CSE 509: System Security Project Report

Malicious Browser Extension with Server Backend

Team Name: Soteria
Members:
Arpith Varghese (110872883)
Aviral Nigam (110849584)
Piyush Shyam Banginwar (110944214)
Shreyas B Prabhu (110899297)

Contents

1 Introduction **2**

2 Client Side **3**
2.1 Components . . . 3
2.1.1 The Manifest File . . . 3
2.1.2 The Background Page . . . 3
2.1.3 UI Pages . . . 3
2.1.4 Content Scripts . . . 3

3 Server side **4**
3.1 Components . . . 4
3.1.1 Elasticsearch . . . 4
3.1.2 Kibana . . . 4
3.1.3 X-Pack Security . . . 4
3.1.4 Sense . . . 4
3.2 Usage . . . 4

4 Architecture **5**
4.1 Overview . . . 5
4.1.1 Explanation . . . 6
4.2 Database Schema . . . 6
4.2.1 Database Schema Overview . . . 6

5 Design **8**
5.1 Useful Functionality . . . 8
5.2 General Design considerations . . . 8
5.3 Leak the browsing history of a user to the attacker's server . . . 8
5.4 Steal usernames and passwords from forms as the user writes them . . . 8
5.5 Steal cookies from outgoing HTTP requests . . . 8
5.6 Stop the user from visiting security websites and reroute them to random websites . . . 9
5.7 Personalized Phishing . . . 9
5.8 JavaScript execution . . . 9
5.9 Communication with Web Pages . . . 9

6 Visualization **10**
6.1 User Information . . . 10

7 Snapshots **12**

Chapter 1

Introduction

Extensions are small software programs that can modify and enhance the functionality of the web browser [1]. Browser extensions have access to everything done by the browser, and can do things like inject ads into webpages, or make "background" HTTP requests to a third-party server. This power can be abused by browser extensions; while Web pages are constrained by the security model of the Web, extensions are not.

For this project we have created a malicious browser extension with functionalities listed below. We picked Google Chrome for our development purpose as it has the highest share of users globally [2]. The functionalities implemented here can be easily extended to other web browsers.

Functionalities:

- Leak browsing history of a user to our server.
- Steal usernames and passwords from forms as the user writes them.
- Steal cookies from outgoing HTTP requests.
- Reroute user from security websites to random websites. The list of such websites being dynamically updatable.
- Dynamic insertion of DOM Objects in a users web page, which can be used for personalised phishing.
- Dynamic insertion of Javascript Objects to steal user information from a web page of choice.

The development has been done using web technologies such as HTML, JavaScript, and CSS. We provide below with the structure of our Client and Server side system, along with the Design and Architecture we have followed to develop our extension.

Chapter 2

Client Side

2.1 Components

2.1.1 The Manifest File

The manifest file, called manifest.json, gives information about the extension, such as the most important files and the capabilities that the extension might use.

2.1.2 The Background Page

Background page of our extension is an invisible page that holds the main logic and control of the extension. This includes all the listeners to different types of events in Javascript to act according to our features.

2.1.3 UI Pages

Extensions can contain ordinary HTML pages that display the extension's UI. For example, a browser action can have a popup, which is implemented by an HTML file. In our case, we keep no UI for the extension to keep it inconspicuous but we instead provide a good user functionality which is popular on the Chrome store.

2.1.4 Content Scripts

A content script is some JavaScript that executes in the context of a page that's been loaded into the browser. Content scripts can read details of the web pages the browser visits, and they can make changes to the pages. This behavior of content scripts makes them an integral part of our implementation.

Chapter 3

Server side

3.1 Components

3.1.1 Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases.

3.1.2 Kibana

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch.

3.1.3 X-Pack Security

X-Pack security features give the right access to the right people.

3.1.4 Sense

Sense is a handy console for interacting with the REST API of Elasticsearch.

