

**SSTI VULNERABILITY**

**Project-Based Internship 2020 Report**

Submitted To

**DataRitz Technologies**

**Duration 7 weeks**

**By-**

Prajwal Srivastava

(**1803210109)**

**Abes Engineering College**

**Abdul kalam Technical University**

**Under the guidance of**

**KRISHNA VIR SINGH**

**CERTIFICATE**

This is to certify that Project Report entitled “SSTI VULNERABILITY” which is submitted by

Prajwal Srivastava in partial fulfillment of the requirement for the summer internship of

CISCO Certified Cyber Ops Associate” in Department of Information Technology of ABES

ENGINEERING COLLEGE is a record of the candidate's own work carried out by her under

my supervision.

**Supervisor**

**Date**

**ACKNOWLEDGEMENT**

*It gives us a great sense of pleasure to present the report of the Project Based*

*Internship 2020 undertaken during CISCO Cyber Ops Associate 2020. We owe*

*special debt of gratitude to Krishna Vir Singh, DataRitz Technologies for his constant*

*support and guidance throughout the course of our work. His constant motivation has*

*been a constant source of inspiration for us. It is only his cognizant efforts that our*

*endeavors have seen light of the day.*

*We also take the opportunity to acknowledge the contribution of team members of*

*DataRitz Technologies for their full support and assistance during the development of*

*the project.*

*We also do not like to miss the opportunity to acknowledge the motivation of*

*Information Technology Department and Abes Engineering College to provide us the*

*opportunity to undergo training at DataRitz Technologies.*

*Signature :* Prajwal Srivastava

*Name:* Prajwal Srivastava

*Roll No:* 1803210109

*Class :* CS - A

**TABLE OF CONTENTS**

DECLARATION

CERTIFICATE

ACKNOWLEDGEMENT

TABLE OF CONTENT

**CHAPTER-1 COURSE DESCRIPTION**

CISCO Certified Cyber Ops Associate

**CHAPTER-2 INTRODUCTION**

1 Aim of the Project

2 Objective of the Project

3 Scope of the Project

**CHAPTER-3 DESCRIPTION**

## What are Template Engines?

1. OVERVIEW TO SSTI

2. DESCRIPTION OF SSTI

3. DETECTION OF SSTI

3.1. Plain Text Context

3.2. Code Context

4. WORKING OF SSTI

5. SSTI CAN BE USED FOR

6. IMPACT OF SSTI

7. PREVENTION OF SSTI

**CHAPTER-4 TOOL DESCRIPTION**

1 INTRODUCTION TO BURP SUITE

2 DIFFERENT TOOLS IN BURP SUITE

Spider

Proxy

Intruder

Repeater

Sequencer

Decoder

Extender

Scanner

**CHAPTER-5 IMPLEMENTATION AND RESULTS**

IMPLEMENTATION

Vulnerable URL

Vulnerable Parameter

Payloads

How to reduce this vulnerability

Impacts

SCREENSHOT OF RESULT

Portswigger Academy Lab 1

XVWA Virtual Environment Exploitation

Burpsuite Tool

REFERENCES

**CHAPTER 1**

**COURSE DESCRIPTION**

**CISCO Certified Cyber Ops Associate**

Cisco's CCNA Cyber Ops certification provides individuals with the

knowledge to identify and respond to security incidents. This certification

provides a path to working in a Security Operations Center (SOC) and

security positions. As a CCNA level certification, Cyber Ops provides

introductory knowledge so one may be aware of the security landscape,

understand security concepts and general networking. We learn topics

such as networking concepts and IP addressing, as well as security

concepts including access control models, risk assessment, and the CIA

triad. We will also review cryptography methods and host-based analysis

details, as well as security monitoring tools, and attack methods used by

threat actors.

The program has one training course and one exam that covers the

foundational skills, processes, and Knowledge you need to prevent, detect,

analyze and respond to cybersecurity incidents as per SOC team

Main topics are:

* Security Concepts
* Security Monitoring
* Different OS - Windows , Linux
* Host-based Analysis
* Network Intrusion Analysis
* Security Policies and Procedures
* Access Control Model for Digital Assets
* Malware Analysis and Implementation
* Cryptography and the Public Key Infrastructure
* Incident Response and Handling

**CHAPTER 2 INTRODUCTION**

**To Project**

1. **Aim of the Project:** To learn and explore Server side template injection (SSTI) vulnerability.

**2 Objective of the Project:** To have a whole idea about the

Server side template Injection attack -

What is SSTI ?

How is it performed ?

What are the impacts of SSTI attack ?

**3. Scope of the Project:**  Server Side Template In is in the

current OWASP Top Ten Most Critical Web Application Security Risks –

and the second most prevalent web application vulnerability. It is thought to

exist in two-thirds of all applications.

SSTI vulnerabilities are easily discovered by attackers. Respectively, they

must also be easily discoverable by defenders

**CHAPTER 3**

**DESCRIPTION**

Web applications frequently use template systems such as “Twig” and “FreeMarker” to embed dynamic content in web pages and emails. Template Injection occurs when user input is embedded in a template in an unsafe manner. Consider a marketing application that sends bulk emails, and uses a Twig template to greet recipients by name.

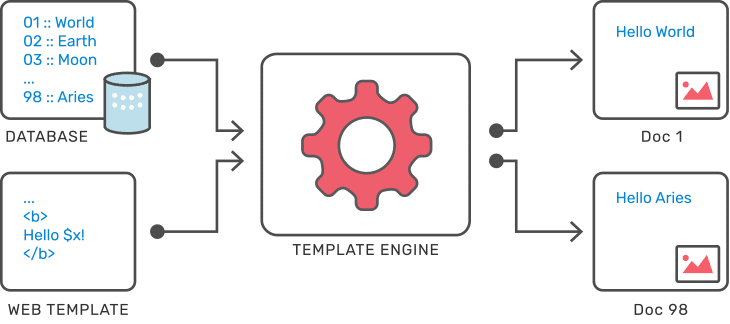
Template Injection can arise both through developer error, and through the intentional exposure of templates in an attempt to offer rich functionality, as commonly done by wikis, blogs, marketing applications and content management systems. Intentional template injection is such a common use-case that many template engines offer a 'sandboxed' mode for this express purpose. Unlike XSS, Template Injection can be used to directly attack web servers' internals and often obtain Remote Code Execution (RCE) turning every vulnerable application into a potential pivot point.

