11.1 SESSION HIJACKING CONCEPTS

- Hijacking Basics
- Hijacking Web Sessions
- Token Examples
- Why Hijacking is Successful
- Hijacking vs Spoofing

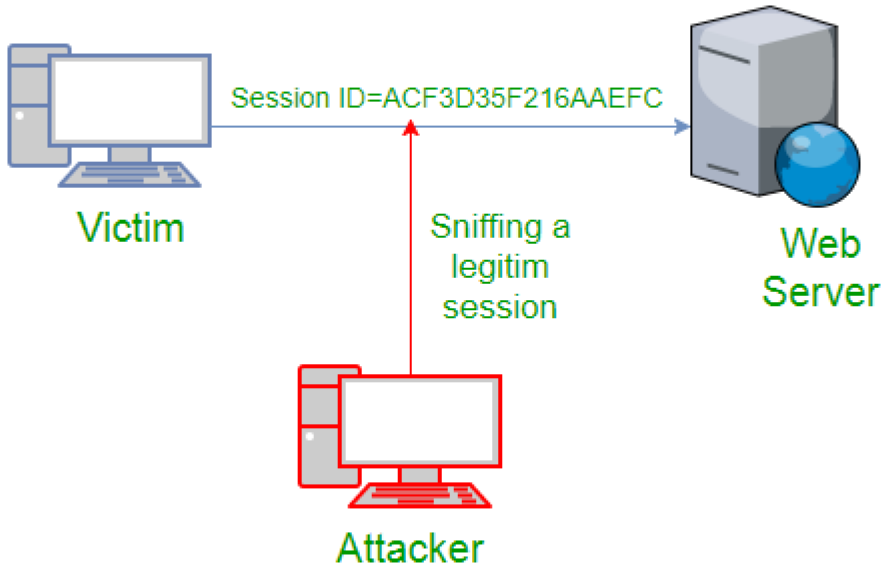
WHAT IS SESSION HIJACKING?

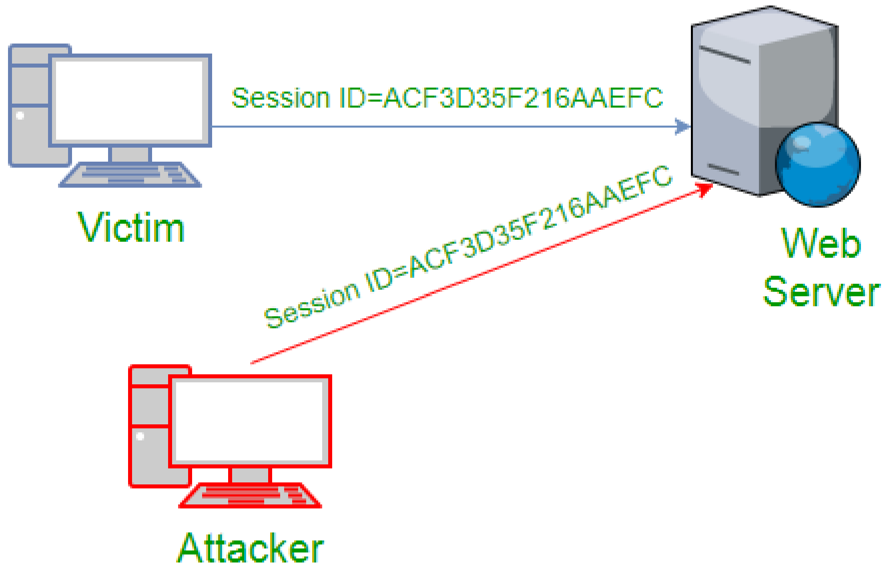
- The act of "taking over" someone else's session after they have established it
- Usually aimed at web browsers
- Can sometimes be done at the network level
- The server does not realize that someone else is masquerading as the client
- The victim (user) also may not realize their session has been hijacked
  - The attacker and victim might be running parallel sessions
  - The server would see this as two sessions by the same client

WEB SESSION HIJACKING BASICS

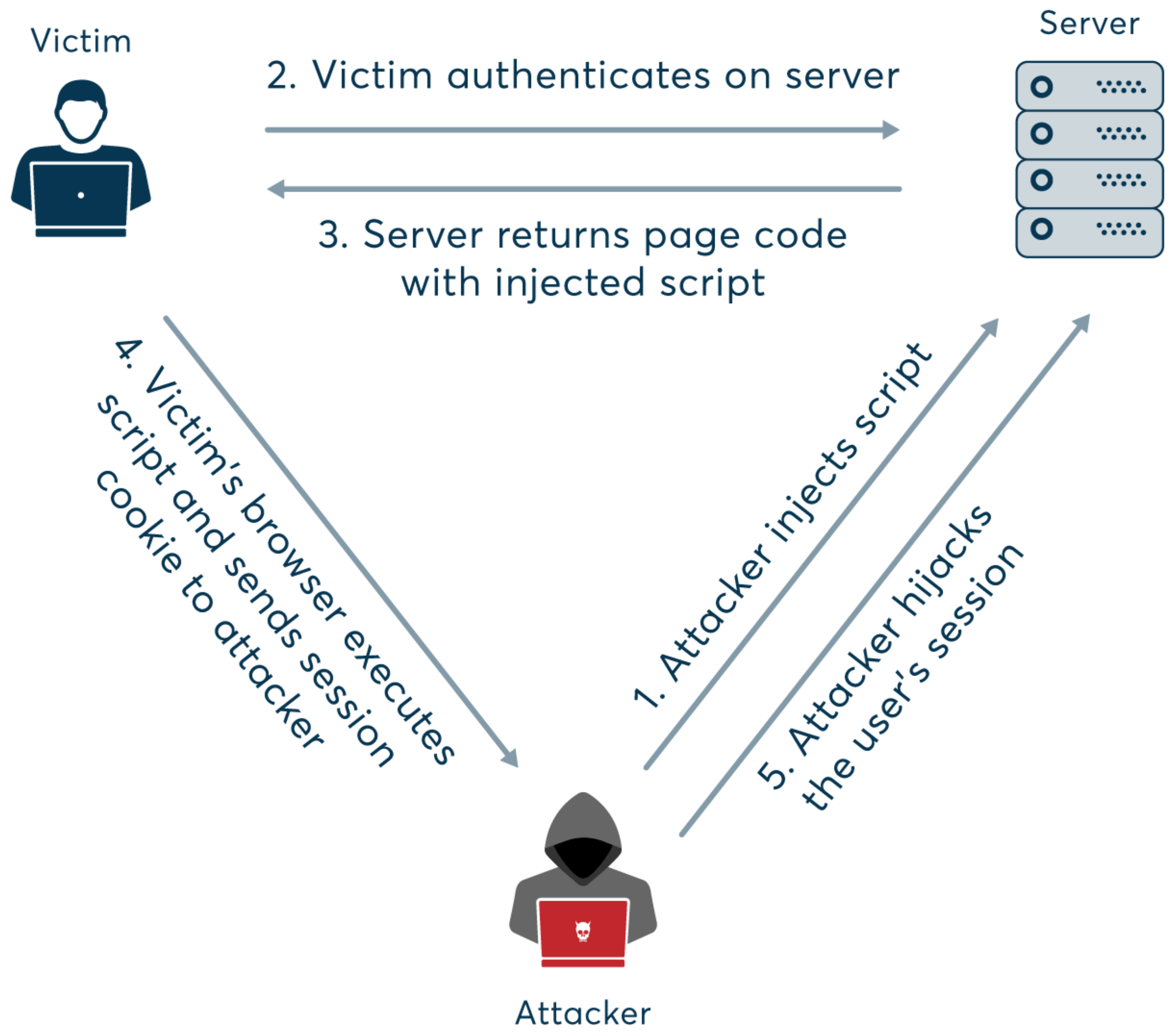
1. After a web client authenticates, the web server sends a session identification token (key) to the client
2. The attacker steals the token or successfully guesses its value
3. The attacker then takes over the connection while the session is in progress
4. The attacker does not need to authenticate again
   - Authentication usually happens only at the beginning of the session
   - The attacker only needs to present the token to the server to show they already authenticated

WEB SESSION HIJACKING EXAMPLE #1





WEB SESSION HIJACKING EXAMPLE #2


Victim
Server
2. Victim authenticates on server
3. Server returns page code with injected script
4. Victim's browser executes script and sends session cookie to attacker
1. Attacker injects script
5. Attacker hijacks the user's session
Attacker

HIJACKING WEB SESSIONS

- A Session Hijacking attack compromises the client's session token
  - Steal or predict a valid session token
  - Gain unauthorized access to the Web Server
- HTTP communication uses many different TCP connections, the web server needs a method to recognize every user's connections.
- The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication.
- A session token is normally composed of a string of variable width and it could be used in different ways
- For example:
  - In a URL, the token is a cookie included in the the header or body of an HTTP request
  - It could also be a JSON Web Token (JWT)

TOKEN EXAMPLES

- URLs with embedded Cookie

```
http://www.example.com/PHPSESSID=298zf09hf012fh2
http://www.example.com/userid=sup3r4n0m-us3r-1d3nt1f13r
```

- JSON Web Token

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.
eyJzdWIiOiJ1c2Vycy9Uek1Vb2NNRjRwIiwibmFtZSI6IlJvYmVydCBUb2tlbiBNYW4iLCJz
Y29wZSI6InNlbGYgZ3JvdXBzL2FkbWlucyIsImV4cCI6IjEzMDA4MTkzODAifQ
.
1pVOLQduFWW3muii1LExVBt2TK1-MdRI4QjhKryaDwc
```

WHY SESSION HIJACKING IS SUCCESSFUL

- Lack of account lockout for invalid session IDs
- Session expiration time is indefinite
- Session IDs are small or the ID generation algorithm is weak
- Vulnerability of most TCP/IP computers
- Session IDs handled insecurely
- Majority of countermeasures require encryption to work

HIJACKING VS. SPOOFING

- Hijacking
  - Process of taking over active session
  - Needs legitimate user to make/authenticate connection
- Spoofing
  - Process of initiating new session using stolen credentials
  - Attacker pretends to be a user/machine to gain access

11.2 COMPROMISING A SESSION TOKEN

- Cookie-based Authentication
- Token-based Authentication
- JWT
- Stealing a Token

WHAT IS COOKIE-BASED AUTHENTICATION?

- The traditional, **stateful** web authentication mechanism
- Provides "proof" to the website that the user has already been authenticated
  - The website can "trust" a browser that presents the cookie
- Lifetime of a cookie:
  1. User enters their login credentials
  2. Server verifies the credentials are correct and creates a session which is then stored in a database
  3. A cookie (text file) with the session ID is placed in the user's browser
  4. On subsequent requests, the session ID is verified against the database and if valid the request processed
  5. Once a user logs out of the app, the session is destroyed both client-side and server-side

WHAT IS TOKEN-BASED AUTHENTICATION?

- A token is (usually) a JSON Web Token (JWT)
  - Digitally signed JSON object (key/value pair)
  - Can be Base-64 encoded
- Token-based authentication is **stateless**
  - The server does not keep a record of which users are logged in
  - Does not keep track of which JWTs have been issued
  - Every request to the server is accompanied by the token which the server uses to verify the authenticity of the request
- Token-based authentication has gained prevalence over the last few years due to the rise of:
  - single page applications
  - web APIs
  - Internet of Things (IoT)

EXAMPLE OF JWT

JWT Format uses key-value pairs

```
HEADER.PAYLOAD.SIGNATURE
```

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

```
{
    "sub": "123456789",
    "name": "Moo",
    "admin": true
}
```

```
{   7WK5T79u5mIzjIXXi2oI9
    Fglmgivv7RAJ7izyj9tUyQ
}
```

Whole thing is then serialized (put in one line) and typically Base64 encoded:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyAKCSJzdWIiOiAiMTIzNDU2Nzg5IiwgCgkibmFtZSI6ICJNb28iLCAKCSJhZG1pbiI6IHRydWUgCn0=.
7WK5T79u5mIzjIXXi2oI9Fglmgivv7RAJ7izyj9tUyQ
```

TOKEN LIFETIME

1. User enters their login credentials.
2. Server verifies the credentials are correct and returns a signed token.
3. This token is stored client-side:
   - most commonly in local storage.
   - but can be stored in session storage or a cookie as well.
4. Subsequent requests to the server include this token as an additional Authorization header
   - or through one of the other methods mentioned above.
5. The server decodes the JWT and if the token is valid processes the request.
6. Once a user logs out:
   - the token is destroyed client-side.
   - no interaction with the server is necessary.

HOW IS A SESSION TOKEN USED?

- Because HTTP communication uses many different TCP connections, the web server needs a method to recognize every user's connections
- The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication
- The token is used in different ways:
  - In the URL
  - In the header of the http requisition as a cookie
  - In other parts of the header of the http request
  - In the body of the http requisition.
- A Session Hijacking attack compromises the session token
  - By stealing or predicting a valid session token
  - This gains unauthorized access to the Web Server

WAYS TO OBTAIN A SESSION TOKEN

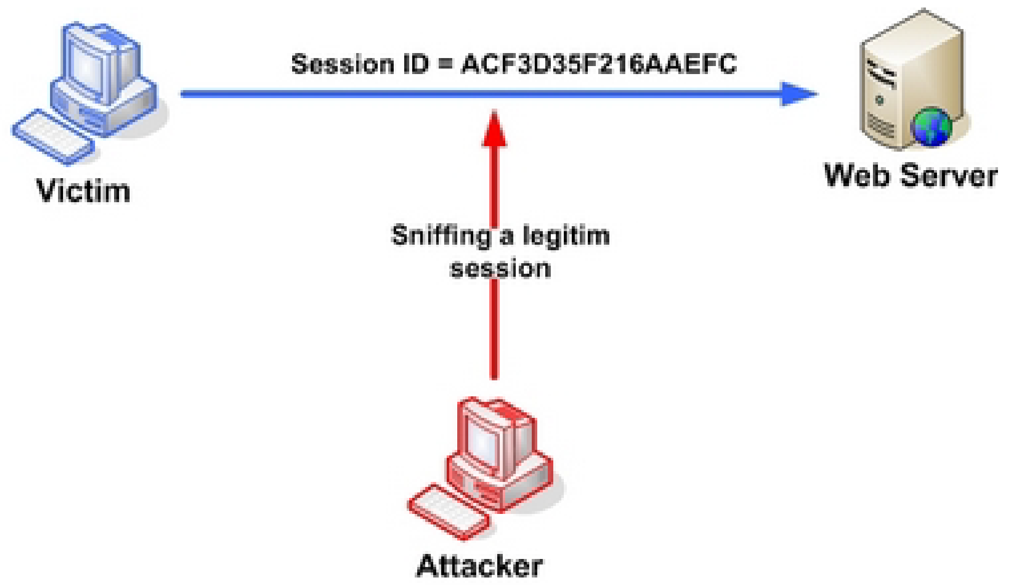
- Stealing
  - Attacker steals session IDs using various techniques
    - Sniffing
    - XSS
    - Malicious site
- Guessing
  - Attacker looks at variable parts of session IDs to try to guess what they are
- Brute Force
  - Attacker keeps trying different session IDs until the right one is found

