Sichere Programmierung

Klaus Kusche

Angriffe

- ***Datendiebstahl***
  (z.B. zum Verkauf der Daten)
- ***Datenzerstörung, Datenverschlüsselung, Datenveränderung***
  (z.B. zur Erpressung)
- ***DOS: "Denial of Service"*** = Ausfall der IT-Dienste
- ***Ressourcen-Diebstahl:***
  - Rechenzeit, z.B. für Mining
  - Netz für DDOS, als Spam-Schleuder, CC-Server, ...
  - Speicherplatz (z.B. für illegale Inhalte)

Motivation für Angriffe

- ***"Ich bin so cool"***
- *Wirtschaftl. Gründe, Konkurrenz*:
  Wirtschaftsspionage,
  Wettbewerbsvorteil, Rufschädigung
  Börsenmanipulation
- Persönliche *Schadensabsicht* (Wut & Zorn),
  z.B. *Ex-Mitarbeiter*,
  unzufriedene Mitarbeiter / Kunden
- Politische / *geheimdienstliche* Absichten
- ***Geld…***

Geld, Angreifer-Seite (1)

- ***Erpressung:***
  - *Wiederherstellung* der Daten (Krypto-Attacken)
  - *Beendigung* des DOS
  - *Nicht-Veröffentlichung* der Daten
  - *Erpressung Dritter* (mit kompromittierenden Daten)
- ***Verkauf:***
  - Der gestohlenen *Daten*
    (Kreditkarten, Zugangsdaten, Betriebsgeheimnisse, ...)
  - Der illegalen Zugänge / *Backdoors*
  - Der *Exploits*

Geld, Angreifer-Seite (2)

- ***Nutzung der Daten:***
  - Finanz- oder Bestell-Betrug
    (*Kreditkarten*-Daten usw.)
  - *Fake Id*'s für Konto-Eröffnung,
    Kreditbetrug, Sozialbetrug, ...
  - Nutzung geklauter Daten zum wirtsch. Vorteil:
    Betriebsgeheimnisse (Know-How/SW), *Kundendaten*
- Ausnutzung der Dummheit der Nutzer:

"Passen geklaute E-Mails & Passwords auch bei anderen Diensten?"

Geld, Opfer-Seite (1)

- ***Betriebsstillstand***, Produktionsausfall
- ***Maschinenschaden*** usw.
- Kosten für ***Daten-Wiederherstellung***,
  SW-Bereinigung (oft: externe Hilfe)
- Schäden durch ***Datenverlust***:
  - Buchhaltung, Kundendaten, ...
  - entwickelte Software bei SW-Firmen,
    Firmen-Know-How,
    zukünftige Produktplanungen, ...

  ==> Konkurs?!

Geld, Opfer-Seite (2)

- ***Erpressungs***-Kosten
- ***Umsatzminderung*** durch
  Rufschädigung, Kundenverlust, ...
- ***Strafe*** nach DSGVO
- ***Schadenersatzzahlungen*** an Dritte
- Kosten der ***Fehlerbehebung***
  und des ***Rollouts*** der abgesicherten Version
  (z.B. Kfz-Rückruf zum Update)

Allgemeines (1)

Über drei Viertel aller Sicherheitslücken
*entfallen auf wenige **"Standard-Fehlerarten"**!*

***"Echt falscher"** Code* (= *macht das Falsche*) ist *selten*!
Stattdessen meist:

Sicherheitslücke = Fehlender Code

- *Fehlende Input-Prüfungen*
- Fehlende Returncode-Prüfungen, ...

oder auch: Verwendung *unsicherer Konstrukte*

==> "Schlamperei" & "Fehlendes Wissen"

Allgemeines (2)

Jeder kann jeden Code
auf Sicherheitslücken untersuchen!

