Security Implications of 3rd Party Resources in WWW

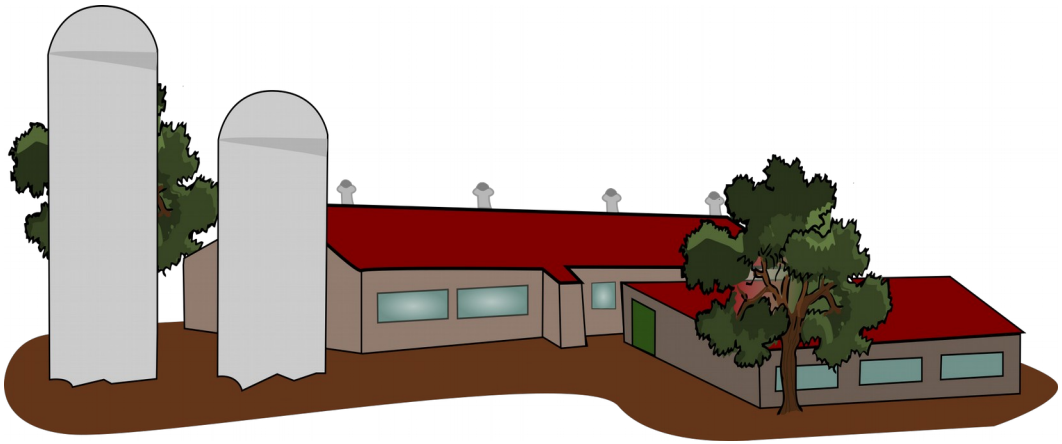
Kārlis Podiņš

Outline

- Cross-site scripting (XSS) attack on web 2.0
- Defeats active content blockers (e.g. NoScript)
  - because of use of external resources in web pages
- Large-scale scanning to examine use of Content Security Policy in web pages

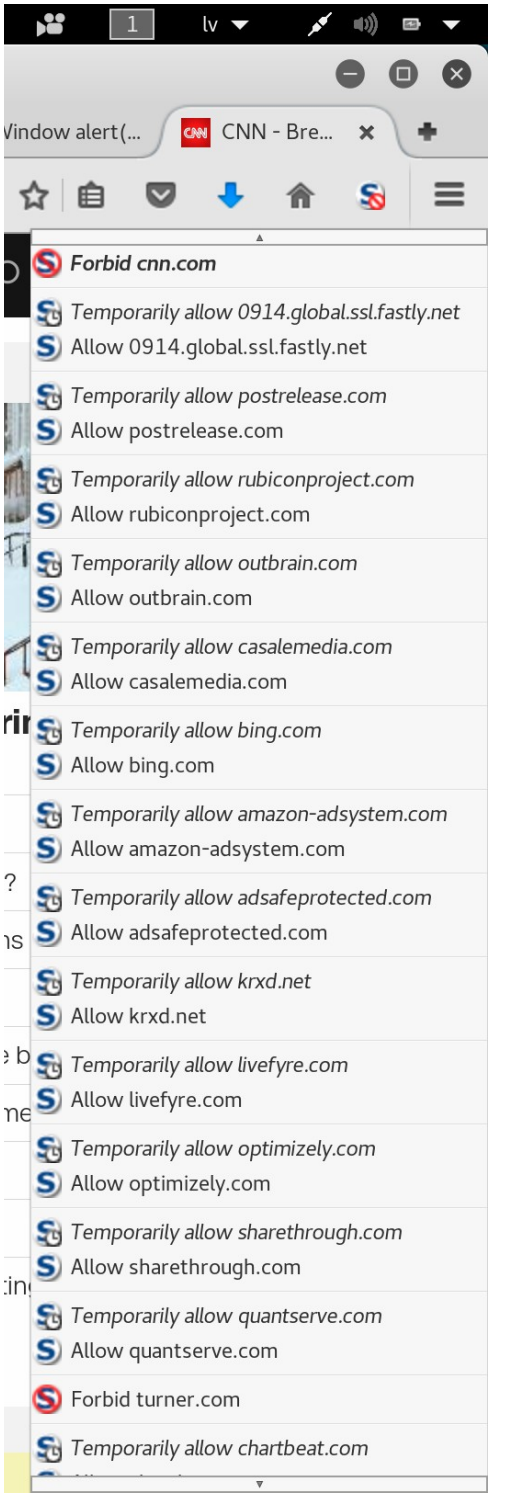
Method

- Review known phenomena in new circumstances
- Reasoning
  - Security often silo’ed
  - Study interaction for real-life performance