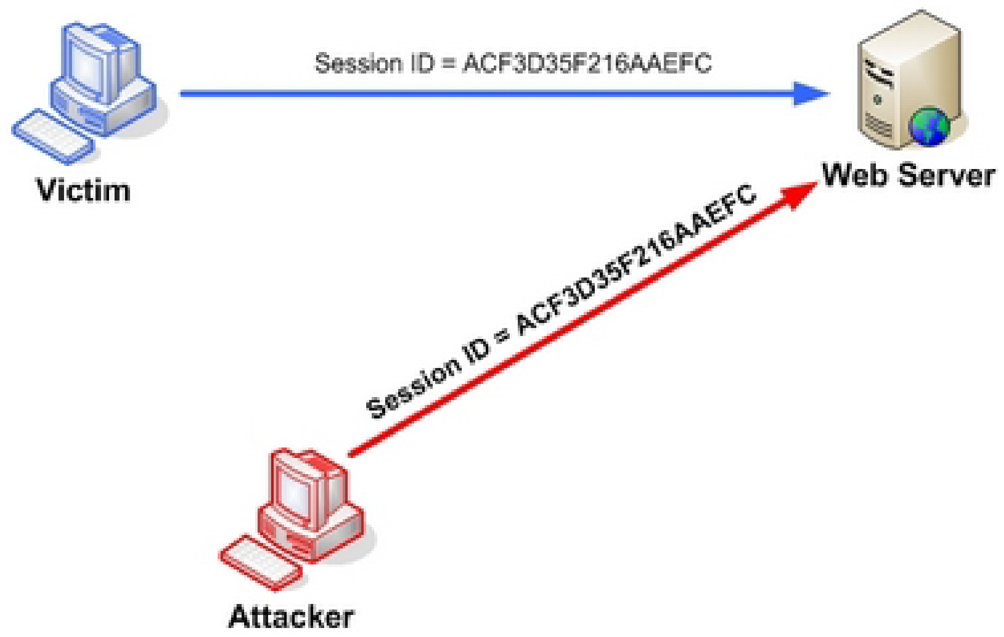
SESSION HIJACKING METHODS

- Command Injection
  - Attacker injects malicious code into the target server
- Session ID Prediction
  - Attacker takes over the session
- Session Desynchronization
  - Attacker breaks the connection with target machine
- Monitoring
  - Attacker watches the TCP segment flow and predicts the TCP sequence number
- Sniffing
  - Attacker intercepts a token
- Attacker gains access to a machine that still has an active session
  - User has stepped away
  - Access is via RAT
  - Session has no logout or expiration time

SESSION SNIFFING

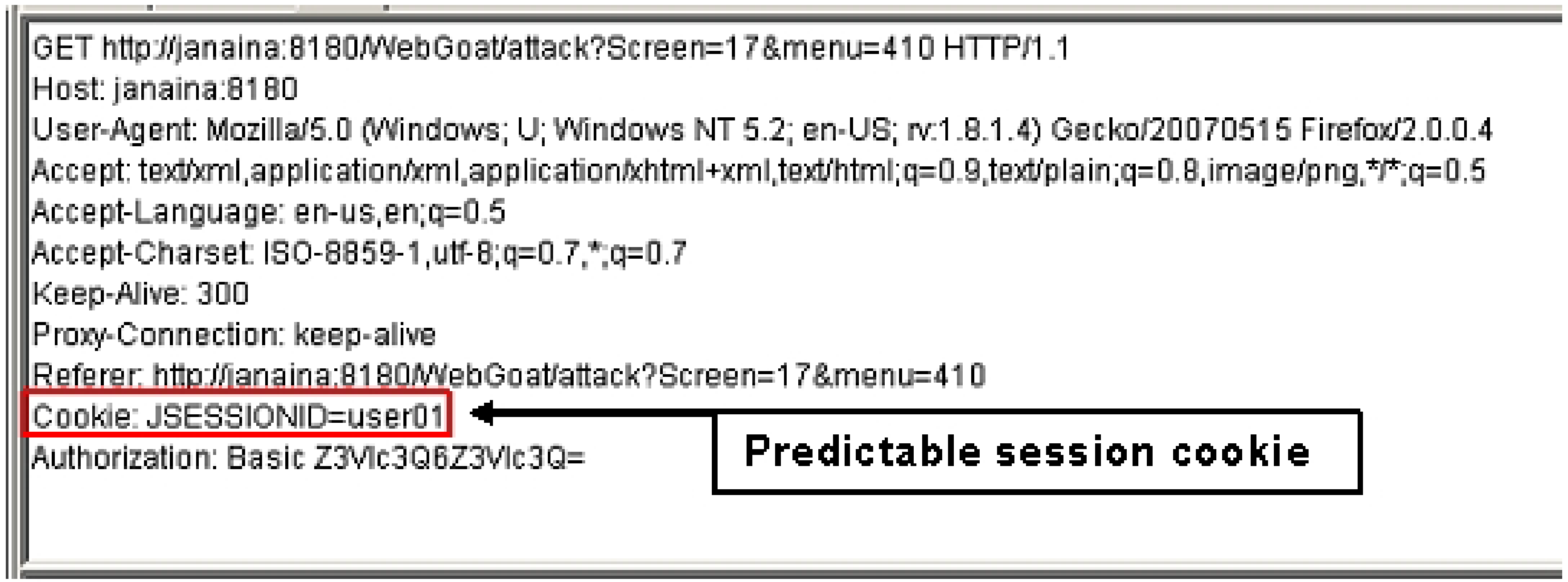
- AKA side-jacking
- Use a sniffer (Wireshark, Kismet) to capture a valid session token (Session ID)
- Reuse the token to gain unauthorized access





SESSION PREDICTION

- AKA predictable session token
- Attacker analyzes the website's session ID generation process
- Attacker then predicts a valid session ID value and gets access
- Looking at the example, "user02" would be a good prediction



```
GET http://janaina:8180/WebGoat/attack?Screen=17&menu=410 HTTP/1.1
Host: janaina:8180
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://janaina:8180/WebGoat/attack?Screen=17&menu=410
Cookie: JSESSIONID=user01
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
```

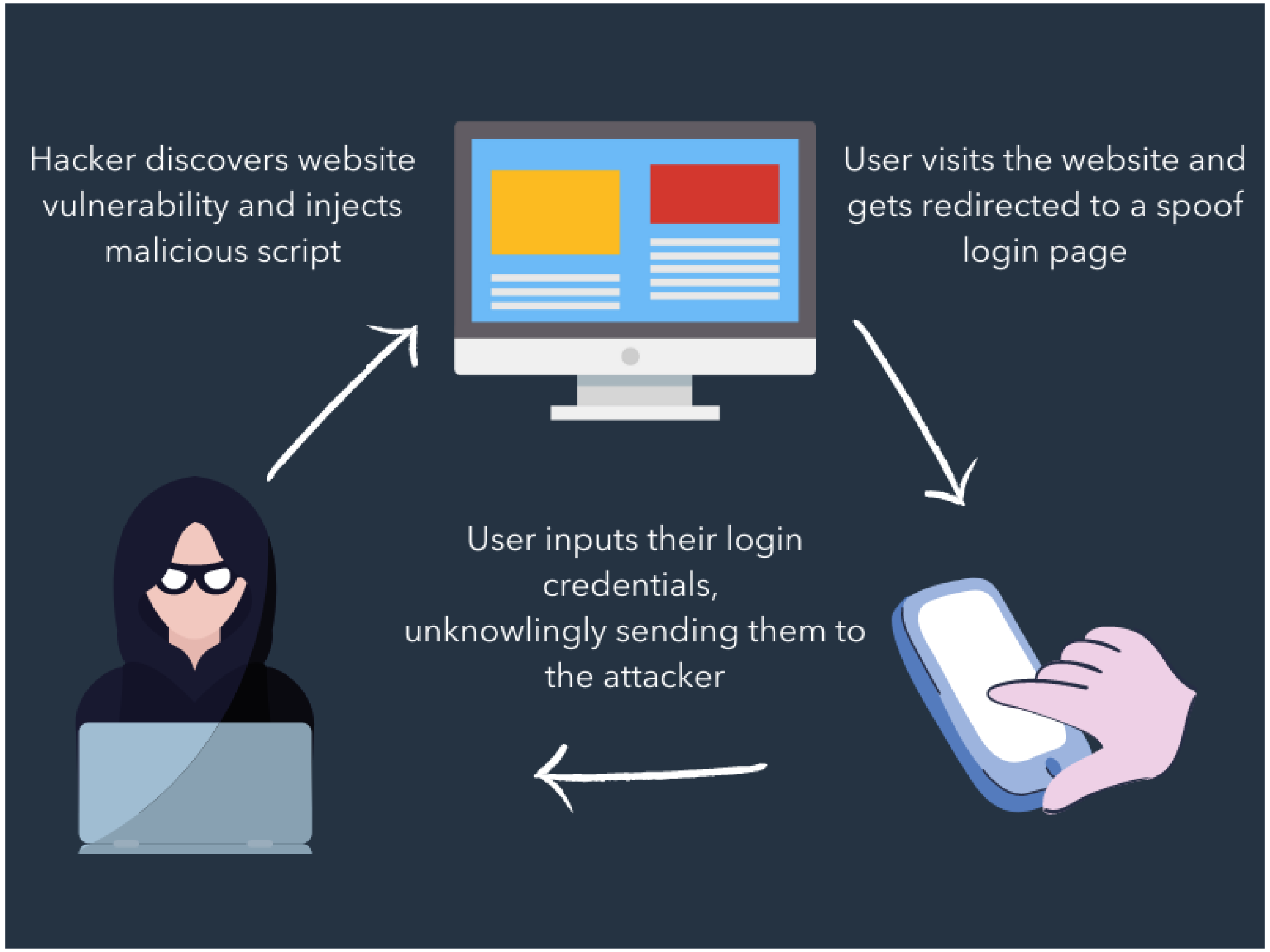
11.3 XSS

- Stored XSS
- Reflected XSS
- DOM XSS

WHAT IS CROSS-SITE SCRIPTING? (XSS)

- A popular and effective attack type
- Takes advantage of a client's trust in a server
- Malicious code (typically JavaScript) is inserted into a web page
- While the victim views the page, their browser executes the malicious code in the background
- Made possible when the web app does not validate user input
- Requires some level of social engineering
- Can compromise a session by stealing its cookie

XSS ATTACK FLOW


Hacker discovers website vulnerability and injects malicious script
User visits the website and gets redirected to a spoof login page
User inputs their login credentials, unknowingly sending them to the attacker

JAVASCRIPT

- JavaScript is a programming language which runs on web pages inside your browser
  - It is included in the HTML that forms the web page you see
  - It adds functionality and interactivity to the web page
  - It's used extensively on all major applications and CMS platforms.
- Unlike server-side languages such as PHP, JavaScript code inside your browser cannot impact the website for other visitors
  - It only performs actions within the user's browser
- While JavaScript is client-side and does not run on the server, it can be used to interact with the server by performing background requests
  - The web page does not display the script to the user
  - It is executed silently in the background

JAVASCRIPT AND XSS

- An attacker exploits a vulnerability on the website’s software
- They inject their own script which is executed by the victim’s browser
- The script can steal the session cookie/token, entice the user with phishing, or perform unwanted actions in the user’s name

CROSS-SITE SCRIPTING TYPES

- Stored XSS
- Reflected XSS
- DOM XSS

Note: You will learn more about compromising web apps in the CEH Module "Hacking Web Applications"

STORED XSS

- AKA Persistent XSS
- Malicious code is stored permanently in a website database
- Victims *later* unknowingly run that code
- An attacker injects a malicious script onto the site that others will read
  - The attacker posts to a forum, product review, or some other feedback page
  - The "legitimate" posting contains a hidden malicious script (easy to do in HTML)
  - As visitors read the posting, their browser is also running the malicious script in the background
  - JavaScript `document.cookie` property in the script is a common way to capture a cookie

Requires some social engineering to entice victims to read the attacker's post

STORED XSS EXAMPLE

- A vulnerable web app allows visitors to post a comment:

```
POST /post/comment HTTP/1.1
Host: vulnerable-website.com
Content-Length: 100
postId=3&comment=Love+this+product!&name=Moo+Dharma&email=moo%40example.com
```

- Visiting users will see the following post:

```
Love this product!
```

STORED XSS EXAMPLE (CONT'D)

- An attacker posts this comment:

<p>Yeah it's great<script>*some bad command here*</script></p>

- Visiting users will see the following post:

```
Yeah it's great
```

- At the same time the bad command is running in the background

REFLECTED XSS

- AKA Non-persistent XSS
- An attack in which a web app receives data in an HTTP request
  - Includes that data within the immediate response in an unsafe way
- Malicious code is run in the victim's browser *in the context of their current session* with the website
  - If you want to steal the user's cookie, you need them to first obtain a cookie
  - While they still have that session going, you need the same browser (in the same session) to execute the malicious code
  - If you simply send them a malicious link, it will open another instance of the browser, which will be in a different session

REFLECTED XSS (CONT'D)

- The injected script is reflected off the webserver as part of:
  - An error message
  - A search result
  - Any other server response
- The attacker must search the web app for any place where user input directly triggers a response
  - A search field is a very common choice
- Once a vulnerable insertion point is found, the attacker can craft a link containing it (and the malicious script) and send it to the victim

Requires some social engineering to entice the victim to click a pre-created malicious link

REFLECTED XSS EXAMPLE

1. Some vulnerable web app has a search function
   - Users enter a word (search term) in a form
   - The form creates a URL containing the search term. For example:

```
https://www.example.com/search?term=chocolate
```

2. When the user presses enter, the web app searches for the term AND echoes the user-supplied search term back :

```
<p>You searched for: chocolate</p>
```

REFLECTED XSS EXAMPLE (CONT'D)

3. The attacker creates a URL with malicious code in place of a normal search term:

```
https://www.example.com/search?term=<script>some+malicious+code</script>
```