## What are Template Engines?

Web developers uses template engines populate dynamic data into modern web pages. This enable them to separate business logic with presentation logic. When web pages come from a web template, they can structure the component of web pages in such a way that can be modified independently of each other. A component can include anything like header, footer, content such as videos, images, audio. Templates Engines are commonly used to:

* Displays information about users, products, companies
* Displays gallery of photos, videos.
* Sell products online
* Sends bulk emails

### **Some examples of Template Engines are:**



**Description of SSTI:**

Intentional template injection is such a common use-case that many template engines offer a 'sandboxed' mode for this express purpose. This paper defines a methodology for detecting and exploiting template injection, and shows it being applied to craft RCE zerodays for two widely deployed enterprise web applications. Generic exploits are demonstrated for five of the most popular template engines, including escapes from sandboxes whose entire purpose is to handle user-supplied templates in a safe way

custom\_email = {{7\*7}}  
 output : 49

**Detection**

This vulnerability can appear in two distinct contexts, each of which requires its own detection method

#### **1. Plaintext context**

Most template languages support a freeform 'text' context where you can directly input HTML. It will typically appear in one of the following ways:

smarty=Hello {user.name}  
Hello user1freemarker=Hello ${username}  
Hello newuserany=<b>Hello</b>  
<b>Hello<b>

This frequently results in XSS, so the presence of XSS can be used as a cue for more thorough template injection probes. Template languages use syntax chosen explicitly not to clash with characters used in normal HTML, so it's easy for a manual blackbox security assessment to miss template injection entirely. To detect it, we need to invoke the template engine by embedding a statement. There are a huge number of template languages but many of them share basic syntax characteristics. We can take advantage of this by sending generic, template-agnostic payloads using basic operations to detect multiple template engines with a single HTTP request:

smarty=Hello ${7\*7}  
Hello 49  
freemarker=Hello ${7\*7}

Hello 49

#### **Code context**

User input may also be placed within a template statement, typically as a variable name:

personal\_greeting=username  
Hello user01

This variant is even easier to miss during an assessment, as it doesn't result in obvious XSS and is almost indistinguishable from a simple hashmap lookup. Changing the value from username will typically either result in a blank result or the application erroring out. It can be detected in a robust manner by verifying the parameter doesn't have direct XSS, then breaking out of the template statement and injecting HTML tag after it:

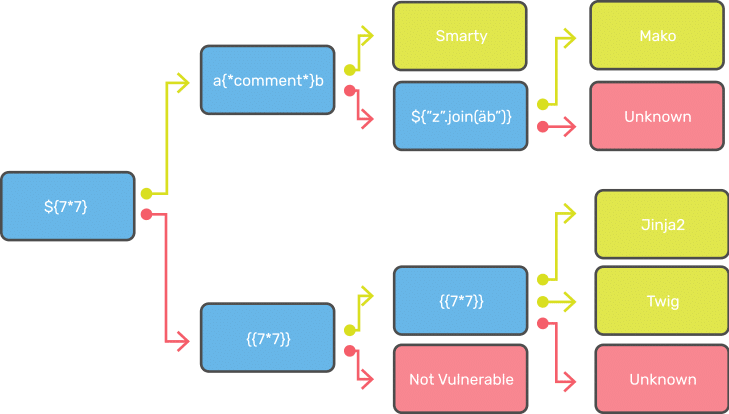
personal\_greeting=username<tag>  
Hello  
  
personal\_greeting=username}}<tag>  
Hello user01 <tag>

### **[Identify](https://portswigger.net/research/server-side-template-injection" \l "Identify)**

After detecting template injection, the next step is to identify the template engine in use. This step is sometimes as trivial as submitting invalid syntax, as template engines may identify themselves in the resulting error messages.

However, this technique fails when error messages are supressed, and isn't well suited for automation. We have instead automated this in Burp Suite using a decision tree of language-specific payloads.

Green and red arrows represent 'success' and 'failure' responses respectively. In some cases, a single payload can have multiple distinct success responses - for example, the probe {{7\*'7'}} would result in 49 in Twig, 7777777 in Jinja2, and neither if no template language is in use.



### **Exploit**

***Read :*** The first step after finding template injection and identifying the template engine is to read the documentation. The importance of this step should not be underestimated; one of the zeroday exploits to follow was derived purely from studious documentation perusal. Key areas of interest are:

* 'For Template Authors' sections covering basic syntax.
* 'Security Considerations' - chances are whoever developed the app you're testing didn't read this, and it may contain some useful hints.
* Lists of builtin methods, functions, filters, and variables.
* Lists of extensions/plugins - some may be enabled by default.

***Explore* :** Assuming no exploits have presented themselves, the next step is to explore the environment to find out exactly what you have access to. You can expect to find both default objects provided by the template engine, and application-specific objects passed in to the template by the developer. Many template systems expose a 'self' or namespace object containing everything in scope, and an idiomatic way to list an object's attributes and methods

***Attack :*** At this point you should have a firm idea of the attack surface available to you and be able to proceed with traditional security audit techniques, reviewing each function for exploitable vulnerabilities.

