Sicherheitslücken in Webanwendungen

Philipp Hagemeister

Lehrstuhl für Rechnernetze
Heinrich-Heine-Universität Düsseldorf

Rechtliches

Ziel dieses Vortrags ist es zu lernen, wie man sich vor Angreifern schützen kann.

Dazu muss man verstehen, wie Angreifer vorgehen.

Ohne Erlaubnis des Angegriffenen ist das Durchführen von Angriffen jeder Art ist strafbar!

Rechtliches

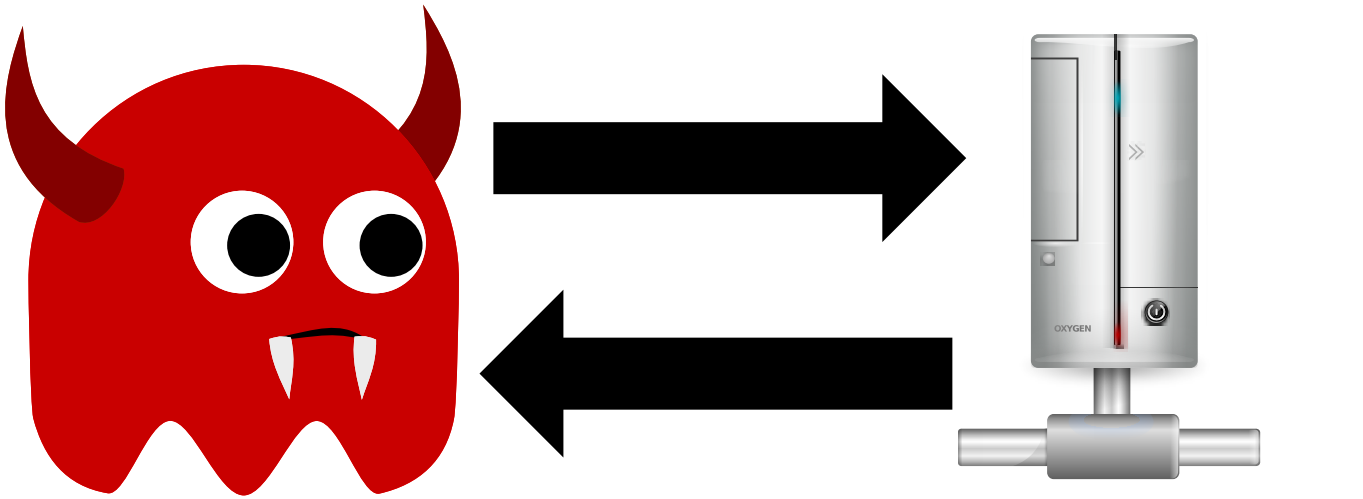
Ziel dieses Vortrags ist es zu lernen, wie man sich vor Angreifern schützen kann.

Dazu muss man verstehen, wie Angreifer vorgehen.

Ohne Erlaubnis des Angegriffenen ist das Durchführen von Angriffen jeder Art ist strafbar!

Bitte nur eigene Systeme zu Testzwecken angreifen!

Angriffsszenario (1)



Was kann hier angegriffen werden?

Angriffe gegen (Web)server

Remote-Desktop-Zugriff mit schwachem Passwort?

Angriffe gegen (Web)server

Remote-Desktop-Zugriff mit schwachem Passwort?

Firewalls einsetzen, sichere Passwörter,
Remote-Desktop beschränken!

Angriffe gegen (Web)server

Remote-Desktop-Zugriff mit schwachem Passwort?

Firewalls einsetzen, sichere Passwörter,
Remote-Desktop beschränken!

Rumänische Hackerbande mit 500.000 Kreditkartendaten verhaftet

Diese hatten sie sich bei australischen Handelsketten beschafft. Dort war es ihnen via RDP gelungen, in die Infrastruktur einzudringen und die Kassensoftware auszuspionieren. Mit rund 30.000 der gestohlenen Daten hatten sie sich bereits jeweils rund 1000 Australische Dollar ergaunert.

Angriffe gegen Server (2)

Buffer Overflows im Betriebssystem, Systembibliotheken, TCP/IP-Stack, Benutzerbibliotheken, Webserver, Anwendungsserver, Anwendung, ...

Angriffe gegen Server (2)

Buffer Overflows im Betriebssystem, Systembibliotheken, TCP/IP-Stack, Benutzerbibliotheken, Webserver, Anwendungsserver, Anwendung, ...

Mehr Code in Java (oder einer anderen modernen Programiersprache) schreiben!

Angriffe gegen Server (2)

Buffer Overflows im Betriebssystem, Systembibliotheken, TCP/IP-Stack, Benutzerbibliotheken, Webserver, Anwendungsserver, Anwendung, ...

Mehr Code in Java (oder einer anderen modernen Programiersprache) schreiben!

Oracle Java SE CVE-2013-2471 Buffer Overflow Vulnerability

Risk
High

Date Discovered
June 18, 2013

Description
Oracle Java SE is prone to a buffer-overflow vulnerability in Java Runtime Environment. An attacker can exploit this issue to execute arbitrary code in the context of the current user. This vulnerability affects the following supported versions: 7 Update 21 , 6 Update 45 , 5.0 Update 45

Webanwendungen angreifen

Wie sehen typische Schwachstellen in **Web**anwendungen aus?

Wie kann man sich dagegen verteidigen?

Manipulation von Pfadangaben

- Angenommen eine Bank bietet Downloads an
- Alle diese Dateien liegen in `/var/www/downloads`
- Der Code zum Ausgeben der Downloads sieht so aus:

```
String path = request.getParameter("path");
File f = new File("/var/www/downloads/" + path);
InputStream is = new FileInputStream(f);
IOUtils.copy(is, response.getOutputStream());
```

Manipulation von Pfadangaben

- Angenommen eine Bank bietet Downloads an
- Alle diese Dateien liegen in `/var/www/downloads`
- Der Code zum Ausgeben der Downloads sieht so aus:

```
String path = request.getParameter("path");
File f = new File("/var/www/downloads/" + path);
InputStream is = new FileInputStream(f);
IOUtils.copy(is, response.getOutputStream());
```

Wenn wir irgendeine Datei *woanders* im Dateisystem lesen wollen – wie können wir das erreichen?

Path Traversal

- Wir könnten als Dateinamen z. B. folgendes übergeben:

  `../../../etc/passwd`

- Das Programm würde daraus den folgenden Datei-Pfad zusammenbauen:

  `/var/www/downloads/../../../etc/passwd`

- Das Betriebssystem würde daraus folgendes machen:

  `/etc/passwd`

Path Traversal

- Wir könnten als Dateinamen z. B. folgendes übergeben:

  `../../../etc/passwd`

- Das Programm würde daraus den folgenden Datei-Pfad zusammenbauen:

  `/var/www/downloads/../../../etc/passwd`

- Das Betriebssystem würde daraus folgendes machen:

  `/etc/passwd`

Ups.

Path Traversal – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von Path-Traversal-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in einen Dateinamen übernommen werden…

…ohne zu überprüfen, ob sie Steuerzeichen enthalten!

Path Traversal – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von Path-Traversal-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in einen Dateinamen übernommen werden…

…ohne zu überprüfen, ob sie Steuerzeichen enthalten!

Was kann man dagegen tun?

Path Traversal – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von Path-Traversal-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in einen Dateinamen übernommen werden…

…ohne zu überprüfen, ob sie Steuerzeichen enthalten!

Was kann man dagegen tun?

- Vertraue keinen von außen gelesenen Daten!
- Bei Benutzereingaben immer überprüfen, ob sie nur gültige Zeichen enthalten
- …

Path Traversal an ungewöhnlichen Stellen

```
SAXParserFactory spf = SAXParserFactory.
                                newInstance();
SAXParser p = spf.newSAXParser();
String xml = readInput("xml");
InputStream is = new ByteArrayInputStream(
    xml.getBytes());
p.parse(is, new DefaultHandler());
```

Path Traversal mit SAX

Problemlose Eingabe:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/
     svg11.dtd">
<svg>
  ...
</svg>
```

Path Traversal mit SAX

Problemlose Eingabe:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/
     svg11.dtd">
<svg>
  ...
</svg>

<!DOCTYPE dt SYSTEM
\"file:/home/buchhaltung/rechnung20140112.pdf\">
```

Glücklicherweise normalerweise keine Ausgabe des Dateiinhalts

SQL Injection

Der Entwickler wollte SQL-Statements zusammenbauen, die folgendermaßen aussehen:

```
SELECT msg FROM ueberweisungen
WHERE user='user' AND msg LIKE '%filter%';
```

Den Wert von *user* kann ich nicht beeinflussen – *filter* kann ich aber frei wählen.