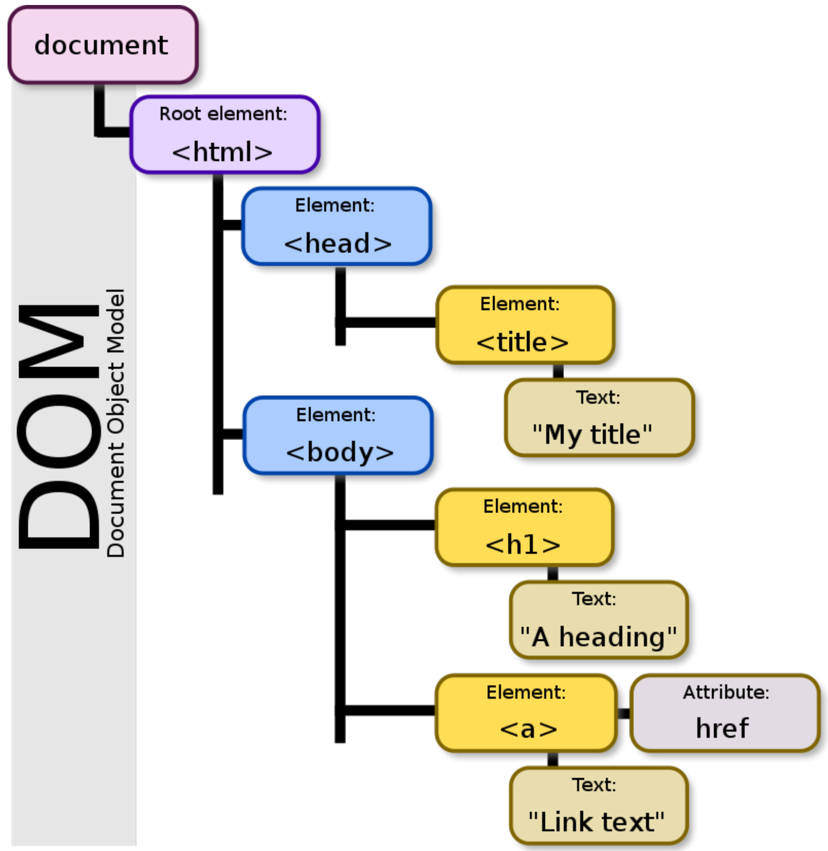
4. The attacker puts this malicious URL in a phishing email, SMS, etc. and sends it to the user
5. The user just has to click the link, thinking it will lead to something else
6. The vulnerable site will reflect the malicious code back to the victim's browser
7. The browser executes the code (which could include stealing the user's session token)

DOM XSS

- AKA DOM-based Cross-Site Scripting
- Abuses the Document Object Model
  - A standard way to represent HTML objects in a hierarchical manner
- In a DOM XSS attack, the malicious JavaScript code is inserted directly into the victim's browser
  - Unlike the other XSS attacks which upload (or reflect) malicious code off the server
- There are several HMTL objects that are particularly suited for DOM XSS:
  - window.location
  - document.url
  - document.location
  - document.referrer

DOCUMENT OBJECT MODEL (DOM)

- DOM is a programming interface for HTML and XML documents
- DOM treats an HTML/XML document as a tree structure,
- Each node in the tree is an "object" representing part of the document
  - The document itself is also considered an object
- Objects have "methods" that can be used programmatically to change the content of the document

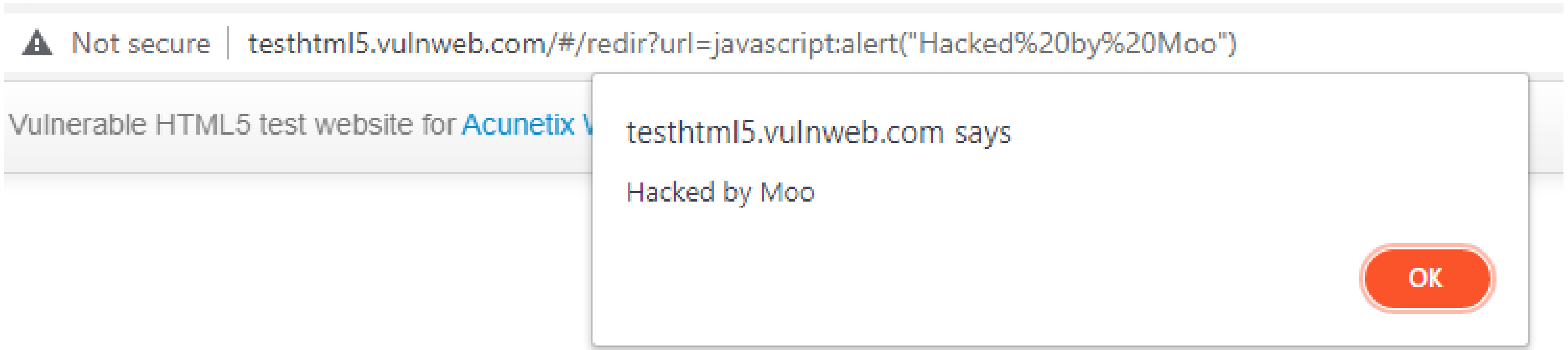


DOM EXAMPLE

- A "malicious" link with JavaScript is clicked by an unsuspecting victim

```
http://testhtml5.vulnweb.com/#/redir?url=javascript:alert(%22Hacked%20by%20Moo%22)
```

- The JavaScript is processed on the client browser


Not secure | testhtml5.vulnweb.com/#/redir?url=javascript:alert("Hacked%20by%20Moo")

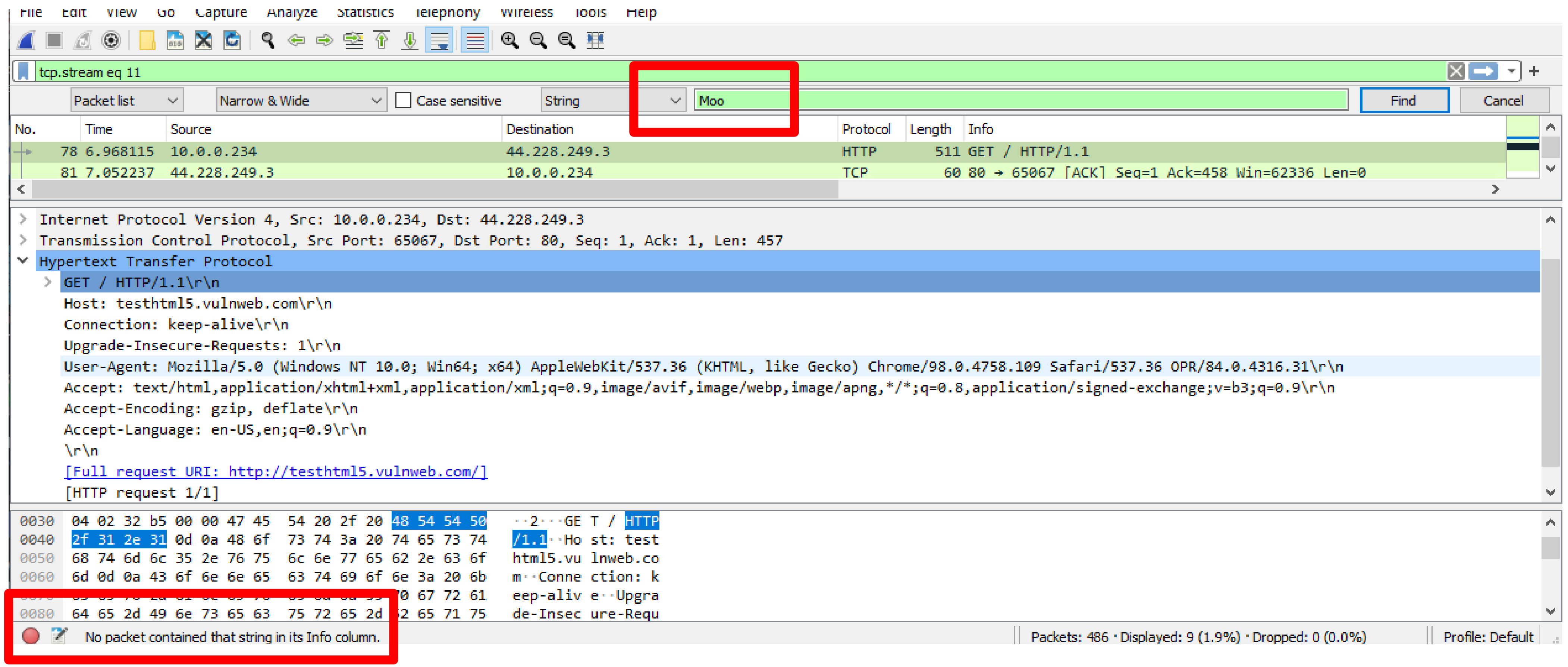
Vulnerable HTML5 test website for Acunetix V

testhtml5.vulnweb.com says

Hacked by Moo

DOM EXAMPLE (CONT'D)

- Wireshark capture confirms the JavaScript was not sent to the server
- No sign of the script in any of the packets


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Packet list
Narrow & Wide
Case sensitive
String
Find
Cancel

```
tcp.stream eq 11
Moo

No.  Time      Source        Destination    Protocol  Length  Info
  78 6.968115  10.0.0.234    44.228.249.3   HTTP         511  GET / HTTP/1.1
  81 7.052237  44.228.249.3  10.0.0.234     TCP           60  80 → 65067 [ACK] Seq=1 Ack=458 Win=62336 Len=0

> Internet Protocol Version 4, Src: 10.0.0.234, Dst: 44.228.249.3
> Transmission Control Protocol, Src Port: 65067, Dst Port: 80, Seq: 1, Ack: 1, Len: 457
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: testhtml5.vulnweb.com\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.109 Safari/537.36 OPR/84.0.4316.31\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: en-US,en;q=0.9\r\n
    \r\n
    [Full request URI: http://testhtml5.vulnweb.com/]
    [HTTP request 1/1]

0030  04 02 32 b5 00 00 47 45  54 20 2f 20 48 54 54 50   ··2···GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f  73 74 3a 20 74 65 73 74   /1.1··Ho st: test
0050  68 74 6d 6c 35 2e 76 75  6c 6e 77 65 62 2e 63 6f   html5.vu lnweb.co
0060  6d 0d 0a 43 6f 6e 6e 65  63 74 69 6f 6e 3a 20 6b   m··Conne ction: k
                                                   70 67 72 61   eep-aliv e··Upgra
0080  64 65 2d 49 6e 73 65 63  75 72 65 2d             2 65 71 75   de-Insec ure-Requ

No packet contained that string in its Info column.    Packets: 486 · Displayed: 9 (1.9%) · Dropped: 0 (0.0%)    Profile: Default
```

XSS COUNTERMEASURES

- Include input validation/sanitization in the web app
- Use a tool such as Burp Suite to scan your website or web application regularly
- Restrict user input to a specific allow list
  - Provide a drop-down menu that a user must choose from
- Avoid/restrict HTML in inputs
  - Require all input be text only
  - Sanitize all input to remove any possible code
- Sanitize all inputted values
  - Escape all unsafe characters so that they don't result in HTML
- Use HTTPOnly Flags on Cookies
  - This prevents JavaScript from reading the content of the cookie
- Use a Web Application Firewall to help pre-screen all input

11.4 CSRF

- Cross-site Request Forgery
- CSRF Considerations

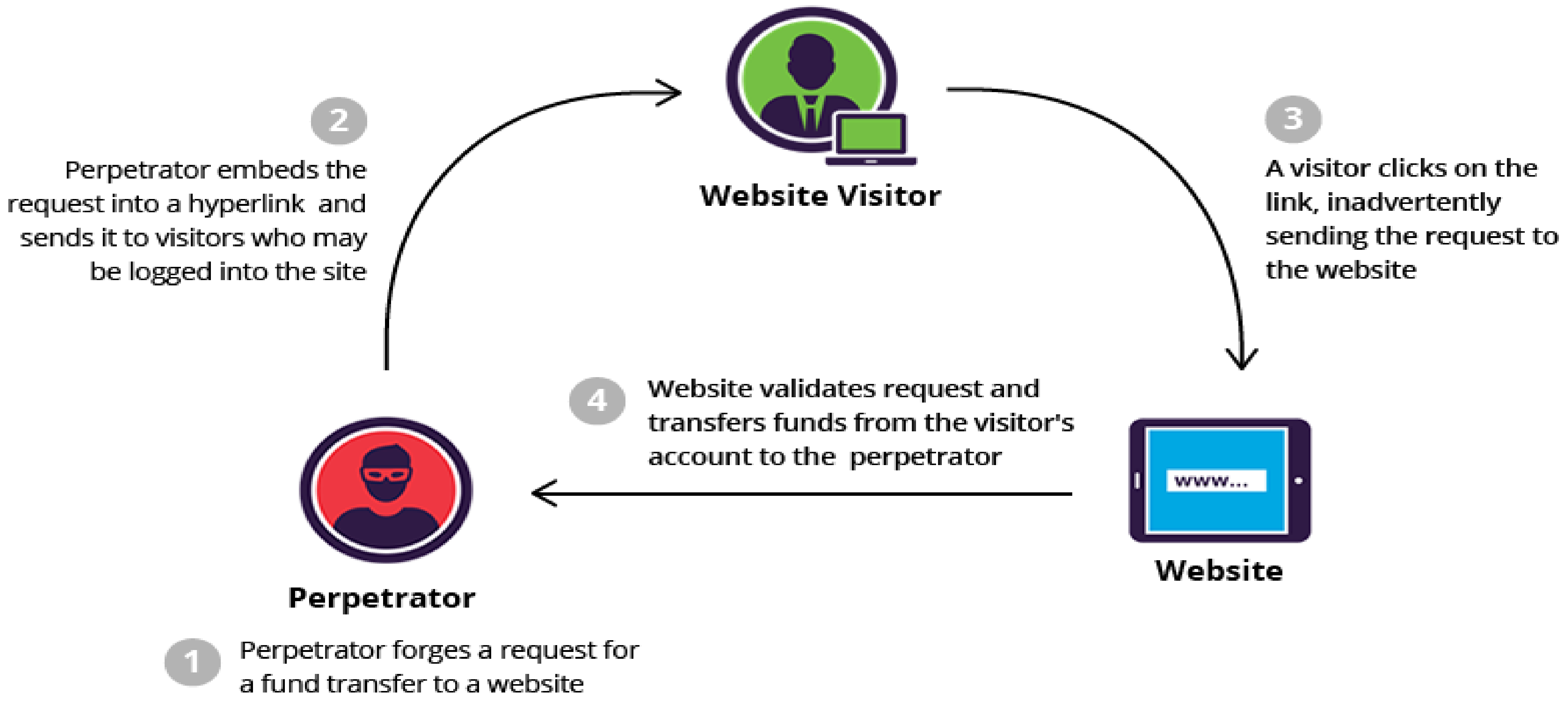
CROSS-SITE REQUEST FORGERY (CSRF)

- Exploits a server's trust in the client
- Takes advantage of a saved authentication to access sensitive data
- The attacker tricks an authenticated victim into unwittingly executing an undesirable action
  - The victim authenticates/is already authenticated to the web site
  - The attacker sends a malicious link to the victim
  - The link instructs the victim's browser to perform an unwanted action in the background
- A CSRF attack can force the unwitting victim to:
  - Transfer funds
  - Make unauthorized purchases
  - Change their email address or contact details
  - etc.
- If the victim has an administrative account, the attacker can expand the attack to other areas in the web site

CSRF SCENARIO

1. Visit your bank's site, log in.
2. Then visit the attacker's site (e.g. sponsored ad from an untrusted organization).
3. Attacker's page includes hidden form with same fields as the bank's "Transfer Funds" form.
4. Form fields are pre-filled to transfer money from your account to attacker's account.
5. Attacker's page includes JavaScript that submits form to your bank.
6. When form gets submitted, browser includes your cookies for the bank site, including the session token.
7. Bank transfers money to attacker's account.
8. The form can be in an iframe that is invisible, so you never know the attack occurred.

