בס"ד

A TEE Password Vault – with a Google Chrome Extension

By David Berger and Eyal Seckbach
At Lev Academic Institute
(a.k.a. Jerusalem College of Technology)
June 2022 (2nd Semester)

Faculty: Computer Science
Course: Trusted Execution Environment
Instructor: Barak Einav

TABLE OF CONTENTS

Motivation of Product:

What’s on the market now:

There are many different programs (*software*) which store a User’s passwords (and websites), often storing them in a cloud, theoretically accessible from any location.

Our idea - A Local Password Vault in a TEE:

Instead of storing this sensitive data in a cloud/software, we have a *hardware* solution - we will store passwords in a **local TEE (Trusted Execution Environment)**. A Host app on the computer can communicate with the TEE, and open a WebSocket (with local host) which will communicate with a Google Chrome Extension, allow the User to quickly save pairs of URLs and passwords

General Strengths of a local Password Vault (vs software):

- Hardware is often more secure than software (objectively); less bugs and backdoors, less attack surfaces (attacker must run on the local machine, and listen to sockets)
- Implementation, management is much simpler

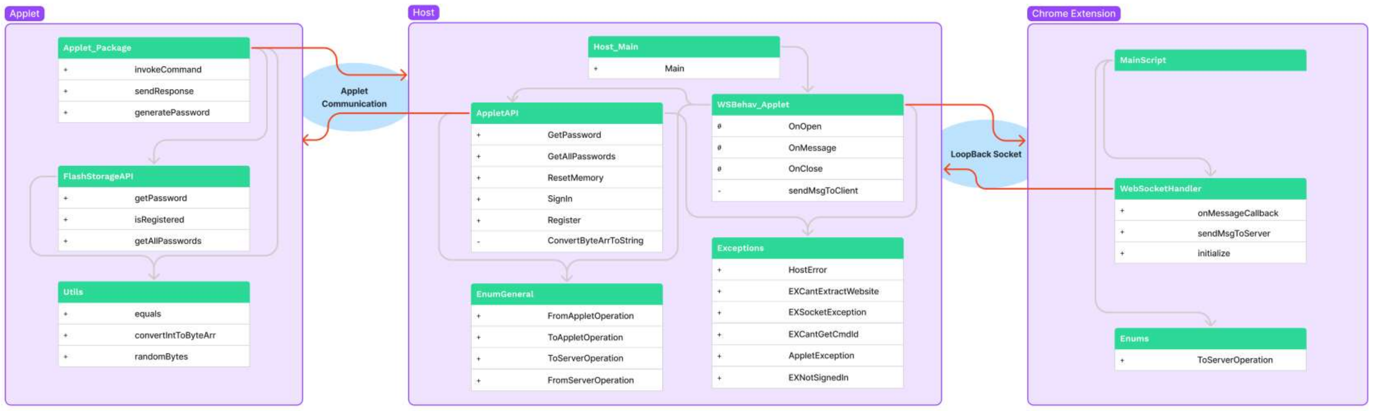
General Weaknesses of a local Pwd Vault (vs software):

- Inaccessible if user is not near computer
- Not easily portable to another computer (even to switch permanently)
- Not as scalable
- If something happens to the Hardware, the information is lost

However - in our implementation - Our password vault's hardware is secured - see Full Security Analysis.

Overview

Our project has 3 parts:



(1) The Chrome Extension (**JavaScript, HTML** - independently learned) -- the UI
(2) The Host/Server Application (**C#**)
(3) The Applet that runs in a **TEE** (Trusted Execution Environment) using Intel's DAL (**java**) -- holds the Data

Details

The extension communicates with the Server via a **Websocket**, and the Host communicates with the Applet with **Byte arrays** (Intel's built-in system).
The extensions pulls the URL from the browser, and sends to the Host (and to the Applet). The Applet returns a randomly generated password (using **Sha1 algorithm**, based on a **time seed** given by Host). The pairs of URL's and Password are stored in the Applet's secure **Flashstorage**. All communication is echoed in the console opened by the a Host (helpful for debugging).
For **2-point security**, the user must use his computer ("*what you have*") and enter a master password ("*what you know*") before accessing any information.
The extension pulls the URL from the browser and **injects a script** into the browser (using **Chrome's ActiveTab**), automatically locating and entering the password in the appropriate text box on the website.
The application is versatile, and can be run in chrome, or in any JS IDE (just uncheck the checkbox)

Source Code: https://github.com/David-YY-Berger/Password-Vault-Chrome-Extension

Full Security Analysis:

The Attacker:

A Password Vault would be attacked by anyone wanted to access the data inside. The Attacker probably will not care about changing the data ("**integrity**) because even if the password vault returns false passwords, the website will not function for the user. It is possible that he will want to generate his own passwords and supply them to the user, but at that point he would have to be running on the user's local machine. Furthermore - he could not access any previous passwords, rather he could just compromise future passwords or new websites that the User visits (it is likely that by then, if he is running on the local machine, he will have already done other damage....).
Availability is also an unlikely target...A user would never pay a high ransom for his passwords back, he would just create new passwords with the website. Again, the Attacker cannot actually access the passwords within the fault.
The only real motivation here would be to attack the Vault's **Confidentiality**. To protect against this attack, the information is secured in the TEE, and protected by 2-point security (as mentioned in the Overview).