3.2 Usage

To store the information extracted from the client and extension configurations, we use elasticsearch. The main reason behind using elasticsearch is that it provides fast read/writes, its highly scalable, distributed and provides an inbuilt REST interface for interaction.

We also use Kibana to visualize the data which is the dashboarding that comes with elasticsearch. Both of these above technologies are combined with X-Pack Security so as to protect them against any un-authorized usage. Native elasticsearch and kibana do not have an inbuilt authentication module, making them insecure for usage, hence Shield needs to be integrated with them to compensate for the lack in security.

Sense plugin within kibana has been used as a REST client to update and extract data to/from elasticsearch. Since elasticsearch provides REST interface, Sense could easily be replaced by another REST client e.g., Postman, curl, etc.

Chapter 4

Architecture

4.1 Overview

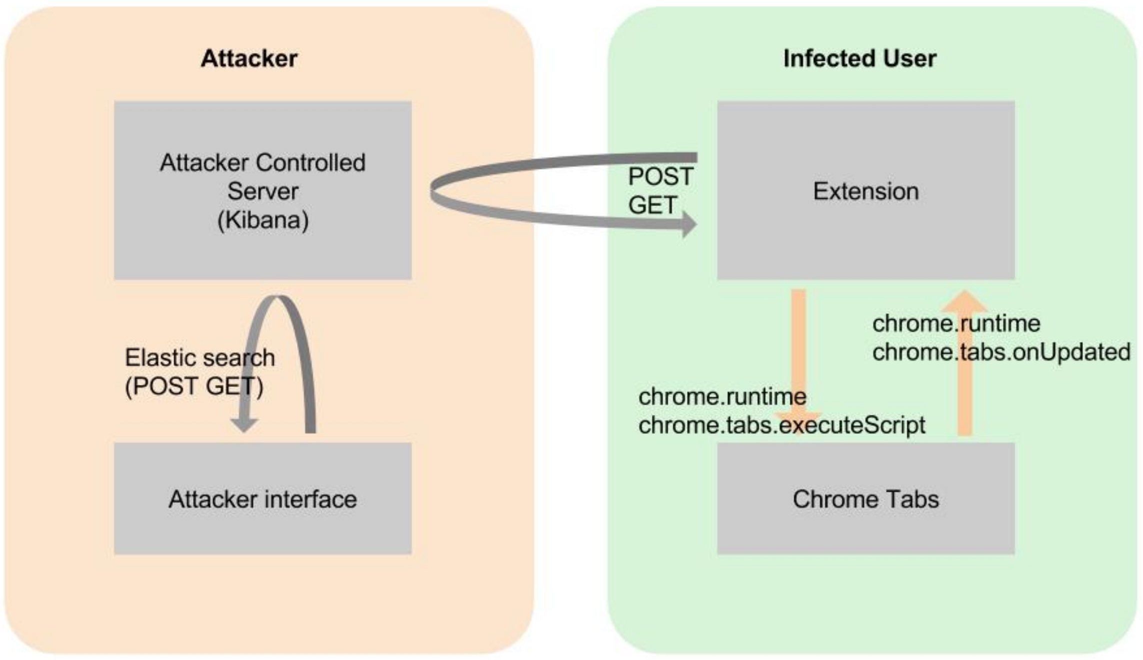


Figure 4.1: Architecture Overview

4.1.1 Explanation

The system can be divide in to two major parts, the client or victim side and the server or attacker side. The client has the extension installed on his/her browser which keeps track of every required events or actions being performed by the user on the browser tabs. The extension then sends the information to an attacker controlled server (Kibana in our case) via network requests, which stores all the data received. The attacker has control to this server and can pull/push data to it using a REST interface.

