Source Code Analyse

Ein praktikabler Ansatz

Marc Ruef

www.scip.ch

Hacking Day
11. Juni 2014, Zürich

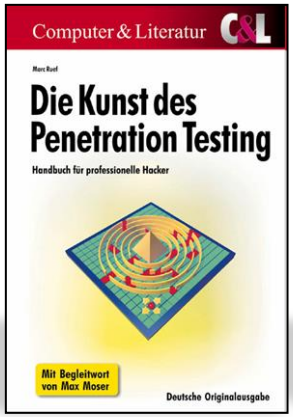
Agenda | Source Code Analyse

1. Einführung	
Wer bin ich	2 min
Was ist das Ziel	2 min
2. Source Code Analyse	
Eintrittspunkte	5 min
Austrittspunkte	3 min
Subroutinen	5 min
Logische Abläufe	5 min
Grafische Schnittstellen	2 min
Parallelisierung	2 min
Experimentell	3 min
Dokumentation	2 min
3. Abschluss	
Zusammenfassung	2 min
Fragerunde	10 min

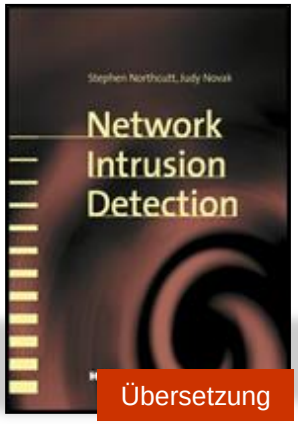
Einführung | Wer bin ich?

Name	Marc Ruef
Beruf	Mitglied der Geschäftsleitung, scip AG, Zürich
Private Website	http://www.computec.ch
Letztes eigenes Buch	«Die Kunst des Penetration Testing», Computer & Literatur Böblingen, ISBN 3-936546-49-5

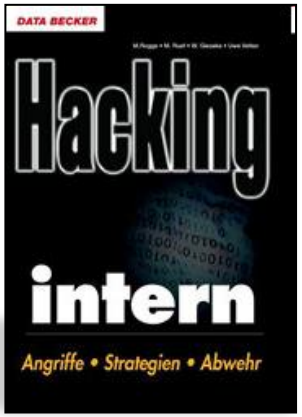
2013



2007



2004



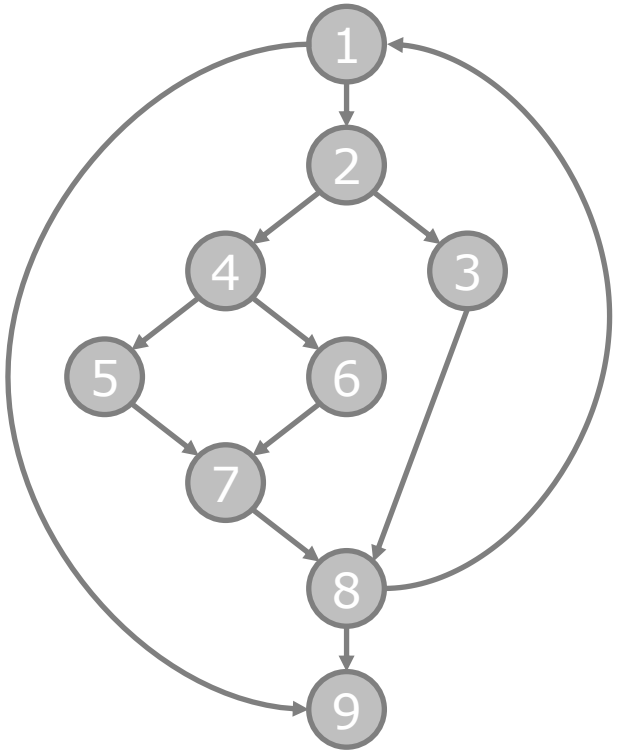
2002

Einführung | Was ist das Ziel

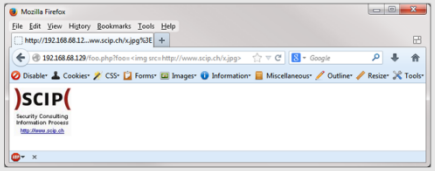
- Ziel einer Source Code Analyse
  - Identifikation von Schwachstellen im Quelltext einer Software
    - Fehler
    - Unsicherheiten
    - Ineffizienz
    - Unschönheiten

Einführung | Der praktikable Ansatz

- Akademischer Ansatz ist sehr aufwendig



- Praktikabler Ansatz soll schnelle und akkurate Resultate liefern

High	Reflektives Cross Site Scripting	ID 42
Risiko	5.0/10 (CVSS2#AV:N/AC:L/Au:N/C:N/I:P/A:N)	
Datei	/foo.php	
Funktion	getResult()	
Parameter	$_GET['foo'] (extern), $str (lokal)	
Zeilen	09-11	
Beschreibung	Durch eine fehlerhafte Eingabeüberprüfung wird es möglich, eigenen Script-Code zu injizieren, der bei der Ausgabe übernommen wird. Dadurch ist es Möglich, das Erscheinungsbild und Verhalten der Webapplikation zu manipulieren. 	
Massnahme	Die Eingabe sollte mit einer Whitelist auf ihre Korrektheit hin geprüft oder die Ausgabe mittels htmlentities() dargestellt werden.	
OWASP	OWASP-DV-001	
Links	http://www.scip.ch/?labs.20110914 (Eingabeprüfung im Detail)	