SQL Injection

Der Entwickler wollte SQL-Statements zusammenbauen, die folgendermaßen aussehen:

```
SELECT msg FROM ueberweisungen
WHERE user='user' AND msg LIKE '%filter%';
```

Den Wert von *user* kann ich nicht beeinflussen – *filter* kann ich aber frei wählen. Angenommen ich wähle:

```
user = philipp
filter = name' OR 'a%'='a
```

Dann erhalten wir folgende SQL-Anfrage:

```
SELECT * FROM ueberweisungen
WHERE owner='philipp' AND msg LIKE
'%name' OR 'a%'='a%';
```

SQL Injection

Der Entwickler wollte SQL-Statements zusammenbauen, die folgendermaßen aussehen:

```
SELECT msg FROM ueberweisungen
WHERE user='user' AND msg LIKE '%filter%';
```

Den Wert von *user* kann ich nicht beeinflussen – *filter* kann ich aber frei wählen. Angenommen ich wähle:

```
user = philipp
filter = name' OR 'a%'='a
```

Dann erhalten wir folgende SQL-Anfrage:

```
SELECT * FROM ueberweisungen
WHERE owner='philipp' AND msg LIKE
'%name' OR 'a%'='a%';
```

Jetzt werden *alle* Einträge der Tabelle ausgegeben, denn der Test (... `OR 'a%'='a'`) ist immer wahr!

SQL Injection

Viele Datenbankserver unterstützen Anfragen mit mehreren durch Semikolon getrennten SQL-Statements. Setzen wir doch einmal

filter = `egal'; DELETE FROM users; -`

Dann erhalten wir insgesamt folgende Anfrage:

```
SELECT * FROM items
WHERE owner='philipp' AND filter LIKE '%egal';
DELETE FROM users; -%';
```

SQL Injection

Viele Datenbankserver unterstützen Anfragen mit mehreren durch Semikolon getrennten SQL-Statements. Setzen wir doch einmal

filter = `egal'; DELETE FROM users; -`

Dann erhalten wir insgesamt folgende Anfrage:

```
SELECT * FROM items
WHERE owner='philipp' AND filter LIKE '%egal';
DELETE FROM users; -%';
```

Ups.

SQL Injection

Angenommen eine Anfrage für einen Microsoft SQL Server sieht so aus:

```
SELECT item,price FROM product
WHERE category='$user_input' ORDER BY price
```

SQL Injection

Angenommen eine Anfrage für einen Microsoft SQL Server sieht so aus:

```
SELECT item,price FROM product
WHERE category='$user_input' ORDER BY price
```

MS SQL erlaubt (wie einige andere Datenbankserver) das Ausführen von Shell-Kommandos in SQL-Statements. Zum Beispiel mit:

```
$user_input = ' exec master..xp_cmdshell 'dir' --
```

SQL Injection

Angenommen eine Anfrage für einen Microsoft SQL Server sieht so aus:

```
SELECT item,price FROM product
WHERE category='$user_input' ORDER BY price
```

MS SQL erlaubt (wie einige andere Datenbankserver) das Ausführen von Shell-Kommandos in SQL-Statements. Zum Beispiel mit:

```
$user_input = ' exec master..xp_cmdshell 'dir' --
```

Dann erhalten wir:

```
SELECT item,price FROM product
  WHERE category=' '                    ← nutzloses SELECT
exec master..xp_cmdshell 'dir'          ← böse
--' ORDER BY price                      ← MS-SQL-Kommentar
```

SQL Injection – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von SQL-Injection-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in ein SQL-Statement übernommen werden…

…ohne zu überprüfen, ob sie SQL-Steuerzeichen enthalten!

SQL Injection – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von SQL-Injection-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in ein SQL-Statement übernommen werden…

…ohne zu überprüfen, ob sie SQL-Steuerzeichen enthalten!

Was kann man dagegen tun?

SQL Injection – Ursachen, Gegenmaßnahmen

Die „Wurzel“ von SQL-Injection-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in ein SQL-Statement übernommen werden…

…ohne zu überprüfen, ob sie SQL-Steuerzeichen enthalten!

Was kann man dagegen tun?

- Vertraue keinen von außen gelesenen Daten!
- Bei Benutzereingaben immer überprüfen, ob sie nur gültige Zeichen enthalten
- In SQL-Statements eingefügte Zeichenketten immer richtig codieren (**nicht** `SELECT`, `DROP`, `...` ausfiltern)
- Parametrisierte SQL-Statements verwenden
- Viele Bibliotheken und Frameworks unterstützen das!

Prepared Statements in Java

Falsch:

```
String sql = "SELECT msg FROM ueberweisungen "
  + "WHERE user='" + USER + "' "
  + AND filter LIKE '%"
  + request.getParameter("filter") + "%'";
Statement s = createStatement();
ResultSet rs = s.executeQuery(sql);
```

Prepared Statements in Java

Falsch:

```
String sql = "SELECT msg FROM ueberweisungen "
  + "WHERE user='" + USER + "' "
  + AND filter LIKE '%"
  + request.getParameter("filter") + "%'";
Statement s = createStatement();
ResultSet rs = s.executeQuery(sql);
```

Korrekt:

```
String sql = "SELECT msg FROM ueberweisungen "
  + "WHERE user=? "
  + AND filter LIKE ?";
PreparedStatement s = conn.prepareStatement(sql);
s.setString(1, USER);
s.setString(2,
  "%'" + request.getParameter("filter") + "%'");
ResultSet rs = s.executeQuery();
```

ORMs

- ORMs wie EJB/hibernate vermeiden das Problem
- Kein SQL $\Rightarrow$ keine SQL Injection!

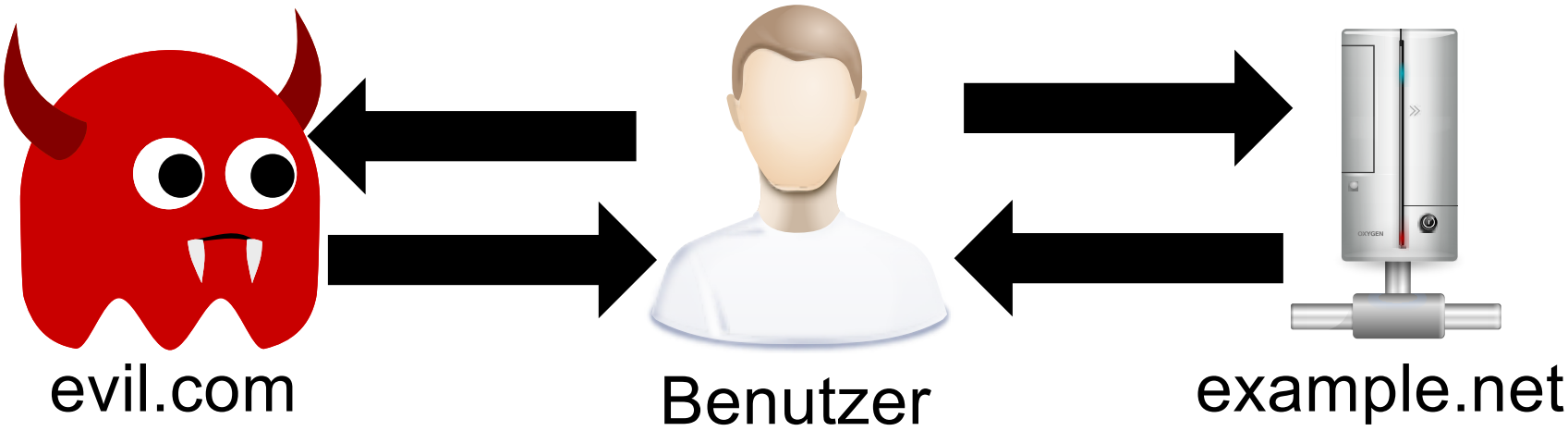
ORMs

- ORMs wie EJB/hibernate vermeiden das Problem
- Kein SQL $\Rightarrow$ keine SQL Injection!

- Aber Achtung: SQL-ähnliche Sprachen (z.B. HQL, LINQ) haben das gleiche Problem!