4.2 Database Schema

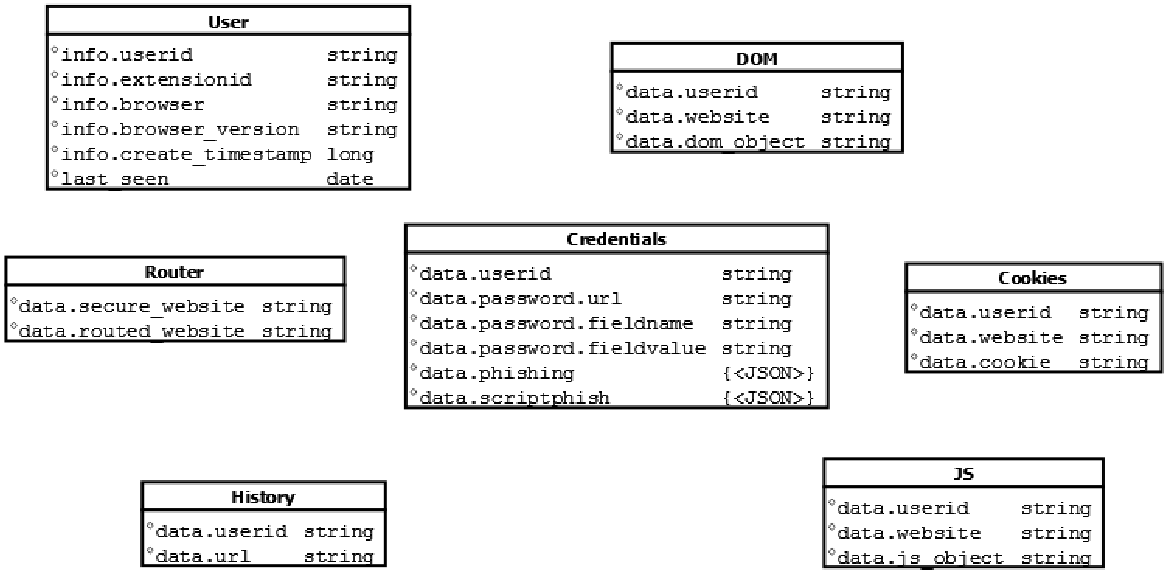


Figure 4.2: Schema

4.2.1 Database Schema Overview

User Table

The table stores all the informational data about a user, such as the unique user ID, the extension ID on their system, the Browser used, Broswer Version, extension installation timestamp and the last time the user was seen/online.

Router

This table maintains a global list of security websites and the link they should be rerouted to, if an user visits any of these links. The table is user agnostic.

History

This table maintains the History of each user in the form of a time stamp and the link visited as key value pairs.

Credentials

This maintains all the stolen credentials of each user by the client side.

Credentials

Similar to credentials, this one stores all the stolen cookies.

DOM

This is a table of all the DOM objects that can be inserted in the user's webpage and is stored in the form strings per userID and weblink combination.

JS

Similar to DOM, this has the Javascript objects in form of strings to be injected to user webpages.

Chapter 5

Design

5.1 Useful Functionality

It is important that our extension provides some useful functionality in order to both convince users to install our extension and to hide the malicious nature of the extension. We did a quick survey of popular extensions on Chrome Store and found that extensions which replace the new tab page with a dashboard are quite popular. So we decided to create an extension which replaces the new tab page with a custom page that displays a daily motivational quote along with a nice background image every time a user opens a new tab.

5.2 General Design considerations

We want to uniquely identify users when their data is sent to the server. So we generate a user-id when the extension is first installed and store it in Chrome Sync storage . This user-id is sent along with every request to the server. We also store the extension-id, which is necessary to communicate between webpages and the extension.

5.3 Leak the browsing history of a user to the attacker's server

Chrome provides an api chrome.history.search which retrieves the history of the user given a query parameter. We use it to get all the history of the user when the extension is first installed and send it to the server, afterwards every time the user visits a new page, we send the corresponding url and the timestamp in epoch to the server. History is stored as a json object containing the timestamp followed by the visited url.