BASIC CSRF EXAMPLE


Website Visitor
2
Perpetrator embeds the request into a hyperlink and sends it to visitors who may be logged into the site
3
A visitor clicks on the link, inadvertently sending the request to the website
4
Website validates request and transfers funds from the visitor's account to the perpetrator
www...
Website
Perpetrator
1
Perpetrator forges a request for a fund transfer to a website

CSRF EXAMPLE

- Typical GET request for a $5,000 bank transfer to my friend Moo:

```
GET https://mybank.com/transfer.do?account=Moo&amount=$5000 HTTP/1.1
```

- Attacker would change the recipient to their account instead:

```
GET https://mybank.com/transfer.do?account=TheAttacker&amount=$5000 HTTP/1.1
```

- Attacker embeds the request into a harmless-looking hyperlink:

```
<a href="https://mybank.com/transfer.do?account=TheAttacker&amount=$5000">Click for more information</a>
```

- The attacker now needs to send out a phishing email to as many bank customers as possible, enticing them to click the link

CSRF EXAMPLE (CONT'D)

- If the bank's website only uses POST requests, it's not possible to frame malicious requests using an <a> href tag
- However, the attack can be delivered in a <form> tag
  - It can even be a self submitting form

```
<body onload="document.forms[0].submit()>
<form id=”csrf” action="https://mybank.com/transfer.do" method="POST">
<input type="hidden" name="account" value=“TheAttacker"/>
<input type="hidden" name="amount" value="$5000"/>
</form>
</body>
```

- The above form above does not have a submit button
  - It will be triggered without a user's knowledge and consent
  - Instead, the button is replaced by a line of JavaScript:

```
document.getElementById('csrf').submit();
```

CSRF CONSIDERATIONS

- The power of CSRF is that it's difficult to detect
  - The attack is carried out by the user's browser as if user requested it
  - The user could enter same URL manually and get same result
  - It's nearly impossible for the browser to distinguish CSRF from normal activity
- CSRF can be difficult for an attacker to execute
  - Requires finding a form that can permit malicious instructions
  - Requires knowing the right values that aren't obscured
  - Sites that check the referrer header will disallow requests from different origins

CSRF COUNTERMEASURES

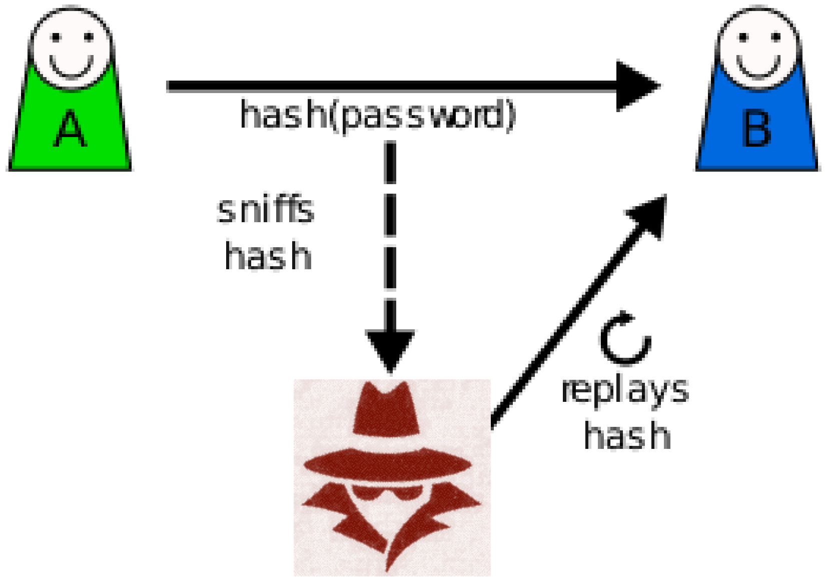
- Implement CSRF tokens
  - A unique, unpredictable secret value generated by the web app
  - The client must present the token for every request
- Do not use GET requests for state-changing operations
- Use the OWASP CSRF Cheat Sheet for guidance when developing the web app

11.5 OTHER WEB HIJACKING ATTACKS

- Session Replay
- Session Fixation
- MITB
- MITM

SESSION REPLAY ATTACK

- Attacker listens in on conversation between user and server
- Attacker obtains user's authentication token
- Attacker replays request to server using obtained token and gains unauthorized server access



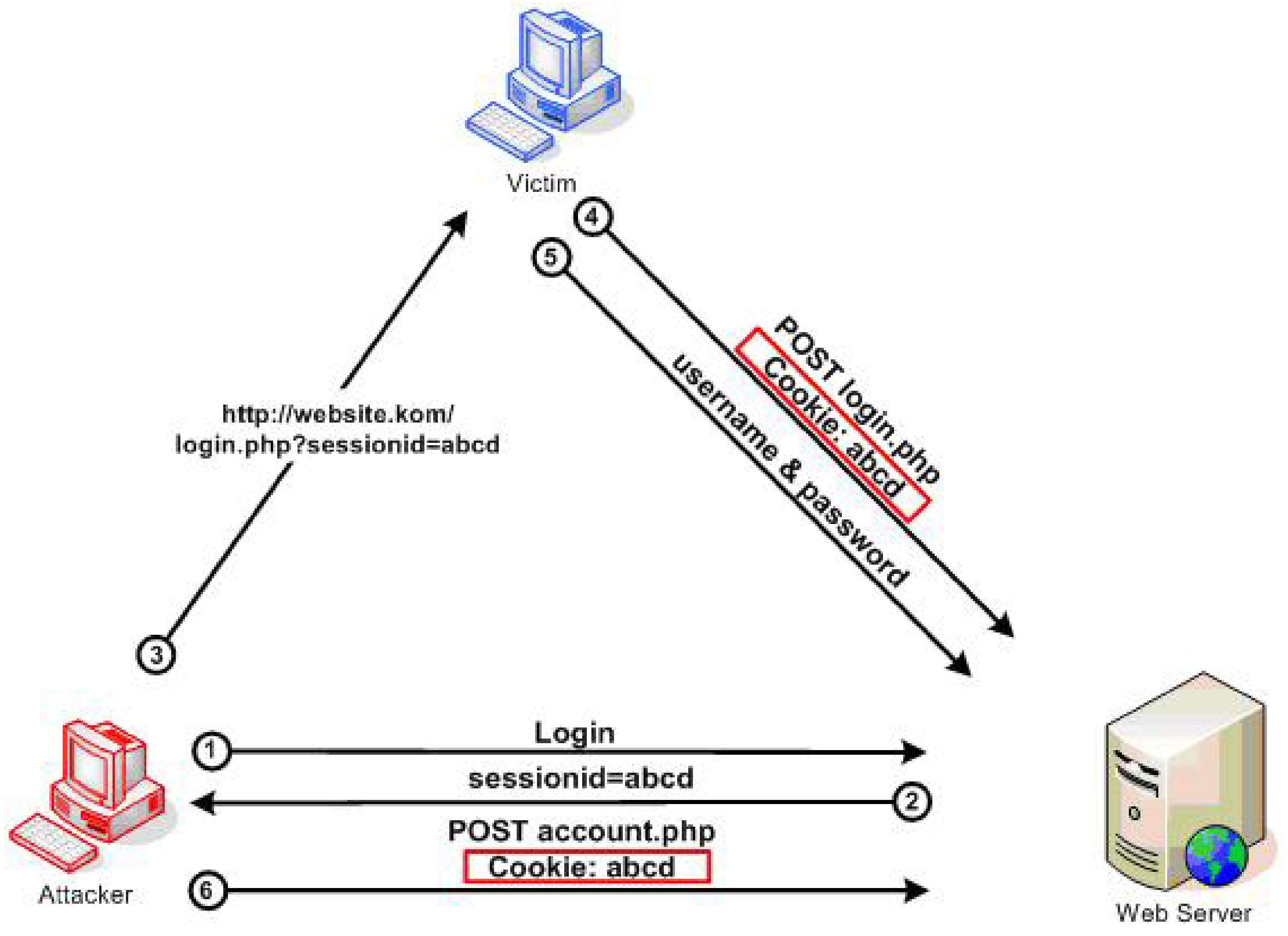
SESSION FIXATION ATTACK

- AKA Session Donation Attack
- Permits an attacker to hijack a valid user session
- Exploits a limitation in the way a vulnerable web app manages the session ID
- An attacker obtains legitimate web app session ID
  - Tricks the victim's browser into using it
- Session fixation execution techniques include:
  - Session token in URL argument
  - Session token in hidden form field
  - Session ID hidden in cookie

SESSION FIXATION SCENARIO

1. The attacker visits the web application login page
   - Receives a legitimate session ID generated by the web application
   - The attacker does not log in, but saves the session ID
2. The attacker tricks a victim into using the session ID
   - Injection, man-in-the-middle attack, social engineering, etc.
   - The victim goes to the website with the attacker's session ID and logs in
3. The web app now thinks anyone with the session ID is legitimate
4. The attacker uses the session ID to access the web application
   - Takes over the user session
   - Impersonates the victim

SESSION FIXATION ATTACK EXAMPLE



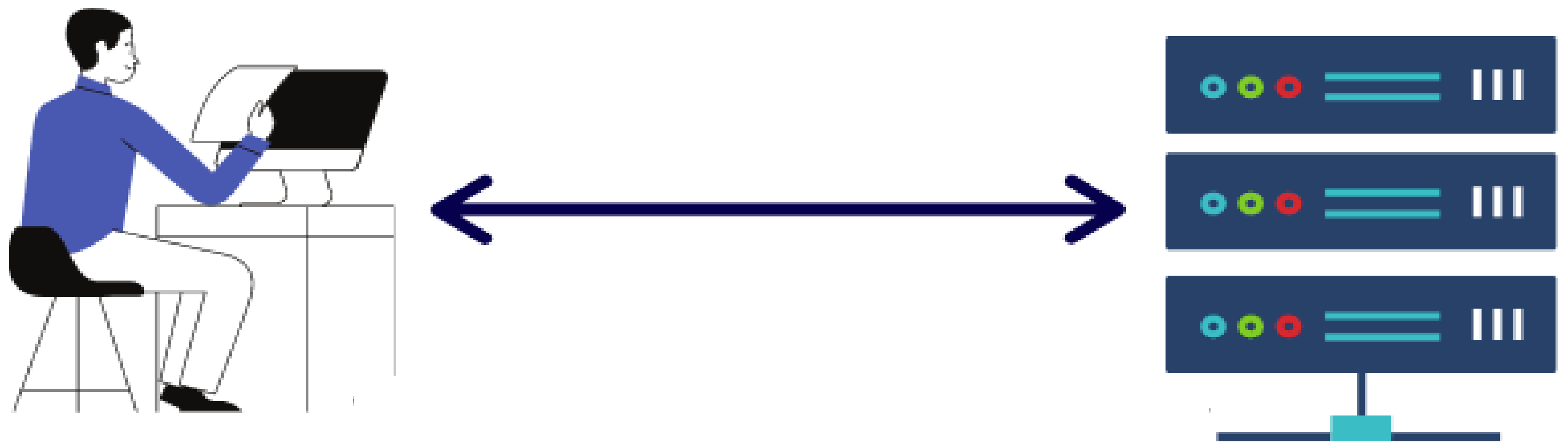
MAN-IN-THE-MIDDLE ATTACK (MITM)

- AKA Monkey-in-the-Middle
- A general term for an attacker inserting themselves into an existing session to intercept messages
- Uses various techniques to split TCP connection into two sessions:
  - Victim-to-attacker
  - Attacker-to-server
- Once inserted, the attacker can read/modify/insert fraudulent data into the communication
- You can capture a session cookie by reading the HTTP header
- You can also change the amount of a money transaction inside the application context

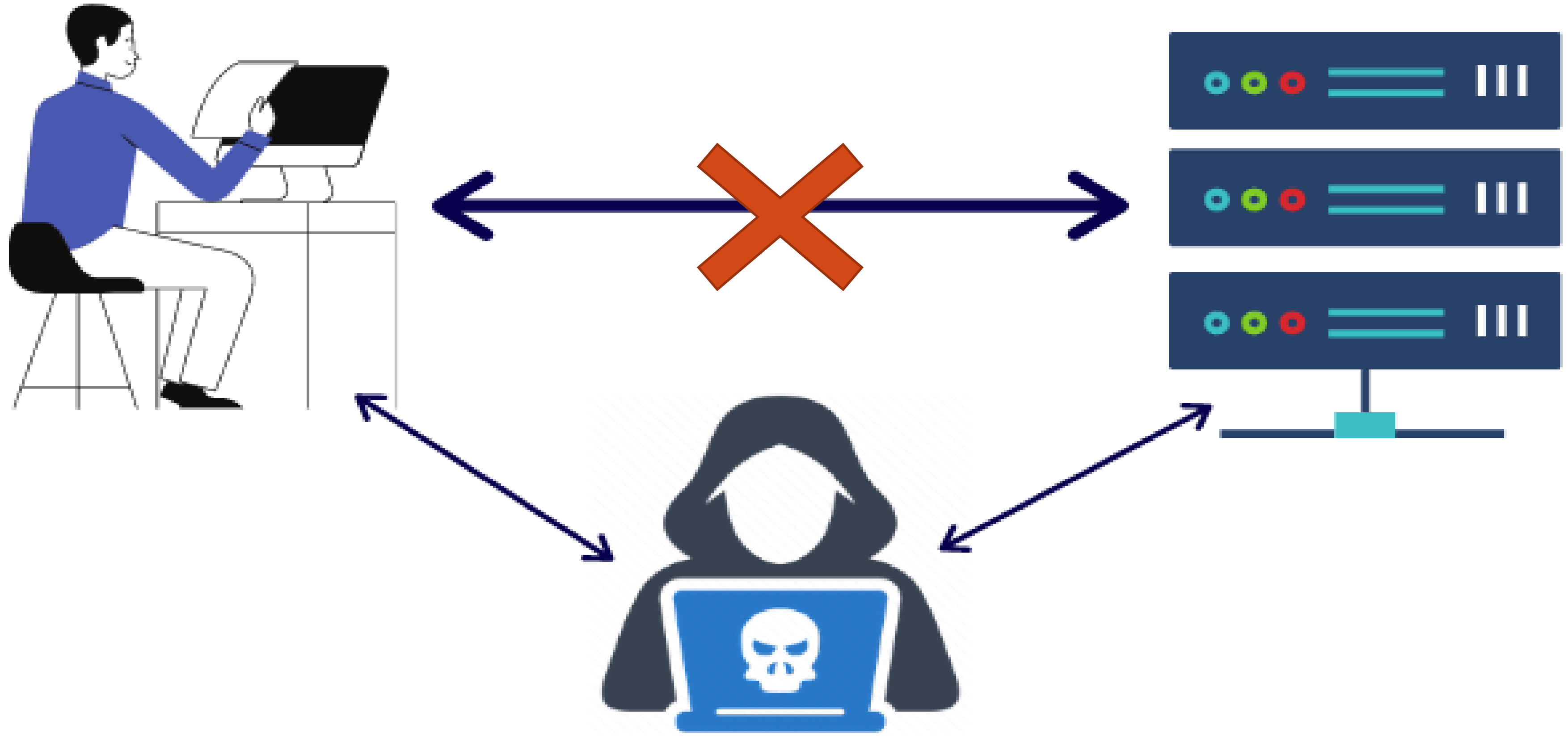
HOW TO ACCOMPLISH MITM

- ARP spoofing
- DNS poisoning
  - Modify records on the authoritative DNS server
  - Modify the cache of the local DNS server
  - Inject fake cached lookups into the victim machine
  - Modify the victim machine's HOSTS file with fake name to IP mappings
- Rogue wireless access point
- Malicious links

