Application Security (apsi)

Lecture at FHNW

Lecture 7, 2020

Arno Wagner, Michael Schläpfer, Rolf Wagner

<arno@wagner.name>, <{michael.schlaepfer, rolf.wagner}@fort-it.ch>

Agenda

- 12:15 – 13:00:
  - Introduction OWASP & WebGoat
  - CSRF & SOP
- 13:15 – 14:00: Guest Speaker Patrick Schaller
- 14:15 – 15:00: WebGoat Q&A Session 1

OWASP – Open Web Application Security Project

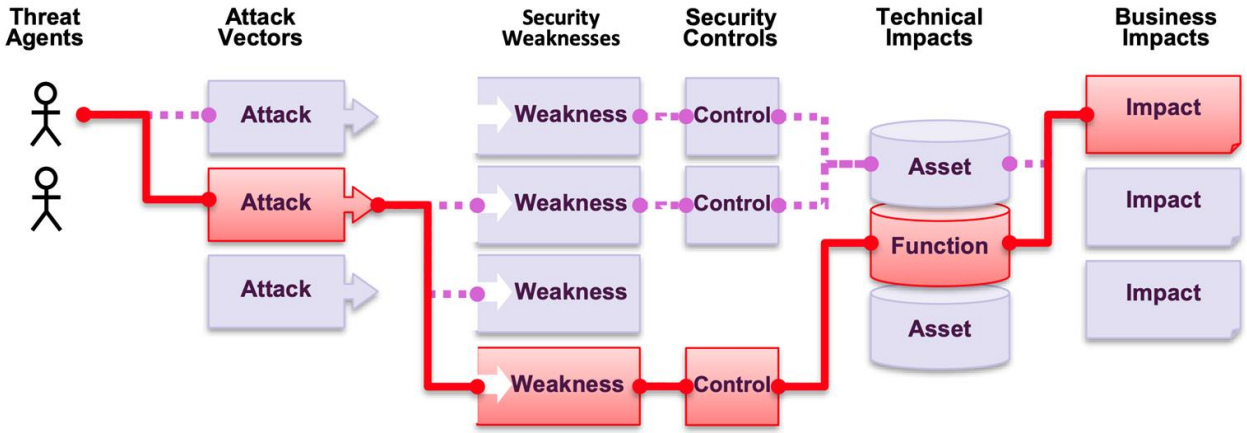
- Online community producing freely-available articles, methodologies, and tools for web application security
- Founded in 2001 (non-profit organization)
- 32’000+ Volunteers worldwide
- Numerous local chapters (e.g., Switzerland)
- Talks and conferences

Most prominent products:

- OWASP Top Ten: Most critical security risks to web applications
- OWASP Zed Attack Proxy (ZAP): Web App Scanner
- OWASP WebGoat: Insecure application (Learn to Hack)

OWASP Top Ten

- First edition In 2003, updated every couple of years (current version is 2017)
- Powerful und widely adopted awareness information for web application security (WAFs implement protection mechanisms based on this)
- Description and countermeasures for the most critical security risks
- Catalogue of the most critical web application security risks



OWASP Top Ten

- For years, same topics but changing rankings
- Large revision in 2017

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

WebGoat

https://owasp.org/www-project-webgoat/

Learn the hack - Stop the attack

WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components.

WebWolf the small helper

WebWolf is a separate web application which simulates an attackers machine:

- Host a file
- E-mail client
- Landing page for incoming requests

WebGoat

Learning in three steps:

- Explain the vulnerability
- Learn by doing
- Explain mitigation

We expect that you put ~8h effort into WebGoat.
Best Case: All Lessons ;-)
Minimal:

- A1 Injection (Intro & Advanced)
- A2 Broken Authentication
- A7 XSS

Cross Site Request Forgery (CSRF)

A malicious web site or email tricks (…) a user's web browser to call a trusted site when the user is authenticated

- Victim must be logged into the target site at attack time, as browser requests automatically include all (session) cookies
- Attack commands can come from a second site the victim visits at the same time

Example:

- A site offers help configuring a specific router model. It also contains the following 1x1 "image":

```
<img src=”http://192.168.1.1/admin/config/outsideInterface?nexthop=123.45.67.89”
alt=”pwned” height=”1” width=”1”/>
```

- If the router accepts this command (e.g. because of cookie-based session authentication), the router now sends all traffic through 123.43.67.89 and the attacker (that owns the IP) can listen to it.