5.4 Steal usernames and passwords from forms as the user writes them

The extension listens to the event chrome.webNavigation.onCompleted and on event trigger, it checks if the webpage contains any form with the tag password. If a matching form is found then we add an event listener to log keystrokes for each of the input fields in this form. This data is sent to the extension which eventually saves it to the server.

5.5 Steal cookies from outgoing HTTP requests

We have added a listener that gets invoked every time a header is created and sent. We inspect the header to check if it contains a cookie field, and if so we save it in the server along with the website name.

5.6 Stop the user from visiting security websites and reroute them to random websites

There is a mapping of security urls and the corresponding reroute urls stored in the server. We sync this information with our local chrome storage every minute. Each time a new request is generated, we intercept the request, extract the request url and compare it with our blacklist of security websites, and if it matches we reroute the user to a different web-page.

5.7 Personalized Phishing

DOM Object Insertion: For each user, we store a mapping of the website and the DOM object to be inserted. This information is retrieved on every user web request. If the user requested a page in our mapping then we add our custom DOM to the DOM of the requested page.

5.8 JavaScript execution

Javascript insertion: Similar to DOM mapping, we store a mapping of the website and the javascript to be executed for each user. This information is retrieved on every user web request and executed on the tab that is requesting this information using the eval function.

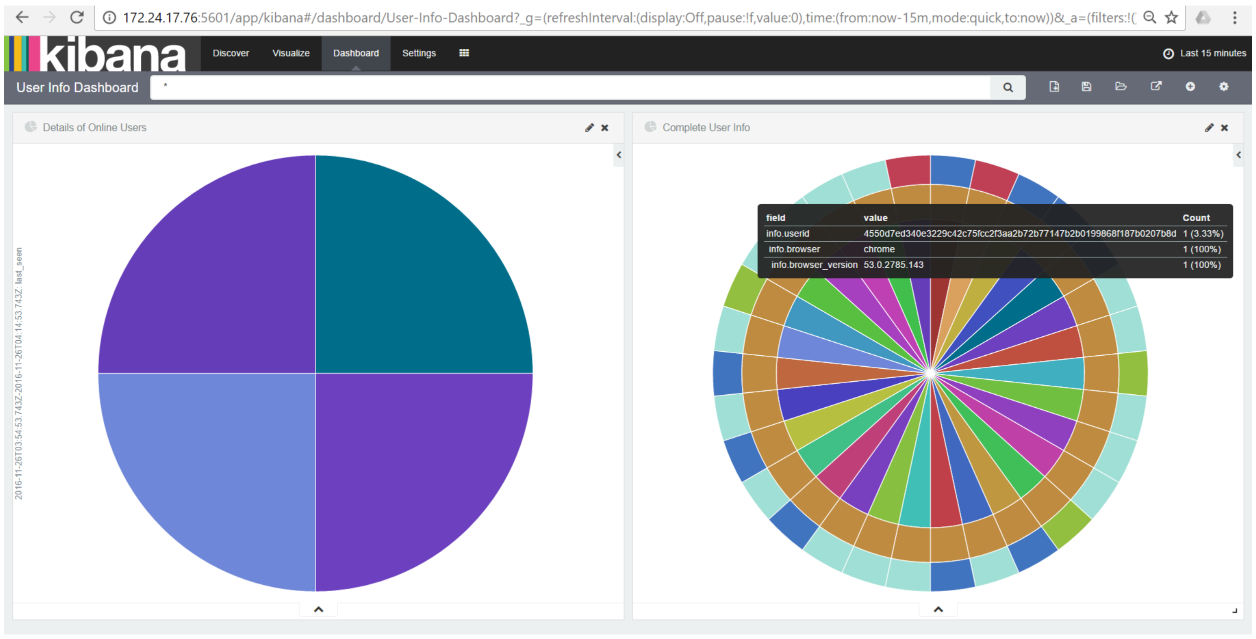
5.9 Communication with Web Pages

As part of supported functionalities, we need a medium of communication between the pages of the extension and also between a webpage and the extension. We use the chrome APIs chrome.runtime.onConnect/onConnectExternal and port.onMessage to establish a communication channel between two parties, which is essential is use-cases such as phishing/stealing user data from a website.