Assets:

Our top priority is the (1) **passwords in the Vault**. This is protected well by the TEE's structure.
Our second priority is the (2) **algorithm to generate the passwords**. If the attacker understands our algorithm - he can easily produce identical passwords, and has effectively broken into our vault. We understand that *Security through Obscurity* is foolish, and therefore use a **Sha1 Algorithm** and a **time seed** (mentioned in Overview). This is not an ideal solution; the random seed is based on time, which is not a genuinely random variable... and therefore before marketing this product, we would generate the seed based on a *truly random variable* (like the User's move of a mouse, or temperature that day, or the combination of different factors).

Attack Surfaces:

User input - The current version does not protect against malicious user inputs (like an **SQL injection**). However, because the chrome extension programmatically takes the URL, there is very little user input in general (except for signing in, and register) - most operations are done thru buttons. And therefore our program is relatively user to protect.
Sniffing - This is potentially a huge problem - because if the Attacker simply **audits** the communication over the WebSocket, he has effectively breached

any password retrieved by the User. However - because all communication runs on the **localhost** (127.0.0.0), in order to sniff this channel, the attacker must physically be on this computer (which is unlikely). And even if the attacker takes the machine - everything is protected by 2 point security.
The only concern is if the attackers manages to run a malicious code on the user's computer (auditing the messages) which can replay them to the attacker from afar. We assume that the user has a safe PC.
Hardware Attack - quote from Intel's website:

"Intel Trusted Execution Technology provides hardware-based mechanisms that help protect against software-based attacks and protects the confidentiality and integrity of data stored or created on the client PC."
And therefore, we are not concerned about a hardware breach.

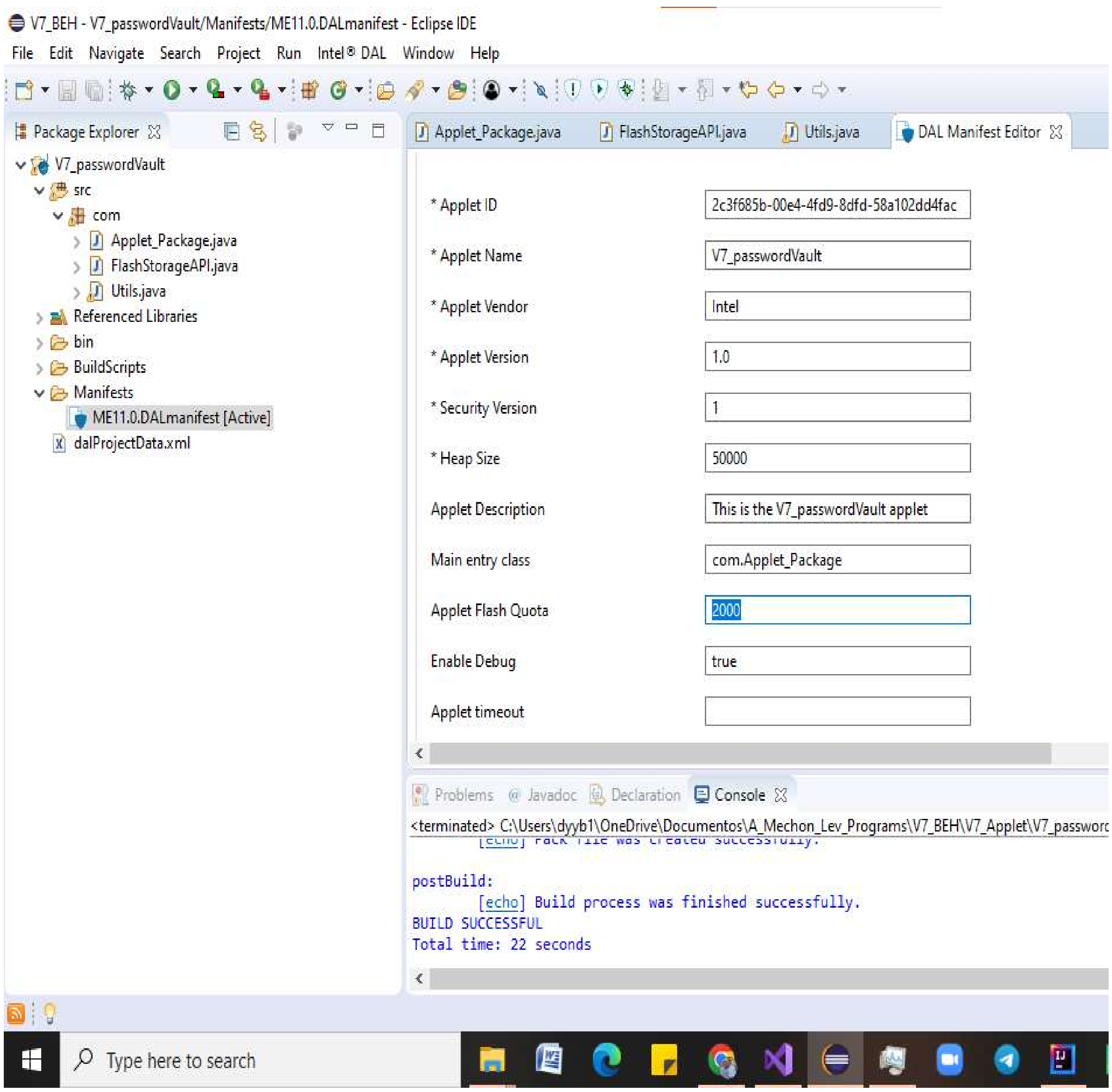
Response to a Breach:

We would recommend that the User the "GetAllPasswords" functionality to see the websites saved by our application, and then immediately resets the data in the Vault. Then, the User must change his personal password of every website. The Attacker would be immediately stopped. We would then require the User to notify us, and we would track (from the User's computer's **Ledger**) when the attacker broke in, understand the attack surface breached, and protect another breach in the future. Lastly, we would change our random see, and continue to market our product.

How to run our Program (after downloading the Source Code):

You must download Intel's DAL: https://www.intel.com/content/www/us/en/developer/tools/dal/sdk-get-started.html

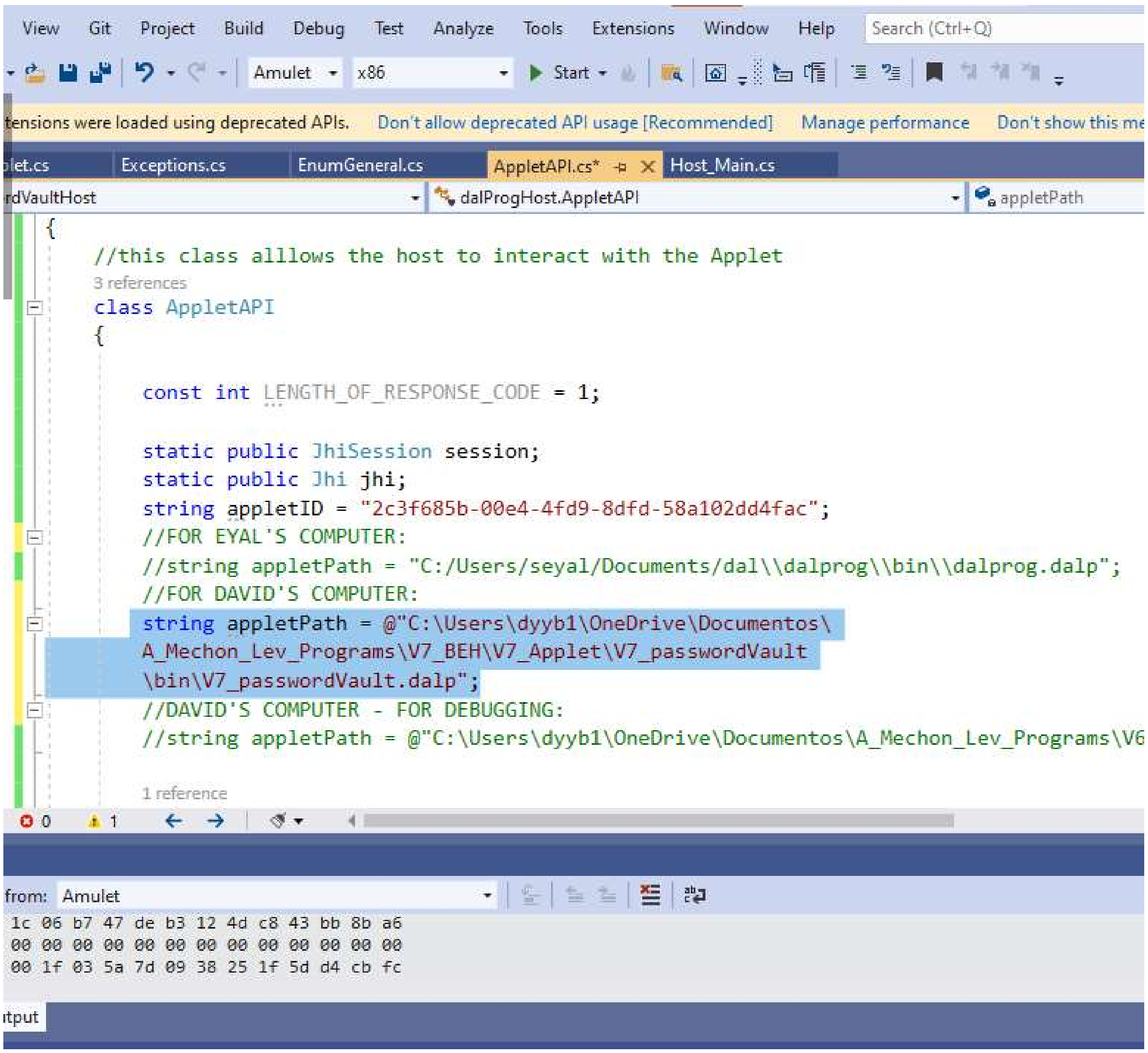
(1) Ensure that Intel's DAL is working, and compile the program into an **".dalp"** file. **Increase the flash storage size** in the Manifest:



* Applet ID	2c3f685b-00e4-4fd9-8dfd-58a102dd4fac
* Applet Name	V7_passwordVault
* Applet Vendor	Intel
* Applet Version	1.0
* Security Version	1
* Heap Size	50000
Applet Description	This is the V7_passwordVault applet
Main entry class	com.Applet_Package
Applet Flash Quota	2000
Enable Debug	true
Applet timeout	

```
<terminated> C:\Users\dyyb1\OneDrive\Documentos\A_Mechon_Lev_Programs\V7_BEH\V7_Applet\V7_passworc
        [echo] Pack file was created successfully.

postBuild:
        [echo] Build process was finished successfully.
BUILD SUCCESSFUL
Total time: 22 seconds
```

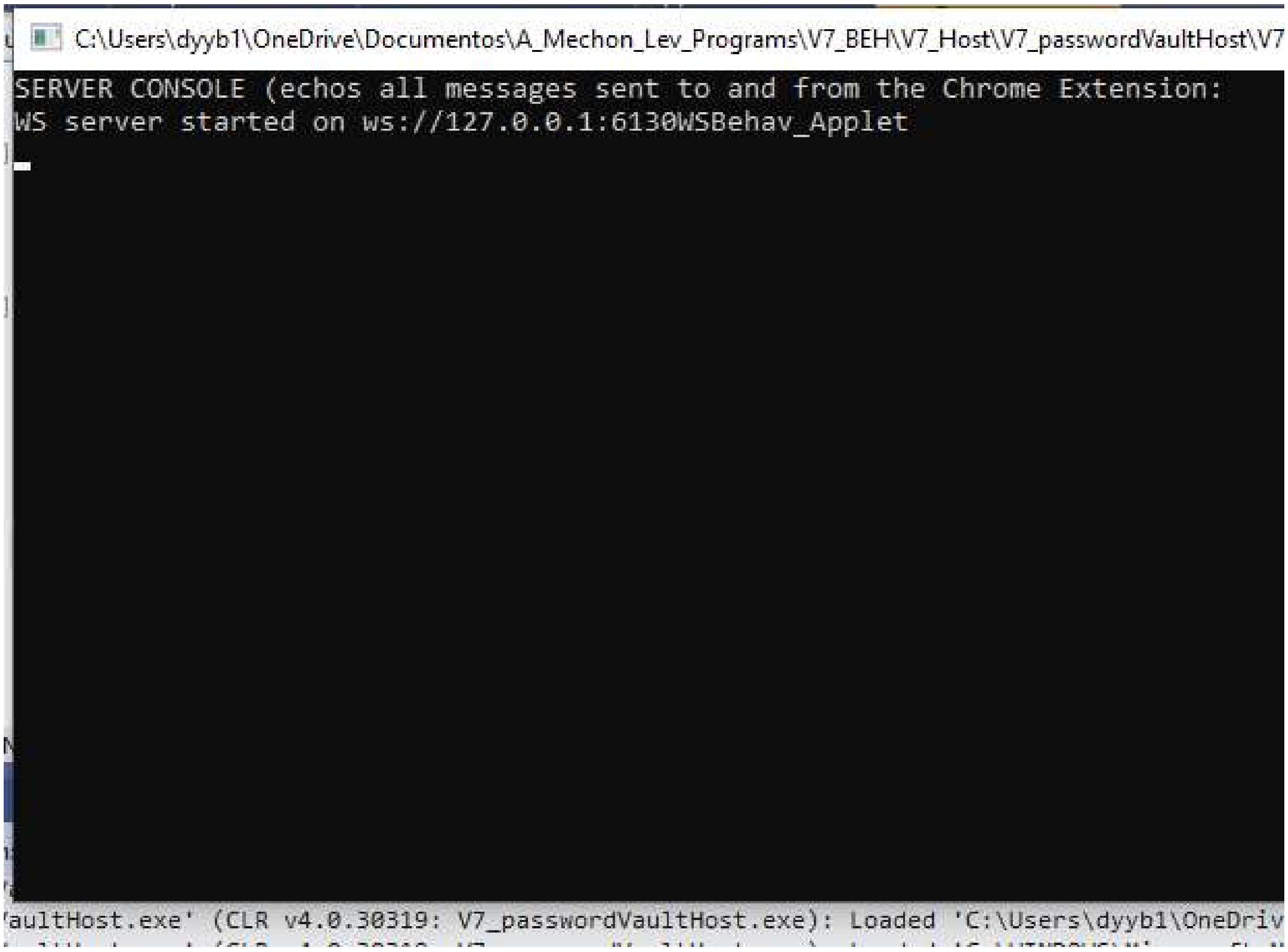
(2) Set up the Host program, ensure that the **applet file path** (must be for local computer!) and **applet Id** are correct