- Durch ***Reverse Engineering*** = Disassemblierung
- Durch ***gezielte Tests*** mit konstruiertem Input

==> *Irgendwann werden die Lücken gefunden!*
(u.a. wegen finanziellem Anreiz)

Nachträgliche Behebung ist meist
viel teurer als *"gleich richtig machen"*
(finanziell, vom Ruf her, ...)

Typische Ursachen für Lücken

Große Mehrheit aller Lücken:

- ***Speicherüberschreiber***
- ***Fehlende Input-Filterung***

==> *Jeweils eigener Foliensatz*

- **Restliche** Lücken:

  Mehrere verschiedene "kleinere" Ursachen...

  http://cwe.mitre.org/top25/#Listing

==> In diesem Foliensatz: Ein paar Beispiele...

“Triviale” Id-Vergabe (1)

Fortlaufende / vorhersagbare Nummern …

- … in ***Session Id’s, Cookies*** usw.

  ==› *“Kann ich eine fremde Session übernehmen, wenn ich die Session Id in meinen Requests um 1 erhöhe?”*

- … in ***URL’s für Downloads & Dokumente***

  ==› *“Welche (fremden) Dateien bekomme ich, wenn ich die File-Nummer in der Download-URL um 1 erhöhe?”*

“Triviale” Id-Vergabe (2)

- “Verstreute”, nicht fortlaufende,
  nicht vorhersagbare Id’s vergeben
  (z.B. Zufallszahlen, Hashwerte)

  ==> *“benachbarte” Werte* dürfen *nicht gültig* sein

  ==> *gezieltes Erzeugen* “gültiger” Id’s
  muss *sehr unwahrscheinlich* sein
  (*lange* Zeichenkette, Prüfsumme!)

  ==> *Lange “Totzeit”* vor dem Recycling einer Id
  (oder gar nicht recyceln)

“Triviale” Id-Vergabe (3)

- Download-URL’s usw. nicht blind vertrauen, *direkt beim Zugriff nochmal prüfen*:

  *Ist **diese URL** in **dieser Session** gültig?*

  = Gehört das Dokument zu dieser URL zum Benutzer in dieser Session?

  (oder schickt jemand eine Dokument-URL, die einem ganz anderen Benutzer gehört?)

  Gültigkeit der Session allein oder der URL allein reicht oft nicht!

“Triviale” Id-Vergabe (4)

- Vor allem bei URL’s,
  die eine *Aktion auslösen* (Buchung, Bestellung, ...):

  Einmal-Id’s verwenden!

  ==> Selber Request mit selber Id
  wird *kein zweites Mal akzeptiert!*

  (Schutz gegen Replay-Attacken)

Nur Client-seitige Prüfungen (1)

Clientseitige Prüfungen taugen
nur zur schnelleren/besseren *Benutzerführung*,
nicht zur Sicherheit!

- *Client* könnte reverse-engineert & *manipuliert*, durch fremdes Programm ersetzt, ... sein!
- *"Man in the middle"* könnte Requests "on the fly" manipulieren!

==> *Daten vom Client sind nie vertrauenswürdig!*

==> Alle *sicherheits-relevanten Prüfungen* müssen *am Server* stattfinden!

Nur Client-seitige Prüfungen (2)

Insbesondere *Prüfung der Session-Rechte*:

Der Server darf sich nicht darauf verlassen,
dass der Client nur solche Requests schickt,
die diese Session schicken darf / im GUI anbietet!

Beispiel:
Session als *"normaler Benutzer"* angemeldet

==> GUI bietet gar keine Klickflächen (URL's)
für administrative Befehle an

==> Was passiert, wenn diese Sitzung trotzdem
administrative Requests schickt???

Exzessive Rechte

Rechte aller ***angelegten Files, Dir's, DB's, IPC's, ...***
explizit möglichst restriktiv setzen,
nicht auf Defaults vertrauen!