External content in WWW

- You don’t load a webpage, you load the internet
- >90% of TOP 1M pages use external content
  - When visiting example.com:
    - Internal = something.example.com
    - External = awesomecdn.lol
- Breakup by [Kumar]
  - Tracking 75.4%,
  - CDN 65.2%,
  - API 39%,
  - Advertising 42.2%,
  - Social media 39.7%



Old Teaching

```
From:      gmail
Subject:   change your password
Body:      Somebody has your password, change
it ASAP at google-security.com
```

- For 10+ years:
  - Updates
  - Check what you click
    - `google-something.com` is bad
  - Use NoScript
- Users slowly starting to get it

New Teaching

- fbcdn.com
- ssl-images-amazon.com
- akamaihd.net
- delphi.lv
- itvnet.lv
- ...

Father of all* cyber attacks

Cross Site Scripting (XSS)

- Data interpreted as code
  - Von Neumann architecture
- Subclass of code injection attacks
  - HTML injection
- Enter comment:
  - <script>alert(“Pwned”)</script>

XSS

- P - a benign web page
  - vulnerable to XSS
- P contains user input
  - in backend database (stored XSS)
  - or volatile - stored in URL only (transient XSS)
- User input contains JavaScript
- Best practice defence against XSS is web server output sanitizing[owasp]
  - all user-supplied input could be validated before storing

XSS – direct vs indirect

- Direct attack
  - Attacker’s script can be reliably stored and retrieved from backend database
  - = full control
    - exploitation of external resources unnecessary
  - Visiting page P equals to visiting a rogue page
  - Challanges (from attacker viewpoint)
    - storage limit insufficient
    - all attack scripts on victim’s server = faster incident response
- Indirect attack
  - To overcome challanges, attackers usually store their scripts on external resources

XSS vs NoScript

- NoScript – browser plugin for restricting domains for active content
- P - legacy web page
  - NoScript reliably helps
  - User enables scripts from domain P
    - Simple dynamic content, in-site navigation, search etc
    - Allows initial attacker's scripts to execute
      - Attacker's secondary scripts stored on external domains blocked
        - Unless user allows scripts from `evil.com`
- P - typical modern web page
  - Heavy use of external resources
  - Some external resources are required for basic functionality
    - User has to allow execution of scripts from some domains
  - Black-box model
    - No clear naming policy
    - Attacker can register any available domain name
      - Race condition
      - User allows domains in random pattern
        - User likely to allow to enable execution of attacker's scripts

