobre como a cifra de Vigenère funciona na Wikipédia. As cifras de Vigenère podem ser decifradas geralmente usando o Exame Kasiski, que é discutido na página Wikipédia.

Assuma que criptografar a letra A com a chave não resulta em mudança; criptografar a letra B resulta em um incremento de uma casa no alfabeto; criptografar C resulta em um incremento de duas casas, e assim por diante. Assuma também que o texto simples original contém apenas letras maiúsculas (AZ) e sem espaços ou pontuação.

Gere um arquivo de texto pleno de forma aleatórie e encontre a cifra considerando a chave a chave de 21 letras a seguir. Lembre que o texto plano deve ter pelo menos a quantidade de caracteres que a chave abaixo, ou seja, o texto plano precisa ter pelo menos 21 caracteres.
Chave: TESSOFTHEDURBERVILLES

O que entregar? Um arquivo de texto chamado *vigenere.cipher* contendo o texto cifrado.

Nota histórica: Em novembro de 2019, foi descoberto que a empresa de segurança Fortinet estava usando "criptografia XOR com uma chave estática" em alguns produtos, que é semelhante a uma cifra Vigenère e tem (falta de) propriedades de segurança semelhantes. `https://seclists.org/bugtraq/2019/Nov/38`

3 Parte 2: extensão de comprimento

Na maioria dos aplicativos, você deve usar MACs como HMAC-SHA256 em vez de funções hash criptográficas simples (por exemplo, MD5, SHA-1 ou SHA-256)

porque os hashes, também conhecidos como resumos, não atendem às nossas expectativas de segurança intuitivas. O que realmente queremos é algo que se comporte como uma função pseudo-aleatória, que os HMACs parecem aproximar e as funções de hash não.

Uma diferença entre funções hash e funções pseudo-aleatórias é que muitos hashes estão sujeitos à extensão de comprimento. Muitas funções hash comuns seguem uma abordagem chamada de construção Merkle-Damgard. Cada um é construído em torno de uma função de compressão f e mantém um estado interno s, que é inicializado com uma constante fixa. As mensagens são processadas em blocos de tamanho fixo aplicando a função de compressão ao estado atual e ao bloco atual para calcular um estado interno atualizado, ou seja, $s_{i+1} = f(s_i, b_i)$. O resultado da aplicação final da função de compressão torna-se a saída da função hash.

Uma consequência desse projeto é que, se conhecermos o hash de uma mensagem de n blocos, podemos encontrar o hash de mensagens mais longas aplicando a função de compactação para cada bloco $b_{n+1}, b_{n+2}, ...$ que queremos adicionar. Esse processo é chamado de extensão de comprimento e pode ser usado para atacar muitas aplicações de funções de hash.

3.1 Primeiro Experimento: Familiarização com o Python MD5

Para experimentar essa ideia, usaremos uma implementação Python da função de hash MD5, embora SHA-1 e SHA-256 sejam vulneráveis a extensão de comprimento da mesma maneira. Você pode baixar o módulo `pymd5` do Moodle e aprender a usá-lo executando `pydoc pymd5`. Para acompanhar esses exemplos, execute o Python no modo interativo e execute o comando

```
from pymd5 import md5, padding
```

Considere a string `''Use HMAC, não hashes''`. Podemos calcular seu hash MD5 executando:

```
from pymd5 import md5, padding
m = ''Usar HMAC, n o hashes''
h = md5()
h.atualizar(m)
imprimir(h.hexdigest())
```

ou, de forma mais compacta,

```
imprimir(md5(m).hexdigest())
```

A saída deve ser `3ecc68efa1871751ea9b0b1a5b25004d`.

O MD5 processa mensagens em blocos de 512 bits, portanto, internamente, a função hash preenche m para um múltiplo desse comprimento. O preenchimento consiste no bit 1, seguido por quantos bits 0, se forem necessários, seguido por

uma contagem de 64 bits do número de bits na mensagem não preenchida. Se o 1 e a contagem não couberem no bloco atual, um bloco extra será adicionado. Você pode usar a função $padding(count)$ no módulo $pymd5$ para calcular o preenchimento que será adicionado a uma mensagem de bit de contagem.