View Git Project Build Debug Test Analyze Tools Extensions Window Help Search (Ctrl+Q)
Amulet x86 Start
tensions were loaded using deprecated APIs. Don't allow deprecated API usage [Recommended] Manage performance Don't show this me
let.cs Exceptions.cs EnumGeneral.cs AppletAPI.cs* Host_Main.cs
rdVaultHost dalProgHost.AppletAPI appletPath

```
{
    //this class allows the host to interact with the Applet
    3 references
    class AppletAPI
    {

        const int LENGTH_OF_RESPONSE_CODE = 1;

        static public JhiSession session;
        static public Jhi jhi;
        string appletID = "2c3f685b-00e4-4fd9-8dfd-58a102dd4fac";
        //FOR EYAL'S COMPUTER:
        //string appletPath = "C:/Users/seyal/Documents/dal\\dalprog\\bin\\dalprog.dalp";
        //FOR DAVID'S COMPUTER:
        string appletPath = @"C:\Users\dyyb1\OneDrive\Documentos\
        A_Mechon_Lev_Programs\V7_BEH\V7_Applet\V7_passwordVault
        \bin\V7_passwordVault.dalp";
        //DAVID'S COMPUTER - FOR DEBUGGING:
        //string appletPath = @"C:\Users\dyyb1\OneDrive\Documentos\A_Mechon_Lev_Programs\V6

        1 reference
```

```
1c 06 b7 47 de b3 12 4d c8 43 bb 8b a6
00 00 00 00 00 00 00 00 00 00 00 00 00
00 1f 03 5a 7d 09 38 25 1f 5d d4 cb fc
```

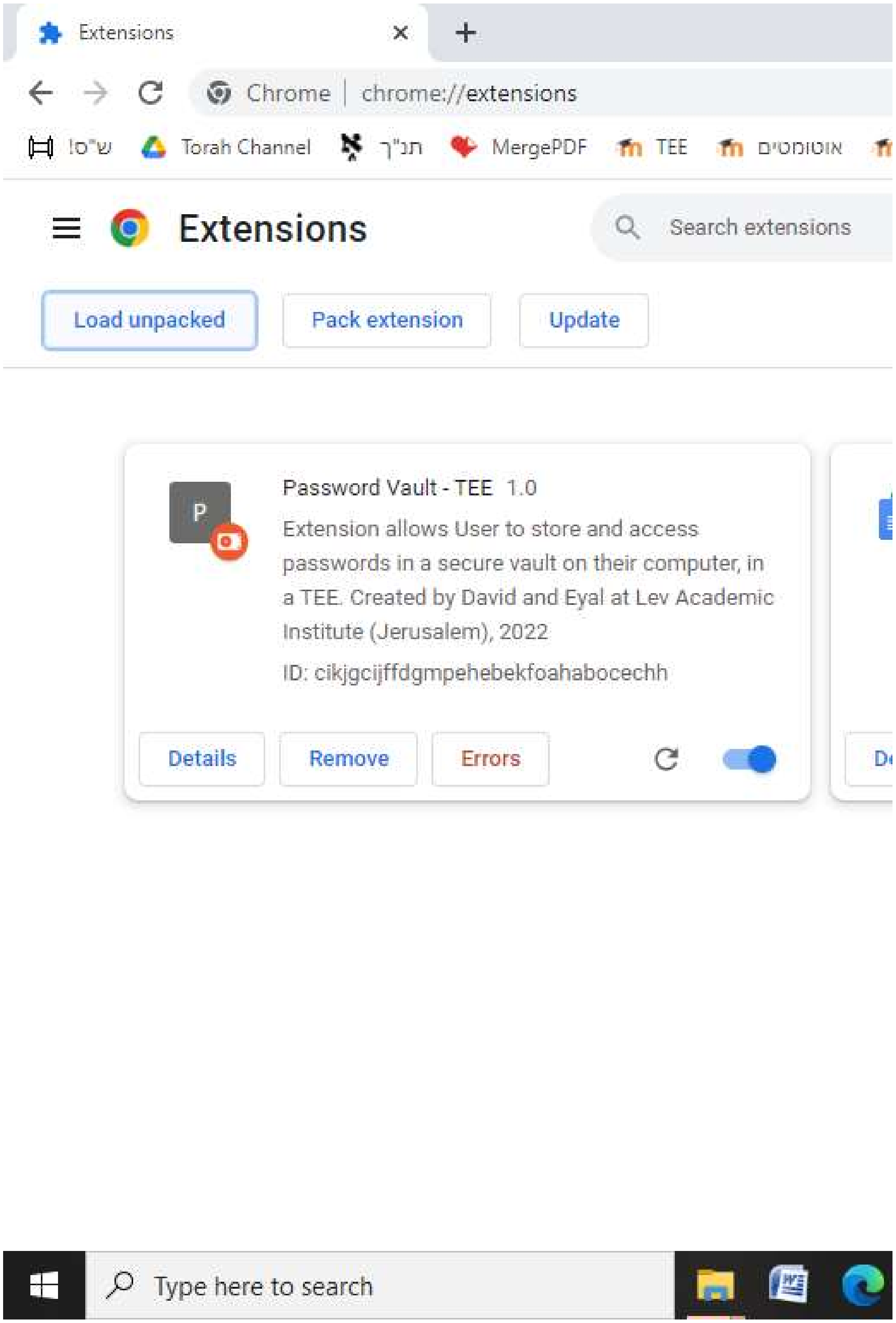
(3) Run the Server (from the Host application)



```
C:\Users\dyyb1\OneDrive\Documentos\A_Mechon_Lev_Programs\V7_BEH\V7_Host\V7_passwordVaultHost\V7
SERVER CONSOLE (echos all messages sent to and from the Chrome Extension:
WS server started on ws://127.0.0.1:6130WSBehav_Applet
```

```
aultHost.exe' (CLR v4.0.30319: V7_passwordVaultHost.exe): Loaded 'C:\Users\dyyb1\OneDriv
```