Einführung | Struktur einer Software

Eintrittspunkte | Ermöglichen Manipulationen

Eintrittspunkt	Exponiertheit	Einfachheit
Argumente	mittel-hoch	hoch
Umgebungsvariablen	gering	hoch
Dateien	mittel-hoch	gering-hoch
Datenbanken	gering-hoch	gering-hoch
HTTP Dateiuploads	mittel-hoch	hoch
HTTP GET-Parameter	hoch	mittel-hoch
HTTP POST-Parameter	mittel-hoch	mittel-hoch
HTTP Cookies	mittel	mittel-hoch
HTTP Sessions	gering	gering

Eintrittspunkte | Beispiele

Eintrittspunkt	PHP	ASP	JSP
Argumente	• $argv • $_SERVER['argv']		
Interaktive Eingabe	• fgets()		
Umgebungs-variablen	• $_ENV, getenv() • apache_getenv() • $_SERVER	• Request.ServerVariables • objShell.ExpandEnvironmentStrings()	• System.getenv() • System.getProperty()
Dateien	• fread() • fgets() • file() • file_get_contents()	• file.OpenAsTextStream() • objBinRead.readBinFile()	• InitialContext().lookup()
HTTP Dateiuploads	• $_FILES	• FileUploadControl • objUpload("foo").SaveAs	• request.getPart()
HTTP GET-Parameter	• $_GET, $_REQUEST • $_SERVER ['QUERY_STRING'] • $HTTP_GET_VARS	• Request.QueryString	• getParameter() • getParameterValues() • ${param['foo']} • ${param.foo}
HTTP POST-Parameter	• $_POST • $_REQUEST • $HTTP_RAW_POST_DATA • $HTTP_POST_VARS	• Request.Form • Request["foo"]	• getInputStream() • getReader() • ${param['foo']} • ${param.foo}
HTTP Cookies	• $_COOKIE	• Request.Cookies	• request.getCookies(), ${cookie['foo']}, ${cookie.foo}
...			

Eintrittspunkte | Suchen und Finden

```
01  maru@debian:~$ grep -H -n -r
    '$_GET\|$_POST\|$_SERVER\|$_COOKIE\|$_FILE' *.php
02  foo.php:3:if($_GET['a']  'foo'){
03  foo.php:5:}elseif($_POST['b']  'bar'{
04  foo.php:6:    echo htmlentities($_POST['c']);
```

Eintrittspunkte | Alternative Referenzierungen in PHP

- http://example.com/?foo=bar
  - $_GET['foo']
  - $_REQUEST['foo']
  - $_SERVER['QUERY_STRING']
    - parse_str($_SERVER['QUERY_STRING'], $arr); echo $arr['foo'];
    - preg_match('/foo=([^&]*)/', $_SERVER['QUERY_STRING'], $matches); echo $matches[1];
    - substr($_SERVER['QUERY_STRING'], strpos($_SERVER['QUERY_STRING'], 'foo='));
  - $HTTP_GET_VARS['foo'] (bis 4.1.0)

Austrittspunkte | Beispiele

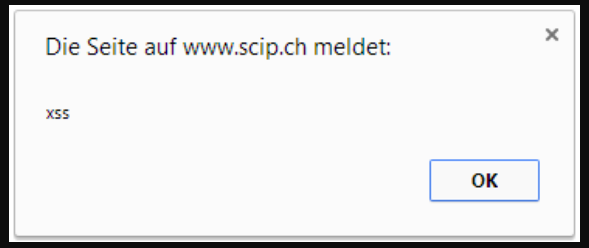
Austrittspunkt	PHP	ASP	JSP
Ausgabe	• echo • print • printf() • fprintf() • sprintf() • vprintf() • print_r()	• Response.Write	• out.println() • out.print()
Datei schreiben	• fwrite() • file_put_contents()	• file.Write() • file.WriteLine()	• InitialContext().bind()
MySQL Query	• mysqli_query() • mysqli_multi_query() • mysqli_real_query() • mysqli_send_query() • mysqli_stmt_execute()	• objConn.Execute()	• executeQuery() • executeUpdate() • execute()
Umgebungs-variablen	• putenv()		• System.setenv() • System.setProperty()

Austrittspunkte | Erschliessen von Schwachstellen

- Pufferüberlauf ⇒ überlange Eingaben
- Format String ⇒ überlange Eingaben
- Directory Traversal ⇒ ../ Zeichensequenzen
- OS Command Injection ⇒ Sonderzeichen + Chaining
- HTML Injection ⇒ HTML Anweisungen
- Cross Site Scripting ⇒ HTML/JS Anweisungen
- SQL Injection ⇒ SQL Anweisungen + Chaining
- Open Redirects ⇒ URL/Link