It's important to approach this in the context of the wider application - some functions can be used to exploit application-specific features. The examples to follow will use template injection to trigger arbitrary object creation, arbitrary file read/write, remote file include, information disclosure and privilege escalation vulnerabilities

## Exploit development

## Sometimes, thirty seconds of documentation perusal is sufficient to gain RCE. For example, exploiting unsandboxed Smarty is as easy as:

{php}echo `id`;{/php}

Mako is similarly easy to exploit:

## <% import os x=os.popen('id').read() %> ${x}

**Impact of SSTI**

Server-side template injection can impact in various ways from Information disclosure to XSS to Remote Code Execution.

If user-supplied templates are a business requirement, how should they be implemented? We have already seen that regexes are not an effective defense, and parser-level sandboxes are error prone. The lowest risk approach is to simply use a trivial template engine such as Mustache, or Python's Template. MediaWiki has taken the approach of executing users' code using a sandboxed Lua environment where potentially dangerous modules and functions have been outright removed. This strategy appears to have held up well, given the lack of people compromising Wikipedia. In languages such as Ruby it may be possible to emulate this approach using monkey-patching.

**PREVENTION OF SSTI**

As we now know about Template injection, and how to identify and Exploit it, Now let’s move to the mitigation part. Mitigation defers depending on which Template Engine is being used. Below are the best-suggested mitigations:

1. Framework and Library updates.
2. Input Sanitization.
3. Sandboxing.

Template engines are server-side sandboxes. As a result, allowing untrusted users to edit templates introduces an array of serious risks, which may or may not be evident in the template system's documentation. Many modern technologies designed to prevent templates from doing harm are currently immature and should not be relied on except as a defense in depth measure. When Template Injection occurs, regardless of whether it was intentional, it is frequently a critical vulnerability that exposes the web application, the underlying webserver, and adjacent network services.

**CHAPTER-4**

**TOOL DESCRIPTION**

**Tool Name: Burp Suite**

**Introduction to Burp Suite:**

Burp or Burp Suite is a set of tools used for penetration testing of web applications. It is developed by the company named Portswigger, which is also the alias of its founder Dafydd Stuttard. BurpSuite aims to be an all in one set of tools and its capabilities can be enhanced by installing add-ons that are called BApps. it is the most popular tool among professional web app security researchers and bug bounty hunters. Its ease of use makes it a more suitable choice over free alternatives like OWASP ZAP. Burp Suite is available as a community edition which is a free, professional edition that costs $399/year and an enterprise edition that costs $3999/Year.

**Different tools in Burp Suite:**

**Spider:** It is a web spider/crawler that is used to map the target web application. The objective of the mapping is to get a list of endpoints so that their functionality can be observed and potential vulnerabilities can be found. Spidering is done for a simple reason that the more endpoints you gather during your recon process, the more attack surfaces you possess during your actual testing.

**Proxy :** BurpSuite contains an intercepting proxy that lets the user see and modify the contents of requests and responses while they are in transit. It also lets the user send the request/response under monitoring to another relevant tool in BurpSuite, removing the burden of copy-paste. The proxy server can be adjusted to run on a specific loop-back ip and a port. The proxy can also be configured to filter out specific types of request-response pairs.

**Intruder :** Burp Intruder works by taking an HTTP request (called the "base request"), modifying the request in various systematic ways, issuing each modified version of the request, and analyzing the application's responses to identify interesting features.

For each attack, you must specify one or more sets of [payloads](https://portswigger.net/burp/documentation/desktop/tools/intruder/payloads), and the [positions](https://portswigger.net/burp/documentation/desktop/tools/intruder/positions) in the base request where the payloads are to be placed. Numerous methods of generating payloads are available (including simple lists of strings, numbers, dates, brute force, bit flipping, and many others). Payloads can be placed into payload positions using [different algorithms](https://portswigger.net/burp/documentation/desktop/tools/intruder/positions" \l "attack-type). Various tools are available to help analyze the results and identify interesting items for further investigation

Burp Intruder is a very flexible tool and can help automate all kinds of tasks when testing web applications. The most common use cases for Intruder fall into the following categories:

* [Enumerating identifiers](https://portswigger.net/burp/documentation/desktop/tools/intruder/using" \l "enumerating-identifiers)
* [Harvesting useful data](https://portswigger.net/burp/documentation/desktop/tools/intruder/using" \l "harvesting-useful-data)
* [Fuzzing for vulnerabilities](https://portswigger.net/burp/documentation/desktop/tools/intruder/using" \l "fuzzing-for-vulnerabilities)

**Repeater:**

Burp Repeater is a simple tool for manually manipulating and reissuing individual HTTP and WebSocket messages, and analyzing the application's responses. You can use Repeater for all kinds of purposes, such as changing parameter values to test for input-based vulnerabilities, issuing requests in a specific sequence to test for logic flaws, and reissuing requests from [Burp Scanner issues](https://portswigger.net/burp/documentation/desktop/tools/target/site-map" \l "issues-view) to manually verify reported issues.

The main Repeater UI lets you work on multiple different messages simultaneously, each in its own tab. When you send messages to Repeater, each one is opened in its own numbered tab. You can rename tabs by double-clicking the tab header.

### Sending HTTP requests

### HTTP request history

**Sequencer:**

The sequencer is an entropy checker that checks for the randomness of

tokens generated by the webserver. These tokens are generally used for

authentication in sensitive operations: cookies and anti-CSRF tokens are

examples of such tokens. Ideally, these tokens must be generated in a fully

random manner so that the probability of appearance of each possible

character at a position is distributed uniformly. This should be achieved

both bit-wise and character-wise. An entropy analyzer tests this hypothesis

for being true. It works like this: initially, it is assumed that the tokens are

random. Then the tokens are tested on certain parameters for certain

characteristics. A term significance level is defined as a minimum value of

probability that the token will exhibit for a characteristic, such that if the

token has a characteristic probability below significance level, the hypothesis that the token is random will be rejected. This tool can be used to find out the weak tokens and enumerate their construction.

**Decoder:**

Decoder lists the common encoding methods like URL, HTML, Base64, Hex, etc. This tool comes handy when looking for chunks of data in values of parameters or headers. It is also used for payload construction for various vulnerability.