(4) Upload the extension’s folder to your google account (**"load unpacked"**)

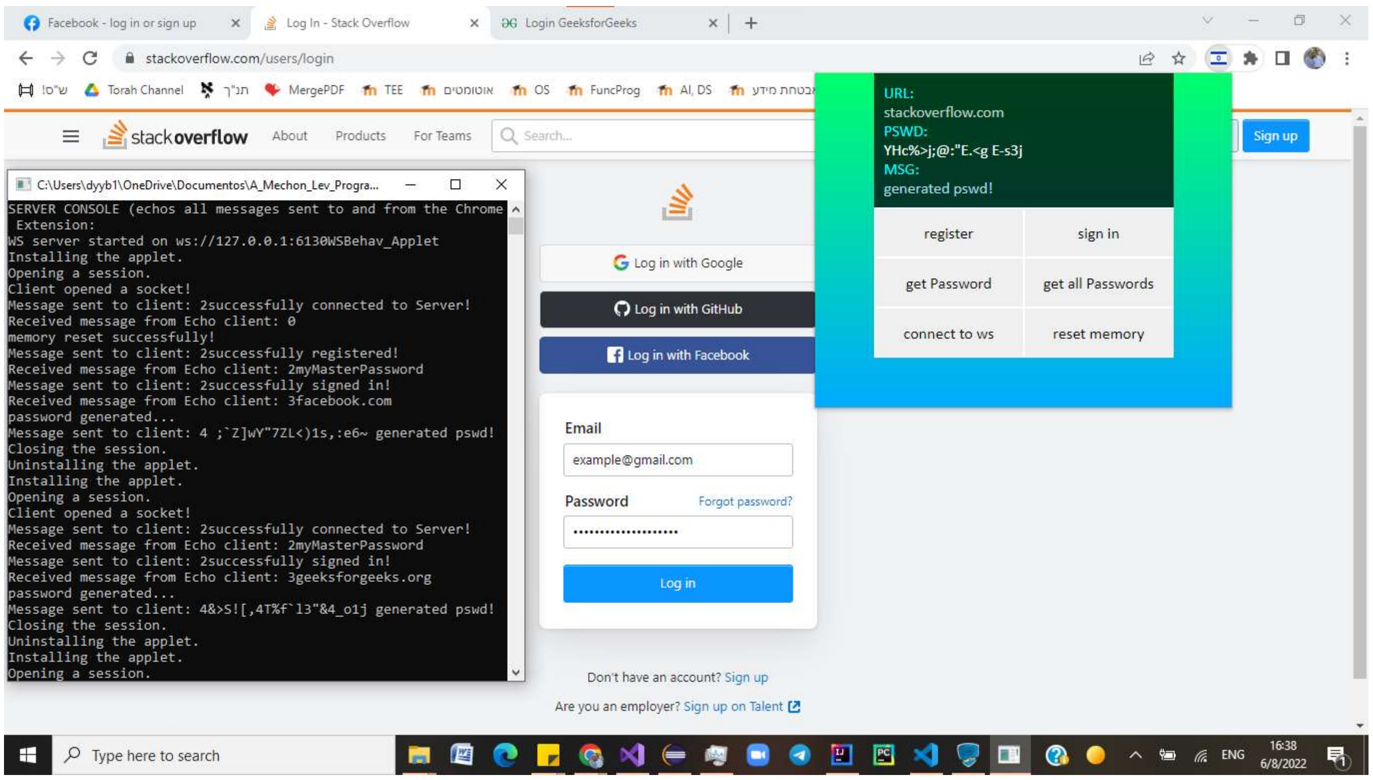


Password Vault - TEE 1.0

Extension allows User to store and access passwords in a secure vault on their computer, in a TEE. Created by David and Eyal at Lev Academic Institute (Jerusalem), 2022