Austrittspunkte | Beispiel eines Cross Site Scripting

```
01  <?php
02
03  // Eingabe:
04  // http://example.com/?foo=<script>alert('xss');</script>
05
06  // Eintrittspunkt
07  $foo = $_GET['foo'];
08
09  // Austrittspunkt
10  echo $foo;
11
12  ?>
```



Austrittspunkte | Program Slicing

- Forward Slicing
  - Eintrittspunkt ⇒ Austrittspunkt
  - Lesefluss wirkt natürlicher
  - Typischerweise beim ersten Durchsehen des Codes

- Backward Slicing
  - Austrittspunkt ⇒ Eintrittspunkt
  - Alternative als Vier-Augen-Prinzip
  - Effizienter bei gewissen Angriffstechniken (z.B. XSS)

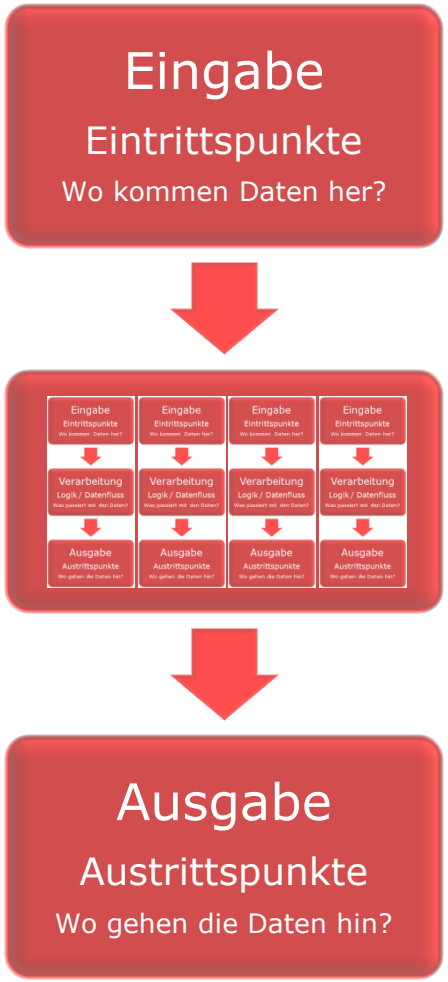
Subroutinen | Intrinsische Ein-/Austrittspunkte

- Parameter
  - Können entgegengenommen werden
  - Können unterschiedliche Konventionen aufweisen
    - Call by Value
    - Call by Reference
    - …
- Rückgabewerte
  - Können zurückgegeben werden
  - Können unterschiedliche Datentypen darstellen
    - Boolean
    - Integer
    - String
    - …

Subroutinen | Intrinsische Ein-/Austrittspunkte

```
01  <?php
02
03  // Uebergebe Benutzereingabe
03  $result = getResult($_GET['foo']);
04
05  // Zeige Resultat
06  echo $result;
07
08  // Funktion zur Abarbeitung
09  getResult($str){
10     return $str;   // gebe Resultat zurueck
11  }
12
13  ?>
```

