Sicheres Forth

M. Anton Ertl, TU Wien

Problem

- Angriffe auf Computersysteme
- “Hackerangriff”
- Geheimdienste

Problem für Forth?

- Wir bauen eingebettete Systeme
  Eingaben beschränkt, daher unangreifbar

- Beispiel: TV Gerät mit zehn Knöpfen
  Fernbedienung
  Teletext (Videotext)
  DVB-S/C/T
  Smart TV (Internet, WLAN)

- StuxNet

- Internet of Things

Mögliche Angriffe

- Arbitrary Code Execution

- meist ermöglicht durch **buffer overflow**

- Dangling pointer

- Andere Angriffe
  Lesender buffer overflow (Heartbleed)
  SQL injection
  ...

Buffer overflows in Forth?

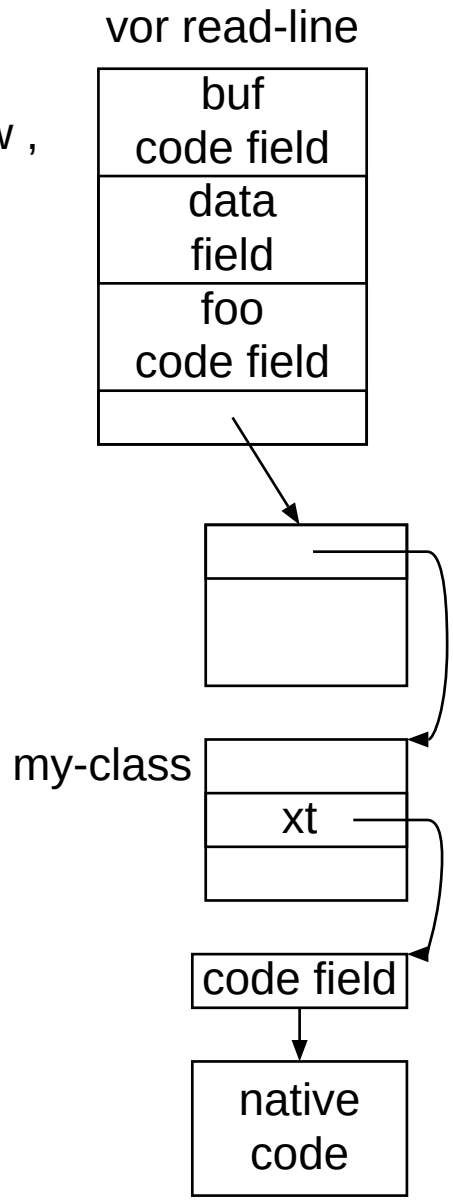
- Länge des Zugriffs explizit
  `move ( c-addr1 c-addr2 u -- )`
  aber: Zwei Längen involviert
  Fehler möglich
  `read-file ( c-addr u file-id -- )`

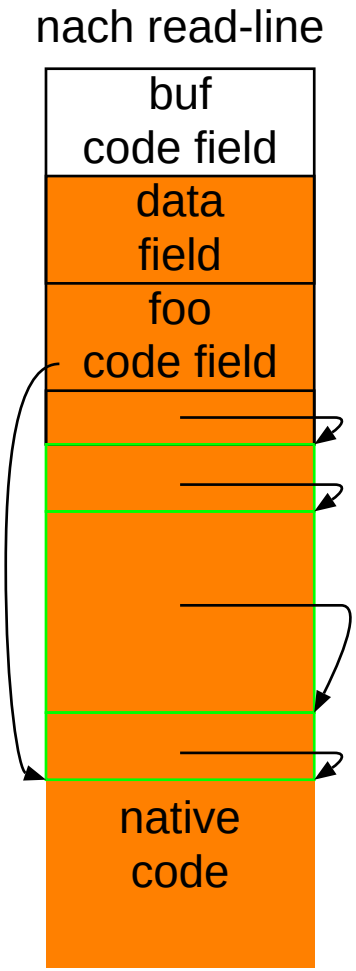
- Speicherzugriffe mit `! c!` etc.

- `xc!+?` statt `xc!+`

Hilft der separate Return-Stack?