MITM EXAMPLE

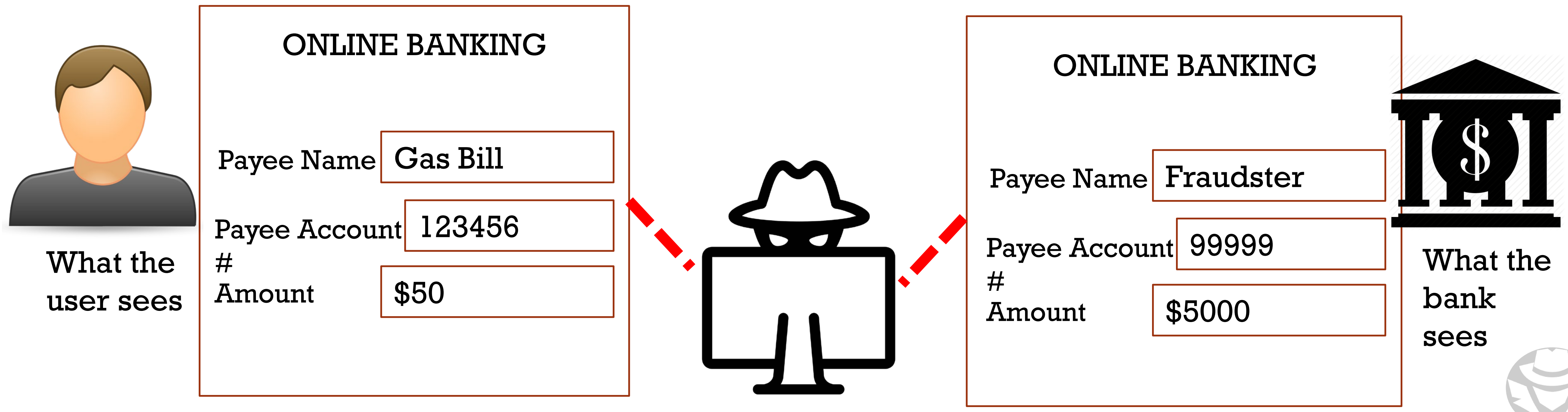


MITM EXAMPLE



MAN-IN-THE-BROWSER ATTACK

- Similar to MITM
- Attacker uses a Trojan to intercept calls between the browser and its libraries/security mechanisms
- Primary objective is to manipulate Internet banking transactions
- Customer makes the payment, but malware changes the destination and amount



ONLINE BANKING

Payee Name	Gas Bill
Payee Account #	123456
Amount	$50

ONLINE BANKING

Payee Name	Fraudster
Payee Account #	99999
Amount	$5000

ADDITIONAL ATTACK TYPES

- Compression Ratio Info-leak Made Easy (CRIME):
  - A client-side attack that exploit vulnerabilities present in the data compression feature of protocols such as SSL/TLS, SPDY*, and HTTPS
- BREACH:
  - An exploit against HTTPS when using HTTP compression (SSL/TLS compression)
  - Based on the CRIME security exploit
- Forbidden Attack
  - A type of MITM
  - Exploits the reuse of a cryptographic nonce during the TLS handshake

*SPDY is a Google protocol that manipulates HTTP traffic. It attempts to reduce page load latency, thus speeding up web traffic

11.6 NETWORK-LEVEL SESSION HIJACKING

- TCP Session Hijacking
- Source Routed Packets
- RST Hijacking
- Blind Hijacking
- ICMP/ARP Spoofing
- UDP Hijacking

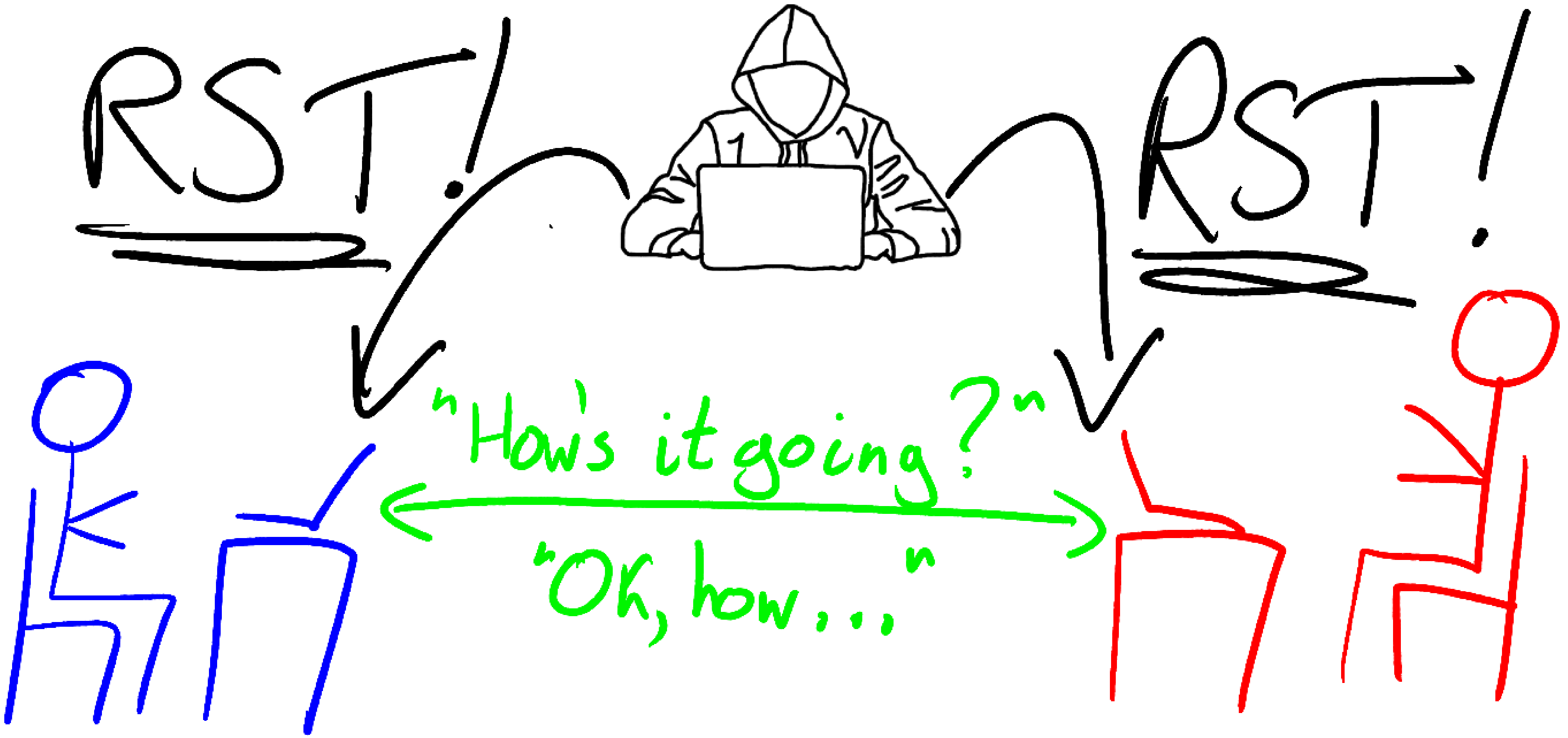
TCP SESSION HIJACKING

- Take a user's or client's place after it has established a TCP connection with a server
- Enables a connection without providing credentials
- Conditions:
  - Cleartext protocol used
  - Attacker needs to observe and correctly predict TCP sequencing numbers
  - Packets can't be digitally signed
- Process:
  - Watch the client/server TCP sequence numbers
  - Send spoofed TCP FIN packets to the client
  - Spoof your IP or MAC to the server
  - When the client disconnects, continue communicating with the server via the spoofed address

RST HIJACKING

- A common way to deauthenticate a client
- Attacker sends spoofed TCP segments to the client with the RST flag raised
- Victim (typically client) thinks the other side (typically server) has closed the connection
- Attacker takes the client's place

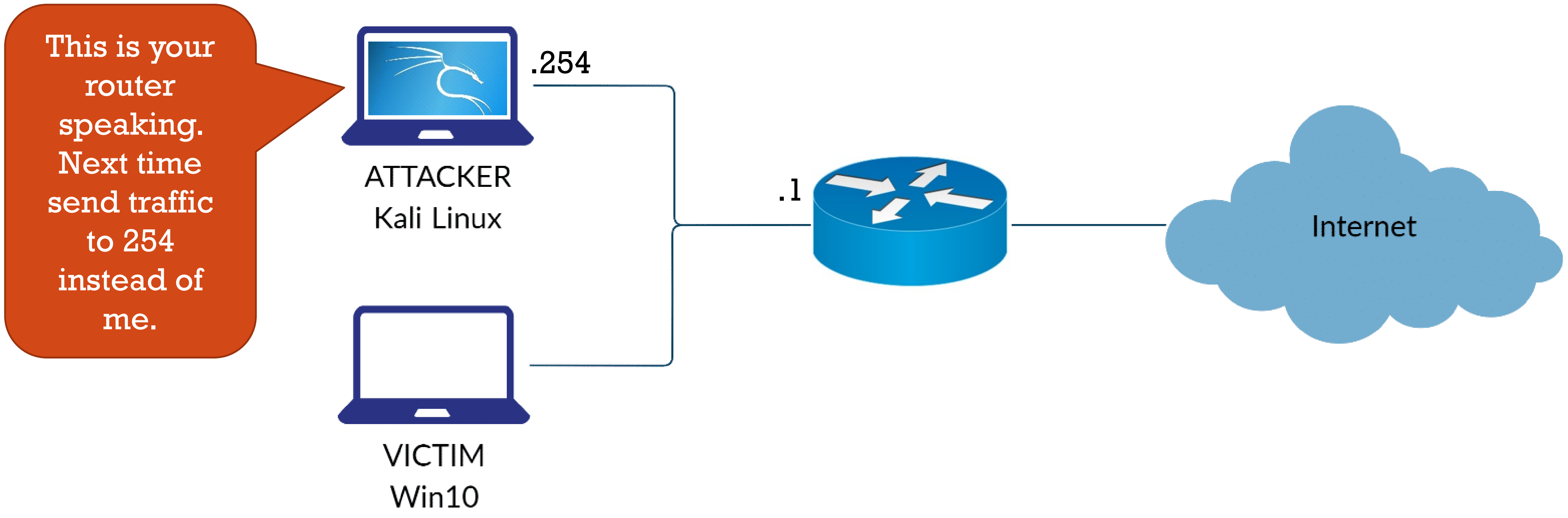
RST EXAMPLE



ICMP REDIRECT/ARP SPOOFING

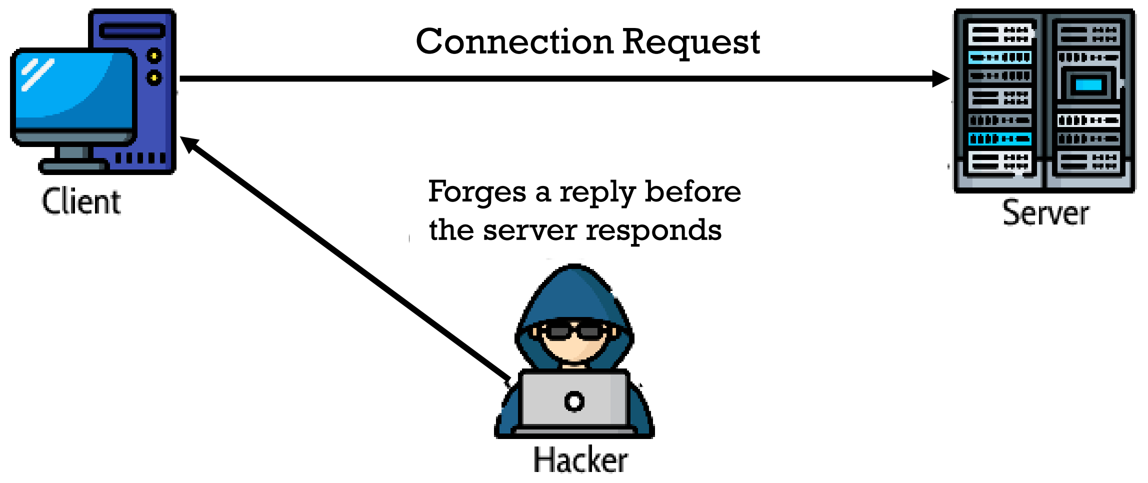
- Two common techniques to redirect traffic to the attacker
- Both require that the client and the attacker be on the same network segment
- ICMP Redirect
  - Attacker sends spoofed ICMP redirect messages to the client
  - Tells the client that it should no longer use its current default gateway
  - Instead, the attacker is the client's "new" default gateway
- ARP Spoofing
  - Attacker sends fake ARP replies re-mapping the server/router IP address to the attacker's MAC address
  - The client will put the attacker's MAC in the destination field of the Ethernet or Wi-Fi frame

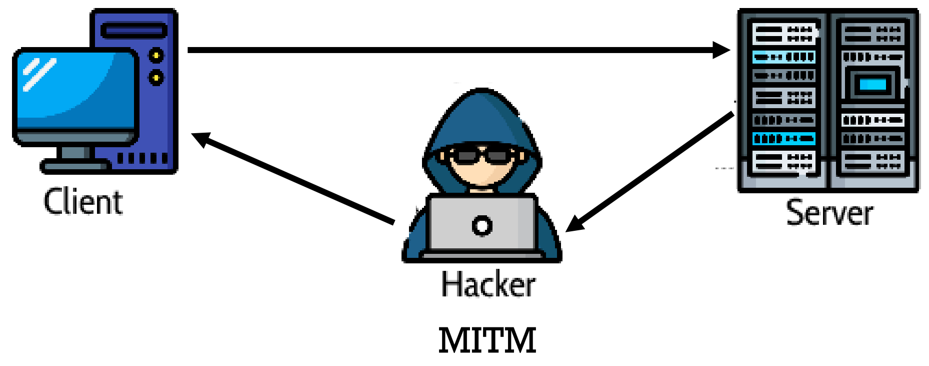
ICMP REDIRECT EXAMPLE


This is your router speaking. Next time send traffic to 254 instead of me.
.254
ATTACKER
Kali Linux
VICTIM
Win10
.1
Internet