ID: cikjgcijffdgmpehebekfoahabocechh

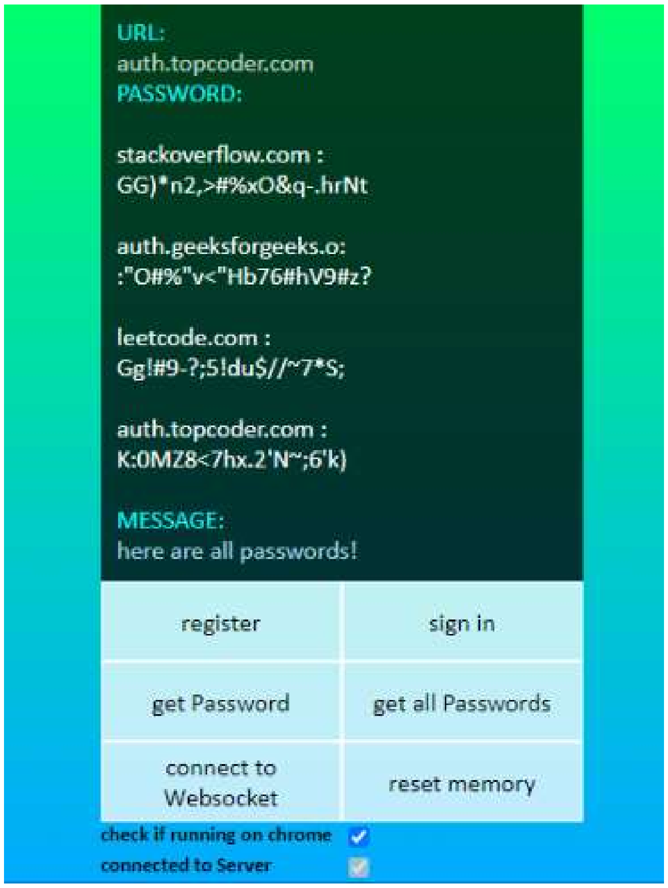
(5) Enjoy



URL:
stackoverflow.com
PSWD:
YHc%>j;@:"E.<g E-s3j
MSG:
generated pswd!

C:\Users\dyyb1\OneDrive\Documentos\A_Mechon_Lev_Progra...

```
SERVER CONSOLE (echos all messages sent to and from the Chrome
 Extension:
WS server started on ws://127.0.0.1:6130WSBehav_Applet
Installing the applet.
Opening a session.
Client opened a socket!
Message sent to client: 2successfully connected to Server!
Received message from Echo client: 0
memory reset successfully!
Message sent to client: 2successfully registered!
Received message from Echo client: 2myMasterPassword
Message sent to client: 2successfully signed in!
Received message from Echo client: 3facebook.com
password generated...
Message sent to client: 4 ;`Z]wY"7ZL<)1s,:e6~ generated pswd!
Closing the session.
Uninstalling the applet.
Installing the applet.
Opening a session.
Client opened a socket!
Message sent to client: 2successfully connected to Server!
Received message from Echo client: 2myMasterPassword
Message sent to client: 2successfully signed in!
Received message from Echo client: 3geeksforgeeks.org
password generated...
Message sent to client: 4&>S![,4T%f`13"&4_o1j generated pswd!
Closing the session.
Uninstalling the applet.
Installing the applet.
Opening a session.
```



URL:
auth.topcoder.com
PASSWORD:

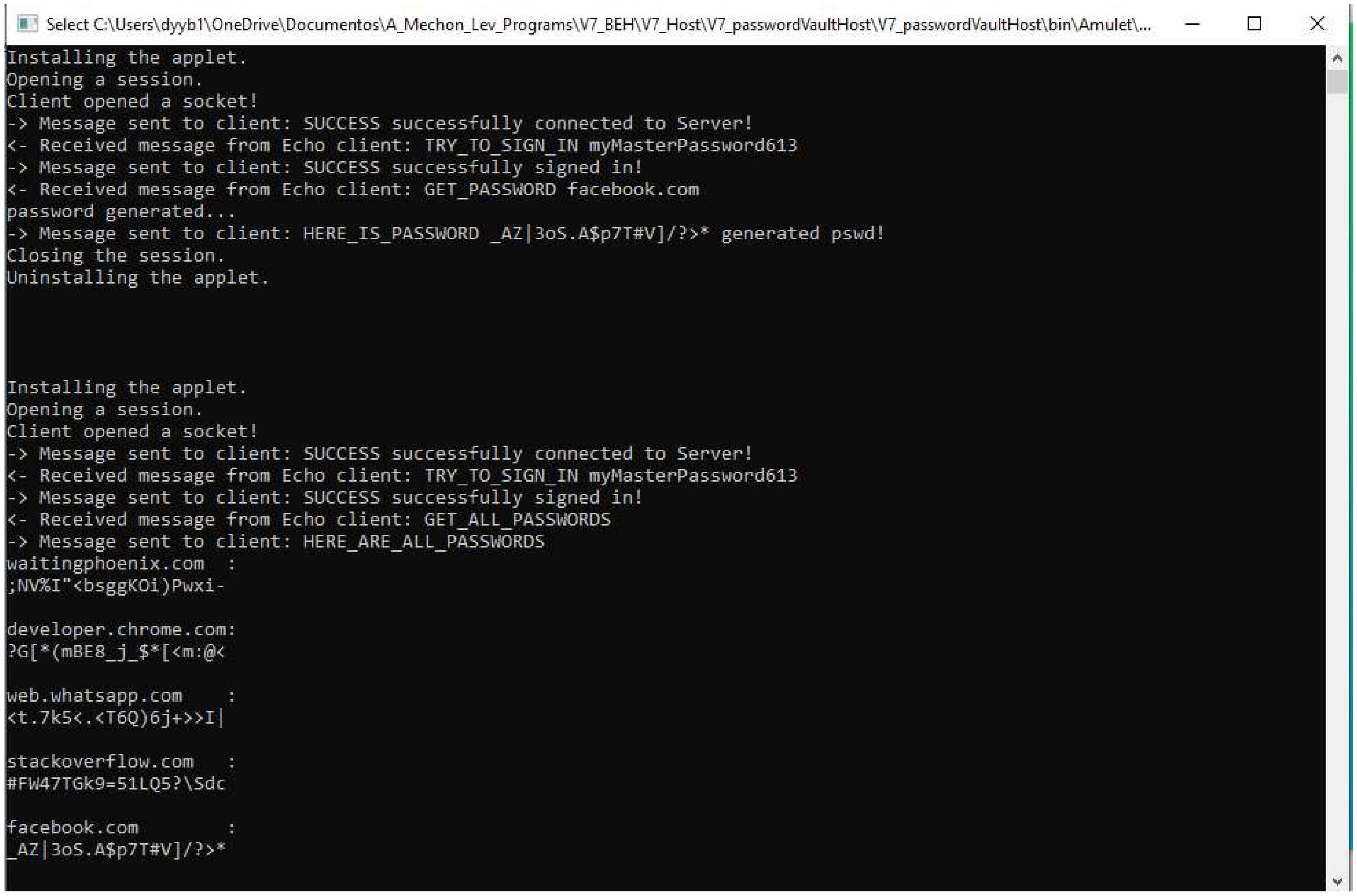
stackoverflow.com :
GG)*n2,>#%xO&q-.hrNt

auth.geeksforgeeks.o:
:"O#%"v<"Hb76#hV9#z?

leetcode.com :
Gg!#9-?;5!du$//~7*S;

auth.topcoder.com :
K:0MZ8<7hx.2'N~;6'k)

MESSAGE:
here are all passwords!



Select C:\Users\dyyb1\OneDrive\Documentos\A_Mechon_Lev_Programs\V7_BEH\V7_Host\V7_passwordVaultHost\V7_passwordVaultHost\bin\Amulet\...

```
Installing the applet.
Opening a session.
Client opened a socket!
-> Message sent to client: SUCCESS successfully connected to Server!
<- Received message from Echo client: TRY_TO_SIGN_IN myMasterPassword613
-> Message sent to client: SUCCESS successfully signed in!
<- Received message from Echo client: GET_PASSWORD facebook.com
password generated...
-> Message sent to client: HERE_IS_PASSWORD _AZ|3oS.A$p7T#V]/?>* generated pswd!
Closing the session.
Uninstalling the applet.



Installing the applet.
Opening a session.
Client opened a socket!
-> Message sent to client: SUCCESS successfully connected to Server!
<- Received message from Echo client: TRY_TO_SIGN_IN myMasterPassword613
-> Message sent to client: SUCCESS successfully signed in!
<- Received message from Echo client: GET_ALL_PASSWORDS
-> Message sent to client: HERE_ARE_ALL_PASSWORDS
waitingphoenix.com  :
;NV%I"<bsggKOi)Pwxi-

developer.chrome.com:
?G[*(mBE8_j_$*[<m:@<

web.whatsapp.com    :
<t.7k5<.<T6Q)6j+>>I|

stackoverflow.com   :
#FW47TGk9=51LQ5?\Sdc

facebook.com        :
_AZ|3oS.A$p7T#V]/?>*
```

Points about the source code:

- Relevant **Exceptions** in the Host
- Clear, **Universal Emuns**
- **Organized API's**, black box (Extension users "websocketAPI", Host users "AppletAPI", Applet uses "FlashStorageAPI")

Future ideas:

☐ Enabling Multiple users (on one computer) to use the TEE

☐ A better random seed (ex physical world generated instead of time...)

☐ Allowing User to hide the Console's Echo

☐ Storing both the password and username with the URL

☐ Running a background script, offering User to save password in the Vault whenever User enters a password on a website

☐ Encrypting or Signing (or both) communication (Host <-> Applet, and Host <-> Chrome Extension)