**Extender:**

BurpSuite supports external components to be integrated into the tools suite to enhance its capabilities. These external components are called BApps. These work just like browser extensions. These can be viewed, modified, installed, uninstalled in the Extender window. Some of them are supported on the community version, but some require the paid professional version.

**Scanner:**

The scanner is not available in the community edition. It scans the website automatically for many common vulnerabilities and lists them with information on confidence over each finding and their complexity of exploitation. It is updated regularly to include new and less known vulnerabilities.

**CHAPTER 5**

**IMPLEMENTATION AND RESULTS**

**CHAPTER 5**

**IMPLEMENTATION AND RESULTS**

**CHAPTER 5**

**IMPLEMENTATION AND RESULTS**

**Implementation :**

Using Portswigger Lab the Vulnerability SSTI can be exploit

And also using the XVWA virtual Machine to exploit the SSTI Vunerability and also executing Remote Code Execution (RCE) to gets its user,id,pwd etc.

**Vulnerable Url:**

<https://ac9d1f7a1f5685c980212e82005900a6.web-security-academy.net/>

**Vulnerable Parameter :**

1. “View Details” button (Portswigger Lab)
2. Search box (in XVWA Machine)

**Payloads**

1. <%=7\*7%> : it is encoded in url parameter
2. <%= system("rm /home/carlos/morale.txt") %>
3. In XVWA {{7\*7}}
4. And {{7\*’7’}}
5. XVWA RCE : {{\_self.env.registerUndefinedFilterCallback("exec")}}{{\_self.env.getFilter("id")}}

**How to reduce this vulnerability:**

Template Injection is only apparent to auditors who explicitly look for it, and may incorrectly appear to be low severity until resources are invested in assessing the template engine's security posture. This explains why Template Injection has remained relatively unknown up till now, and its prevalence in the wild remains to be determined.

**Impacts**

Server-side template injection can impact in various ways from Information disclosure to XSS to Remote Code Execution.

**Screenshots of Result :**

●

Verifying whether the user-supplied values are being verified.

●

Verifying whether the user-supplied values are being verified.

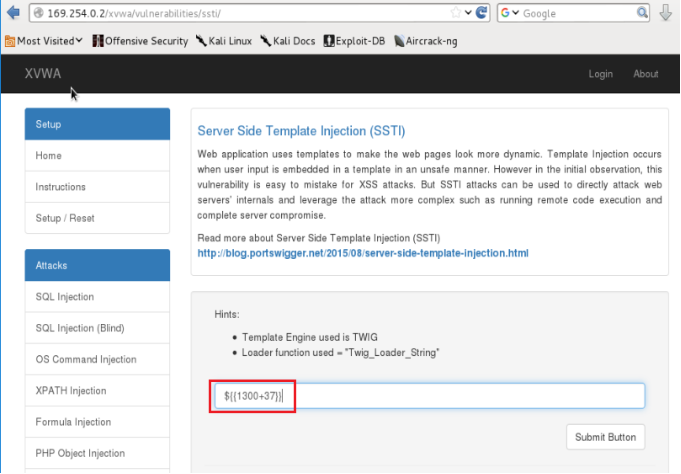
ifying whether the

●

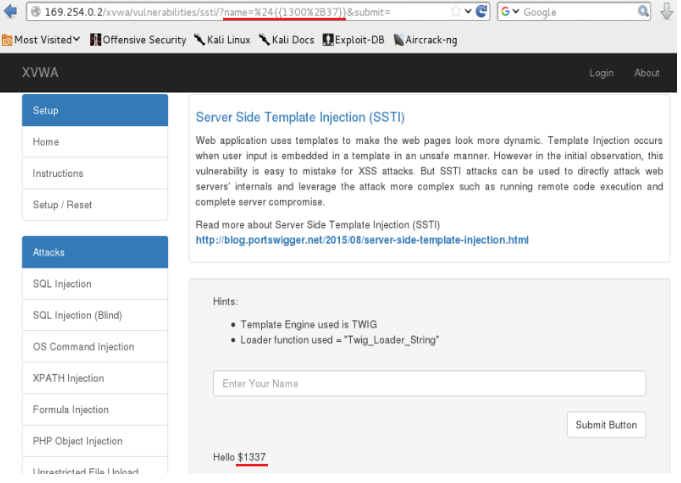
●

**Screenshots of Result :**

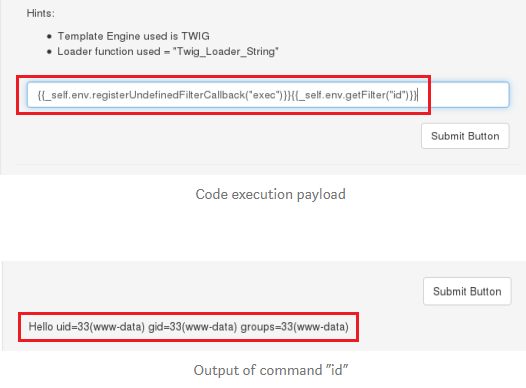
**Input :** ${{300+37}} or {{7\*7}} or {{7\*”7”}}