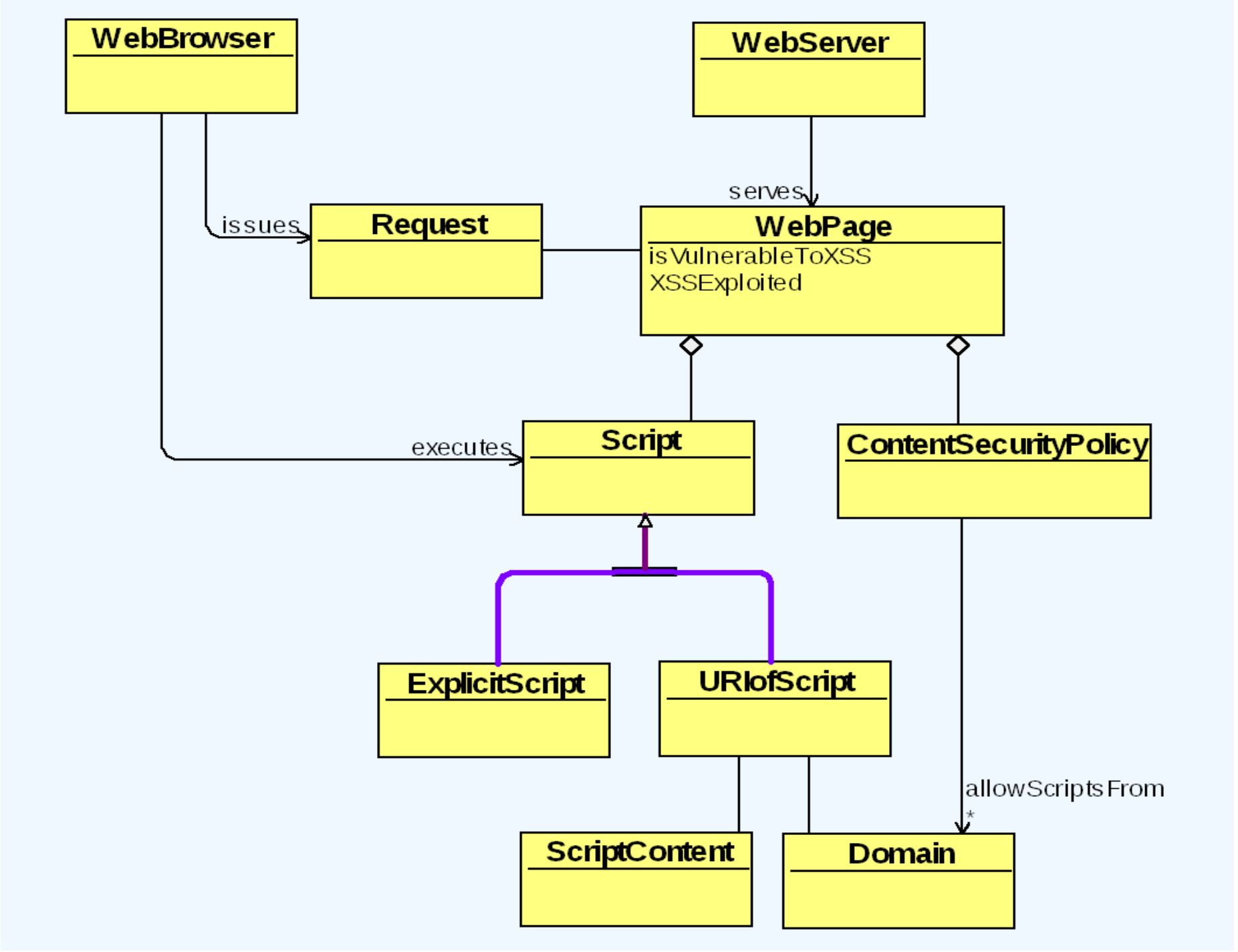
NoScript vs External Resources

- P uses resources from commercial 3rd party C
  - attacker can purchase service from C too
    - Bitcoin
    - Stolen CC
    - Demo period
  - Depending on policy used by C, legitimate resources of P and malicious resources are not easily distinguishable by either user or security software
- Service provider policy
  - Subdomains
    - N2435PORIUaASOPI.awesomecdn.lol
  - Path
    - awesomecdn.lol/po2i43r5a0ou2

Content Security Policy (CSP)