Server-Code *aufspalten*:

- Entweder alle Operationen mit erhöhten Rechten *gleich beim Start* machen,
  dann *erhöhte Rechte abgeben* (eigene Server-Uid)
- Oder 2 getrennte Prozesse:
  - 1 x *erhöhte Rechte*, ***so wenig Code wie möglich***
  - 1 x *normale Rechte*, *Großteil* des Codes

File-Tricks (1)

Angreifer missbraucht
normale Filezugriffe eines Programmes.

Beispiel 1:

Der Server will eine *temporäre Datei schreiben*.

Der Angreifer *verlinkt* diesen Dateinamen
mit **`/etc/passwd`** (geht, ohne **`root`** zu sein!)

==> Der Server (falls **`root`**) *überschreibt*
`/etc/passwd` mit unsinnigen Inhalten

==> Sehr schwerer *Denial of Service*!

File-Tricks (2)

Beispiel 2:

Der Server erzeugt *dynamisch* einen *File* (z.B. HTML), um diesen später *an die Clients* zu schicken.

Der Angreifer schafft es, diesen File zwischendurch durch einen *Link auf* **`/etc/shadow`** zu ersetzen.

==> **`/etc/shadow`** wird *auf den Clients angezeigt*,
falls der Server mit **`root`**-Rechten läuft...
(auch *ohne* **`root`**-Rechte des Angreifers!)

File-Tricks (3)

Ursachen:

- Anlegen von Files in *für alle benutzbaren Dir's* (z.B. **`/tmp`**)
- ... mit *fixen oder erratbaren Namen* (z.B. berechnet aus Server-Pid, Datum & Uhrzeit, ...)
- ... mit *nicht atomarem* *"prüfen ob noch nicht existiert"* & *"zum Schreiben öffnen"*

==> ***Race Condition*** beim *File erzeugen!*

File-Tricks (4)

Korrektes Anlegen von Files in öffentlichen Dir's:

- Minimal-Anforderung:
  Mit *nicht vorhersehbarem Namen*
  Existenz-Prüfung & Anlegen *atomar*
- Besser:
  Atomar *Unterverzeichnis* anlegen
  Dessen *Rechte* auf eigenen User beschränken
  Files nur *in diesem Unterverzeichnis* anlegen

Vordef. Funktionen: **`tmpfile`**, **`mkstemp`**, **`mkdtemp`**

Krypto-Sünden (1)

- *Selbstgestrickte* Krypto-Algorithmen
- *Passwörter* im Klartext,
  mit Zweiweg-Verschlüsselung, ohne Salz,
  ohne Brute-Force-Bremse, ...
- *Hart eincodierte* User & Passwords
  (vor allem Admin-Accounts, "Hintertüren"),
  hart eincodierte Schlüssel & Zertifikate
- *Automatisches Anlegen* von Admin-Accounts
  und von Backend-Usern (z.B. DB-Accounts)
  ohne Passwort oder *mit fixem Password*

Krypto-Sünden (2)

- *Pseudo-Zufall*, wo echter Zufall sein sollte,
  Pseudo-Zufall mit fixem Seed, ...
- "Inoffizielle" Zertifikate, die Benutzer zwingen,
  im Browser eine Ausnahme zu akzeptieren
- *Automatischer Update* ohne krypt. Absicherung
  - des Update-Hosts & der Verbindung
  - des File-Inhaltes

  DNS-Attacken, Man in the Middle, ...
  zum Einschleusen von manipuliertem Code

  ==> 2 mal Verschlüsselung + Signatur/Zertifikat

Was tun?

- Hausinterne Programmier-*Richtlinien*,
  Programmierer & Tester *schulen*!
- Öffentliche sichere *Programmier-Standards*
  (z.B. CERT Secure Coding Standards für C, C++, Java, ...)
- *Tools* für Sicherheits-Qualitätssicherung
  - Statische Analyse
  - ASAN, valgrind, ...
  - Fuzzer
- (Public) *Code Reviews*, Bounties
- Externe Audits & *Penetration Tests*