****

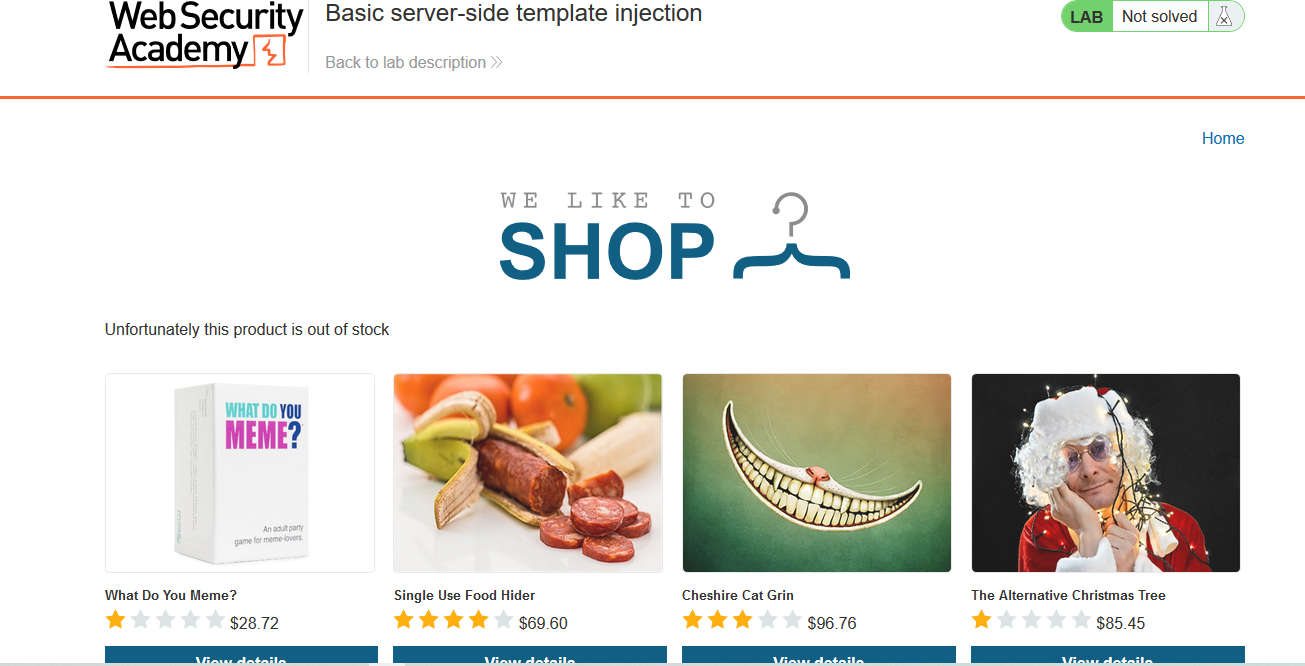
Output : $1337

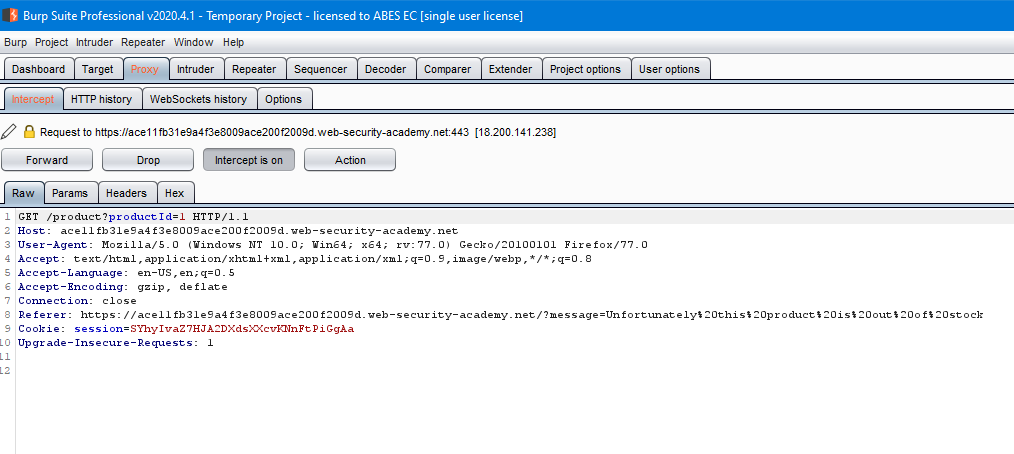


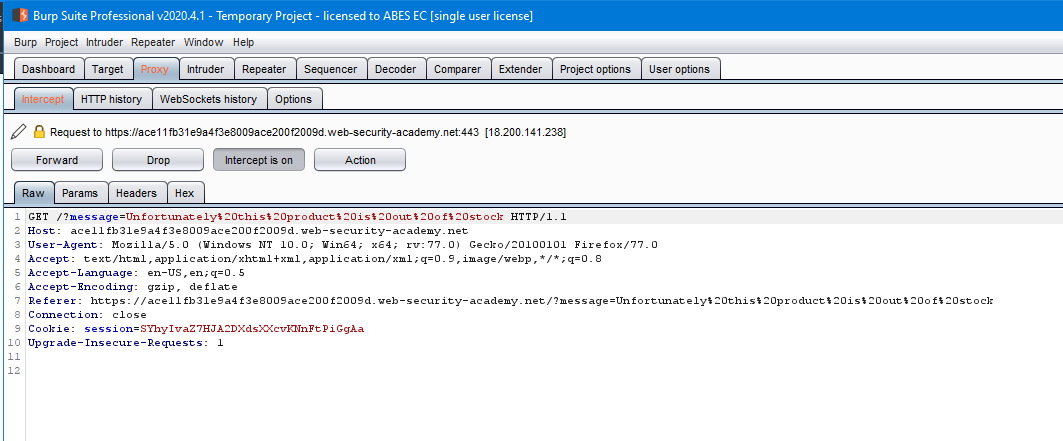
Taking Remote Access over the XVWA Machine