Angriffsszenario revisited

Idee: Wir greifen nicht mehr direkt den Server an!
Stattdessen: Benutzer auf *unsere* Seite `evil.com` locken



Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

Angriffsvorbereitung

Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

- Phishing-E-Mail

Angriffsvorbereitung

Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

- Phishing-E-Mail
- Links zu `evil.com` in Wikipedia etc. einschleusen

Angriffsvorbereitung

Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

- Phishing-E-Mail
- Links zu `evil.com` in Wikipedia etc. einschleusen
- Irgendeine Webseite hacken!

Angriffsvorbereitung

Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

- Phishing-E-Mail
- Links zu `evil.com` in Wikipedia etc. einschleusen
- Irgendeine Webseite hacken!
- Noch besser: Werbenetzwerk hacken!

Angriffsvorbereitung

Wie können wir den Benutzer dazu bringen `evil.com` zu besuchen?

- Phishing-E-Mail
- Links zu `evil.com` in Wikipedia etc. einschleusen
- Irgendeine Webseite hacken!
- Noch besser: Werbenetzwerk hacken!

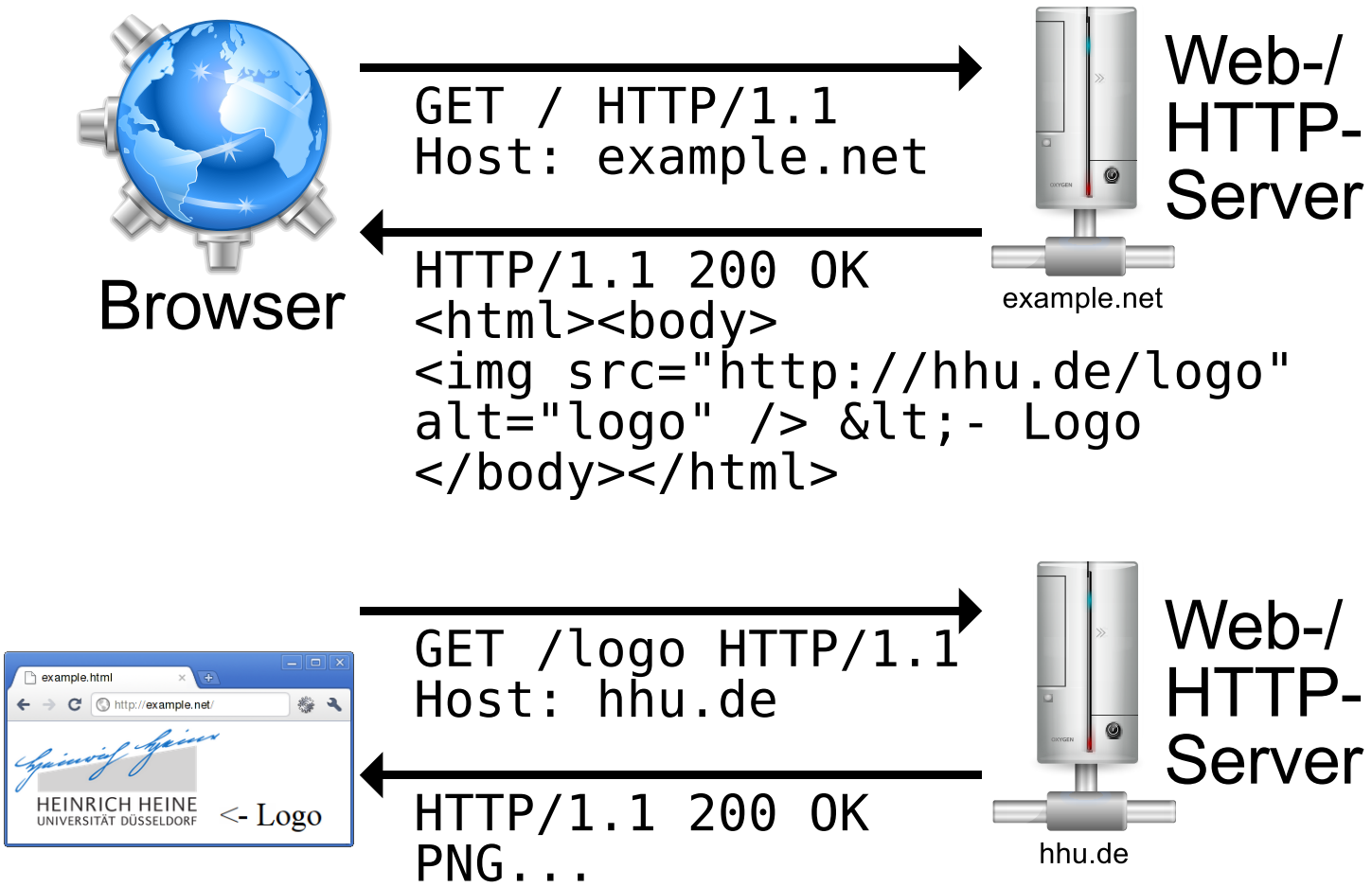
22.09.2014 14:44 « Vorige | Nächste »

Doubleclick und Zedo lieferten virenverseuchte Werbung aus

vorlesen / MP3-Download

Das große Werbenetzwerk Zedo und die Google-Tochter Doubleclick sollen nach Angaben eines Antivirenherstellers fast einen Monat lang Schadcode über ihre Werbung verteilt haben. Auch größere Webseiten wie Last.fm waren betroffen.

Ex-Kurs: HTTP und HTML („Web”)



- Sonderzeichen werden durch *entity references* dargestellt
  - z. B. `<`, `&#60;` oder `&#x3c;` für das Kleiner-Zeichen <.

Formulare

```
<form method="GET"
      action="http://example.net/send">
<input type="text" name="message"
       value="Hello, World" />
<input type="submit" value="Send" />
</form>
```

⇒ Browser sendet ...

```
GET /send?message=Hello%2C+World HTTP/1.1
Host: example.net
```

Formulare

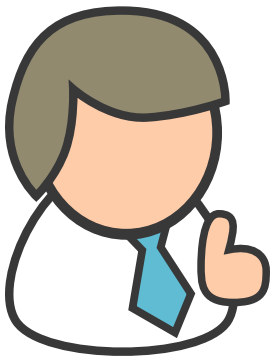
```
<form method="GET"
      action="http://example.net/send">
<input type="text" name="message"
       value="Hello, World" />
<input type="submit" value="Send" />
</form>
```

⇒ Browser sendet ...

```
GET /send?message=Hello%2C+World HTTP/1.1
Host: example.net
```

Wie kann ein Angreifer das ausnutzen?

Angriffsszenario gegen GET-Formulare



```
GET / HTTP/1.1
Host: evil.com
```

evil.com

```
HTTP/1.1 200 OK
<img
src="http://vulnerable.net/
➥send?message=Hello%2C+hack"/>
```

```
GET /send?message=Hello
➥%2C+hack HTTP/1.1
Host: example.net
```

example.net

Cross-Site Request Forgery (CSRF)

- GET-Anfrage verändert Daten **mit den Rechten des Benutzers** (wenn der eingeloggt/IP zugelassen ist)
  - Nachricht im Namen des Benutzers senden
  - Mit Rechten des Benutzers Inhalte hinzufügen, ändern, oder löschen
  - ...

Cross-Site Request Forgery (CSRF)

- GET-Anfrage verändert Daten **mit den Rechten des Benutzers** (wenn der eingeloggt/IP zugelassen ist)
  - Nachricht im Namen des Benutzers senden
  - Mit Rechten des Benutzers Inhalte hinzufügen, ändern, oder löschen
  - ...

Wie lösen wir das Problem?

Cross-Site Request Forgery (CSRF)

- GET-Anfrage verändert Daten **mit den Rechten des Benutzers** (wenn der eingeloggt/IP zugelassen ist)
  - Nachricht im Namen des Benutzers senden
  - Mit Rechten des Benutzers Inhalte hinzufügen, ändern, oder löschen
  - ...

Wie lösen wir das Problem?

GET-Anfragen dürfen keine Daten verändern!

HTTP-Standard: „(...) GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. ”

Cross-Site Request Forgery (CSRF) – POST

- Verwende POST für datenverändernde Anfragen:

```
<form method="POST"
      action="http://example.net/send">
<input type="text" name="message"
       value="Hello, World"/>
<input type="submit" value="Send" />
</form>
```

Problem gelöst?

