2021136124 조병하

10장 2번

2. TLS에서 Diffie-Hellman을 사용할 경우 서버의 값만 인증하고, 클라이언트의 값은 인증하지 않는다. 하지만 서명 외에 추가로 확립된 세션키를 이용하여 계산되는 MAC 값을 서로 교환한다. 아래와 같이 한 쪽만 인증하는 형태로 프로토콜을 수행할 경우 중간자 공격 관련 어떤 문제가 있는지 논하시오.

$$\begin{array}{ll} \text{Msg 1.} & A \rightarrow B: \quad g^a \\ \text{Msg 2.} & B \rightarrow A: \quad g^b, \text{Sig}.B(g^a||g^b||A) \end{array}$$

힌트. 중간자가 g^a를 g^c로 바꾸면 어떻게 진행되는지, 또 g^b를 g^c로 바꾸면 어떻게 진행되는지 생각해 보세요.

A.
중간자가 g^a를 g^c로 바꾸면:

클라이언트는 중간자로부터 수신한 g^c를 사용하여 세션 키를 계산합니다.
중간자는 서버와 별개의 세션 키를 생성하여 클라이언트와 통신합니다.
이로써 중간자는 클라이언트와 서버 간의 통신을 도청하거나 조작할 수 있습니다.

중간자가 g^b를 g^c로 바꾸면:

서버는 중간자로부터 수신한 g^c를 사용하여 세션 키를 계산합니다.
중간자는 클라이언트와 별개의 세션 키를 생성하여 서버와 통신합니다.
이로써 중간자는 클라이언트와 서버 간의 통신을 도청하거나 조작할 수 있습니다.

따라서, 클라이언트와 서버 간의 Diffie-Hellman 키 교환 단계에서 중간자가 클라이언트 또는 서버로부터 수신한 값을 변조할 수 있다면, 중간자는 클라이언트와 서버 사이의 통신을 제어할 수 있게 됩니다. 이로 인해 중간자는 암호화된 데이터를 엿들을 수 있거나, 악의적인 데이터를 주고받는 등의 공격을 수행할 수 있습니다. 따라서 클라이언트와 서버 양쪽 모두의 값을 인증하는 것이 중요합니다.

11장 4번

4. 패스워드 해싱할 때 보통 소금을 추가하며, 이 소금값을 해시값과 함께 보관한다. k비트 소금값을 보관하지 않고 서버는 가능한 모든 k비트를 이용하여 해시값을 확인하도록 할 수 있다. 물론 계산 비용 때문에 긴 길이를 사용할 수 없지만 성능에 문제가 안 되는 길이(예: 16bit)를 사용한다면 사전 공격에 도움이 되는지 논하고, 이 방법을 사용하면 발생할 수 있는 다른 문제점은 없는지 논하시오.

A.
소금(salt)을 사용하여 패스워드 해싱을 할 때, 일반적으로 소금값을 해시값과 함께 보관하는 것이 보안 강화를 위한 일반적인 방법입니다. 그러나 소금값을 보관하지 않고 서버가 가능한 모든 비트를 사용하여 해시값을 확인하는 방법도 사용할 수 있습니다.

짧은 길이(예: 16비트)의 소금을 사용하는 경우, 사전 공격에 도움이 되는 문제가 발생할 수 있습니다. 소금을 사용하지 않는 경우, 동일한 패스워드에 대해 항상 동일한 해시값이 생성되므로, 공격자는 사전을 구축하여 해시값을 미리 계산할 수 있습니다. 이 사전 공격은 빠르게 수행될 수 있으며, 일치하는 해시값을 찾을 수 있다면 패스워드를 알아낼 수 있게 됩니다.

또한, 짧은 길이의 소금을 사용하는 경우 가능한 소금값의 조합이 제한되어 패스워드 공격에 대한 공간이 축소될 수 있습니다. 즉, 소금값이 매우 제한적이라면 공격자가 모든 가능한 소금값에 대해 해시를 계산해볼 수 있으므로 패스워드를 추측하는 데 도움이 됩니다.

또한, 짧은 길이의 소금을 사용하는 경우 일부 패스워드들은 동일한 소금값을 공유할 가능성이 있습니다. 이는 패스워드의 다양성과 안전성을 감소시키는 요소가 될 수 있습니다.

이 방법을 사용할 때 발생할 수 있는 다른 문제점은 다음과 같습니다:

소금을 사용하지 않는 경우, 동일한 패스워드에 대해 항상 동일한 해시값이 생성되므로 공격자가 여전히 사전 공격을 수행할 수 있습니다.
소금값이 제한적이고 예측 가능하다면, 공격자가 소금값에 대해 미리 계산된 해시값을 보유하여 공격에 사용할 수 있습니다.
긴 소금값을 사용하는 경우, 해시 연산의 성능에 영향을 줄 수 있습니다. 따라서 소금값의 길이는 적절하게 선택되어야 합니다.
요약하면, 짧은 길이의 소금을 사용하는 경우 패스워드 해싱에 대한 보안성이 감소하고 사전 공격에 취약해질 수 있습니다. 적절한 길이의 소금을 사용하는 것이 안전하며, 보안 강화를 위한 일반적인 권장 사항입니다.