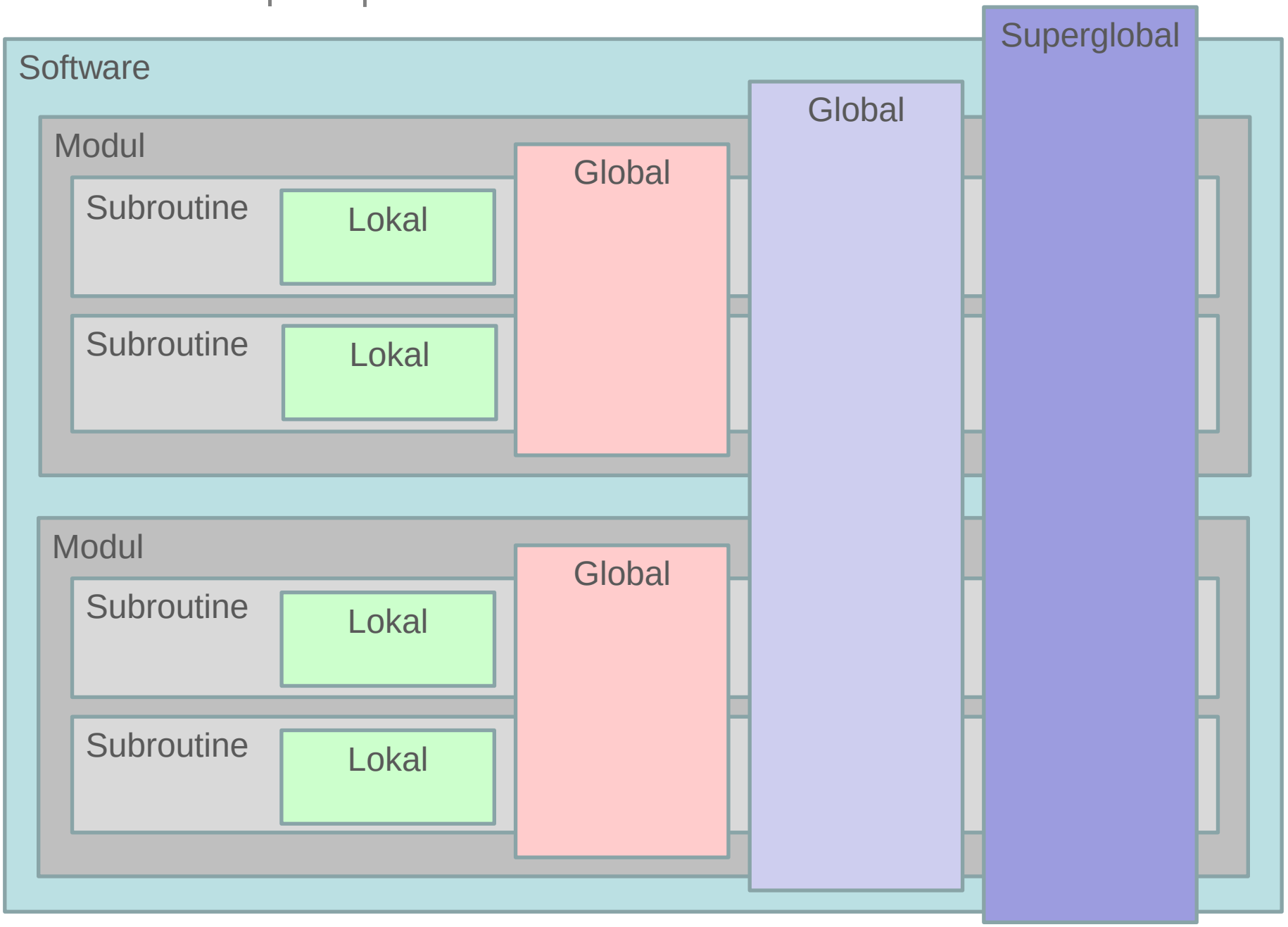
Subroutinen | Mikrokosmische Betrachtung



Subroutinen | Scope

- Objekte kennen einen Sichtbarkeitsbereich
- Globale Objekte
  - Sind «übergeordnet»
  - Können durch einzelne Routinen angesteuert/eingebunden werden
    - Superglobal ⇒ echo $_GET['foo'];
    - Global ($GLOBALS) ⇒ echo $GLOBALS['foo'];
    - Global (Keyword) ⇒ global $foo; echo $foo;
- Lokale Objekte
  - Sind nur durch die jeweiligen Routinen nutzbar
  - Sollten nach dem Beenden der Routine durch den Garbage Collector zerstört werden

Subroutinen | Scope



Subroutinen | Scope Beispiel

```
01  <?php
02
03  // $_GET ist superglobal
04  // $result ist gobal
05  $result = getResult($_GET['foo']);
06  echo $result;
07
08  getResult($str){
09      // $str ist lokal
10      return $str;
11  }
12
13  ?>
```

Logische Abläufe | Kontrollstrukturen

- Kontrollstrukturen bestimmen Programmablauf
  - if, elseif, else
  - for, foreach, while, do, until
  - switch, case
  - break, continue
  - goto, return, exit

Logische Abläufe | Fehler 1: Zuweisung anstatt Vergleich

```
01  <?php
02
03  $result = getResult($_GET['foo']);
04  echo $result;
05
06  getResult($str){
07      // Parameter wird zugewiesen; ist immer TRUE
08      if($str = 'bar'){
09          return $str;
10      }
11  }
12
13  ?>
```

Logische Abläufe | Fehler 2: Toter Code

```
01  <?php
02
03  $result = getResult($_GET['foo']);
04  echo $result;
05
06  getResult($str){
07     if($str > 0){
08        // Kann ausgefuehrt werden
09     }elseif($str > 10){
10        // Kann niemals ausgefuehrt werden
11     }else{
12        // Kann ausgefuehrt werden
13     }
12  }
13
14  ?>
```

Logische Abläufe | Fehler 3: Typenunsichere Prüfung

```
01  <?php
02
03  $result = getResult($_GET['foo']);
04  echo $result;
05
06  getResult($str){
07     if(strpos($str, '<')){
08        return '';   // nur falls an Position >0
09     }else{
10        return $str; // auch falls an Position 0
11     }
12  }
13
14  ?>
```

Logische Abläufe | Typenunsichere Prüfung bei PHP

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	arr()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
arr()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Beispiel wird gerne mit PHP gemacht. Aber viele andere typenunsichere Sprachen verhalten sich ähnlich «irrational».