Chapter 6

Visualization

6.1 User Information

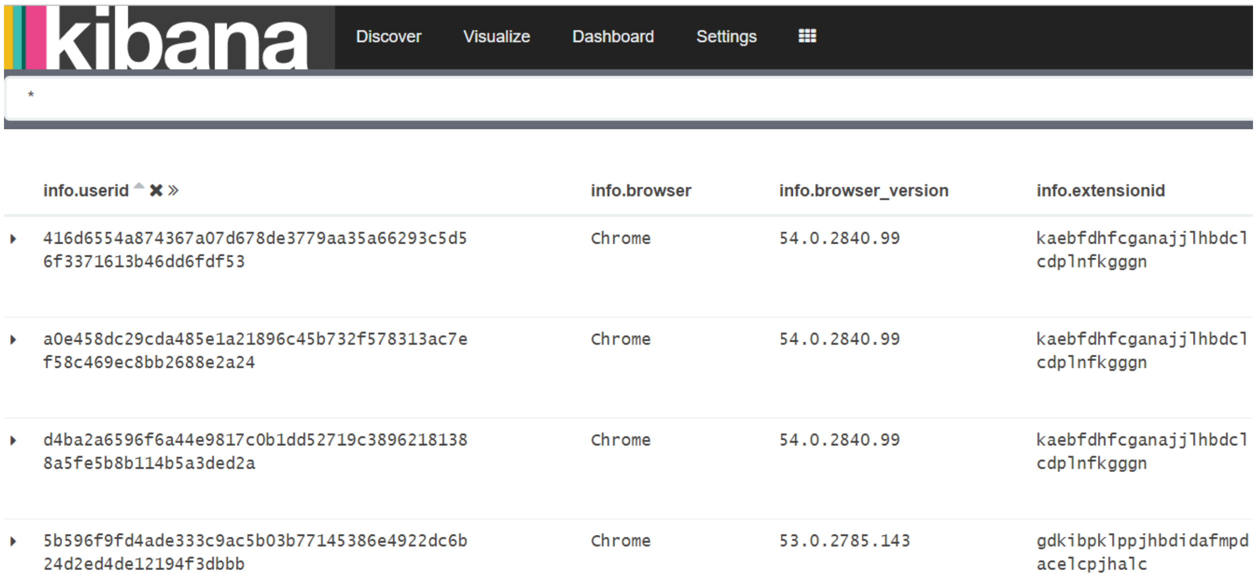


172.24.17.76:5601/app/kibana#/dashboard/User-Info-Dashboard?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-15m,mode:quick,to:now))&_a=(filters:!(
kibana
Discover
Visualize
Dashboard
Settings
Last 15 minutes
User Info Dashboard
Details of Online Users
Complete User Info
2016-11-26T03:54:53.743Z-2016-11-26T04:14:53.743Z: last_seen

field	value	Count
info.userid	4550d7ed340e3229c42c75fcc2f3aa2b72b77147b2b0199868f187b0207b8d	1 (3.33%)
info.browser	chrome	1 (100%)
info.browser_version	53.0.2785.143	1 (100%)

Figure 6.1: Dashboard

Kibana provides us with Dash Boarding capabilities, one example of which can be seen above. The above graph shows the current online users and the relevant info associated with them. While the right picture shows the cookies collected for each user and the urls the cookies are associated with. The visualization can also be done in a simpler tabular form with the correct filters configured.

