Vorlesung Sicherheit

Dennis Hofheinz

ITI, KIT

03.07.2014

Überblick

Überblick

Erinnerung

- **Problem:** Authentifikation eines Benutzers
- **Üblich:** Passwortbasierte Authentifikation

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

  - Naheliegender Angriff: Wörterbuchangriff
  - Wörterbuch sehr groß, deshalb komprimiert (Rainbow Tables)
  - **Wichtig:** erfordert aufwändige Vorberechnung, rentiert sich erst, wenn viele Passwörter angegriffen werden
  - **Deshalb:** Passwörter „gesalzen“ abspeichern
  - Alternative: GPU-basierte parallele Brute-Force-Angriffe

Überblick

Motivation

- Aktuelles Forschungsthema: wie kann man sich durch eigene *Position* authentifizieren?
- **Szenario:** einzige auszeichnende Eigenschaft geographische Position von Prover P (bzw. Nutzer U aufgefasst als Prover)
  - Insbesondere Annahme: Angreifer haben gewissen Mindestabstand Δ von ehrlichem P
  - Weitere Annahme: Daten können gerichtet oder ungerichtet, aber nur mit $\leq$ Lichtgeschwindigkeit übertragen werden
  - Mehrere V_i möglich (und notwendig!), deren geschickte Positionierung entscheidend
  - Aber: auch jedes V_i sollte Mindestabstand Δ von P haben
- **Motivation (Beispiel):** Botschaften in fremden Ländern

Sicherheitsmodell

- **Frage:** was wollen wir eigentlich?
- Betrachte Angreifer $\mathcal{A}_1, \ldots, \mathcal{A}_m$
  - Positionierung beliebig, aber in $\geq \Delta$-Abstand von P
  - $\mathcal{A}_i$-Positionen dürfen von V_i-Positionen abhängen
- Ziel der $\mathcal{A}_i$: Prover P „impersonieren“
  - Also: $\mathcal{A}_i$ schaffen es, bei „deaktiviertem“ P, die V_i zum Akzeptieren zu bringen

Erste Überlegungen

- Wir brauchen mehr als einen Verifier V:
  - Angenommen, es gibt nur einen Verifier V
  - Dann könnte sich Angreifer $\mathcal{A}$ direkt zwischen P und V setzen, und V durch Verzögerung „vorgaukeln“, P zu sein
- Hoffnung: mehrere V_i schicken *koordiniert* Nachrichten an P
  - Allerdings: auch die V_i können nur mit Lichtgeschwindigkeit miteinander kommunizieren
- **Frage:** wie könnte das funktionieren?

Ein negatives Resultat

- Tatsächlich Situation auch mit mehreren V_i unangenehm

Theorem (Unmöglichkeit pos.-basierter Authentifikation, informell)

Es existiert kein Protokoll (mit einem oder mehreren Verifiern), das in obigem Modell sicher ist.

Beweisidee.

Setze je einen Angreifer $\mathcal{A}_i$ zwischen V_i und P; dieses $\mathcal{A}_i$ leitet alle eingegangenen Nachrichten von V_i an alle anderen $\mathcal{A}_i$ weiter und reagiert wie P. In jedem $\mathcal{A}_i$ läuft dann eine Δ-verzögerte Simulation von P ab. □

- Details: `https://eprint.iacr.org/2009/364.pdf`

Eine Idee

- Situation hoffnungslos?
- Nein, wir müssen nur mehr Annahmen machen!
- Schwierigkeit hierbei: Annahmen sollten realistisch sein
- **Frage:** wie könnten Angreifer eingeschränkt sein?
- Berechnungsbeschränkung (z.B. PPT) bei Angreifern bringt nichts, weil obiger Angriff effizient ist
- **Generell populär:** Angreifer $\mathcal{A}_i$ *speicherbeschränkt*

Speicherbeschränkte Angreifer

- **Szenario:** *Erzeugen* von großen, „breitfrequenten“ (d.h. über viele Frequenzen verteilten) Datenmengen einfach, vollständiges *Speichern* (oder Weiterleiten) schwierig
- Insbesondere: sogar ehrliche V_i können viele Daten gleichzeitig senden, von dem jedes $\mathcal{A}_i$ nur einen kleinen Teil (etwa auf einer bestimmten Frequenz) speichern oder weiterleiten kann
- **Grundidee:** V_i überfluten P (bzw. die $\mathcal{A}_i$) mit breitfrequenten Datenströmen
- **Beobachtung:** nur P an bestimmter Position erhalten diese Daten gleichzeitig

Beispielprotokoll

- **Beispiel:** V_1 sendet gleichzeitig auf verschiedene Frequenzen verteilte Daten $(X_1, \ldots, X_n)$, und V_2 sendet $i^* \in \{1, \ldots, n\}$

$$\mathrm{V}_1 \xrightarrow{(X_1,\ldots,X_n)} \mathrm{P} \xleftarrow{i^*} \mathrm{V}_2$$