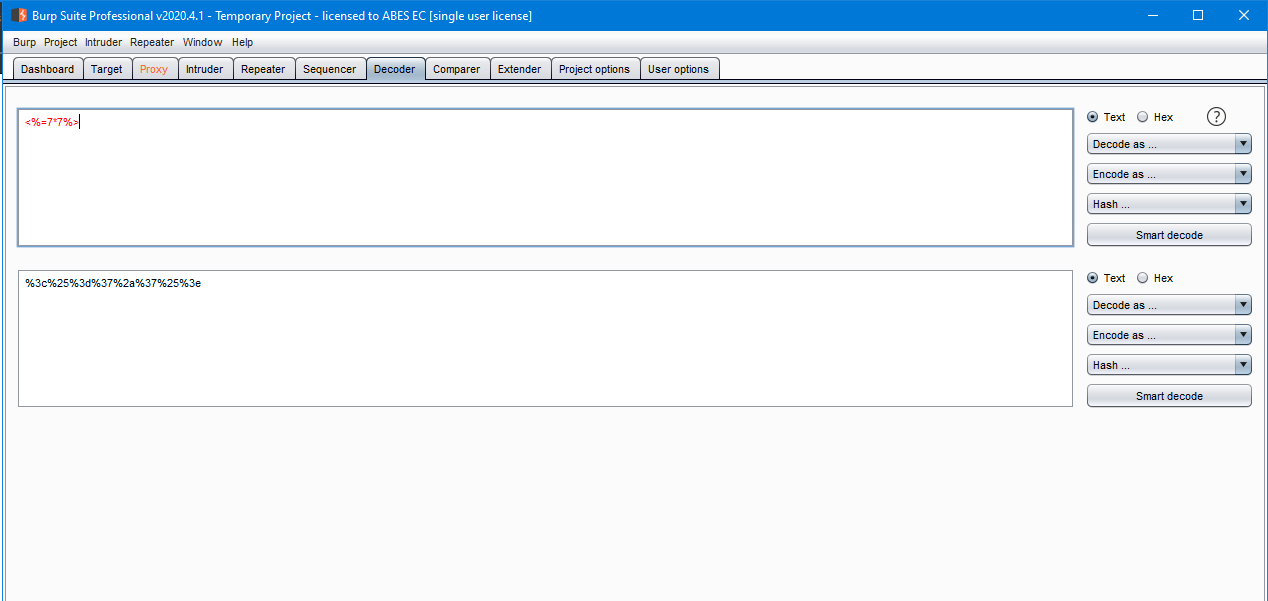
sd

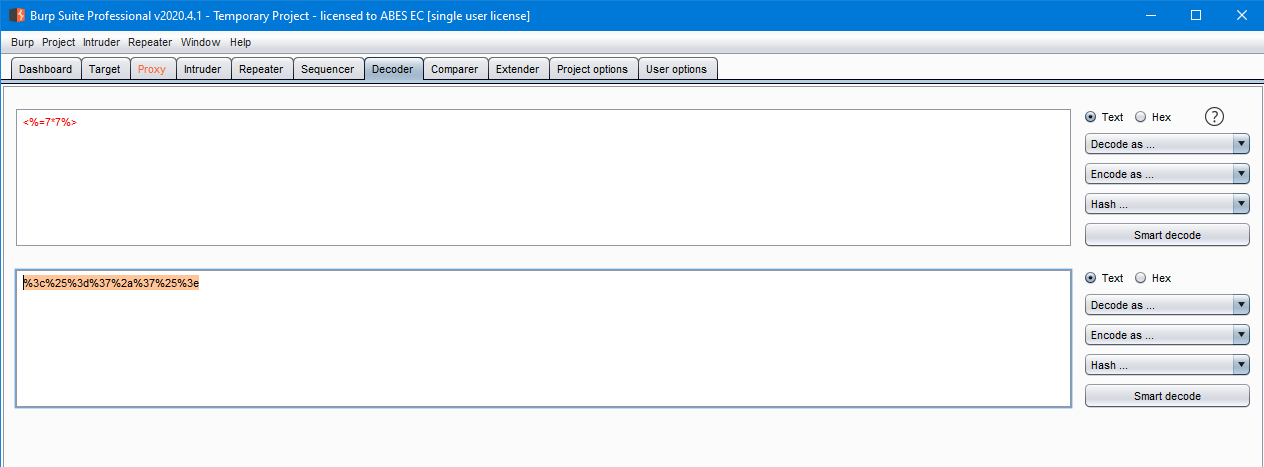
Working In Portswigger Lab :