Cross-Site Request Forgery mit JavaScript

```
<form action="http://example.net/send"
      method="POST">
<input type="hidden" name="message"
      value="Hello, hack" />
</form>

<script type="text/javascript">
window.onload = function() {
document.getElementsByTagName("form")[0].submit();
}
</script>
```

Und was machen wir jetzt?

CSRF – Gegenmaßnahmen

- Das Formular dynamisch erzeugen und eine eindeutige benutzerabhängige Nonce als unsichtbares Feld einbetten
  - typischerweise: die Cookie-ID des Benutzers zusätzlich in das Formular übernehmen
  - ein Zugriff wird nur dann akzeptiert, wenn die Cookie-ID im Formular gleich dem im POST-Request übermittelten Cookie ist
  - der Angreifer kann beliebige Daten ins Formular schreiben, kennt aber den Cookie nicht
- Bei „kritischen“ Operationen zusätzliche Bestätigung vom Benutzer verlangen
- Es gibt Bibliotheken, die das unterstützen!

[Zeller, Felten: Cross-Site Request Forgeries: Exploitation and Prevention, 2008]

HTML-Exkurs (2): Frames

- HTML erlaubt das Einbinden anderer Seiten in die aktuelle mit `<iframe>`
- Anwendungsfälle:
  - Toolbar bei Bildersuche oder Übersetzungsdienst
  - Anfragen im Hintergrund, z. B. für Update ohne Neuladen der Seite (historisch)
  - Navigation in einem Frame, Inhalt in einem anderem (historisch)
- Mit JavaScript kann der Inhalt des Frames eingesehen und verändert werden

HTML-Exkurs (2): Frames

- HTML erlaubt das Einbinden anderer Seiten in die aktuelle mit `<iframe>`
- Anwendungsfälle:
  - Toolbar bei Bildersuche oder Übersetzungsdienst
  - Anfragen im Hintergrund, z. B. für Update ohne Neuladen der Seite (historisch)
  - Navigation in einem Frame, Inhalt in einem anderem (historisch)
- Mit JavaScript kann der Inhalt des Frames eingesehen und verändert werden

Wie kann ein Angreifer das ausnutzen?

Inter-Frame-Angriffe

```
<iframe src="http://example.bank.com/transfer" />

<script type="text/javascript">
window.onload = function() {
  var frame=document.getElementsByTagName
                        ("iframe")[0];
  var doc = frame.contentDocument;
  doc.getElementById("to").value = "evil.com";
  doc.getElementById("amount").value = "6.66 $";
  doc.getElementById("form").submit();
}
</script>
```

Inter-Frame-Angriffe: Analyse

- Verteidigungsmaßnahmen gegen Cross-Site Request Forgery sind wirkungslos!
- Wenn nur ein Klick genügt, dann müssen wir den Frame-Inhalt nicht einmal verändern
  - Vor Klick-Auslösung den Frame mit JavaScript geschickt positionieren („**Clickjacking**“)
- Rendern innerhalb eines Frames kann verhindert werden
  - Mit JavaScript „Framebusting“-Code (kompliziert)
  - HTTP-Header `X-Frame-Options`
  - HTTP-Header `Content-Security-Policy`

[Rydstedt, Bursztein, Boneh, Jackson: Busting Frame Busting]

[Mozilla DN: Introducing Content Security Policy, `https://developer.mozilla.org/en/Security/CSP/Introducing_Content_Security_Policy`]

Die Same-Origin Policy

- Die **Same-Origin Policy** verbietet Interaktion zwischen Inhalten von verschiedenen Domains („origins“)
- Jedes HTML-Dokument, Bild, … bekommt einen Origin (Schema, Host, Port) zugeordnet
- JavaScript-Zugriff auf Inhalt mit anderem Origin löst eine Exception aus
- Beschränkt nicht nur Frames, sondern auch andere Interaktionsmöglichkeiten (XHR, canvas, …)
- Seit Netscape 2 in jedem Browser
- Seiten können Same-Origin-Beschränkungen explizit abschwächen (Cross-Origin Resource Sharing)
- Hilft allerdings nicht gegen Clickjacking

[HTML5: Origin, `http://www.w3.org/TR/html5/origin-0.html`]

[Cross-Origin Resource Sharing, `http://www.w3.org/TR/cors/`]

Cross-Site Scripting (XSS)

- Häufig stellen Webseiten Inhalte dar, die vorher von Benutzern übergeben wurden
  - Suchfeld
  - Benutzername nach Login
  - Social Networks
  - Foren
  - Webmail-Oberflächen
  - ...
- In jedem dieser Beispiele müssen die benutzerdefinierten Inhalte in den HTML-Code eingebettet werden
- Dafür: Programme/Skripte auf dem Server, die dynamisch die HTML-Seite „zusammenbasteln“

Cross-Site Scripting (XSS)

Das könnte in einem ganz einfachen Fall z. B. so aussehen (hier in JSP):

```
<% String eid = request.getParameter("eid"); %>
<html>
...
<body>
...
Employee ID: <%= eid %>
...
</body>
```

Cross-Site Scripting (XSS)

Das könnte in einem ganz einfachen Fall z. B. so aussehen (hier in JSP):

```
<% String eid = request.getParameter("eid"); %>
<html>
...
<body>
...
Employee ID: <%= eid %>
...
</body>
```

Wie kann ein Angreifer das ausnutzen?

Cross-Site Scripting (XSS)

- Angenommen, wir übergeben für den Parameter `eid` den Wert `<h1>Test</h1>`
- Dann sieht die zurückgelieferte HTML-Seite so aus:

```
<html>
...
<body>
...
Employee ID: <h1>Test</h1>
...
</body>
```

Cross-Site Scripting (XSS)

- Statt `<h1>` wären auch beliebige andere HTML-Tags möglich
- Also auch Skripte mit `<script>`!

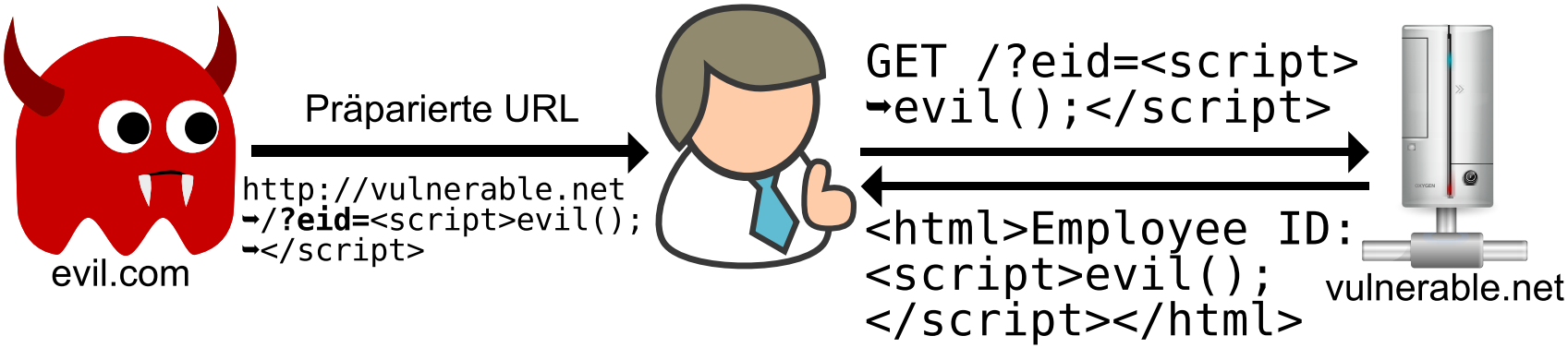
Was ist dann der Origin des eingeschleusten Skripts?

Cross-Site Scripting (XSS)

- Statt `<h1>` wären auch beliebige andere HTML-Tags möglich
- Also auch Skripte mit `<script>`!