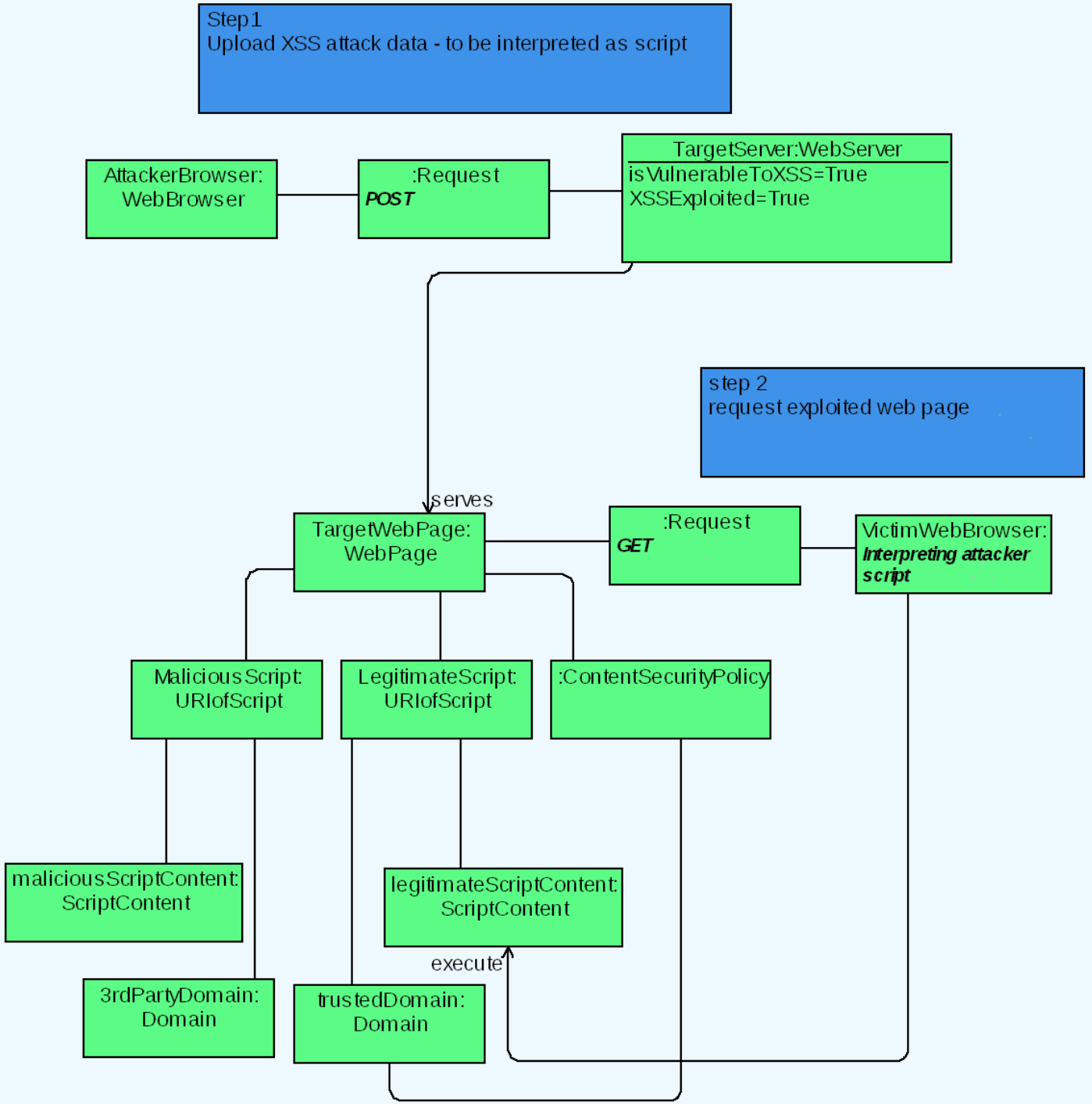
- Mitigate XSS
- Explicitly defined in HTTP header
- Report functionality!
- Cannot restrict path
  - `awesomecdn.lol/naou21rass`
- Useless CSP
  - `*.awesomecdn.lol`