UDP HIJACKING

- UDP Hijacking can happen in one of two ways:
  - Attacker sends a forged server reply to the victim before the legitimate server can reply
  - Attacker intercepts server's reply using man-in-the-middle attack

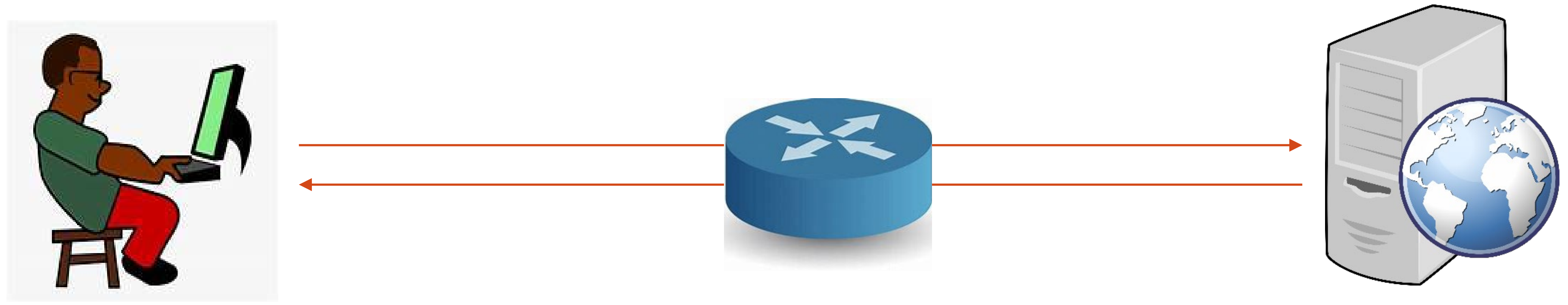




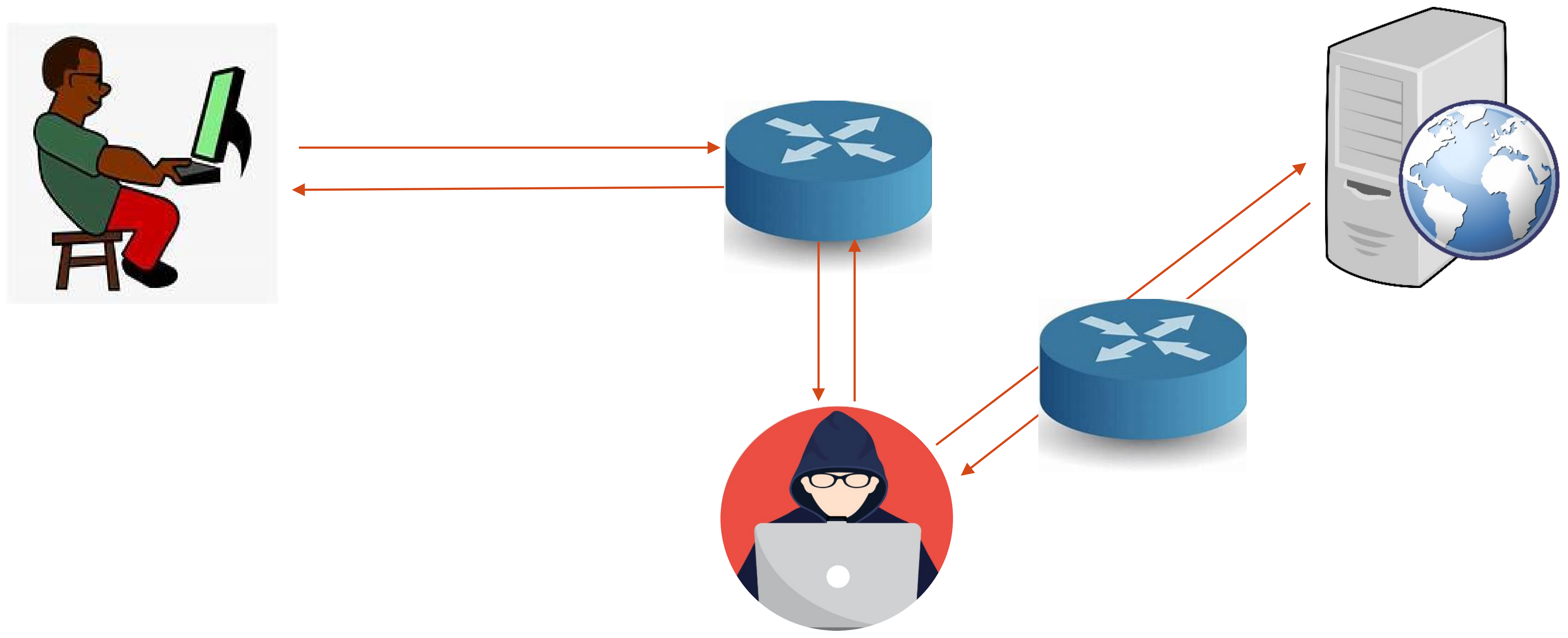
SOURCE-ROUTED IP PACKETS

- A type of MITM
- The attacker does NOT create two sessions
- Instead, the attacker poisons the DNS lookup so the client sends traffic destined for the server to the attacker
- The attacker also manipulates the source routing option in the IP headers of the client's traffic
  - Specifies that the traffic return path from the server passes back through the attacker

SOURCE-ROUTED IP PACKETS EXAMPLE

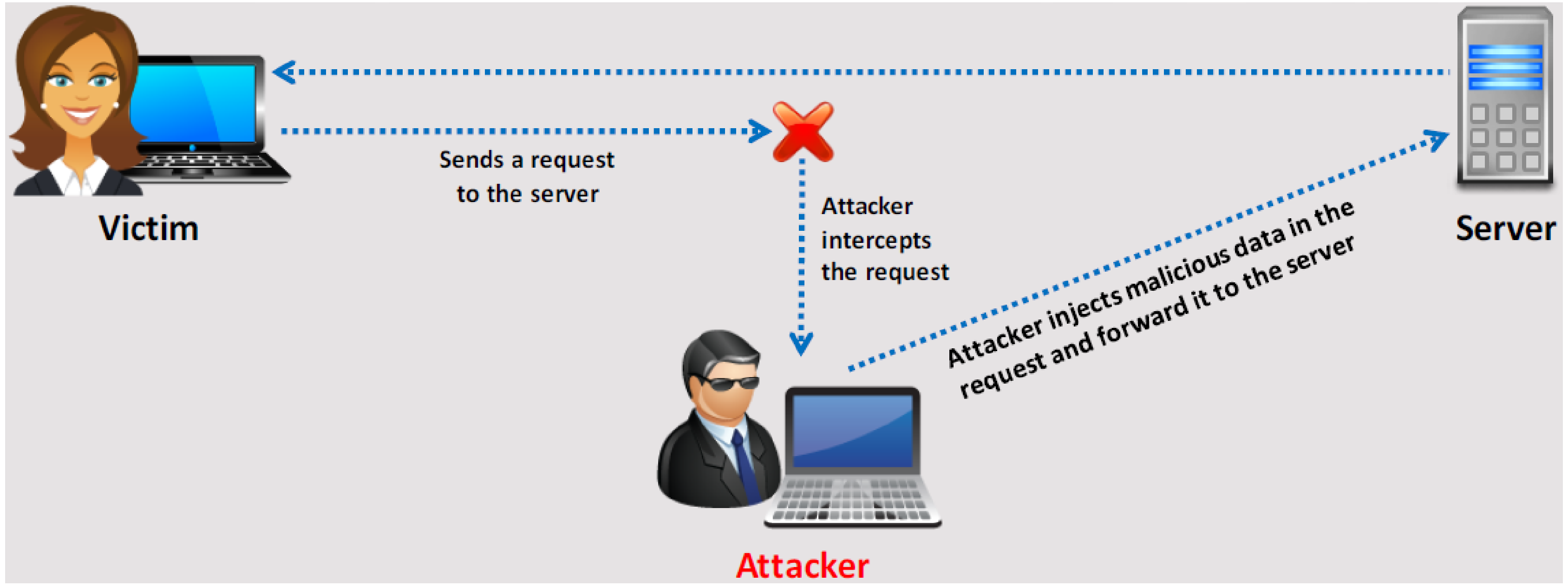


SOURCE-ROUTED IP PACKETS EXAMPLE



BLIND HIJACKING

- Performed if source routing is not possible
- Attacker can only send data/commands – cannot see server's response
  - OK if they can see the results of a command


Victim
Sends a request to the server
Attacker intercepts the request
Attacker injects malicious data in the request and forward it to the server
Server
Attacker

11.7 SESSION HIJACKING TOOLS

- Hijacking Tools
- Mobile Tools

SESSION HIJACKING TOOLS

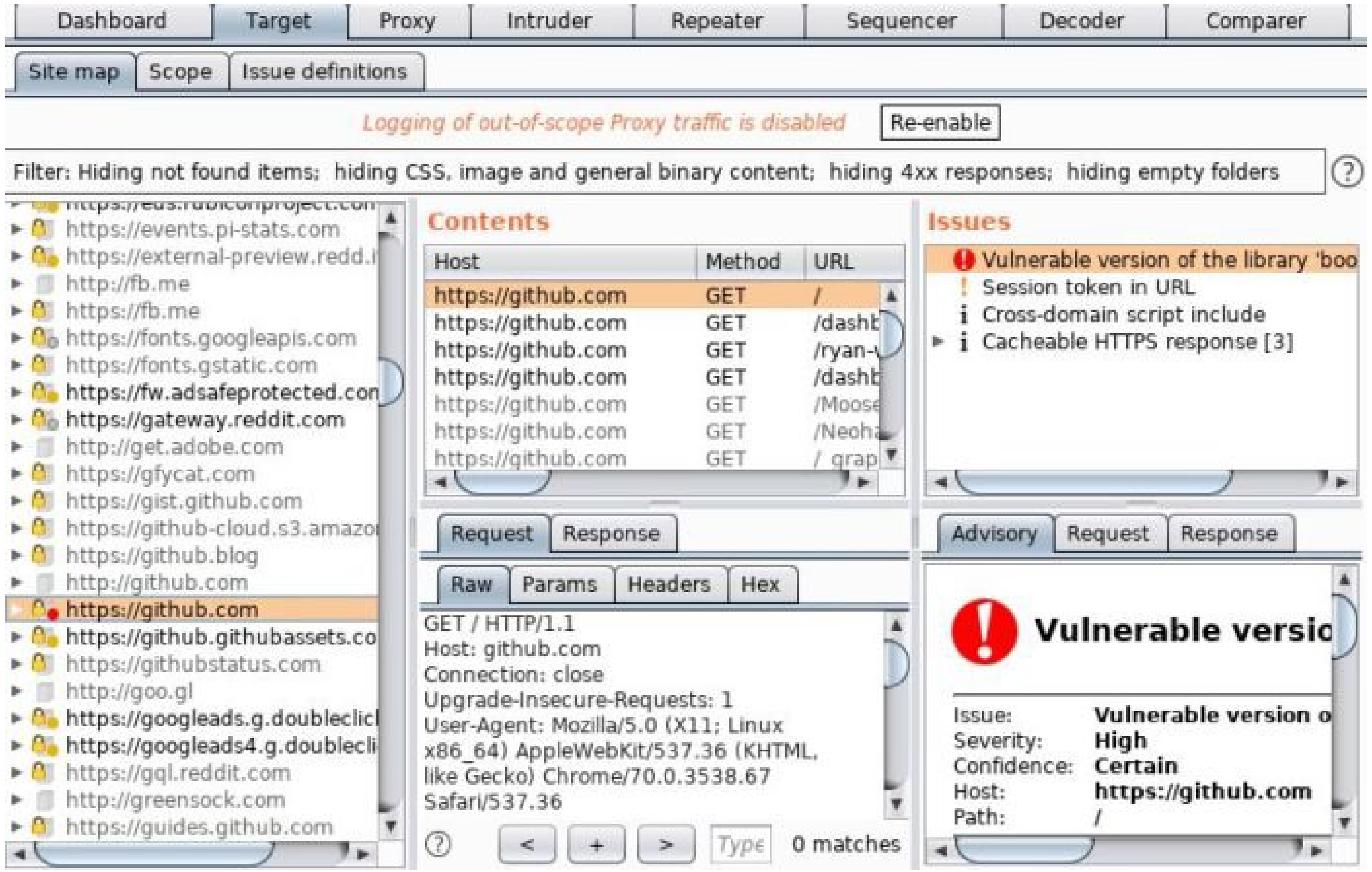
- **Ettercap, bettercap**
  - ARP poisoning and MITM tools
- **T-Sight, Juggernaut, Hunt, Shijack**
  - TCP interactive session hijackers
- **sslstrip**
  - force SSL downgrade to HTTP
  - Used in HTTPS MITM attacks
- **Hamster**
  - cookie sidejacking tool - replaces your cookie with someone else's
- **Ferret**
  - the cookie sniffer used by Hamster

Note: You can use Ettercap, hamster, and ferret together to MITM, sniff, and sidejack cookies

SESSION HIJACKING TOOLS (CONT'D)

- **Burp Suite, OWASP ZAP, Paros**
  - localhost proxies for intercepting and manipulating web app traffic
- **Firesheep**
  - Mozilla Firefox extension
  - Packet sniffer that hijacks browser sessions on unencrypted public Wi-Fi
  - Steals cookies from the user's favorite sites such as Facebook, Twitter, etc.
- **CookieCadger**
  - A Java app that automatically sidejacks and replays insecure HTTP GET requests

BURP SUITE EXAMPLE



Logging of out-of-scope Proxy traffic is disabled

Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

https://eus.rubiconproject.com
https://events.pi-stats.com
https://external-preview.redd.i
http://fb.me
https://fb.me
https://fonts.googleapis.com
https://fonts.gstatic.com
https://fw.adsafeprotected.com
https://gateway.reddit.com
http://get.adobe.com
https://gfycat.com
https://gist.github.com
https://github-cloud.s3.amazo
https://github.blog
http://github.com
https://github.com
https://github.githubassets.co
https://githubstatus.com
http://goo.gl
https://googleads.g.doublecli
https://googleads4.g.doublecli
https://gql.reddit.com
http://greensock.com
https://guides.github.com

Host	Method	URL
https://github.com	GET	/
https://github.com	GET	/dashb
https://github.com	GET	/ryan-
https://github.com	GET	/dashb
https://github.com	GET	/Moose
https://github.com	GET	/Neoh
https://github.com	GET	/_grap

- Vulnerable version of the library 'boo
- Session token in URL
- Cross-domain script include
- Cacheable HTTPS response [3]