http://www.scip.ch/?labs.20120503

Logische Abläufe | Fehler 4: Apple Goto Fail [CVE-2014-1266]

```
01   if ((err = SSLFreeBuffer(&hashCtx)) != 0)
02       goto fail;
03
04   if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
05       goto fail;
06   if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
07       goto fail;
08   if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
09       goto fail;
10   if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
11       goto fail;
12       goto fail;
13   if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
14       (...)
```

Datenverarbeitung | Konstruierte Datensätze am Beispiel PHP

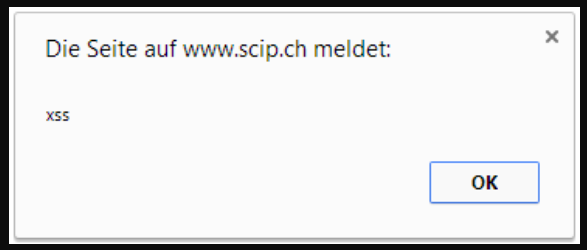
Konstruktion	Strings	Arrays
Konkatenation	• $c = 'a' . 'b'; • $e.= 'd'; • echo $c,$e;	• array_combine() • array_merge() • array_merge_recursive() • array_push() • compact() • implode() / join()
Diskonkatenation	• chunk_split() • explode() • preg_split() • split() • strtok()	
Teilstrings	• preg_match() • substr()	• array_chunk() • array_filter() • array_pop() • array_reduce() • array_shift() • array_slice()
Ersetzen	• preg_filter() • preg_replace() • str_ireplace() • str_replace() • substr_replace()	• array_replace() • array_replace_recursive() • array_splice()
Sortieren	• strrev() • str_shuffle()	• arsort() • array_multisort() • array_reverse() • krsort() • ksort() • natcasesort() • natsort() • shuffle() • sort() • rsort() • uasort() • uksort() • usort()

Datenverarbeitung | Erweiterte Funktionen

Angriff	PHP	ASP	JSP
Directory Traversal	• basename() • realpath()		
Cross Site Scripting	• htmlentities() • htmlspecialchars()	• Server.HTMLEncode()	• escapeHtml()
SQL Injection	• mysql_real_escape_string() • mysqli_real_escape_string() • sqlite_escape_string() • addslashes() • PDO::quote()		

Datenverarbeitung | Str-Manipulation erschliesst Schwachstelle

```
01  <?php
02
03  // Eingabe:
04  // http://example.com/?foo=<script>alert('&bar=xss');</script>
05
06  // Eingaben werden separat vorbereitet
07  $var1 = substr($_GET['foo'], 0, 15);
08  $var2 = substr($_GET['bar'], 0, 15);
09
10  // Ausgabe
11  echo $var1.$var2;
12
13  ?>
```

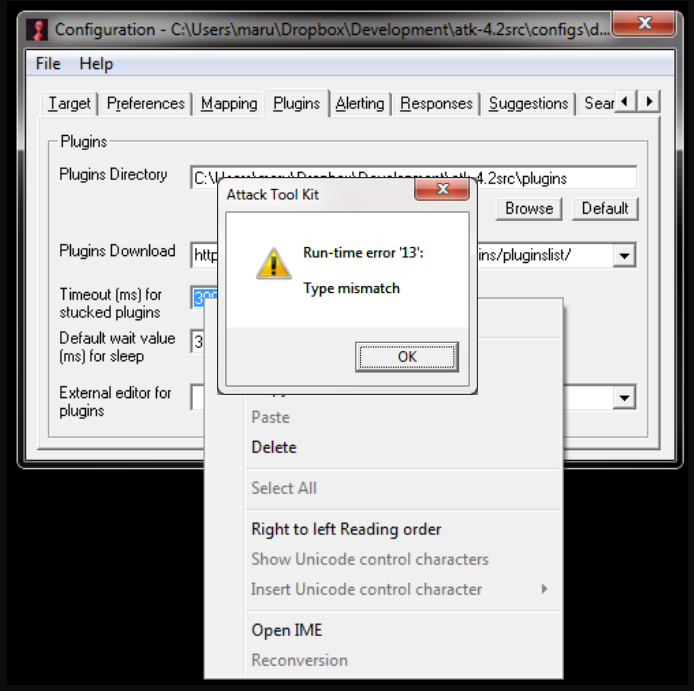


Grafische Schnittstellen | Eine andere Denkweise

- Eintrittspunkte bei prozeduralen Projekten sind simpel
- Bei grafischen Schnittstellen müssen Events verstanden werden
  - Wann wird ausgelöst?
  - Was wird ausgelöst?