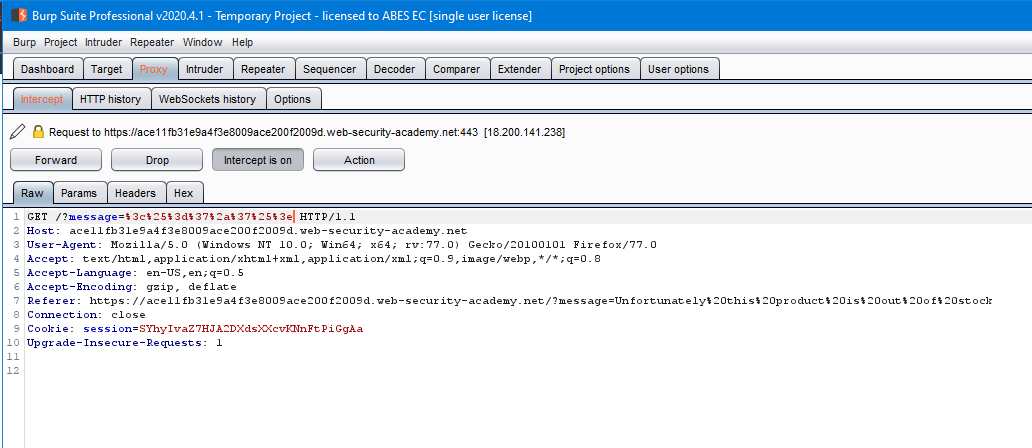


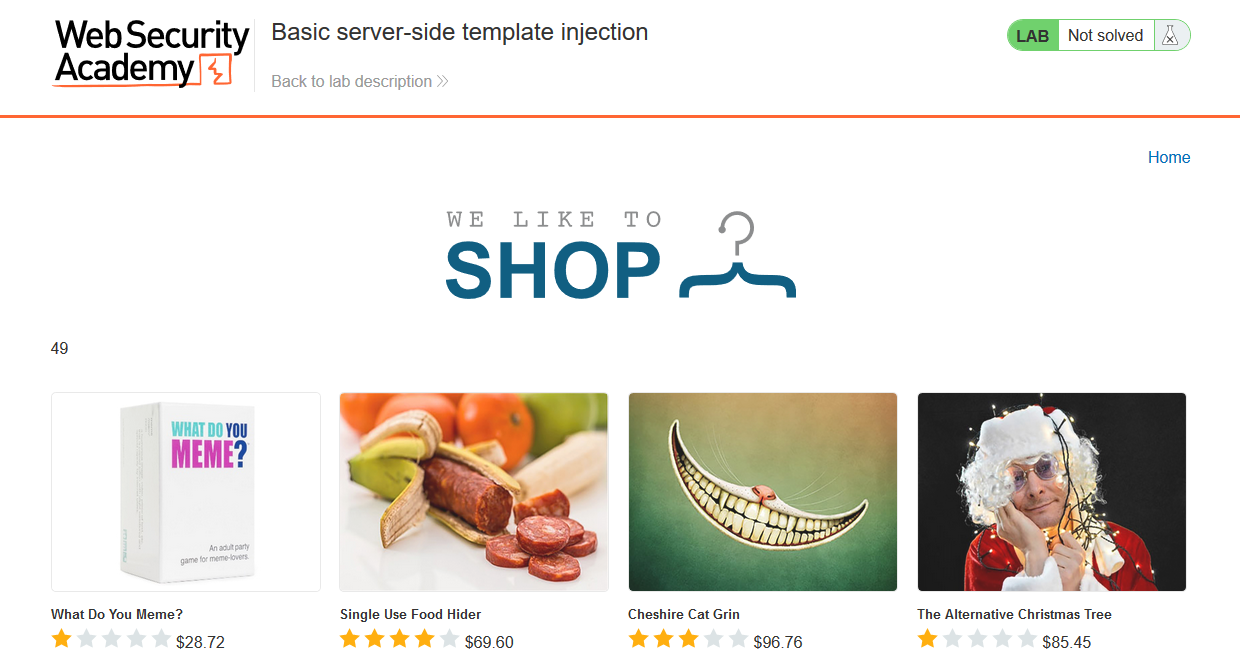


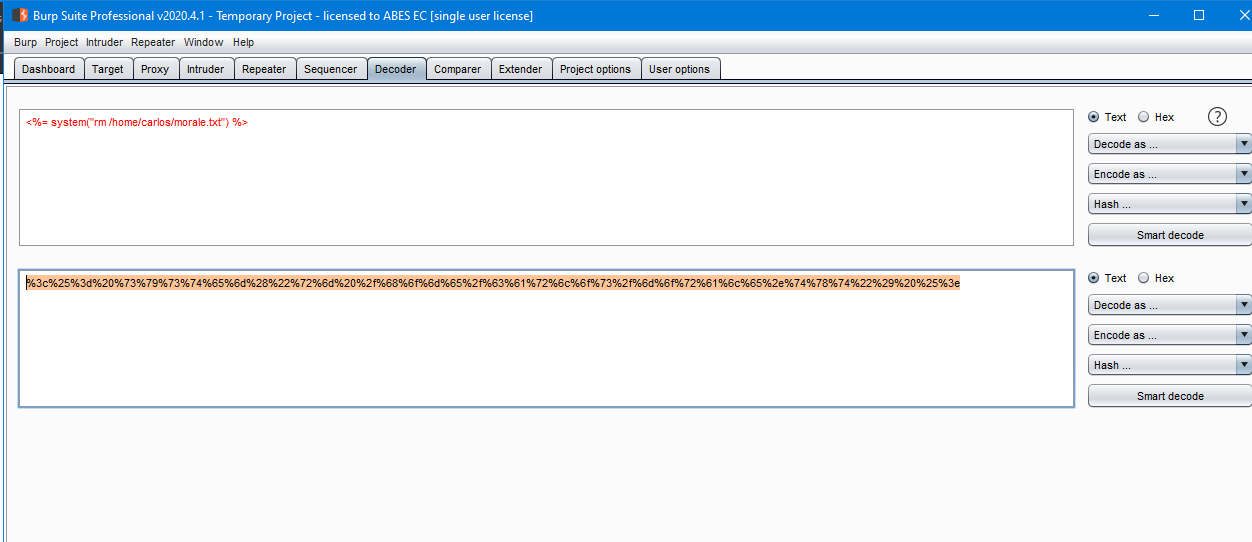


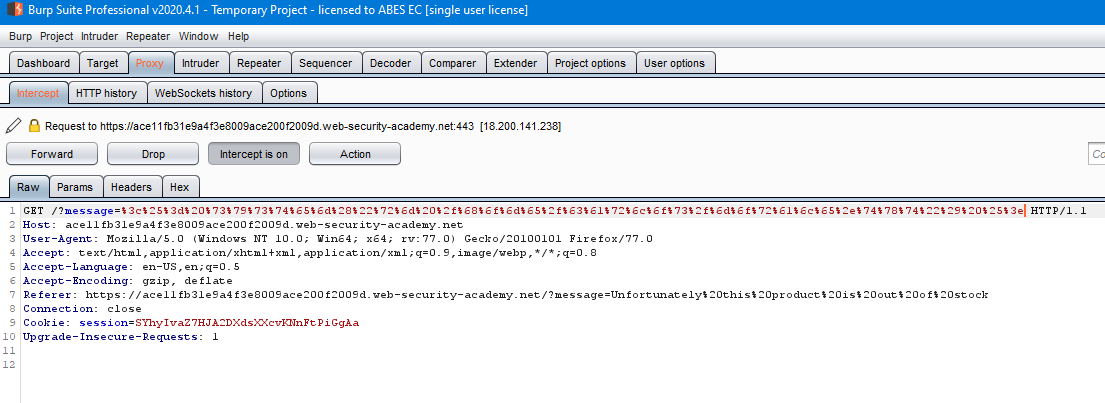


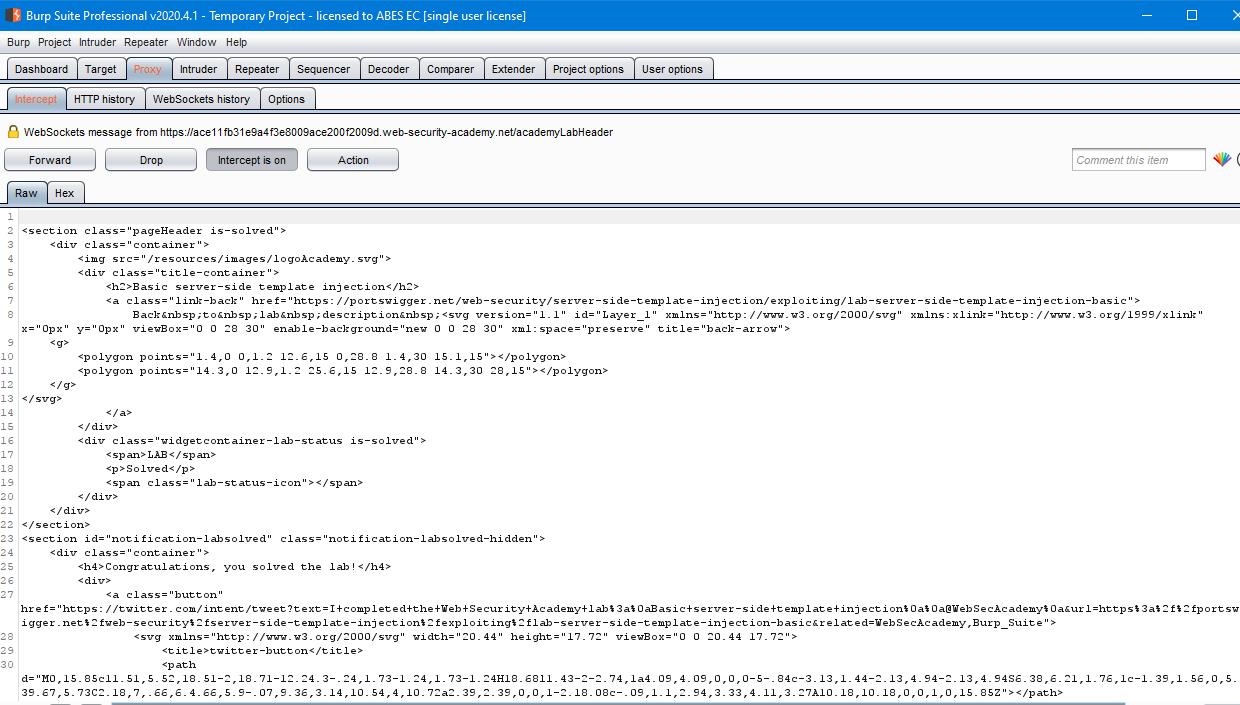