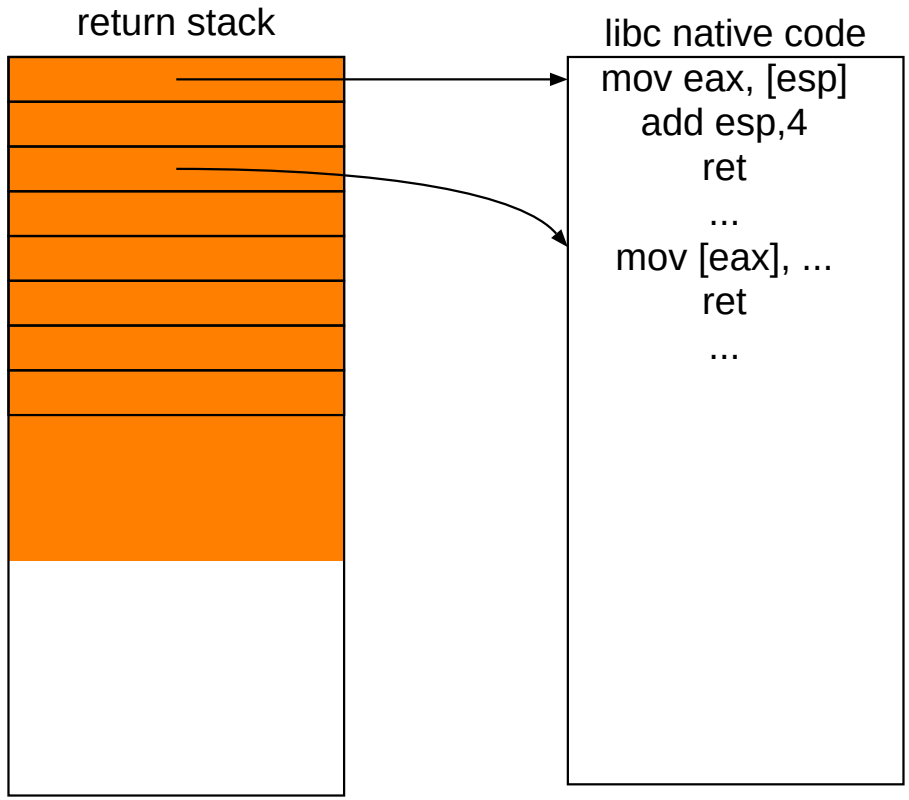
```
create buf 20 allot
create foo my-class new ,
buf 50 my-file read-line
foo @ my-method
```





Helfen nicht-ausführbare Daten?

- NX DEP W^X ...
- Return-Oriented Programming



Was tun?

- Code Audit

- Sichere Programmiersprache
  Kein Zugriff über Buffergrenzen hinaus

Und in Forth?

- Teilen der Programme
- Ein Teil in vollem Forth
  Benötigt Audit gegen Buffer overflows
- Ein Teil in einem sicheren Dialekt
  Mühsamer zu programmieren
  Einfacherer Audit
  Nur Sicherheit gegen Versehen
  Für mehr bräuchte man mehr Typechecking

```
... move ! ...
secure
... move* !* ...
insecure
...  ! !* ...
```

Sicherer Dialekt

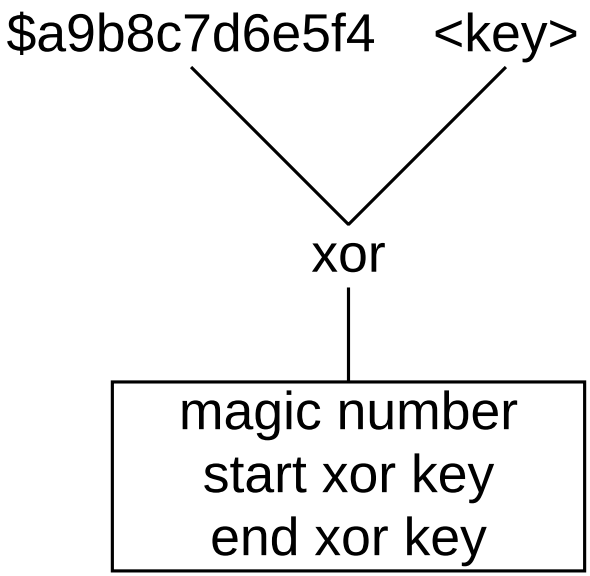
- Bufferdeskriptoren: Start, Ende
- Bei jedem Schreibzugriff übergeben und prüfen
- Alle schreibenden Wörter ersetzen
- `move* ( from to count buf -- )`
  `!* ( x addr buf -- )`
  `read-file* ( c-addr u file-id buf -- u2 ior )`

Typen?

- Zahl oder Zeiger als Deskriptor übergeben?
  `5 a 24 + b !*`

- Deskriptoren “verschlüsseln”

- Magic Number im Deskriptor



Dangling Pointers

- `buf @ free-buf 5 a buf @ !*`
- garbage collection statt `free`
- oder bei `free` Deskriptor überschreiben
  wenn da wieder ein Deskriptor hinkommt, kaputt, aber sicher

Zusammenfassung

- Buffer overflows $\Rightarrow$ beliebiger Code ausführen
- Abwehr durch sicheren Dialekt
  jedes schreibende Wort bekommt Buffer-Deskriptor
- kein Typecheck, daher kein Schutz vor Absicht
- Deskriptoren "verschlüsseln": Schutz gegen Versehen