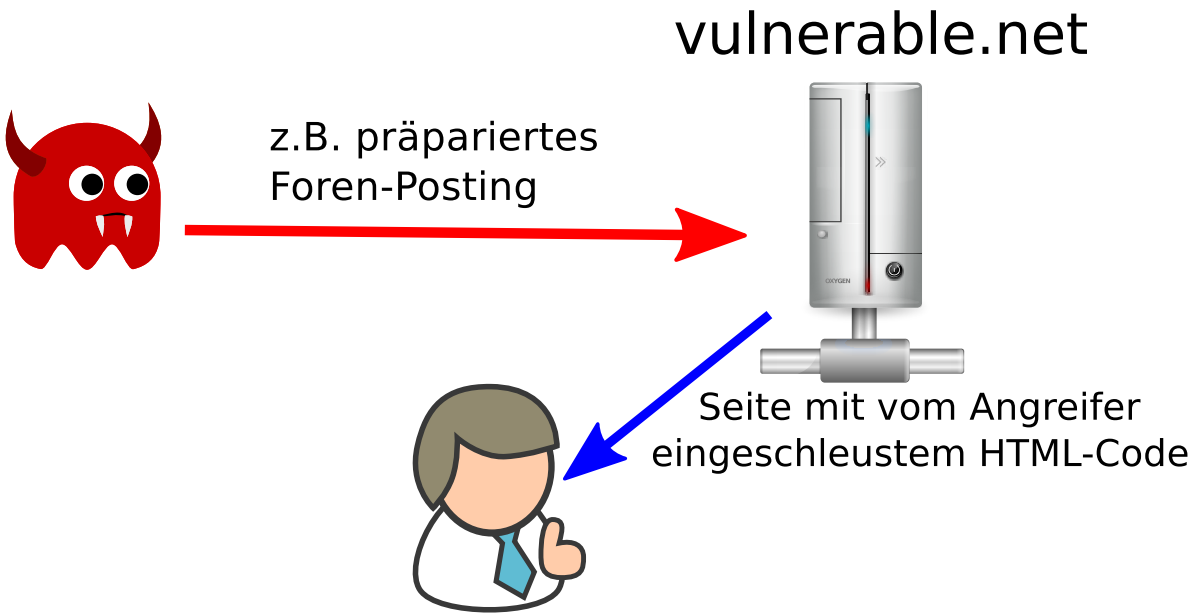
Was ist dann der Origin des eingeschleusten Skripts?

- Origin ist `example.net`, der Browser kann legitimen und eingeschleusten Code nicht unterscheiden



Stored XSS

- Manche Webseiten geben Eingaben nicht (nur) zurück, sondern speichern sie (z.B. in einer Datenbank)
- Anderen Benutzern werden diese Daten dann angezeigt
- Bei fehlerhafter/ausbleibender Kodierung: „Stored XSS“



Möglich wäre so etwas z. B. bei (schlecht programmierten!) Foren oder Social Networks

XSS – Ursachen und Gegenmaßnahmen

Die „Wurzel“ von XSS-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in einen (HTTP-)Ausgabe-Datenstrom übernommen werden…

…ohne zu überprüfen, ob sie (HTML-)Steuerbefehle enthalten!

XSS – Ursachen und Gegenmaßnahmen

Die „Wurzel“ von XSS-Verwundbarkeiten ist, dass zuvor von außen entgegengenommene Daten unverändert in einen (HTTP-)Ausgabe-Datenstrom übernommen werden…

…ohne zu überprüfen, ob sie (HTML-)Steuerbefehle enthalten!

- Vertraue keinen von außen gelesenen Daten!
- Bei Benutzereingaben immer überprüfen, ob sie nur gültige Zeichen enthalten
- In HTML-Code eingefügte Zeichenketten immer richtig kodieren
- Viele Bibliotheken und Frameworks unterstützen das!

XSS – Korrekte Enkodierung

- C#: `System.Web.HttpUtility.HtmlEncode`
- Java:
  `org.apache.commons.lang3.StringEscapeUtils.escapeHtml4`
  oder
  `org.springframework.web.util.HtmlUtils.htmlEscape`
- JavaScript: `_.escape` (Underscore)
- Perl: `HTML::Entities::encode_entities` oder `HTML::Escape`
- php: `htmlspecialchars`
- Python: `cgi.escape`
- Ruby: `CGI.escapeHTML`

- Vorsicht: Verschiedene Zeichensätze verlangen verschiedene Kodierungen
  - Zeichensatz angeben statt den Browser raten lassen!

XSS – Probleme bei Enkodierung

- Problem: Eine einzige vergessene Enkodierung genügt!

XSS – Probleme bei Enkodierung

- Problem: Eine einzige vergessene Enkodierung genügt!
- Besser: XML-Bibliotheken oder Templating (z.B. mustache)

```
<html>
<body>
Hallo <strong>{{name}}</strong>!
Achtung: {{{HTML}}}
</body>
</html>
```

Universal XSS und XSS-Filter

- Die Same-Origin Policy ist essentiell für Sicherheit im Web
- Eine Schwachstelle im *Browser* (**Universal XSS**) betrifft darum sehr viele Benutzer und *alle* Webseiten
- Plugins(z. B. Adobe Flash Player) müssen ebenfalls Same-Origin-Verletzungen verhindern

- Moderne Browser versuchen, mit **XSS-Filtern** verdächtigen (in der Anfrage vorhandenen) Code zu erkennen und durch harmlosen zu ersetzen
- Achtung: XSS-Filter können Universal XSS-Schwachstellen *verursachen*!

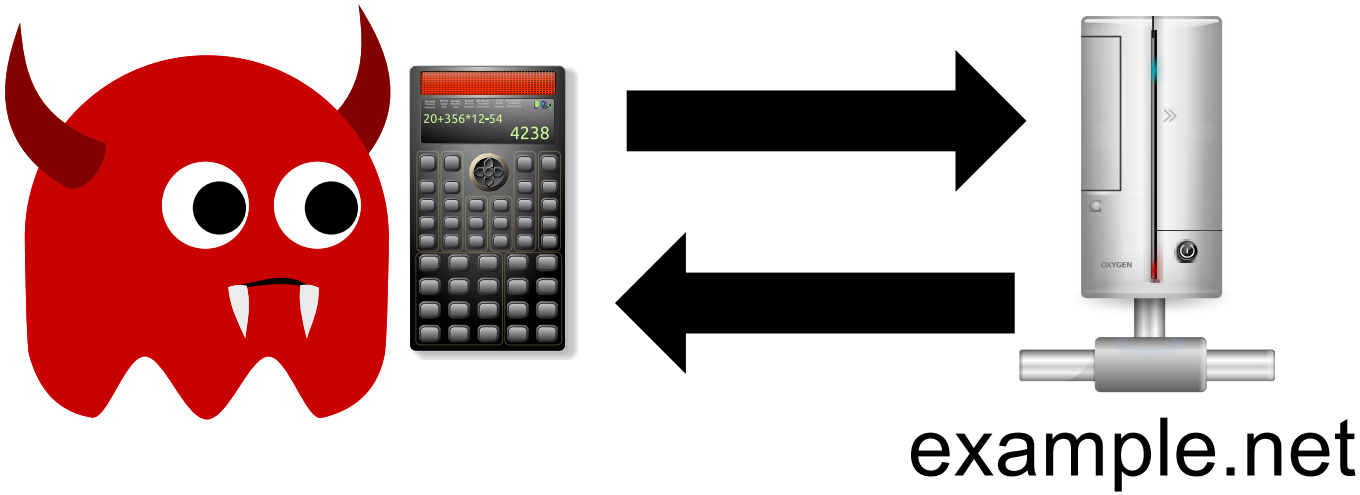
Schadcode-Erkennung (und -beseitigung) ist schwierig!

[Nava, Lindsay: Abusing Internet Explorer 8's XSS Filters]

Angriffsszenario (3)

Kryptografische Angriffe

... und Verteidigungen



Token-Generierung

- In manchen Fällen will man den Benutzer kein Passwort wählen oder übertragen lassen
- Stattdessen: Automatisch generieren
- Z.B. Session-Tokens nach Login
- Gefordert: Zufällig ...

Token-Generierung

- In manchen Fällen will man den Benutzer kein Passwort wählen oder übertragen lassen
- Stattdessen: Automatisch generieren
- Z.B. Session-Tokens nach Login
- Gefordert: Zufällig ...

heise Security

13.05.2008 18:27

Alert!

Schwache Krypto-Schlüssel unter Debian, Ubuntu und Co.

Die OpenSSL-Bibliothek der Linux-Distribution Debian erzeugt seit einem fehlerhaften Patch im Jahr 2006 schwache Krypto-Schlüssel. Der Sicherheitsexperte Luciano Bello entdeckten nun in dem OpenSSL-Paket eine kritische Schwachstelle, die die erzeugten Zufallszahlenfolgen und somit die erzeugten Schlüssel vorhersagbar macht.