CSRF Example 2

- The e-banking site www.bank.com has a CRSF vulnerability
- The victim is currently logged into bank.com
- An email arrives in the mail-client, that unfortunately also is a web-browser and automatically loads images
- The email contains
  `<img src="https://www.bank.com/transfer?amount=1000&destination=attacker">`
- What happens?
- The transfer request is sent to www.bank.com with all cookies for www.bank.com and from the IP address of the logged-in user!
- www.bank.com cannot distinguish this from a legitimate request

CSRF Countermeasures

- Verify Referer / Origin headers, as they are not writable by client-side JavaScript
  - Check, if Referer / Origin = target URL / Domain
- Use SameSite Cookie Attribute for session cookies
  - Defines whether to send cookies along with cross-site requests (none, lax, strict)
  - Lax: will be sent in GET request in top window navigations (a href, window.open())
- Use Synchronizer (CSRF) Tokens for each session or request
  - CSRF should be generated on service side
  - For every client request, the server checks the validity of the token (stateful)
  - CSRF tokens should not be transmitted using cookies or http get
    (but with headers, hidden fields)
- Require user interaction (e.g. individual sign-off on transactions)

Note: More possibilities exist and it make sense to combine them (e.g. SameSite & CSRF Token)

Same-Origin Policy (SOP)

*Under the Same-Origin Policy, a **web browser** permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same origin*

- Requires: Same protocol, host and port

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@**www.example.com**/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:**81**/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://**en.example.com**/dir/other.html	Failure	Different host
http://**example.com**/dir/other.html	Failure	Different host (exact match required)
http://**v2.www.example.com**/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:**80**/dir/other.html	Depends	Port explicit. Depends on implementation in browser.

Same Origin Policy (SOP) and its Limits

Sometimes, SOP is not enough resp. does prevent legitimate requests.

Example:

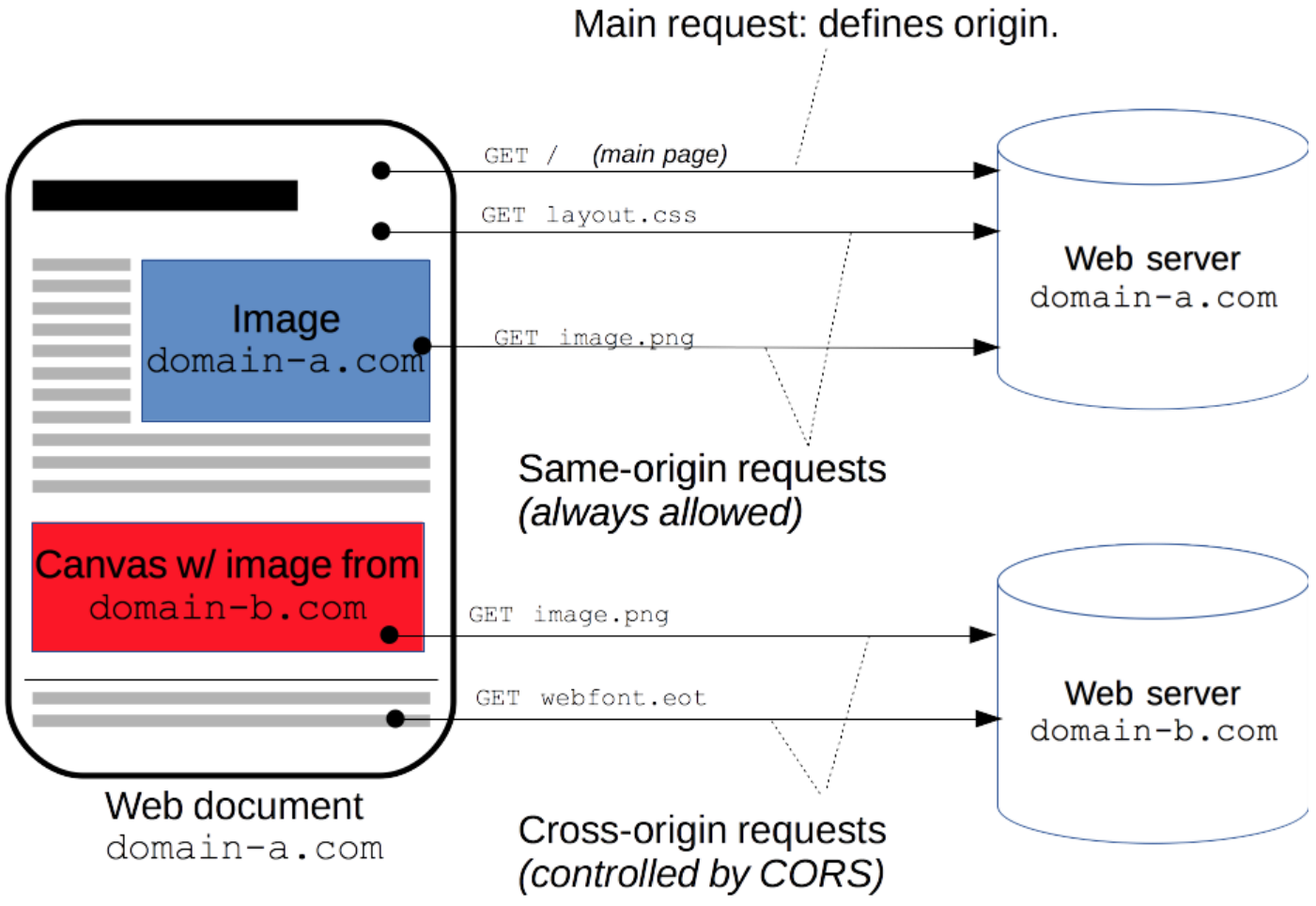
- A web-page may want to embed content from a different server, and request this via a script on client-side
- SOP prevents scripts from accessing a non same-origin page