Grafische Schnittstellen | Fehlerhafte Events

```
01   ''' Textbox erlaubt ausschliesslich Zahleneingaben
02   Private Sub txtProductID_KeyPress(KeyAscii As Integer)
03      Select Case KeyAscii
04         Case vbKey0 To vbKey9
05         Case vbKeyBack, vbKeyClear, vbKeyDelete
06         Case vbKeyLeft, vbKeyRight, vbKeyUp, vbKeyDown, vbKeyTab
07         Case Else
08            KeyAscii = 0
09            Beep
10      End Select
11   End Sub
```



Grafische Schnittstellen | 76 Events für TextBox in VS2013

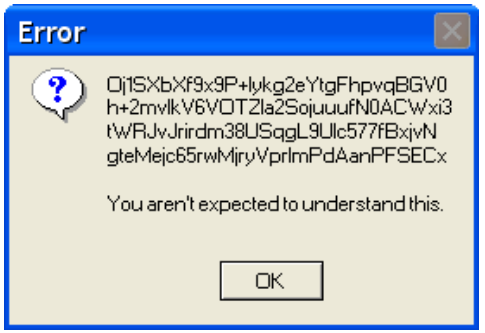
- AcceptsTabChanged
- AutoSizeChanged
- BackColorChanged
- BackgroundImageChanged
- BackgroundImageLayoutChanged
- BindingContextChanged
- BorderStyleChanged
- CausesValidationChanged
- ChangeUICues
- Click
- ClientSizeChanged
- ContextMenuChanged
- ContextMenuStripChanged
- ControlAdded
- ControlRemoved
- CursorChanged
- Disposed
- DockChanged
- DoubleClick
- DragDrop
- DragEnter
- DragLeave
- DragOver
- EnabledChanged
- Enter
- FontChanged
- ForeColorChanged
- GiveFeedback
- GotFocus
- HandleCreated
- HandleDestroyed
- HelpRequested
- HideSelectionChanged
- ImeModeChanged
- Invalidated
- KeyDown
- KeyPress
- KeyUp
- Layout
- Leave
- LocationChanged
- LostFocus
- MarginChanged
- ModifiedChanged
- MouseCaptureChanged
- MouseClick
- MouseDoubleClick
- MouseDown
- MouseEnter
- MouseHover
- MouseLeave
- MouseMove
- MouseUp
- MouseWheel
- Move
- MultilineChanged
- PaddingChanged
- Paint
- ParentChanged
- PreviewKeyDown
- QueryAccessibilityHelp
- QueryContinueDrag
- ReadOnlyChanged
- RegionChanged
- Resize
- RightToLeftChanged
- SizeChanged
- StyleChanged
- SystemColorsChanged
- TabIndexChanged
- TabStopChanged
- TextAlignChanged
- TextChanged
- Validated
- Validating
- VisibleChanged

Grafische Schnittstellen | Validierung

- Wann wird ein Control validiert?
  - Focus
    - GotFocus
    - LostFocus
    - ...
  - Maus Events
    - Click
    - DoubleClick
    - DragDrop
    - DragEnter
    - …
  - Key Events
    - KeyDown
    - KeyPress
    - KeyUp
    - ...

Grafische Schnittstellen | Modalität

- Frames kennen eine Modalität
  - Modal
    - Immer im Fokus
    - Erfordert Close, Hide oder Unload
    - Weiterer Code wird erst danach ausgeführt
  - Modeless
    - Erlaubt Fokuswechsel
    - Weiterer Code wird unmittelbar nach Anzeige ausgeführt
  - Child (Windows)
    - Erlaubt Modal und Modeless
    - Ist an Parent gebunden
    - Verhält sich gleich wie Parent bei Close, Hide, Unload und Minimize



http://www.scip.ch/?vuldb.13481

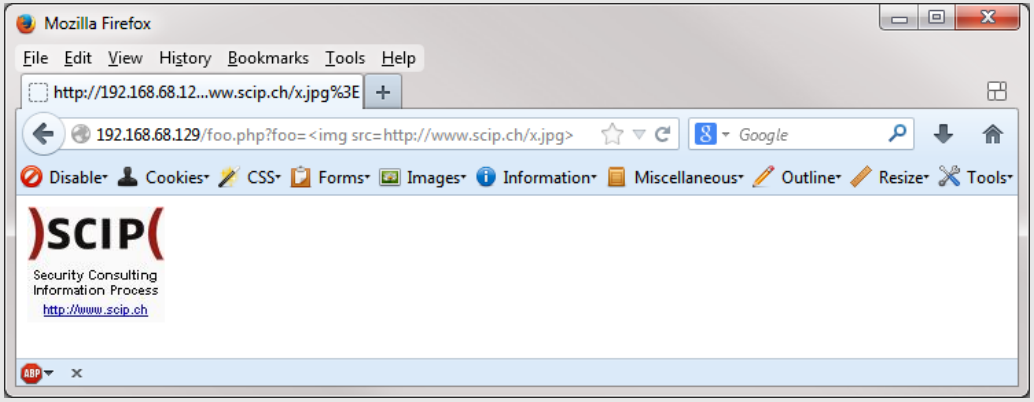
Parallelisierung | Zusätzliche Komplexität

- Abhängigkeiten werden eingeführt
- Nachvollziehbarkeit wird erschwert
- Heisenbugs sind möglich
  - Durch das Beobachten eines Bugs wird dessen Verhalten beeinflusst