Anforderungen an Tokens

- zufällig **und unvorhersagbar!**

Falsch:

```
byte[] token = new byte[16];
Random r = new Random();
r.nextBytes(token);
```

Anforderungen an Tokens

- zufällig **und unvorhersagbar!**

Falsch:

```
byte[] token = new byte[16];
Random r = new Random();
r.nextBytes(token);
```

> *Instances of java.util.Random are not cryptographically secure. Consider instead using SecureRandom to get a cryptographically secure pseudo-random number generator for use by security-sensitive applications.*
> *Random (Java Platform SE 8)*

Anforderungen an Tokens

- zufällig **und unvorhersagbar!**

Falsch:

```
byte[] token = new byte[16];
Random r = new Random();
r.nextBytes(token);
```

> *Instances of java.util.Random are not cryptographically secure. Consider instead using SecureRandom to get a cryptographically secure pseudo-random number generator for use by security-sensitive applications.*
> *Random (Java Platform SE 8)*

Richtig:

```
Random r = new SecureRandom();
r.nextBytes(token);
```

Speichern von Passwörtern

- Viele Leute verwenden Passwörter bei mehreren Diensten
- Dienste werden gehackt
- $\Rightarrow$ Bei Hack eines Dienstes: Angreifer erlangt Zugriff auf E-Mail-Konto, Social Media, ...
- Darum: Passwörter nicht im Klartext abspeichern!

Speichern von Passwörtern

- Besser als Klartext: Hash des Passworts abspeichern
- Beim Login `hash(Eingabe) == gespeicherter Hash` vergleichen
- Dann muss der Angreifer alle Passwörter durchprobieren!

Speichern von Passwörtern

- Besser als Klartext: Hash des Passworts abspeichern
- Beim Login `hash(Eingabe) == gespeicherter Hash` vergleichen
- Dann muss der Angreifer alle Passwörter durchprobieren!
- Alle Passwörter?
- Viele Benutzer wählen unsichere Passwörter (123456)
- oder verwenden einfach zu erratende Benutzer wie (<Hundename><Ziffer>)

Passwörter knacken

- Idee: *Alle* Passwörter hashen, dies effizient speichern
- $\Rightarrow$ Erlaubt schnelle Berechnung von Hash $\rightarrow$ Passwort
- Diese *Rainbow Tables* sind nicht zu groß
  - z.B. MD5, $\leq$ 9 alphanumerische Zeichen: 24GB

Passwörter knacken

- Idee: *Alle* Passwörter hashen, dies effizient speichern
- $\Rightarrow$ Erlaubt schnelle Berechnung von Hash $\rightarrow$ Passwort
- Diese *Rainbow Tables* sind nicht zu groß
  - z.B. MD5, $\leq$ 9 alphanumerische Zeichen: 24GB
- Verteidigung: Verwende für jeden Eintrag irgendein *salt* und speichere `salt` und `hash(Passwort + salt)`
- Dann muss der Angreifer wieder für jeden Eintrag alle Passwörter hashen

Passwörter knacken

- Idee: *Alle* Passwörter hashen, dies effizient speichern
- $\Rightarrow$ Erlaubt schnelle Berechnung von Hash $\rightarrow$ Passwort
- Diese *Rainbow Tables* sind nicht zu groß
  - z.B. MD5, $\leq$ 9 alphanumerische Zeichen: 24GB
- Verteidigung: Verwende für jeden Eintrag irgendein *salt* und speichere `salt` und `hash(Passwort + salt)`
- Dann muss der Angreifer wieder für jeden Eintrag alle Passwörter hashen
- Vorsicht: Mit GPUs kann auch das schnell gehen!
- Darum *stretching*: `hash(hash(..hash(Passwort+salt)..))`

Gute Passwortspeicherung

- Hashing und stretching nicht selber implementieren!
- Sondern PBKDF2/bcrypt/scrypt verwenden

Gute Passwortspeicherung

- Hashing und stretching nicht selber implementieren!
- Sondern PBKDF2/bcrypt/scrypt verwenden
- oder erst gar keine Passwörter speichern!
- sondern: Single Sign-On
  - LDAP
  - Kerberos
  - Facebook Connect
  - OpenID
  - OAuth

Hausaufgabe

Hausaufgabe

Testen Sie **ihre** (Web-)Anwendung!

Hausaufgabe

Hausaufgabe

Testen Sie **ihre** (Web-)Anwendung!

- Durch Blick in den Code
- Mit einem automatischem Schwachstellenscanner
  - arachni
  - skipfish
  - w3af
  - webvulnscan (Forschungsprojekt)

Mehr Lesen und Ausprobieren

- vulnsrv (eine löchrige Web-Anwendung zum „Ausprobieren“ von Sicherheitslücken):
  `https://github.com/phihag/vulnsrv`
- Alternativ: Google Gruyere:
  `http://google-gruyere.appspot.com/part1`
- OWASP Top 10 vulnerabilities:
  `https://www.owasp.org/index.php/Top_10_2013-Top_10`
- Browser Security Handbook:
  `http://code.google.com/p/browsersec/wiki/Main`

Fragen?

Fragen?

Bonus: Authentifizierung (nicht nur) im Web

- Problem: HTTP ist statuslos, aber wir wollen Benutzer-Identität speichern
- Login bei jeder Aktion ist benutzerunfreundlich

Was halten Sie davon?

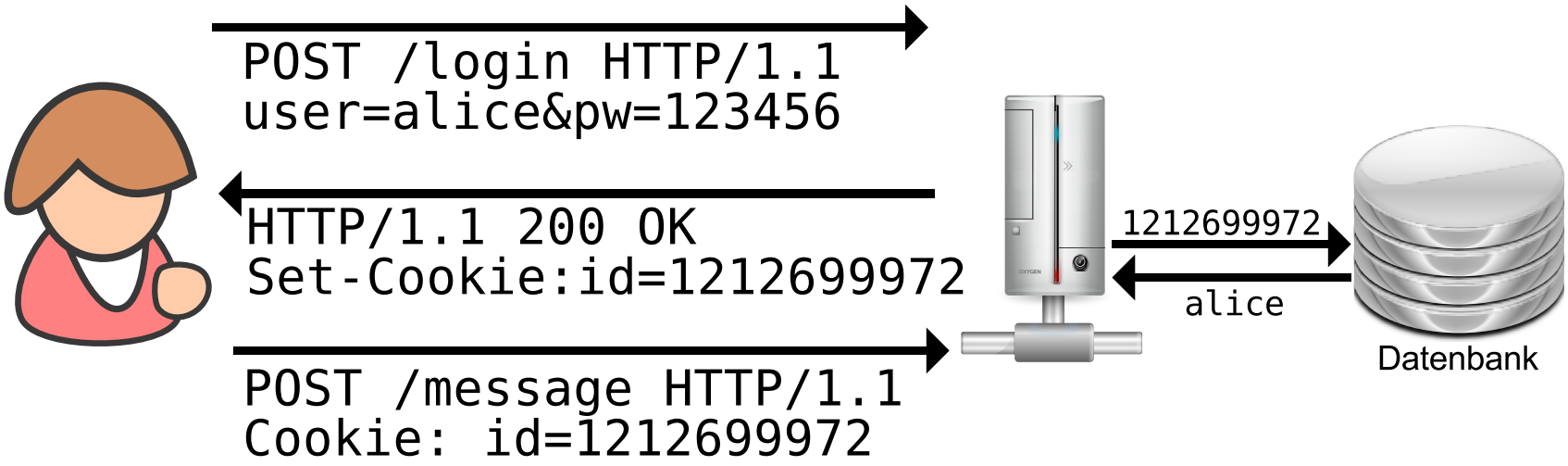
Passwort im Cookie: Bewertung

Was halten Sie davon?

