IT-Sicherheit 2 – Zusammenfassung

XSS:

- Same Origin Policy:
  - Verbietet Skripten das Laden von Inhalten anderer Domains.
  - Protokoll, Domain und Port müssen gleich sein (auch Subdomain)
  - Eingebundene Scripts laufen auf der Seite, nicht auf der Quelldomain
- Problem: Eingegebene Daten werden auf der Seite 1:1 ausgegeben
  - Dadurch einbinden von Skripten in fremde Seiten möglich (Bsp. Forum etc.)
  - Auch Scripte durch gefälschte Links können eingebunden werden.
- Möglichkeiten:
  - Übergabe von Scripts durch http - $_Request Ausgabe
  - Escaping von Input-Feldern mit `>
- Reflected XSS:
  - Daten werden direkt aus Eingabe ausgegeben -> nicht gespeichert
- Stored XSS:
  - Daten werden in Speicher übergeben und von dort aus wieder ausgeben (z.B. DB)
- Teilweise werden Bilder oder andere Daten die HTML enthalten als solche ausgeführt.
- Verteidigung:
  - Analysiere http Anfragen auf Scripts (Client)
  - Analysiere ausgeführte Scripts (Client)
  - Maskieren der Ausgabe (Server)
    - Encoding mit htmlspecialchars etc.
  - Analyse der übergebenen Daten (Server):
    - Akzeptiere nur bestimmte Eingaben
    - Verweigere alle anderen Eingaben

SQL-Injection:

- Problem ist direkte Weiterleitung von Benutzereingaben in SQL-Abfrageb
- Durch , OR 1=1 kann die ganze Tabelle ausgegeben werden
- Rest der Abfrage durch - - oder # auskommentiert
- Mit Having 1=1 kann man anhand ausgegebener Fehlermeldungen die Struktur erkennen

CSRF:

- User ist auf einer Seite angemeldet und besucht währenddessen eine weitere infizierte.
- Auf der infizierten Seite kann nun ein Script sein, welches einen Befehl an die andere Seite sendet. Der Browser des Users hängt die Sessiondaten (Cookie) an. Der Befehl wird ausgeführt.
- So kann man auch Zugriff auf das Lokale Netzwerk erhalten.
- Es kann auch unbemerkt jemand angemeldet werden, um Daten auszuspionieren

Schutzmöglichkeiten:

- Zustandsloses Protokoll (keine Sessions wie http)
- **Secret validation token:**
- Session Identifier:
  - Client sendet Session ID in Cookie und URL.
  - Durch Teilen der URL kann Angreifer ID herausfinden
- Session-Independet Nonce:
  - Zufallsstring wird an Client gesendet und in Cookie gespeichert.
  - Client sendet Cookie und Token in Header und Body zurück.
  - Server prüft Gleichheit (Token im Body/URL kann Angreifer nicht ändern)
  - Vorteil: Angreifer kenn die SessionID nicht.
  - Nachteil: Angreifer kann Nonce überschreiben und eigene bauen.
- Session-Dependent Nonce:
  - Server gibt ein Token mit und speichert es
  - Client sendet Token immer wieder mit
  - Server prüft ob das Token mit dem gespeicherten übereinstimmt
  - Vorteil: Angreifer kann Token nicht fälschen
  - Nachteil: Server muss Token speichern, meist nicht ausreichend implementiert ->NoForge
- MAC SessionID:
  - Alle Hoster einer Seite kennen einen gemeinsamen Schlüssel k.
  - Server generiert durch SessionID und Key einen Token und vergleicht diesen mit dem mitgesendeten.
  - Vorteil: Fälschen des Tokens fällt auf, kein Token muss gespeichert werden
  - Nachteil: Erhöhter Rechenaufwand seitens des Servers
- Referer-Header: Prüfen von wo die Anfrage kam
  - Eine Seite kann sich so selbst vor CSRF schützen
- Custom-Header: Nur durch XHR möglich, dadurch nicht Suchmaschinenfreundlich
  - Browserverlauf versagt mit XHR
- Origin-Header: Speichert ursprüngliche Domain der Anfrage
  - Verfälscht durch Weiterleitungen
- NoForge: Hängt an alle URLs im HTML ein Token. Schlägt bei Browser-generiertem HTML fehl.
  - Problem: Hängt Token auch an externe Links.
  - Schützt nicht gegen Login CSRF.