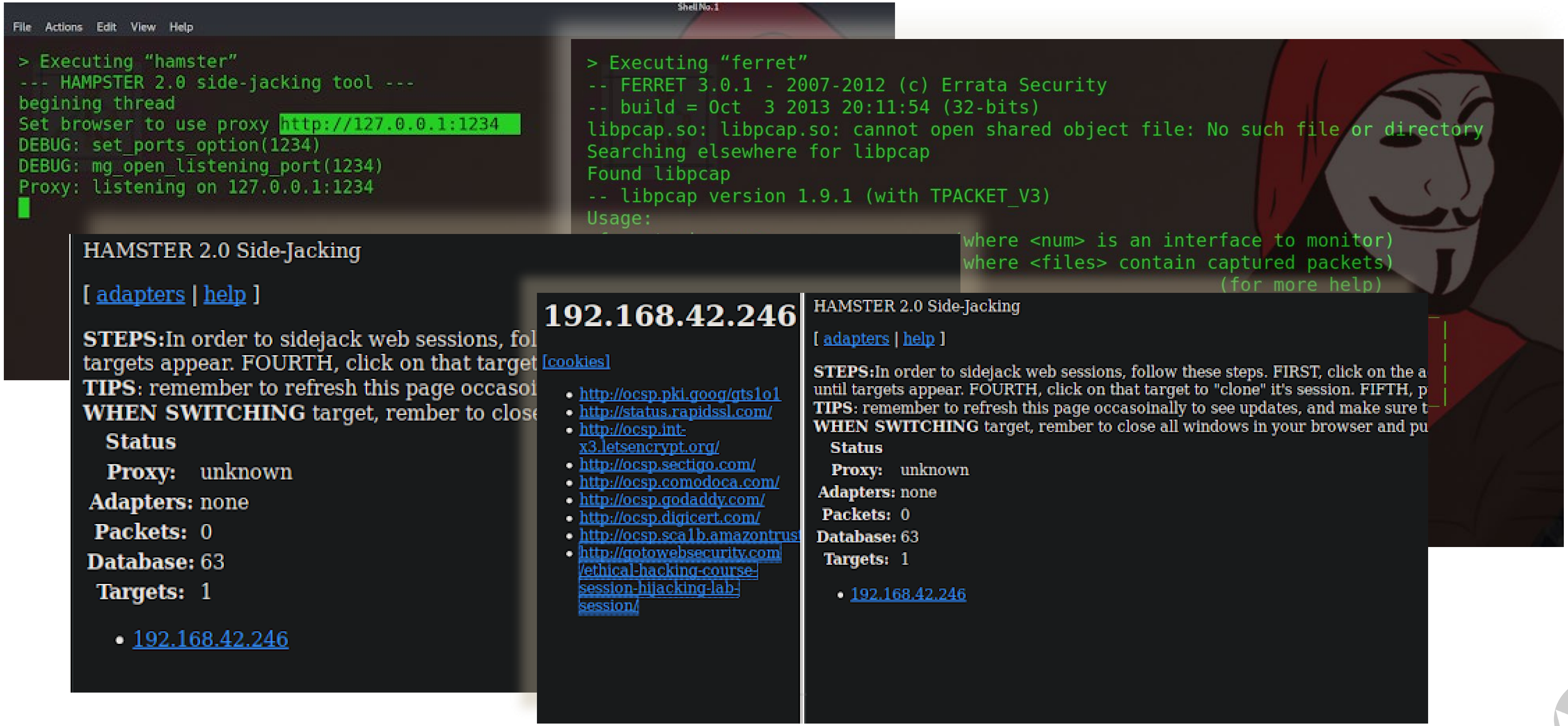
```
GET / HTTP/1.1
Host: github.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux
x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.67
Safari/537.36
```

Type 0 matches

Vulnerable versio

Issue:	**Vulnerable version o**
Severity:	**High**
Confidence:	**Certain**
Host:	**https://github.com**
Path:	**/**

HAMSTER AND FERRET EXAMPLE



```
> Executing “hamster”
--- HAMPSTER 2.0 side-jacking tool ---
begining thread
Set browser to use proxy http://127.0.0.1:1234
DEBUG: set_ports_option(1234)
DEBUG: mg_open_listening_port(1234)
Proxy: listening on 127.0.0.1:1234
```

```
> Executing “ferret”
-- FERRET 3.0.1 - 2007-2012 (c) Errata Security
-- build = Oct  3 2013 20:11:54 (32-bits)
libpcap.so: libpcap.so: cannot open shared object file: No such file or directory
Searching elsewhere for libpcap
Found libpcap
-- libpcap version 1.9.1 (with TPACKET_V3)
Usage:
                                   where <num> is an interface to monitor)
                                   where <files> contain captured packets)
                                                      (for more help)
```

HAMSTER 2.0 Side-Jacking

[adapters | help]

STEPS: In order to sidejack web sessions, fol
targets appear. FOURTH, click on that target
TIPS: remember to refresh this page occasoi
WHEN SWITCHING target, rember to close

Status
Proxy: unknown
Adapters: none
Packets: 0
Database: 63
Targets: 1

- 192.168.42.246

192.168.42.246

[cookies]

- http://ocsp.pki.goog/gts1o1
- http://status.rapidssl.com/
- http://ocsp.int-x3.letsencrypt.org/
- http://ocsp.sectigo.com/
- http://ocsp.comodoca.com/
- http://ocsp.godaddy.com/
- http://ocsp.digicert.com/
- http://ocsp.sca1b.amazontrust
- http://gotowebsecurity.com/ethical-hacking-course-session-hijacking-lab-session/

HAMSTER 2.0 Side-Jacking

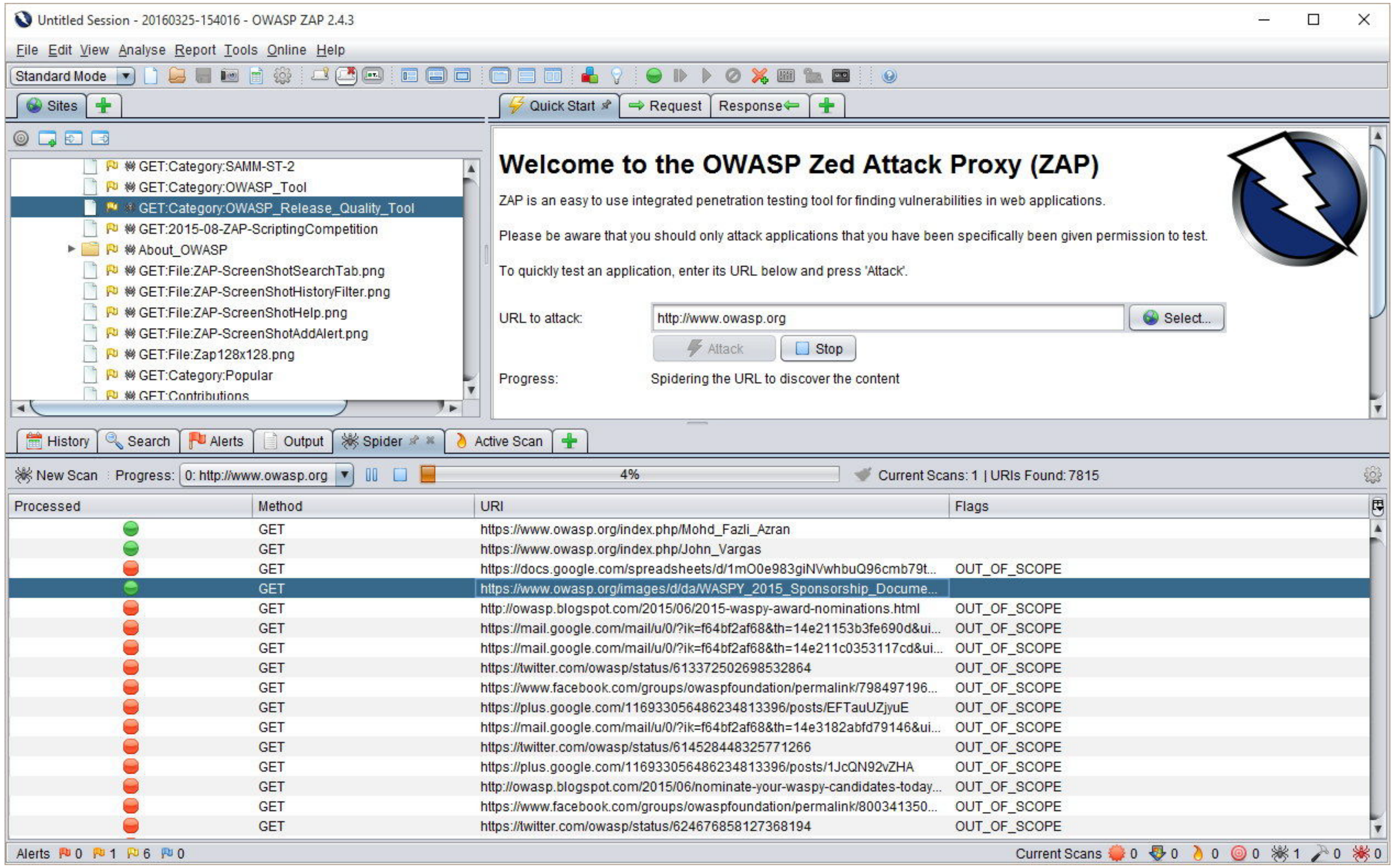
[adapters | help]

STEPS: In order to sidejack web sessions, follow these steps. FIRST, click on the a
until targets appear. FOURTH, click on that target to "clone" it's session. FIFTH, p
TIPS: remember to refresh this page occasoinally to see updates, and make sure t
WHEN SWITCHING target, rember to close all windows in your browser and pu

Status
Proxy: unknown
Adapters: none
Packets: 0
Database: 63
Targets: 1

- 192.168.42.246

ZAP EXAMPLE



Untitled Session - 20160325-154016 - OWASP ZAP 2.4.3

File Edit View Analyse Report Tools Online Help

Standard Mode

Sites

- GET:Category:SAMM-ST-2
- GET:Category:OWASP_Tool
- GET:Category:OWASP_Release_Quality_Tool
- GET:2015-08-ZAP-ScriptingCompetition
- About_OWASP
- GET:File:ZAP-ScreenShotSearchTab.png
- GET:File:ZAP-ScreenShotHistoryFilter.png
- GET:File:ZAP-ScreenShotHelp.png
- GET:File:ZAP-ScreenShotAddAlert.png
- GET:File:Zap128x128.png
- GET:Category:Popular
- GET:Contributions

Quick Start | Request | Response

Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://www.owasp.org

Progress: Spidering the URL to discover the content

History | Search | Alerts | Output | Spider | Active Scan

New Scan Progress: 0: http://www.owasp.org 4% Current Scans: 1 | URIs Found: 7815

Processed	Method	URI	Flags
🟢	GET	https://www.owasp.org/index.php/Mohd_Fazli_Azran	
🟢	GET	https://www.owasp.org/index.php/John_Vargas	
🔴	GET	https://docs.google.com/spreadsheets/d/1mO0e983giNVwhbuQ96cmb79t...	OUT_OF_SCOPE
🟢	GET	https://www.owasp.org/images/d/da/WASPY_2015_Sponsorship_Docume...	
🔴	GET	http://owasp.blogspot.com/2015/06/2015-waspy-award-nominations.html	OUT_OF_SCOPE
🔴	GET	https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e21153b3fe690d&ui...	OUT_OF_SCOPE
🔴	GET	https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e211c0353117cd&ui...	OUT_OF_SCOPE
🔴	GET	https://twitter.com/owasp/status/613372502698532864	OUT_OF_SCOPE
🔴	GET	https://www.facebook.com/groups/owaspfoundation/permalink/798497196...	OUT_OF_SCOPE
🔴	GET	https://plus.google.com/116933056486234813396/posts/EFTauUZjyuE	OUT_OF_SCOPE
🔴	GET	https://mail.google.com/mail/u/0/?ik=f64bf2af68&th=14e3182abfd79146&ui...	OUT_OF_SCOPE
🔴	GET	https://twitter.com/owasp/status/614528448325771266	OUT_OF_SCOPE
🔴	GET	https://plus.google.com/116933056486234813396/posts/1JcQN92vZHA	OUT_OF_SCOPE
🔴	GET	http://owasp.blogspot.com/2015/06/nominate-your-waspy-candidates-today...	OUT_OF_SCOPE
🔴	GET	https://www.facebook.com/groups/owaspfoundation/permalink/800341350...	OUT_OF_SCOPE
🔴	GET	https://twitter.com/owasp/status/624676858127368194	OUT_OF_SCOPE

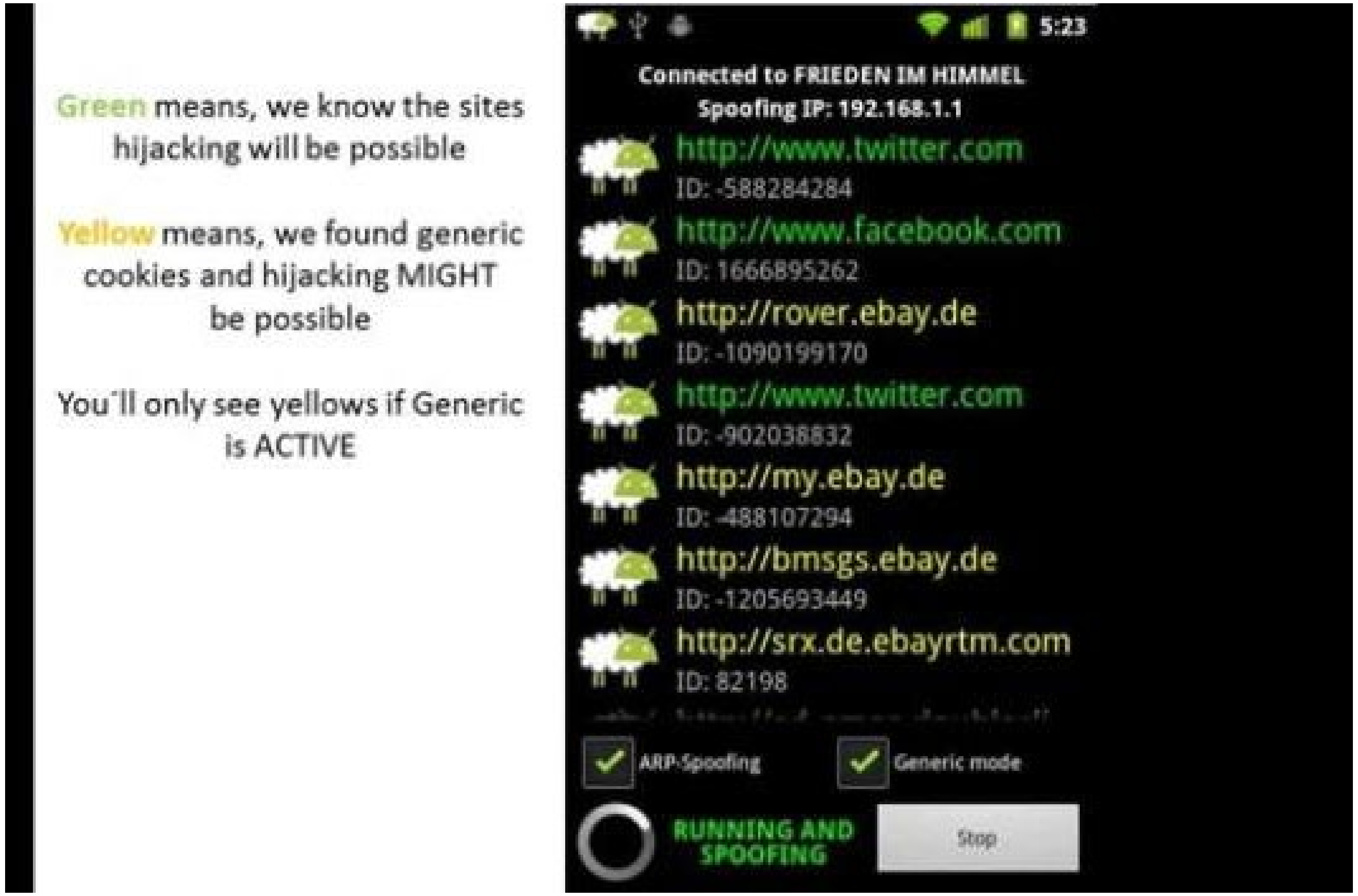
Alerts 0 1 6 0 Current Scans 0 0 0 0 1 0 0