Parallelisierung | Gefahr einer Race Condition

```
01   Private Sub Form_Load()
02      Me.Caption = "Foo 2.0"
03      Screen.MousePointer = vbHourglass
04      With tlbMenu.Buttons
05         .Item(1).Enabled = True
06         .Item(2).Enabled = True
07         .Item(3).Enabled = False 'deaktiviere Debug Moeglichkeiten
08      End With
09      Screen.MousePointer = vbDefault
10   End Sub
```

Dokumentation | Beispiel eines Findings

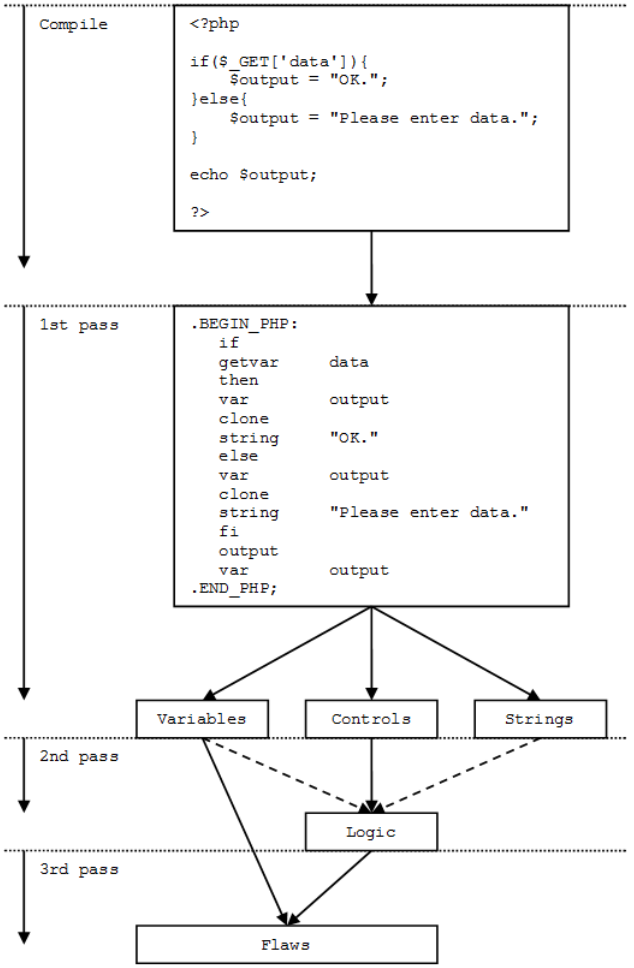
High	**Reflektives Cross Site Scripting**	**ID 42**
Risiko	5.0/10 (CVSS2#AV:N/AC:L/Au:N/C:N/I:P/A:N)	
Datei	/foo.php	
Funktion	getResult()	
Parameter	$_GET['foo'] (extern), $str (lokal)	
Zeilen	09-11	
Beschreibung	Durch eine fehlerhafte Eingabeüberprüfung wird es möglich, eigenen Script-Code zu injizieren, der bei der Ausgabe übernommen wird. Dadurch ist es Möglich, das Erscheinungsbild und Verhalten der Webapplikation zu manipulieren. 	
Massnahme	Die Eingabe sollte mit einer Whitelist auf ihre Korrektheit hin geprüft oder die Ausgabe mittels htmlentities() dargestellt werden.	
OWASP	OWASP-DV-001	
Links	http://www.scip.ch/?labs.20110914 (Eingabeprüfung im Detail)	

Dokumentation | Der Kontext wird wichtig

- Eine Software kann isoliert betrachtet werden
- Das Mitberücksichtigen des Kontexts generiert Qualität
  - Technische Umsysteme
  - Benutzerverhalten
  - Geschäftsprozesse