- Andauernde Übertragung ist unsicher
- Überprüfung ist langsam ($\rightarrow$ hash stretching)

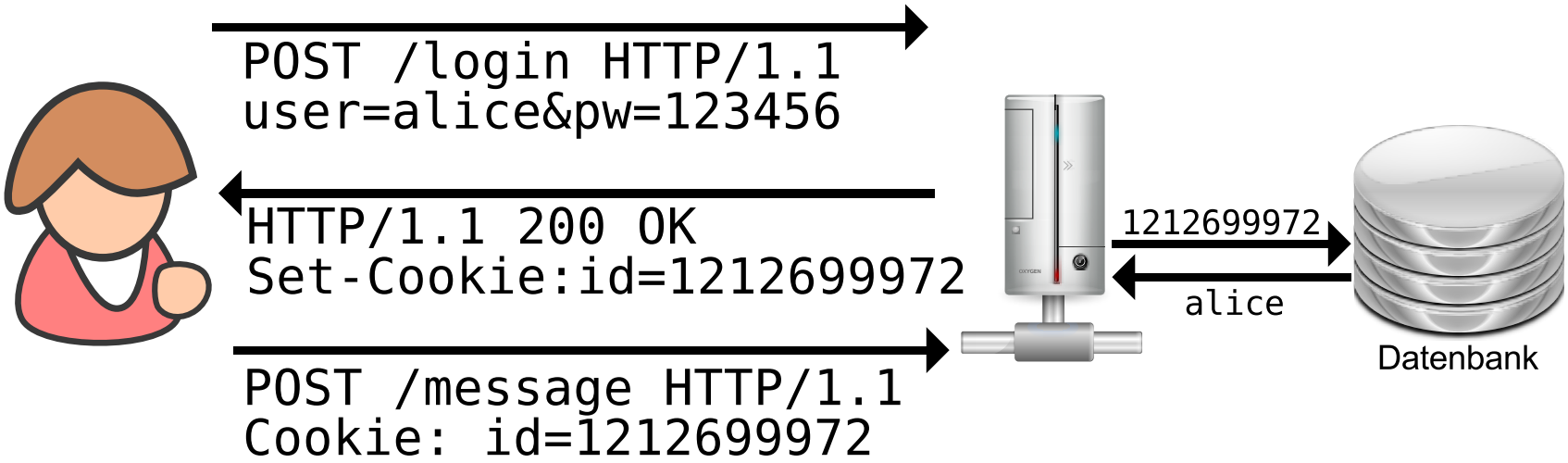
Authentifizierung mit Session-IDs

- Idee: Generiere zufällige ID
- Speichere Assoziation (ID $\rightarrow$ Benutzer) in Datenbank



Was halten Sie davon?

Authentifizierung mit Session-IDs: Probleme



- Sicherheit: Gegeben (wenn ID hinreichend zufällig)
- Problem: Jetzt brauchen wir eine Datenbank!
  - .. und müssen für jeden Login einen Eintrag einfügen!
  - Kein Problem bei einem einzelnen System, skaliert aber nicht

Authentifizierung mit Public Key Cryptography

Verwende digitale Signaturen!

Probleme: Langsam, Denial of Service leicht möglich

Authentifizierung mit Message Authentication Codes

Darum: Verwende einfach hash über Geheimnis und die Nachricht!

Schnell und sicher!

Attacken auf MACs

Beispielimplementierung:

```
def getUser(secret, input_user, input_mac):
    if sha256(secret + input_user) == input_mac:
        return input_user
    else:
        return invalid
```

Ist das sicher?

Attacken auf MACs

Beispielimplementierung:

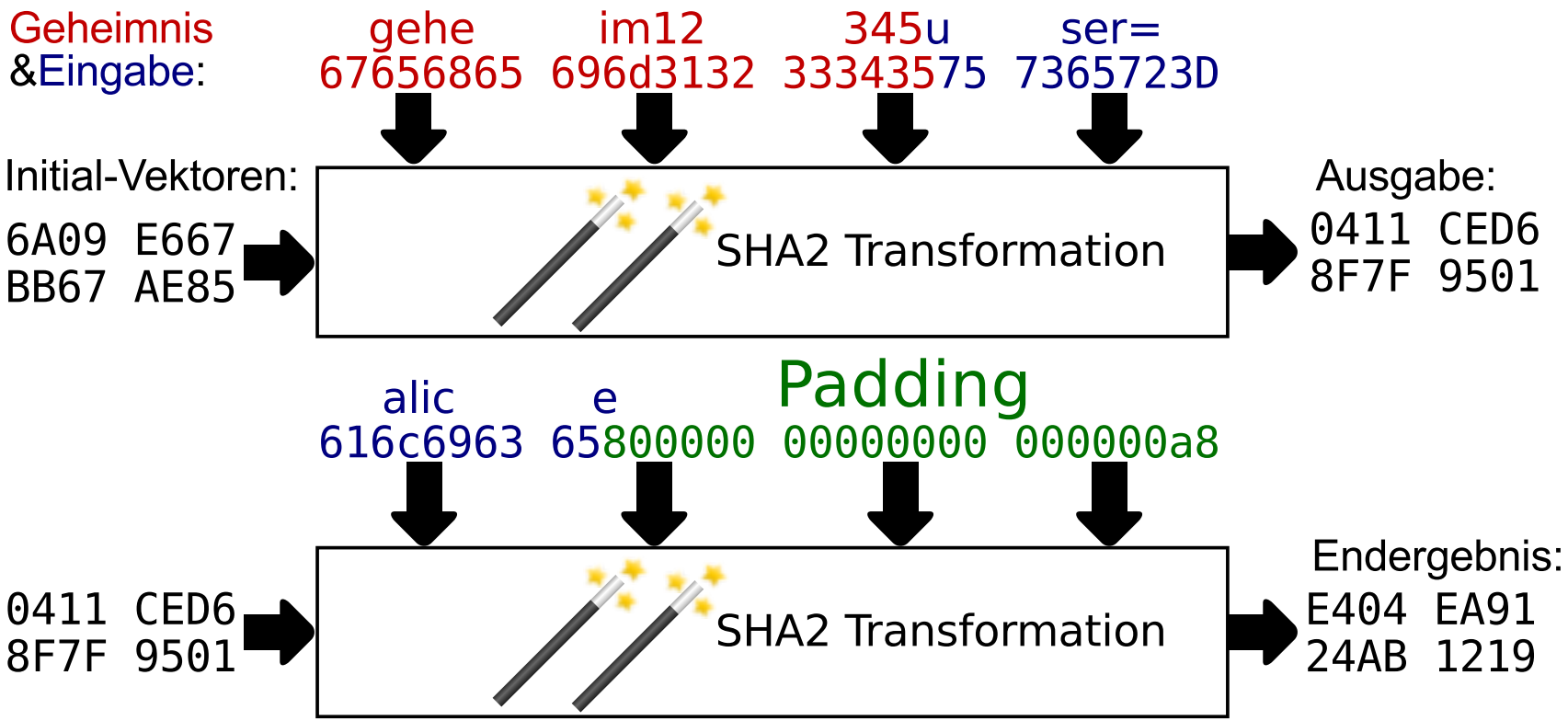
```
def getUser(secret, input_user, input_mac):
    if sha256(secret + input_user) == input_mac:
        return input_user
    else:
        return invalid
```

Ist das sicher?

- Sieht auf den ersten Blick richtig aus!
- Der Teufel steckt im Detail

Ex-Kurs: Konstruktion von Hashes

- Die meisten Hashfunktionen arbeiten blockweise
- Start mit wohldefinierten Werten
- Zustand nach dem letztem Block ist Ausgabe



Ex-Kurs: Padding

- Wenn die Länge der Nachricht kein Vielfaches von 64 Byte ist, brauchen wir Füllbytes (*padding*)
- Außerdem die Länge der Originalnachricht

Warum nicht einfach Nullen als Füllbytes?

Ex-Kurs: Padding

- Wenn die Länge der Nachricht kein Vielfaches von 64 Byte ist, brauchen wir Füllbytes (*padding*)
- Außerdem die Länge der Originalnachricht

Warum nicht einfach Nullen als Füllbytes?