HIJACKING APPS FOR MOBILE DEVICES

- **DroidSheep**
  - Android app that listens for HTTP packets on wireless networks
  - Sidejacks the session and extracts the session ID
- **DroidSniff**
  - Sniffer, MITM, automatic password capture of popular social media sites
- **dSploit**
  - Pentesting suite that runs on Android
  - Wi-Fi scanning, network mapping, port scanning session hijacking, MITM
- **zANTI**
  - ARP poisoning MITM
  - Sniff cookies

MOBILE APP EXAMPLES

DroidSheep



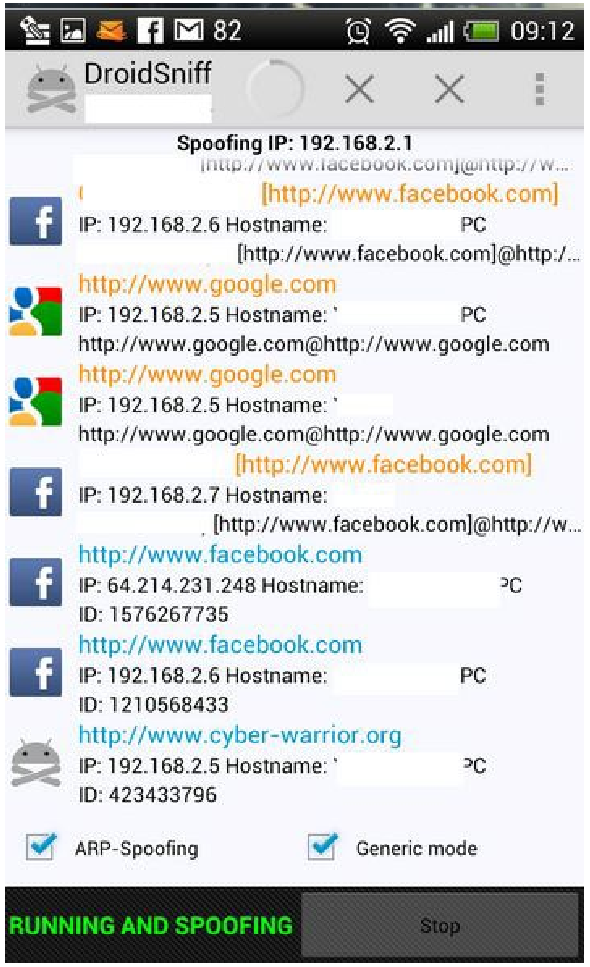
Green means, we know the sites hijacking will be possible

Yellow means, we found generic cookies and hijacking MIGHT be possible

You´ll only see yellows if Generic is ACTIVE

```
5:23
Connected to FRIEDEN IM HIMMEL
Spoofing IP: 192.168.1.1
http://www.twitter.com
ID: -588284284
http://www.facebook.com
ID: 1666895262
http://rover.ebay.de
ID: -1090199170
http://www.twitter.com
ID: -902038832
http://my.ebay.de
ID: -488107294
http://bmsgs.ebay.de
ID: -1205693449
http://srx.de.ebayrtm.com
ID: 82198
```

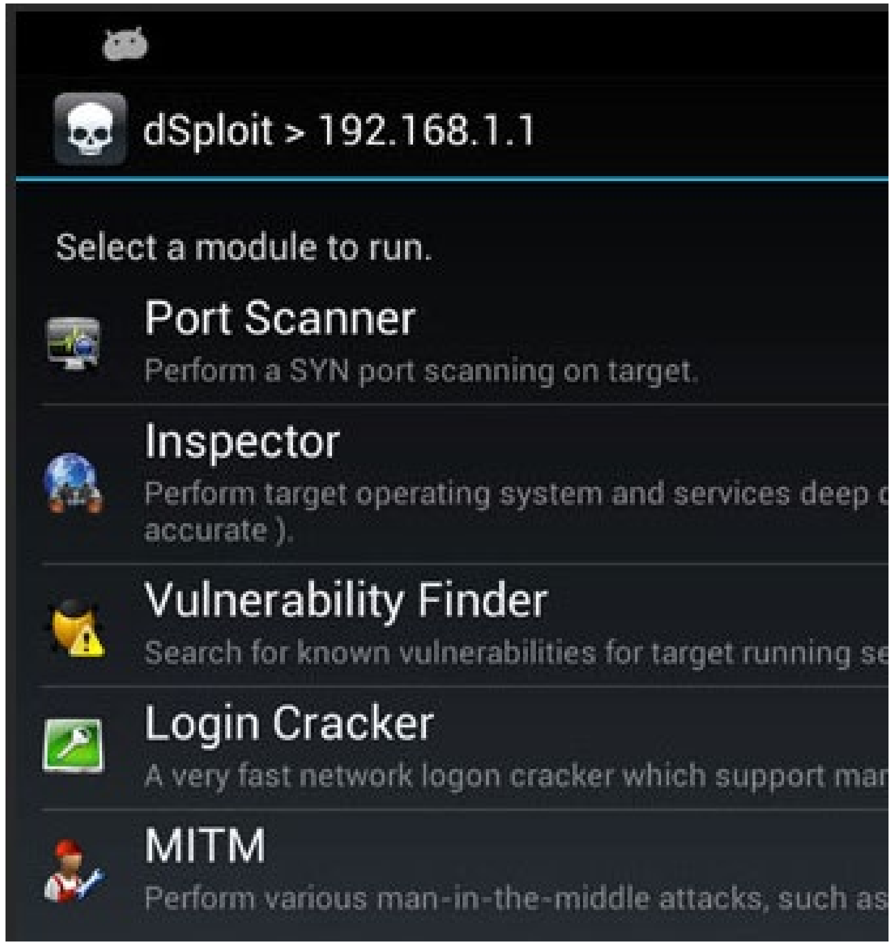
DroidSniff



```
Spoofing IP: 192.168.2.1
http://www.facebook.com]@http://w...
[redacted] [http://www.facebook.com]
IP: 192.168.2.6 Hostname: [redacted] PC
[redacted] [http://www.facebook.com]@http:/...
http://www.google.com
IP: 192.168.2.5 Hostname: [redacted] PC
http://www.google.com@http://www.google.com
http://www.google.com
IP: 192.168.2.5 Hostname: [redacted]
http://www.google.com@http://www.google.com
[redacted] [http://www.facebook.com]
IP: 192.168.2.7 Hostname: [redacted]
[redacted] [http://www.facebook.com]@http://w...
http://www.facebook.com
IP: 64.214.231.248 Hostname: [redacted] PC
ID: 1576267735
http://www.facebook.com
IP: 192.168.2.6 Hostname: [redacted] PC
ID: 1210568433
http://www.cyber-warrior.org
IP: 192.168.2.5 Hostname: [redacted] PC
ID: 423433796
```

MOBILE APP EXAMPLES (CONT'D)

dSploit



dSploit > 192.168.1.1

Select a module to run.

Port Scanner
Perform a SYN port scanning on target.

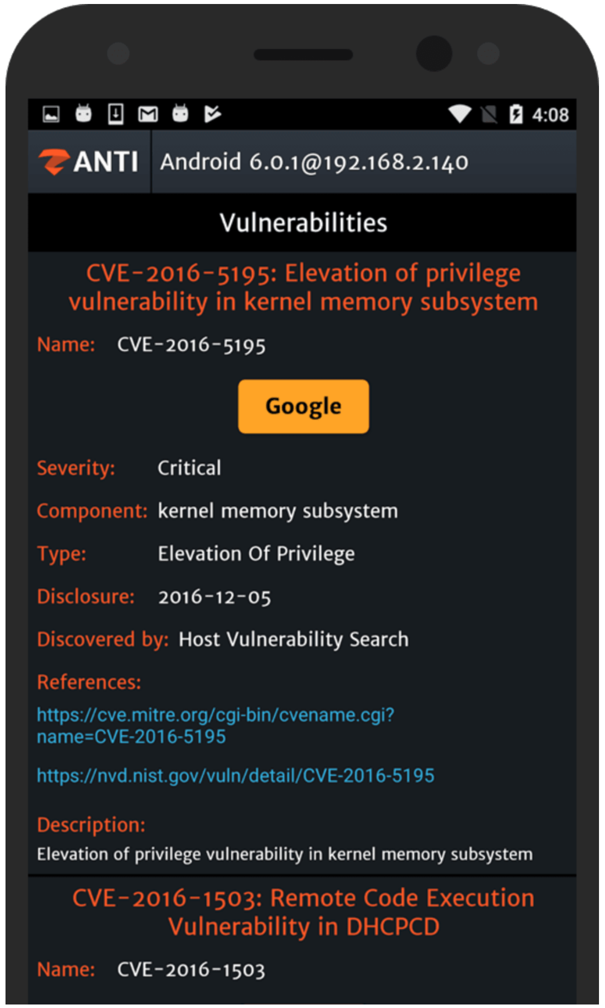
Inspector
Perform target operating system and services deep c
accurate).

Vulnerability Finder
Search for known vulnerabilities for target running se

Login Cracker
A very fast network logon cracker which support mar

MITM
Perform various man-in-the-middle attacks, such as

zANTI



ANTI Android 6.0.1@192.168.2.140

Vulnerabilities

CVE-2016-5195: Elevation of privilege vulnerability in kernel memory subsystem

Name: CVE-2016-5195

Google

Severity: Critical

Component: kernel memory subsystem

Type: Elevation Of Privilege

Disclosure: 2016-12-05

Discovered by: Host Vulnerability Search

References:

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5195

https://nvd.nist.gov/vuln/detail/CVE-2016-5195

Description:
Elevation of privilege vulnerability in kernel memory subsystem

CVE-2016-1503: Remote Code Execution Vulnerability in DHCPCD

Name: CVE-2016-1503

11.8 SESSION HIJACKING COUNTER-MEASURES

- Countermeasures

SESSION HIJACKING COUNTERMEASURES

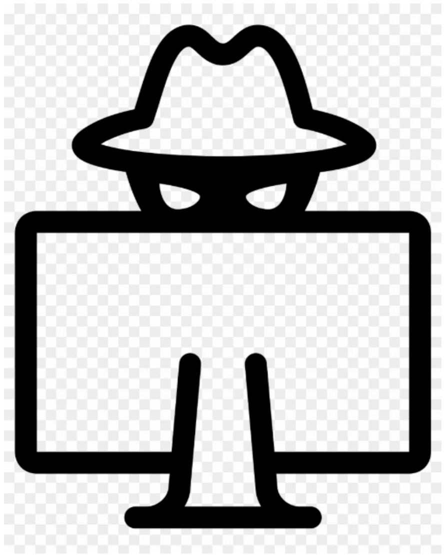
- Protect Session IDs:
  - Use unpredictable (randomized) Session IDs
  - Never use URLs with Sessions IDs
  - Don't Re-use Session IDs
- Use HTTP-Only on Cookies to help prevent XSS (Cross-Site Scripting)
- Regenerate the session key after authentication
- Limit incoming connections
- Minimize remote access
- Set absolute and inactive time limits on sessions
- Use Multi-Factor Authentication
- Use HTTPS or an IPSEC-based VPN to encrypt your connection
- Use OWASP cheat sheets for web app developer best practices.

11.9 SESSION HIJACKING REVIEW

- Review

SESSION HIJACKING REVIEW

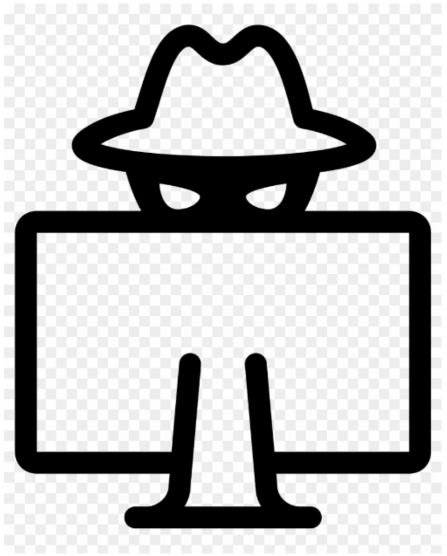
- In session hijacking, the attacker attempts to take over the client's session AFTER the user has authenticated
- Cookies and Java Web Tokens (JWTs) are the most common type of session token
- Session Sniffing (sidejacking) is where you passively sniff and capture the user's token
- Session Prediction is where you can guess what the next token value would be
- Cross-Site Scripting (XSS) is where the client trusts the server
  - The attacker injects malicious code which the client's browser executes in the background



- Stored (persistent) XSS stores the malicious code on a page that others will see
- Reflected XSS uses a web app's search or error functionality to send the malicious command, along with a session token, back to the user
- DOM XSS injects the malicious script into the victim's browser directly, superimposing it on top of a downloaded page

SESSION HIJACKING REVIEW

- Cross-Site Request Forgery (CSRF) is where the server trusts the client (authenticated user)
  - As the user does something else, the CSRF tricks the browser into sending unauthorized commands to the website, which the website will accept and execute
- Session Replay is where the attacker passively sniffs the client's session token and then uses it
- Session Fixation is where the attacker obtains a legitimate session token and then tricks the client into using it while authenticating



- CRIME and BREACH take advantage of protocol vulnerabilities
- MITM can be accomplished through ARP spoofing, ICMP redirect, DNS poisoning, and malicious links
- Source-routed hijacking uses the source routing field in an IP header to instruct routers to send traffic through a different path (the attacker)
  - Blind hijacking is used when source routing is not possible
  - The attacker can relay requests to the server, but cannot see the responses
- TCP session hijacking requires the attacker to predict the next TCP sequence number, deauthenticate the client, and take the client's place
- UDP hijacking intercepts a server's UDP response to a client, sending a fake response in its stead