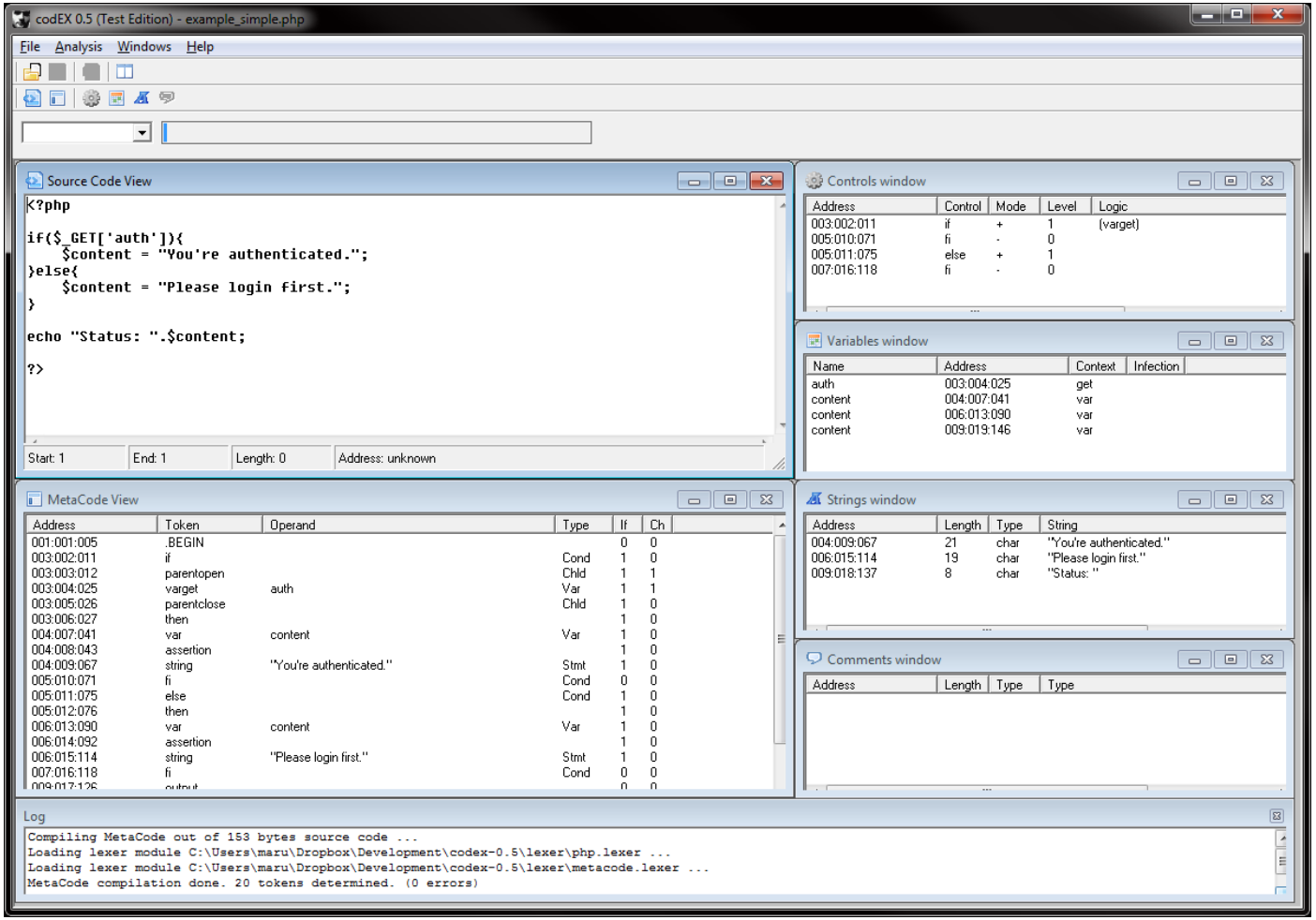
Experimentell | Der Ansatz von codEX

1. Original-Quelltext wird in MetaCode™ umgewandelt
2. Token werden ausgemacht und analysiert
3. Logische Entscheidungen und Datenfluss werden ermittelt
4. Schwachstellen werden identifiziert



http://www.computec.ch/projekte/codex/

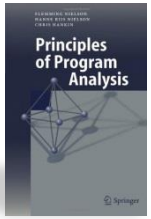
Experimentell | Beispiel einer Analyse mit codEX



Abschluss | Zusammenfassung

- Source Code Analysen sind ein mächtiges Werkzeug
- Damit lassen sich effizient und zuverlässig Schwächen erkennen
- Verständnis für die zugrundeliegende Sprache ist unabdingbar
- Eintritts- und Austrittspunkte sind die Ausgangslage
- Logische Entscheidungen und Datenfluss müssen verstanden werden
- Ein Report muss die Resultate nachvollziehbar festhalten
- Es gibt viele potentielle Möglichkeiten für Verbesserungen

Abschluss | Literatur (Source Code Analyse Allgemein)

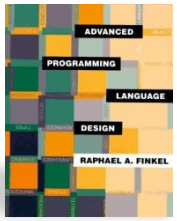


Principles of Program Analysis (2010)

Fleming Nielson, Hanne Riis Nielson, Chris Hankin

Sehr theoretischer aber umfangreicher Einstieg ins Thema der Software-Analyse

ISBN 978-3642084744



Advanced Programming Language Design (1995)

Raphael Finkel

Historische Herleitung von Programmiersprachen und ihren Paradigmen

ISBN 978-0805311914

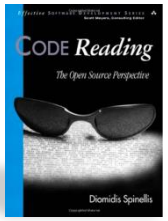
)SCIP(
scip ag

Source Code Analyse - Eine Einführung (2014)

Marc Ruef

Zusammenfassung dieses Vortrags mit zusätzlichen Informationen

http://www.scip.ch/?labs.20140425



Code Reading (2003)

Diomidis Spinellis

Einführung in das Lesen und Verstehen von Code

ISBN 978-0201799408

Abschluss | Fragen

Security is our Business!

scip AG
Jakob-Fügli-Strasse 18
CH-8048 Zürich

Tel +41 44 404 13 13
Fax +41 44 404 13 14
Mail info@scip.ch
Web http://www.scip.ch
Twitter http://twitter.com/scipag

- ☑ Strategy | Consulting
- ☑ Auditing | Testing
- ☑ Forensics | Analysis