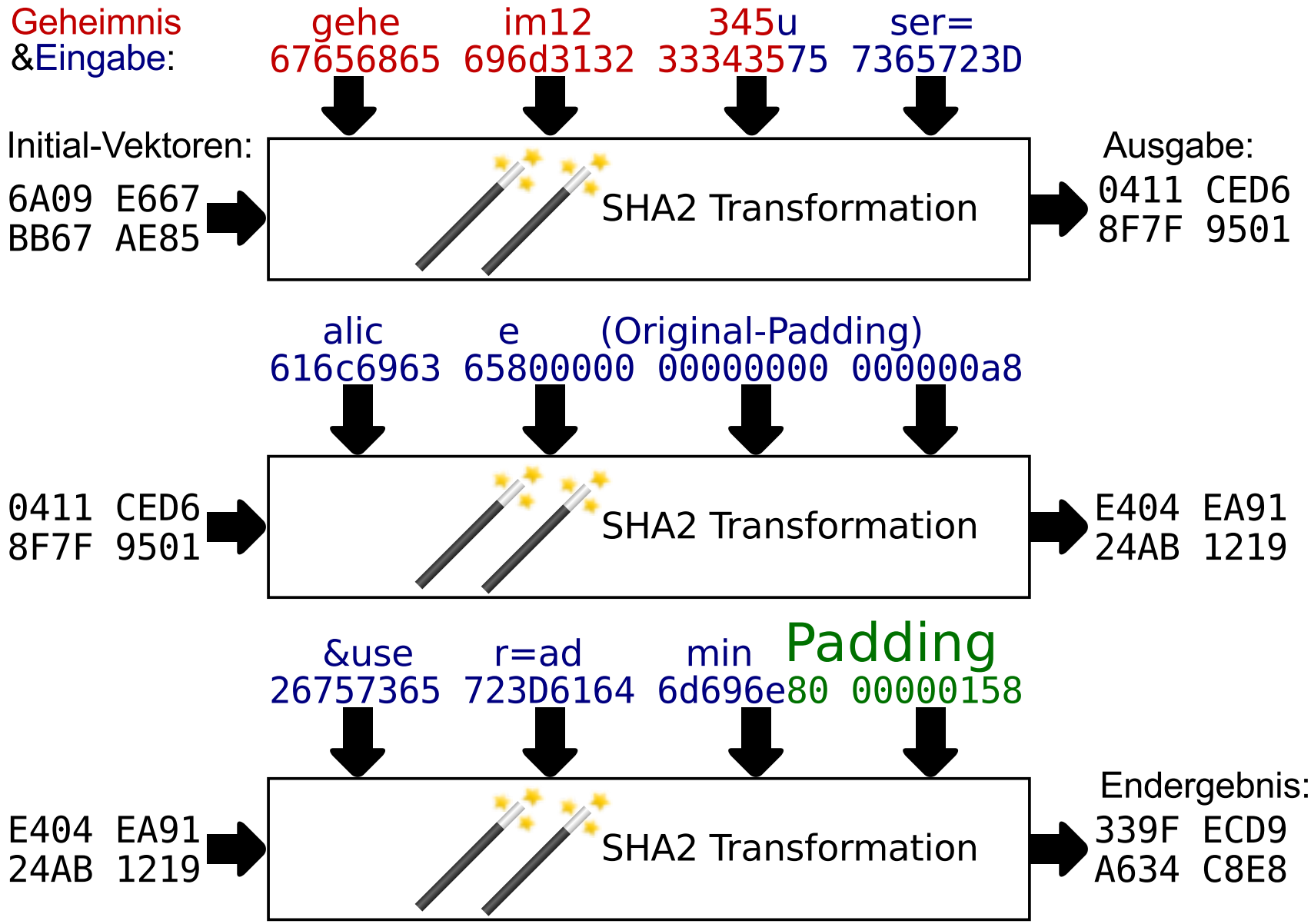
- Ohne Länge gäbe es Kollisionen!
  (da sonst SHA2-256(`AB CD`) == SHA2-256(`AB CD 00`))

- Bei SHA2-256:
  1. Ein Byte `0x80`
  2. n `0x00`-Bytes, so dass (Länge + 1 + 8 + n) mod 64 = 0
  3. Länge der Nachricht in Bit (8 Bytes, 64 bit big-endian)

Length Extension Attack

1. Fordere irgendeine valide Nachricht (z.B. user=alice) an.
2. Damit erhalten wir den Hash der Nachricht
   - = interner Zustand nach Padding
3. Angriff: Füge Padding zur Eingabe hinzu!
4. Und füge danach beliebige weitere Informationen hinzu
   - Z.B. `&user=admin`
5. Berechne dann den finalen Hash von der letzten Zwischenausgabe

Length Extension Attack: Konstruktion



Hausaufgabe: Message Length Extension Attack

Hausaufgabe

Bis Mittwoch, den 25. Juni 2014

Lösen Sie die vulnsrv-Aufgabe *Length Extension Attack*, indem Sie ein Programm `mac_extension` schreiben, dass die Length Extension Attack ausführt.

Beispiellösung: Length Extension Attack

Zuerst holt der Angreifer sich einen valide MAC für irgendeine Nachricht, z.B. indem er sich als Gast einloggt. In unserem Fall gelingt das durch einen POST-Request nach `/mac/login`.
Der Angreifer erhält nun eine valide MAC, wie z.B.
`765af7404861dce62ba7c682b33274a4d689269359080024`
`23c1c391c2a20803!user=Gast&time=1371129615`.

Weiterhin ist dem Angreifer die Länge des Geheimnisses bekannt – falls sie es nicht ist, kann er die folgende Attacke einfach für alle plausiblen Längen (z.B. 4 - 1024) ausprobieren.
In unserem Fall ist das Geheimnis 32 Bytes lang.

Beispiellösung: Length Extension Attack (2)

Nun berechnen wir zuerst das Padding der Originalnachricht. Da diese aus Geheimnis und Nachricht besteht, können wir die Gesamtlänge 32 + 25 = 57 Byte berechnen. Damit ergeben sich die letzten 8 Byte des Paddings als hexadezimale Repräsentation der Länge in Bit, also hier `00 00 00 00 00 00 01 C8`. Das erste Byte des Paddings ist immer `80`.

Zwischen erstem Byte und Länge werden nun so viele Null-Bytes eingefügt, so dass die Gesamtlänge genau ein Vielfaches von 512 Bit / 64 Byte – der Blocklänge von SHA2-256 – ist.
Das sind hier
$64 - ((57 + 1 + 8) \bmod 64) = 64 - (66 \bmod 64) = 64 - 2 = 62$
Byte.

Beispiellösung: Length Extension Attack (3)

Nun wollen wir den Zustand nach dem Padding reproduzieren, um später den endgültigen Hash zu berechnen. Am einfachsten können wir dafür einfach 32 beliebige Zeichen einfügen, damit der interne Zustand (also die Länge und der Fortschritt im gerade aktuellen Block) für die Hash-Bibliothek korrekt aussieht:

```
char* secret = malloc(32);
char* val = "user=Gast&time=1371129615";
SHA256_CTX ctx;
SHA256_Init(&ctx);
SHA256_Update(&ctx, secret, 32);
SHA256_Update(&ctx, val, strlen(val));
SHA256_Update(&ctx, padding, padding_size);
```

Der Zustand des Hash-Contexts ist jetzt natürlich nicht korrekt, da unser Geheimnis nicht korrekt war. Darum setzen wir ihn jetzt auf den gewünschten Hash 765af7404861dce62ba7c682b3... :

```
ctx.h[0] = 0x765af740;
ctx.h[1] = 0x4861dce6;
ctx.h[2] = 0x2ba7c682;
...
```

Beispiellösung: Length Extension Attack (4)

Nachdem der Zustand gesetzt ist, können wir den Hash nun mit dem anzufügenden Inhalt erweitern:

```
const char* inject = "&user=admin";
SHA256_Update(&ctx, inject, strlen(inject));
unsigned char hash[SHA256_DIGEST_LENGTH];
SHA256_Final(hash, &ctx);
```

Der finale MAC (in binärer Form in der Variable `hash`) kann nun für eine beliebige Aktion als Benutzer admin verwendet werden.

Length Extension Attack

- Fix: Verwende HMAC-Schema
- MAC wird dann berechnet als
  MAC = Hash(Key + Hash(Key + Message))
- Damit wird der interne Zustand nicht mehr preisgegeben

```
def getUser(secret, input_user, input_mac):
    if hmac(secret + input_user) == input_mac:
        return input_user
    else:
        return invalid
```

Ist das jetzt sicher?

Timing Attacks

```
def getUser(secret, input_user, input_mac):
    if hmac(secret + input_user) == input_mac:
        return input_user
    else:
        return invalid
```

- Achtung: Laufzeit von Stringvergleich hängt davon ab, welches Zeichen unterschiedlich ist!
- Mit sehr genauem Timing kann man herausfinden, *ab welchem Zeichen der angegebene Hash falsch ist*
- Das erlaubt lineare Suche nach dem richtigen MAC
- Eine sogenannte *Seitenkanalattacke*
- Abhilfe: Immer alle Zeichen vergleichen

Fazit:

- Vorsicht beim Einsatz von Kryptographie!
- Bestehende Funktionen verwenden!