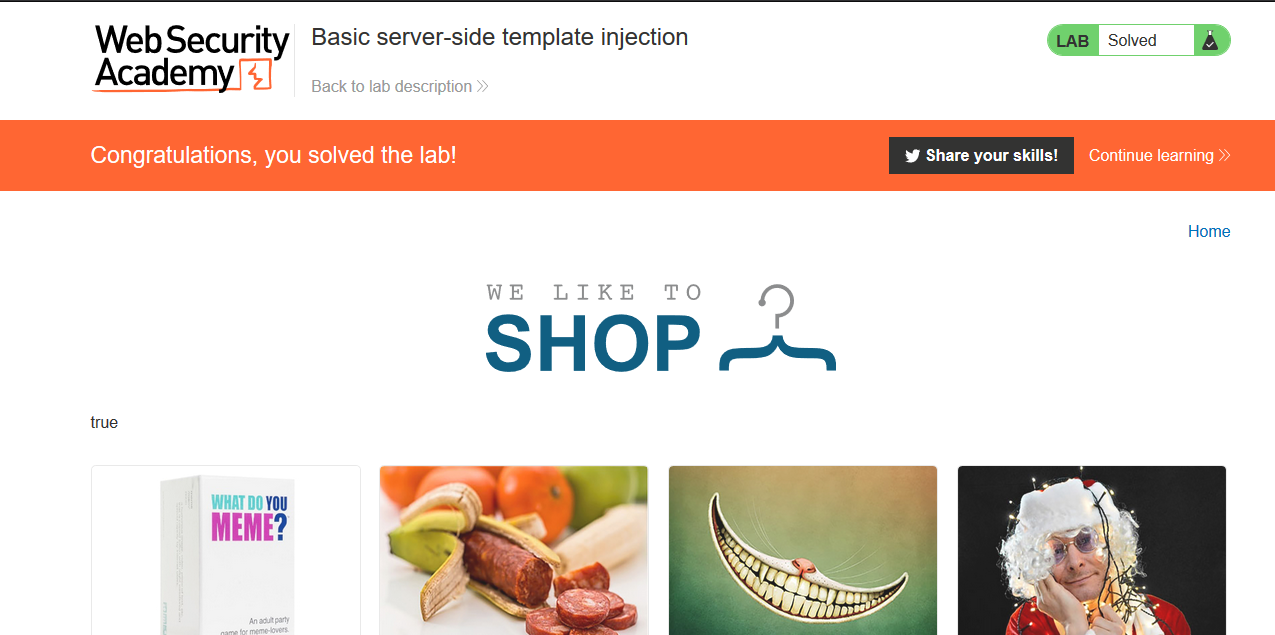












**REFERENCES**

1. **Portswigger Academy:**

<https://portswigger.net/web-security/server-side-template-injection/exploiting/lab-server-side-template-injection-basic>