Weitere Web-Lücken und Sicherheitsmaßnahmen:

Client:

- Eingabedaten sollten serverseitig getestet werden, Client kann verändert werden
- Javascript Schutzmaßnahmen sinnlos

Server:

- Zugriff auf private Dateien per URL-Änderung (RechnungsID etc)
- Direkte Manipulation von Headern, wenn Var in Header landet
- Zugriff auf Dateien per include
- Schadcode in Bildern etc. interpretiert durch Server

Erkennung von Schwachstellen:

Pixy:

Serverseitiger Schutz vor XSS in php

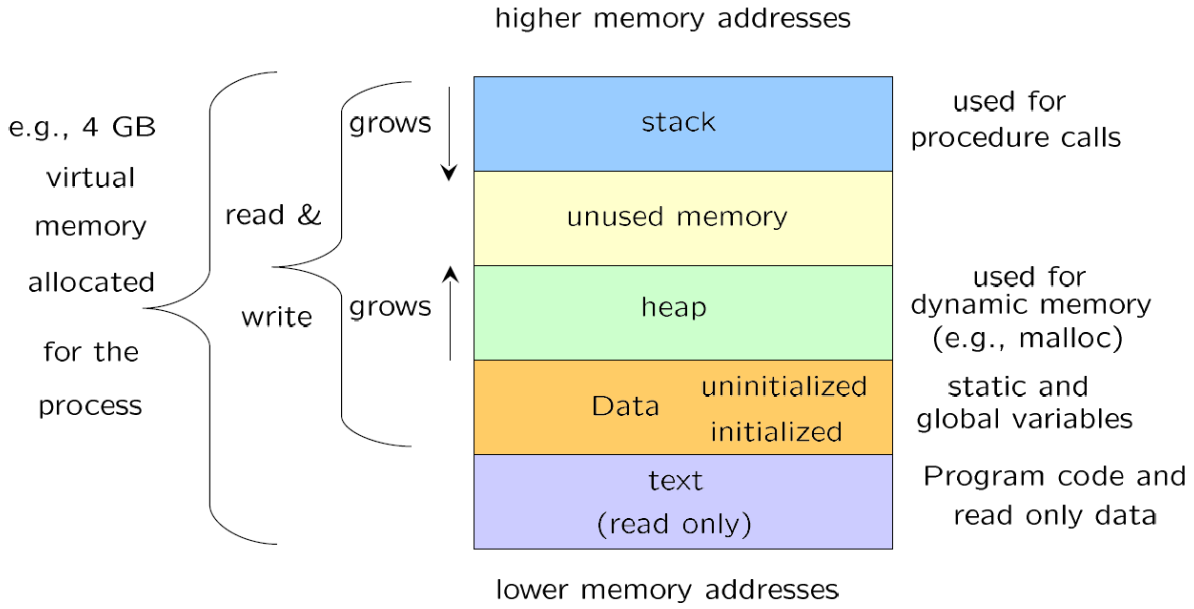
Prüft wo Daten eingeben werden „tainted Values“ und wo sie landen.

Falls sie in „sensitive sink“ landen besteht gefahr

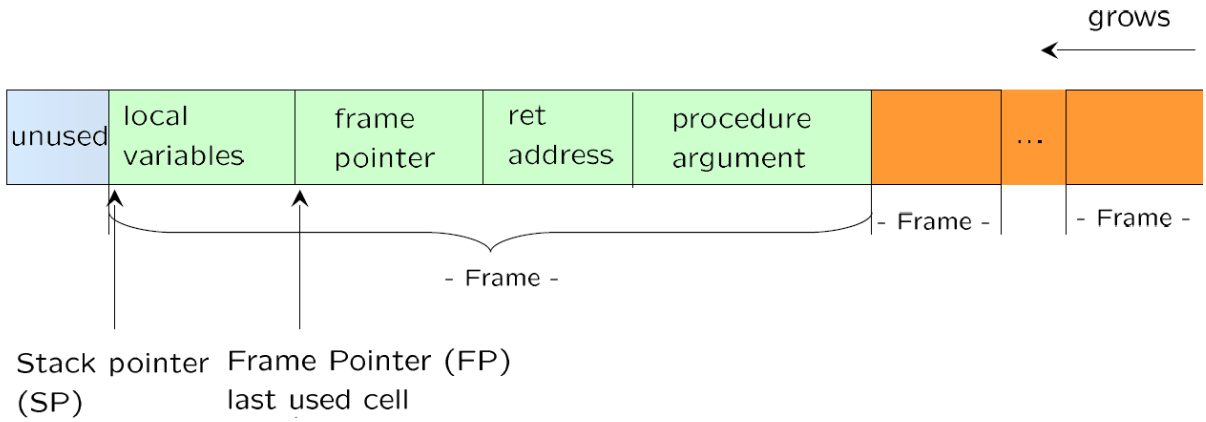
Viele Fehlalarme

Buffer Overflows:

Aufbau des Arbeitsspeichers:



Aufbau des Stacks:



Lokale Variablen können solche auf höherer Adresse, sowie frame pointer und return Adresse überschreiben

So kann durch Überschreiben der Return Adresse eigener Code ausgeführt werden.

Ansätze um Adresse richtig zu überschreiben:

Wiederholtes schreiben der Adresse, sodass eine dieser Wiederholungen die Return Adresse überschreibt.

Setzen von NOP’s an den Anfang des Codes. Die neue Return Adresse fällt hoffentlich

in die „NOP-Rutsche“ (NOP = No Operation Anweisung).

Falls Speicher zu klein kann man Code in Umgebungsvariable stecken. Sie stehen hinter den Argumenten.