Model

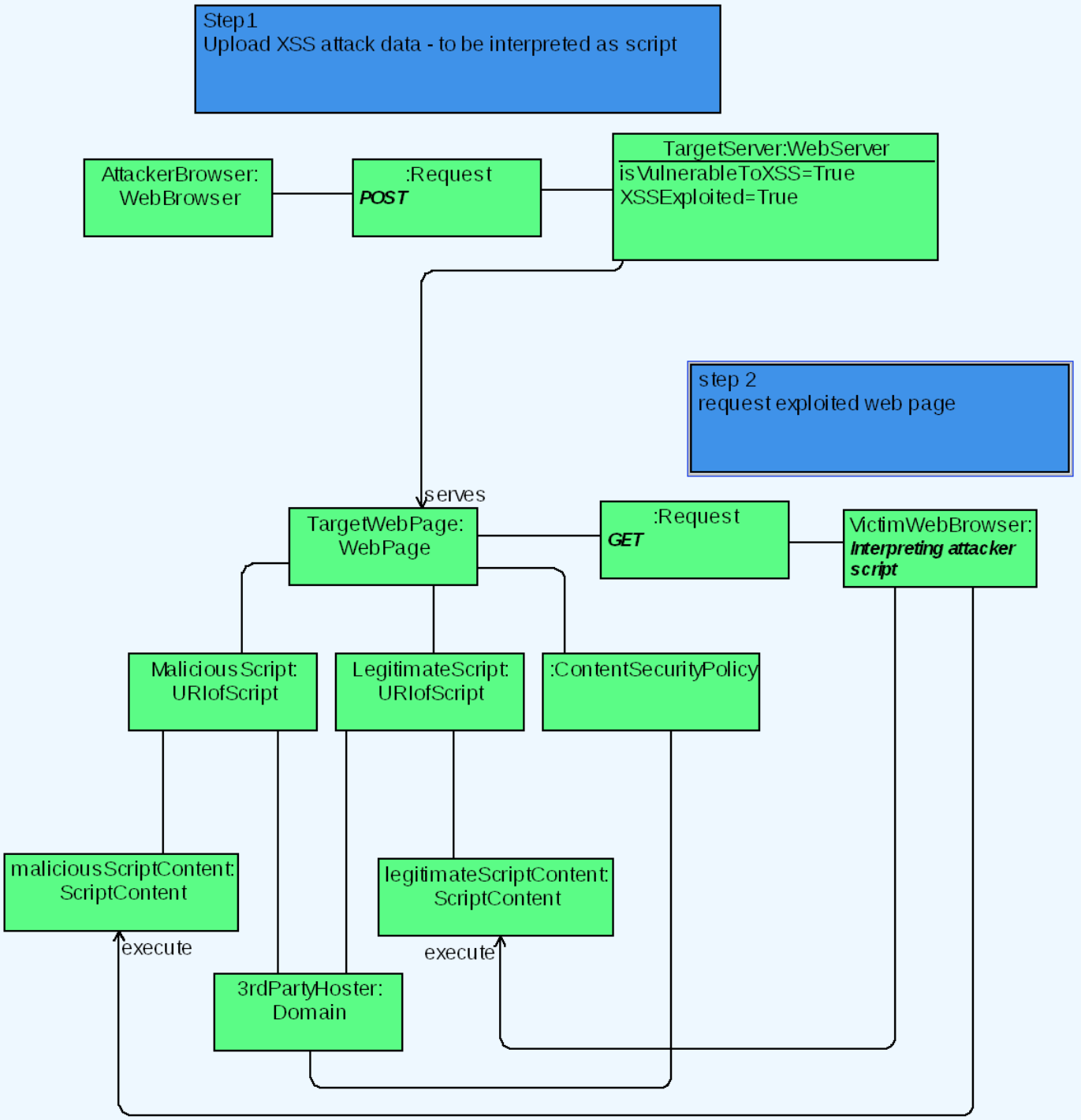


Script executed only if $\texttt{Script} \circ \texttt{URIofScript} \circ \texttt{Domain} \in \texttt{webPage} \circ \texttt{CSP} \circ \texttt{domain}$
Or explicit script

Good CSP


Step1
Upload XSS attack data - to be interpreted as script
AttackerBrowser:
WebBrowser
:Request
POST
TargetServer:WebServer
isVulnerableToXSS=True
XSSExploited=True
step 2
request exploited web page
serves
TargetWebPage:
WebPage
:Request
GET
VictimWebBrowser:
Interpreting attacker script
MaliciousScript:
URIofScript
LegitimateScript:
URIofScript
:ContentSecurityPolicy
maliciousScriptContent:
ScriptContent
legitimateScriptContent:
ScriptContent
execute
3rdPartyDomain:
Domain
trustedDomain:
Domain

Bad CSP


Step1
Upload XSS attack data - to be interpreted as script
AttackerBrowser:
WebBrowser
:Request
POST
TargetServer:WebServer
isVulnerableToXSS=True
XSSExploited=True
step 2
request exploited web page
serves
TargetWebPage:
WebPage
:Request
GET
VictimWebBrowser:
Interpreting attacker
script
MaliciousScript:
URIofScript
LegitimateScript:
URIofScript
:ContentSecurityPolicy
maliciousScriptContent:
ScriptContent
legitimateScriptContent:
ScriptContent
execute
execute
3rdPartyHoster:
Domain

Large-scale examination of CSP

- Dataset – Alexa Top 1M
- 98% don’t give a damn
- 4% of CSPs vulnerable
  - 1/25 fail
  - ~ 1/1000 of total

Results

amazonaws.com	310
cloudfront.net	200
akamaihd.net	152
s3.amazonaws.com	21
githubusercontent.com	31
rackcdn.com	16
edgecastcdn.net	11
kxcdn.com	11
akamaized.net	10
edgesuite.net	8

Take-away

- Revisit advice given to users
  - Circumstances change
- Review naming policies
  - Single domain possible – e.g. youtube
    - TODO – youtube is single purpose cdn, special case
- Script-free fallback functionality
  - Breaks income model
    - TODO unintended consequences?
      - income model --> insecure users
- CSP implementation – careful examination