Possible solution: **Cross-origin resource sharing (CORS)**

Original page is from server A and the request from script goes to B

1) Browser asks B via OPTION request for access, stating A as Origin
2) B allows it via "Access-Control-Allow-Origin: A"
3) Request to B is made

CORS Overview



Agenda

- 12:15 – 13:00:
  - Introduction OWASP & WebGoat
  - CSRF & CORS
- 13:15 – 14:00: Guest Speaker Patrick Schaller
- 14:15 – 15:00: WebGoat Q&A Session 1

Guest Speaker

Agenda

- 12:15 – 13:00:
  - Introduction OWASP & WebGoat
  - CSRF & CORS
- 13:15 – 14:00: Guest Speaker Patrick Schaller
- 14:15 – 15:00: WebGoat Q&A Session 1

WebGoat

The WebGoat exercise environment

In the following two lectures you will get hands-on experience using a vulnerable system (*WebGoat*). You will learn to hack the most common vulnerabilities with this respect.

Setup

The setup is pretty simple. The OWASP offers docker images for *WebGoat* and *WebWolf* (which you as an attacker will use for certain exercises).

Install docker-compose on the apsi-host (vagrant)

To start the environment, the OWASP offers a simple *docker-compose.yml* file, which you will in this folder. However, to use `docker-compose`, you will first have to install it. You will find a simple script in this folder which will do the job for you. First make the *install_docker-compose.yml* file executable:

```
vagrant@apsi-host:~/FHNW-apsi/Vorlesung/v07$ sudo chmod u+x install_docker-compose.sh
```

and then execute it:

```
vagrant@apsi-host:~/FHNW-apsi/Vorlesung/v07$ ./install_docker-compose.sh
```

After installation you may test whether or not `docker-compose` is correctly installed on the demo-machine:

```
vagrant@apsi-host:~/FHNW-apsi/Vorlesung/v07$ docker-compose -v
```

You should now see the version of your `docker-compose` installation.

Start WebGoat and WebWolf

Now, that you have `docker-compose` installed, you may just have to execute it with the given *docker-compose.yml* file:

```
vagrant@apsi-host:~/FHNW-apsi/Vorlesung/v07$ docker-compose up
```

Using WebGoat

After *WebGoat* and *WebWolf* are running properly, you may just use your favorite browser on your host system and point it to:

```
http://localhost:8080/WebGoat
```

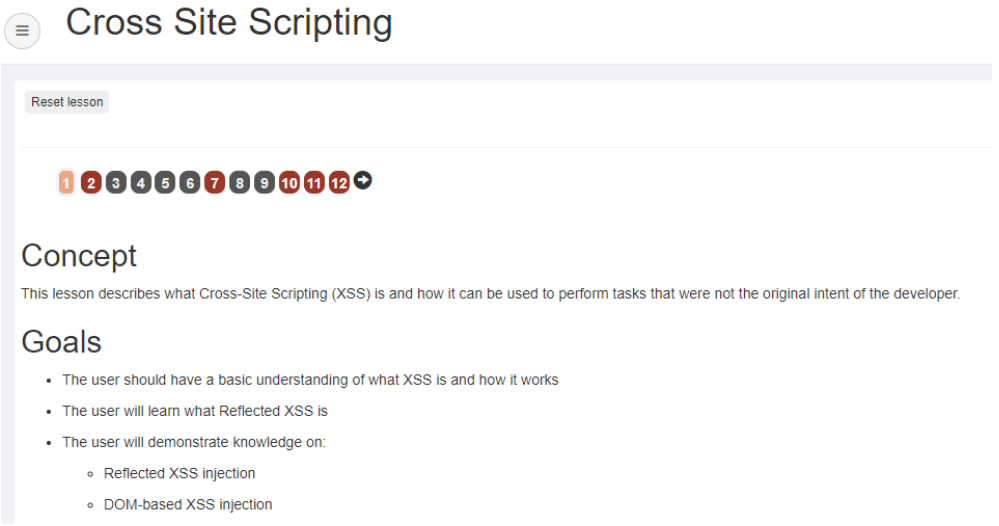
or

```
http://localhost:9090/WebWolf
```

respectively.

If you do so, just register a new user first. You will use these credentials further on to authenticate yourself.

WebGoat



Let's try XSS

XSS – Cross Site Scripting

- Injection attack and listed as 7th out of top 10 vulnerabilities identified by OWASP in 2017.
- Cross site scripting is the method where the attacker injects malicious script into trusted website.
- Types
  - Stored XSS: User inputs (malicious scripts) stored in websites (databases) and displayed to other users
    => Comment section
  - Reflected XSS: URLs with malicious scripts which are directly displayed in web pages to the user (which clicked the URL)
    => Query URL
  - DOM based XSS: URLs with malicious scripts which are directly displayed in web pages to the user (which clicked the URL), with difference that DOM based XSS doesn't even go to
    => URI fragment: https://en.wikipedia.org/wiki/URI_fragment