Mesmo que não soubéssemos m, poderíamos calcular o hash de mensagens mais longas da forma geral $m + padding(len(m) * 8) + suffix$ definindo o estado interno inicial da função MD5 como MD5(m), em vez do valor de inicialização padrão, e definindo o contador de tamanho da mensagem da função como o tamanho de m mais o preenchimento (um múltiplo do tamanho do bloco). Para encontrar o comprimento da mensagem preenchida, adivinhe o comprimento de m e execute $bits = (length_of_m + len(padding(length_of_m * 8))) * 8$.

O pymd5módulo permite especificar esses parâmetros como argumentos adicionais para o md5objeto:

```
h = md5(estado = codecs.
decode(''3ecc68efa1871751ea9b0b1a5b25004d'', ''hex''),
contagem = 512)
```

Agora você pode usar extensão de comprimento para encontrar o hash de uma string mais longa que acrescenta o sufixo "Bom conselho". Para executar a função de compactação sobre x e gerar o hash resultante, simplesmente execute:

```
x = ''Bom conselho''
h.atualizar(x)
imprimir(h.hexdigest())
```

Verifique se o hash resultante é igual ao hash MD5 de $m + padding(len(m) * 8) + x$. Observe que, devido à propriedade length-extension do MD5, não precisamos saber o valor de m para calcular o hash da string mais longa - tudo o que precisamos saber é o comprimento de m e seu hash MD5.

Esta parte da tarefa destina-se a apresentar a extensão de comprimento e familiarizá-lo com o módulo Python MD5 que usaremos; você não precisará enviar nada para isso.

4 Colisões MD5

O MD5 já foi a função hash criptográfica mais usada, mas hoje é considerada perigosamente insegura. Isso ocorre porque os criptoanalistas descobriram algoritmos eficientes para encontrar colisões — pares de mensagens com o mesmo valor de hash MD5. As primeiras colisões conhecidas foram anunciadas em 17 de agosto de 2004 por Xiaoyun Wang, Dengguo Feng, Xuejia Lai e Hongbo Yu. Aqui está um par de mensagens conflitantes que eles publicaram:

Mensagem 1:

```
d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
```

e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70

Mensagem 2:

d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c 5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70

Copie as strings hexadecimais acima em dois arquivos file1.hex e file2.hex. Converta cada grupo de strings hexadecimais em um arquivo binário. (No Linux, execute xxd -r -p file.hex > file.)

1. Quais são os hashes MD5 dos dois arquivos binários? Verifique se são iguais. (openssl dgst -md5 file1 file2)
2. Quais são seus hashes SHA-256? Verifique se são diferentes. (openssl dgst -sha256 file1 file2)

Você não precisa enviar nada para as perguntas acima.

4.1 Gerando colisões

Em 2004, o método de Wang levou mais de 5 horas para encontrar uma colisão em um PC de mesa. Desde então, os pesquisadores introduziram algoritmos de detecção de colisão muito mais eficientes. Você pode calcular suas próprias colisões MD5 usando uma ferramenta escrita por Marc Stevens que usa uma técnica mais avançada.

Você pode baixar a ferramenta fastcoll aqui:
`http://www.win.tue.nl/hashclash/fastcoll_v1.0.0.5.exe.zip`
(executável do Windows)

ou

`http://www.win.tue.nl/hashclash/fastcoll_v1.0.0.5-1_source.zip`
(código-fonte)

Se você estiver compilando fastcolla partir do código-fonte, poderá compilar usando o arquivo makefile que você encontra no Moodle. Você também precisará ter instalado as bibliotecas do Boost. No Ubuntu, você pode instalar usando apt-get install libboost-all-dev. No OS X, você pode instalar o Boost por meio do gerenciador de pacotes Homebrew usando brew install boost.

Gere sua própria colisão com esta ferramenta e responda:

1. Quanto tempo levou? (time ./fastcoll -o file1 file2)
2. Quais são seus arquivos? Para obter um despejo hexadecimal, execute xxd -p file.
3. Quais são seus hashes MD5? Verifique se são iguais.
4. Quais são seus hashes SHA-256? Verifique se são diferentes.
5. O que enviar? Escreva suas respostas em writeup.txt e envie pelo Moodle.