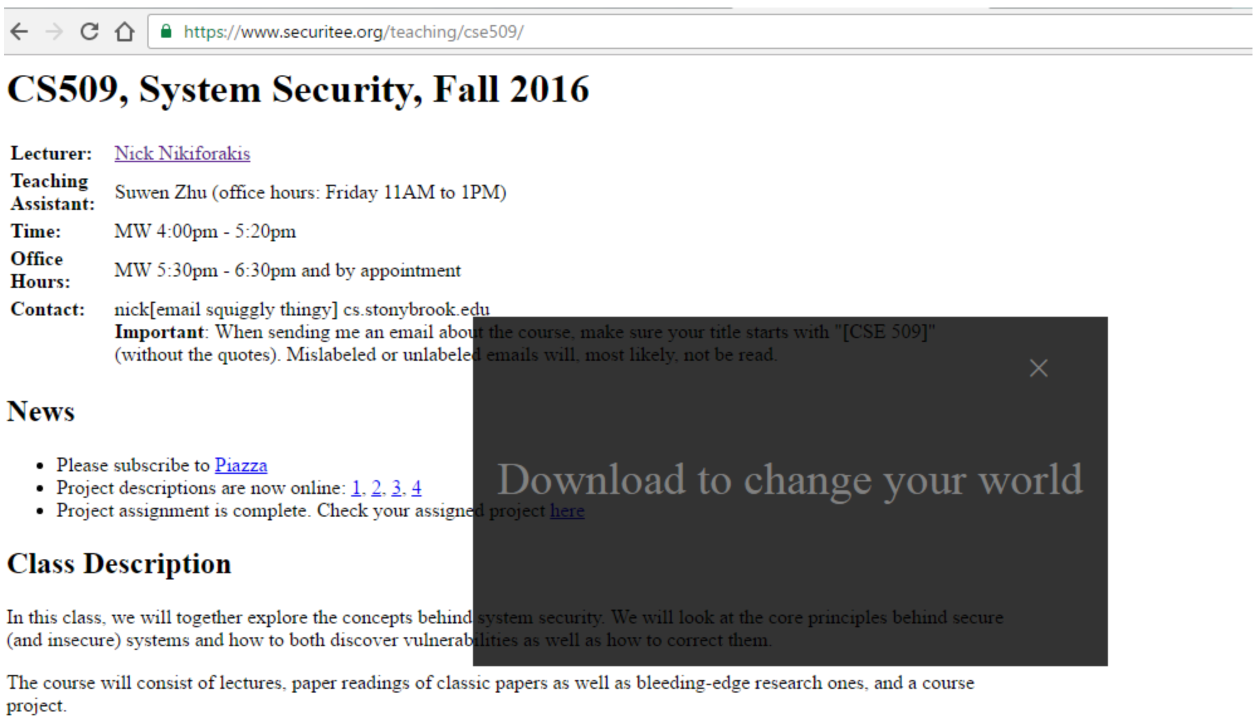


info.userid	info.browser	info.browser_version	info.extensionid
416d6554a874367a07d678de3779aa35a66293c5d5 6f3371613b46dd6fdf53	Chrome	54.0.2840.99	kaebfdhfcganajjlhbdcl cdplnfkgggn
a0e458dc29cda485e1a21896c45b732f578313ac7e f58c469ec8bb2688e2a24	Chrome	54.0.2840.99	kaebfdhfcganajjlhbdcl cdplnfkgggn
d4ba2a6596f6a44e9817c0b1dd52719c3896218138 8a5fe5b8b114b5a3ded2a	Chrome	54.0.2840.99	kaebfdhfcganajjlhbdcl cdplnfkgggn
5b596f9fd4ade333c9ac5b03b77145386e4922dc6b 24d2ed4de12194f3dbbb	Chrome	53.0.2785.143	gdkibpklppjhbdidafmpd acelcpjhalc

Figure 6.2: An example of a dashboard created with filters on Kibana to show User Info.