- V_1 und V_2 können sich vorher über i^* abstimmen
- Zweiter Schritt (eigentliche Authentifikation):

$$\mathrm{V}_1 \xleftarrow{X_{i^*}} \mathrm{P} \qquad \mathrm{V}_2$$

V_1 überzeugt, wenn das richtige X_{i^*} frühestmöglich ankommt

Beispielprotokoll

- Ehrliches Protokoll:

$$\mathrm{V}_1 \xrightarrow{(X_1,\ldots,X_n)} \mathrm{P} \xleftarrow{i^*} \mathrm{V}_2$$

$$\mathrm{V}_1 \xleftarrow{X_{i^*}} \mathrm{P} \qquad \mathrm{V}_2$$

- Angriffssituation (Beispiel: zwei $\mathcal{A}_i$, um Ps Position verteilt):

$$\mathrm{V}_1 \xrightarrow{(X_1,\ldots,X_n)} \mathcal{A}_1 \underbrace{\longleftrightarrow}_{\text{Abstand} \geq 2\Delta} \mathcal{A}_2 \xleftarrow{i^*} \mathrm{V}_2$$

- **Intuition:** $\mathcal{A}_1$ und $\mathcal{A}_2$ können X_{i^*} nicht an V_1 liefern
  - $\mathcal{A}_1$ erfährt i^* zu spät, hat das richtige X_{i^*} nicht gespeichert
  - $\mathcal{A}_2$ wird X_{i^*} erfahren, kann aber nicht rechtzeitig antworten

Mehr positionsbasierte Kryptographie

- Es existieren sichere Authentifikationsprotokolle, wenn Speicher der $\mathcal{A}_i$ hinreichend klein
- Verbesserungen durch Zufallsextraktion möglich
- Szenario auch für Schlüsselaustausch nützlich
- Interessant auch für Quantenkryptographie
  - Sicher gegen Angreifer, die nicht *entangled* sind
  - Wichtig: V_i und P müssen keinen Quantenspeicher haben
- Bei Interesse: `https://eprint.iacr.org/2009/364.pdf`

Überblick

Zusammenfassung

- Benutzerauthentifikation üblicherweise mit Hashes:

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{\mathrm{H}(\mathtt{pw})}$$

- Wichtigste Angriffe: Wörterbuchangriff, Brute Force
- Komprimierung des Wörterbuchs für Angriff möglich
- Neuere Implementierungen verwenden gesalzene Hashes

$$\mathrm{U}_{\mathtt{pw}} \xrightarrow{\mathtt{pw}} \mathrm{S}_{(\mathtt{salt},\mathrm{H}(\mathtt{salt},\mathtt{pw}))}$$

- Einige ältere Implementierung problematisch (Windows $\leq$XP)

(Mehr) aktuelle Forschung

- Untersuchung von Wörterbuchangriffen
- Andere Methoden zur Nutzerauthentifikation
  - **(Weiteres) Beispiel:** biometrische Daten
  - **Herausforderung:** Daten „fuzzy“
  - **(Weiteres) Beispiel:** motorisches Gedächtnis
  - **Genauer:** Nutzer trainiert „Guitar-Hero“-artig Tastenkombination, Authentifikationsprozess überprüft, wie gut Benutzer Kombination kennt
  - **Vorteil:** Nutzer kann Geheimnis selbst unter Zwang nicht preisgeben
  - Mehr dazu: `https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final25.pdf`

Überblick

Überblick

Motivation Zugriffskontrolle

- **Szenario:** Mehrbenutzersystem, gemeinsamer Zugriff auf Dateien mit verschiedenen Sicherheitsstufen
  - Nutzer aus Finanzabteilung soll Zugriff auf Gehaltsdaten haben
  - ... Nutzer aus Forschungsabteilung nicht
- **Erstes Ziel:** Zugriffsrechte von Nutzern verwalten

Beispiel: Unix-Rechteverwaltung auf Dateiebene

- Durchsetzen von Zugriffsbeschränkungen auf Dateiebene
- Üblich in allen neueren Unix-artigen Betriebssystemen
- Beispiel: `ls -l` (ruft Verzeichnisinhalt ab)

```
-rw-------   1   smith   users      4096   Dec 24   secret.txt
-rw-r-----   1   smith   finances   4096   Dec 24   salary.txt
```

- Jede Datei hat Besitzer (z.B. `smith`) und gehört zu Gruppe (z.B. `finances` oder `users`)
- Links stehen Berechtigungen
  - (Nur) `smith` darf `secret.txt` lesen und schreiben
  - Jeder Nutzer aus Gruppe `finances` darf `salary.txt` lesen, aber nur `smith` darf schreiben