11장 7번

7. 10장에서 살펴본 무선랜 보안 프로토콜 중 WPA3은 패스워드 기반 프로토콜을 사용한다. WPA3는 기본적으로 패스워드 기반 대칭키를 무선 장치와 무선 AP 간에 사용하므로 패스워드 기반 프로토콜을 사용하는 것이 필요하다. 그런데 패스워드 기반 프로토콜을 사용한다고 모든 보안 문제가 해결되는 것은 아니다. 무선랜 환경에서 패스워드 기반 프로토콜을 사용하면 어떤 공격을 방어할 수 있고, 어떤 공격은 방어할 수 없는지 설명하시오.

A.
무선랜 환경에서 패스워드 기반 프로토콜을 사용하는 것은 WPA3와 같은 보안 프로토콜에서 사용되는 일반적인 접근 방식입니다. 하지만 패스워드 기반 프로토콜을 사용하더라도 모든 보안 문제가 해결되는 것은 아닙니다. 패스워드 기반 프로토콜은 일부 공격을 방어할 수 있지만, 일부 다른 공격은 여전히 존재할 수 있습니다.

패스워드 기반 프로토콜을 사용하는 경우 어떤 공격을 방어할 수 있는지와 어떤 공격을 방어할 수 없는지를 살펴보겠습니다.

공격을 방어할 수 있는 경우:

도청(평문 공격): 패스워드 기반 프로토콜은 통신을 암호화하여 도청을 방지합니다. 패스워드를 사용하여 장치 간에 대칭키를 생성하고, 해당 키를 사용하여 데이터를 암호화하여 전송하므로 도청 공격을 방어할 수 있습니다.

무차별 대입(브루트 포스 공격): 패스워드 기반 프로토콜은 일반적으로 패스워드를 입력 오류

제한 시도 횟수로 보호하거나, 비밀번호에 대한 잠금 기능을 제공하여 무차별 대입 공격을 방어할 수 있습니다.

공격을 방어할 수 없는 경우:

사전 공격: 패스워드 기반 프로토콜은 패스워드를 기반으로 대칭키를 생성하므로, 공격자가 사전을 사용하여 미리 계산된 해시값과 비교하여 패스워드를 추측하는 사전 공격을 방어할 수 없습니다. 따라서 강력한 패스워드를 사용해야 합니다.

중간자 공격: 패스워드 기반 프로토콜은 통신의 인증과 암호화에만 초점을 맞추므로 중간자 공격을 완전히 방어할 수는 없습니다. 공격자가 무선 장치와 무선 AP 사이에 위치하여 통신을 가로채고 조작하는 중간자 공격은 패스워드 기반 프로토콜로부터 완전히 방어할 수 없는 공격입니다.

요약하면, 패스워드 기반 프로토콜은 도청과 무차별 대입 공격과 같은 일부 공격을 방어할 수 있지만, 사전 공격과 중간자 공격과 같은 다른 공격은 완전히 방어할 수 없습니다. 패스워드의 강도와 사용자의 주의가 중요하며, 추가적인 보안 메커니즘을 사용하여 무선 랜 환경의 보안을 강화해야 합니다.

11장 10번

10. M_1부터 M_n까지 n개 메시지가 있을 때, 이들을 해시 트리를 표현하여 루트에 서명하는 것, $H(M_1||M_2||\dots||M_n)$에 서명하는 것, $H(H(M_1)||H(M_2)||\dots||H(M_n))$에 서명하는 것과 비교하시오.

A.

해시 트리를 사용하여 메시지를 루트에 서명하는 방식, H(M1||M2||...||Mn)에 서명하는 방식, 그리고 H(H(M1)||H(M2)||...||H(Mn))에 서명하는 방식을 비교해보겠습니다.

해시 트리를 사용하여 메시지를 루트에 서명하는 방식:

해시 트리를 구성하기 위해 메시지들을 재귀적으로 분할하여 해시 함수에 적용합니다.
재귀적인 해시 과정을 거쳐 최종적으로 루트 해시값을 계산합니다.
루트 해시값에 서명을 적용하여 전체 해시 트리에 대한 무결성을 보장합니다.
이 방식은 메시지들 간의 의존성을 포함하여 전체 트리 구조를 보호할 수 있습니다.
H(M1||M2||...||Mn)에 서명하는 방식:

모든 메시지를 연결하여 단일 메시지로 만든 후, 해당 메시지에 해시 함수를 적용합니다.
해시된 단일 메시지에 서명을 적용하여 무결성을 보장합니다.
이 방식은 메시지들 간의 순서와 의존성을 포함하여 단일 메시지에 대한 무결성을 보장합니다.
H(H(M1)||H(M2)||...||H(Mn))에 서명하는 방식:

각 메시지를 개별적으로 해시 함수에 적용하여 해시값을 계산합니다.
개별 해시값들을 다시 연결하여 단일 메시지로 만든 후, 해당 메시지에 해시 함수를 적용합니다.
최종적으로 해시된 단일 메시지에 서명을 적용하여 무결성을 보장합니다.
이 방식은 각 메시지의 개별 무결성과 전체 메시지 집합의 무결성을 모두 보장합니다.
이 세 가지 방식은 각각 다른 무결성 보장 수준을 가지고 있습니다. 해시 트리를 사용하는 방식은 전체 트리 구조에 대한 무결성을 보장할 수 있습니다. H(M1||M2||...||Mn)에 서명하는 방식은 단일 메시지에 대한 무결성을 보장하며, H(H(M1)||H(M2)||...||H(Mn))에 서명하는 방식은 각 메시지의 개별 무결성과 전체 메시지 집합의 무결성을 모두 보장할 수 있습니다.

따라서 선택해야 하는 방식은 무결성 보장의 요구사항과 애플리케이션의 특정 상황에 따라 달라질 수 있습니다.