Chapter 7

Snapshots



https://www.securitee.org/teaching/cse509/

CS509, System Security, Fall 2016

Lecturer:	Nick Nikiforakis
Teaching Assistant:	Suwen Zhu (office hours: Friday 11AM to 1PM)
Time:	MW 4:00pm - 5:20pm
Office Hours:	MW 5:30pm - 6:30pm and by appointment
Contact:	nick[email squiggly thingy] cs.stonybrook.edu **Important**: When sending me an email about the course, make sure your title starts with "[CSE 509]" (without the quotes). Mislabeled or unlabeled emails will, most likely, not be read.

News

- Please subscribe to Piazza
- Project descriptions are now online: 1, 2, 3, 4
- Project assignment is complete. Check your assigned project here

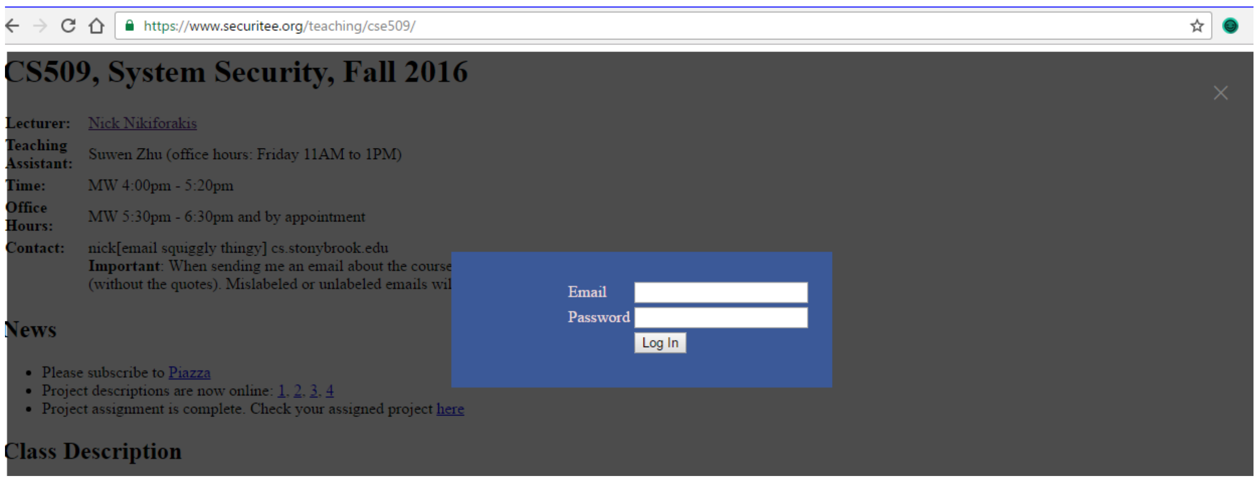
Class Description

In this class, we will together explore the concepts behind system security. We will look at the core principles behind secure (and insecure) systems and how to both discover vulnerabilities as well as how to correct them.

The course will consist of lectures, paper readings of classic papers as well as bleeding-edge research ones, and a course project.

Download to change your world

Figure 7.1: A snapshot of a DOM Object Insertion.



https://www.securitee.org/teaching/cse509/

CS509, System Security, Fall 2016

Lecturer: Nick Nikiforakis
Teaching Assistant: Suwen Zhu (office hours: Friday 11AM to 1PM)
Time: MW 4:00pm - 5:20pm
Office Hours: MW 5:30pm - 6:30pm and by appointment
Contact: nick[email squiggly thingy] cs.stonybrook.edu
Important: When sending me an email about the course
(without the quotes). Mislabeled or unlabeled emails wil

News

- Please subscribe to Piazza
- Project descriptions are now online: 1, 2, 3, 4
- Project assignment is complete. Check your assigned project here

Class Description

Figure 7.2: Snapshot of a phishing attempt.

Bibliography

[1] Chrome Developer Documentation: https://developer.chrome.com/extensions/getstarted

[2] Top Browser Usage: http://gs.statcounter.com/#all-browser-ww-monthly-201608-201608-bar

[3] Elastic Search: http://www.elastic.co/