Dynamische Rechteverwaltung

- **Problem:** Unix-Rechteverwaltung gruppenorientiert, statisch
- **Beispiel:** Gruppen `topsecret`, `secret`, `unclassified`
  - Was, wenn Nutzer manchmal `topsecret`-Daten lesen will, aber meistens auf `unclassified`-Daten arbeitet?
  - Möglich: zwei Nutzerkonten, kann aber unhandlich sein
- **Alternative:** betrachte „aktuellen Sicherheitslevel“ von Nutzern

Überblick

Das Bell-LaPadula-Modell

- **Idee:** Dynamische Rechteverwaltung
- Systemzustand formal modelliert, System kann *sicher* sein, wenn gewisse Regeln erfüllt sind
- Für jede einzelne Zugriffsanfrage (z.B.: Nutzer `smith` will Objekt `gehalt.txt` lesen) wird Gültigkeit festgelegt
- **Hauptresultat:** Gültige Anfrage überführt sicheres System in sicheres System
- **Wichtig:** Unterscheidung genereller/aktueller Sicherheitslevel von Nutzer

Bell-LaPadula-Formalisierung

- Formale Modellierung des Systems:
  - $\mathcal{S}$ Menge von *Subjekten* (z.B. Nutzern)
  - $\mathcal{O}$ Menge von *Objekten* (z.B. Dateien)
  - $\mathcal{A}$ Menge von *Zugriffsoperationen*

    Hier Beschränkung auf $\mathcal{A} = \{\texttt{read}, \texttt{write}, \texttt{append}, \texttt{execute}\}$
  - $\mathcal{L}$ halbgeordnete Menge von *Sicherheitsleveln*

    (Beispiel: $\texttt{topsecret} \geq \texttt{secret} \geq \texttt{unclassified}$)

Bell-LaPadula-Formalisierung

- Formale Modellierung des Systemzustands (B, M, F):
  - Menge B von Elementen $b \in \mathcal{S} \times \mathcal{O} \times \mathcal{A}$ (aktuelle Zugriffe)
    (Beispiel: $b = (\texttt{smith}, \texttt{salary.txt}, \texttt{read})$
  - Eine Zugriffskontrollmatrix $M = (M_{s,o})_{s \in \mathcal{S}, o \in \mathcal{O}}$, deren Einträge Untermengen von $\mathcal{A}$ sind

Beispiel:

	salary.txt	mail	fstab
smith	{read}	{execute}	$\emptyset$
jones	{read, write}	{execute, read, write}	$\emptyset$
spock	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{A}$

  - Ein Tupel $F = (f_s, f_c, f_o)$, wobei folgendes gilt:
    - $f_s : \mathcal{S} \to \mathcal{L}$ beschreibt den *maximalen* Level der Subjekte
    - $f_c : \mathcal{S} \to \mathcal{L}$ beschreibt den *aktuellen* Level der Subjekte
    - $f_o : \mathcal{O} \to \mathcal{L}$ beschreibt den Level der Objekte

Bell-LaPadula-Formalisierung

- **Frage:** wann sollte ein Systemzustand (B, M, F) „sicher“ sein?

Discretionary Security

- **Naheliegend:** aktuelle Zugriffe sollten konsistent mit Berechtigungen in Zugriffskontrollmatrix sein
- **Formalisierung:** „Discretionary-Security“-Eigenschaft

Definition (Discretionary-Security-/ds-Eigenschaft)

Ein Systemzustand (B, M, F) hat die ds-Eigenschaft, wenn für alle $(s, o, a) \in B$ gilt: $a \in M_{s,o}$.

- Notwendig, aber noch nicht hinreichend
- **Frage:** was könnte sonst noch schiefgehen?

Simple Security

- **Auch naheliegend:** wenn Subjekt s auf Objekt o zugreifen will, sollte (maximaler) Level von s mindestens so hoch sein wie der von o
- **Formalisierung:** „Simple-Security“- oder „No-Read-Up“-Eigenschaft

Definition (Simple-Security-/ss-Eigenschaft)

Ein Systemzustand (B, M, F) hat die ss-Eigenschaft, wenn für alle $(s, o, a) \in B$ mit $a = \texttt{read}$ gilt: $f_s(s) \geq f_o(o)$.

- Nach genehmigter Anfrage wird $f_c(s)$ angepasst, sofern nötig
- **Frage:** was könnte sonst noch schiefgehen?

Motivation Star Property

- **Problem:** hochprivilegierte Benutzer veröffentlichen (absichtlich oder unabsichtlich) geheime Daten
- **Lösung:** beschränke *Schreib*zugriffe hochprivilegierter Nutzer
- **Formal:** „Star Property“ oder „No-Write-Down“-Eigenschaft

Definition (Star Property/$\star$-Eigenschaft)

Ein Systemzustand (B, M, F) hat die $\star$-Eigenschaft, wenn für alle $(s, o, a) \in B$ mit $a \in \{\texttt{write}, \texttt{append}\}$ gilt: $f_c(s) \leq f_o(o)$.

- Man beachte, dass $f_c(s)$ und nicht $f_s(s)$ betrachtet wird